38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science

FSTTCS 2018, December 11–13, 2018, Ahmedabad, India

Edited by

Sumit Ganguly

Paritosh Pandya

Editors

Sumit Ganguly
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur, India
sganguly@cse.iitk.ac.in

Paritosh Pandya
Tata Institute of Fundamental Research
Mumbai, India
pandya@tifr.res.in

ACM Classification 2012
Theory of Computation

ISBN 978-3-95977-093-4

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-95977-093-4.

Publication date
December, 2018

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.2018.0

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

http://www.dagstuhl.de/lipics

Contents

38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018).
Editors: Sumit Ganguly and Paritosh Pandya
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This volume constitutes the proceedings of the 38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018) held at Ahmedabad University, Ahmedabad, India from December 10 to December 14, 2017. The FSTTCS conferences are organized annually by the Indian Association for Research in Computing Science (IARCS). The proceedings of FSTTCS 2018 is published as a volume in the LIPIcs series under a Creative Commons license, with free online access to all.

The conference comprised of 4 invited talks and 44 contributed papers. This volume contains the contributed papers and abstracts of invited talks presented at the conference. The contributed papers were selected from a total of 123 submissions. We are grateful to the programme committee for its efforts in the evaluation of the submissions and selection of papers. We also thank the external reviewers for sending their informative and timely reviews. Further, we thank all authors who submitted their work to FSTTCS 2018. We are especially thankful to the invited speakers: *Rupak Majumdar* (MPI-SWS, Saarbrucken, Germany), *A. Prasad Sistla* (University of Illinois, Chicago, USA), *Ola Svensson* (EPFL, Lausanne, Switzerland) and *Santosh Vempala* (Georgia Tech., Atlanta, USA).

The conference had a pre-conference workshop on *Trends in Transformations* organized by *Paul Gastin* (ENS de Cachan, France) and *S.N. Krishna* (IIT Bombay, India). We thank the organizers of the workshop and the speakers in it.

The organizing committee of the conference from Ahmedabad University were responsible for the local and technical arrangements that led to the smooth running of the conference. We thank them for their invaluable efforts. We thank *Easychair* for the conference management tool used for the submission and review process. Finally, we thank Dagstuhl publications for the publication of these proceedings.

Sumit Ganguly and Paritosh Pandya

38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018).
Editors: Sumit Ganguly and Paritosh Pandya
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018).
Editors: Sumit Ganguly and Paritosh Pandya

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

List of Reviewers

Abel Molina
Abhishek Sahu
Abhishek Shetty
Achim Blumensath
Ahmed Bouajjani
Aida Mousavifar
Akash Kumar
Aldric Degorre
Alexander Rabinovich
Amaldev Manuel
Ameya Velingker
Amit Sinhababu
Anand Louis
André van Renssen
Andreas Pavlogiannis
Anil Shukla
Anirban Dasgupta
Anna Adamaszek
Anupam Gupta
Arijit Ghosh
Arindam Khan
Ashish Chiplunkar
Ashish Dwivedi
Ashutosh Gupta
Ashutosh Kumar
B Srivathsan
Benoit Groz
Bharat Adsul
Bjoern Lellmann
Blaise Genest
Bodhayan Roy
C. Aiswarya
Caucal
César Rodríguez
Chaitanya Swamy
Chandra Chekuri
Chandra Chekuri
Chandra Chekuri
Chien-Chung Huang
Chris Schwiegelshohn
Christoph Haase
Damian Straszak
Dániel Marx
David Manlove
Deepak D'Souza
Dejan Jovanović
Dietmar Berwanger
Dietrich Kuske
Emmanuel Filiot
Eugene Stark
Eva-Maria Hols
Fahad Panolan
Francesco Belardinelli
Frank Kammer
Geevarghese Philip
Georg Zetzsche
Gilles Geeraerts
Giorgio Delzanno
Gökalp Demirci
Gopal Pandurangan
Gopalan Nadathur
Hartmut Klauck
Havana Rika
Hrishikesh Karmarkar
Hui Kong
Ismaël Jecker
Jaikumar Radhakrishnan
James Brotherston
Jan Kretinsky
Jan Otop
Jannik Dreier
Jason Reed
Jayesh Chaudhuri
Jean Goubault-Larrecq
Jean-Francois Raskin
Jean-Jacques Levy
Jesper Nederlof
Jiehua Chen
Joachim Gudmundsson
John Kallaugher
Jugal Garg
K. Narayan Kumar
Kamal Lodaya
Karin Quaas
Kavitha Telikepalli
Kent Quanrud
Krishna S
Krishnendu Chatterjee
Laszlo Kozma
Laszlo Vegh

Laure Daviaud
Laure Daviaud
Lingxiao Huang
Luc Dartois
M. Praveen
Madhavan Mukund
Madhukar Y
Magnus Wahlström
Mahesh Viswanathan
Manoj Gupta
Matthew Bauer
Meena Mahajan
Michael Kapralov
Michał Pilipczuk
Mihaela Sighireanu
Minati De
Mirco Giacobbe
Mitchell Jones
Mohamed Faouzi Atig
Mohit Singh
Neeraj Kayal
Nicolas Markey
Nicolas Massocchi
Nikhil Balaji
Nima Roohi
Nisheeth Vishnoi
Nitin Saxena
Noam Touitou
Nutan Limaye
Paritosh Pandya
Parosh Abdulla
Paul Gastin
Paweł Parys
Peter Habermehl
Peter McGlaughlin
Peter Rossmanith
Petr Novotny
Philipp Meyer
Pierre-Alain Reynier
Piyush Kurur
Piyush Srivastava
Pooja Kulkarni
Prahladh Harsha
Prakash Saivasan
Pranav Bisht
R.B. Sandeep
Raghunath Tewari
Rajarshi Ray
Rajat Mittal
Rajesh Chitnis
Rakesh Venkat
Ramchandra Phawade
Ramprasad Saptharishi
Rasmus Ibsen-Jensen
Richard Mayr
Robert Simmons
Rohit Gurjar
Roland Meyer
Rucha Kulkarni
Rüdiger Ehlers
S P Suresh
S. Akshay
Sadra Yazdanbod
Saket Saurabh
Saladi Rahul
Samuel Mimram
Sanath Kumar Krishnamurthy
Sanjiva Prasad
Sanjiva Prasad
Satyadev Nandakumar
Sebastian Arming
Seeun William Umboh
Shibashis Guha
Shweta Agrawal
Siddharth Barman
Siddharth Bhandari
Siu On Chan
Sivakanth Gopi
Soumya Paul
Sourav Chakraborty
Srijita Kundu
Srikanth Srinivasan
Stefan Kratsch
Stefan Schwoon
Subodh Sharma
Sucheendra K. Palaniappan
Sudeshna Kolay
Suguman Bansal
Sumanta Ghosh
Sumit Ganguly
Supratik Chakraborty
Suprovat Ghoshal
Tamás Király
Thomas Ferrere
Thomas Thierauf
Till Fluschnik

Tobias Nipkow
Udi Boker
Uli Fahrenberg
Uli Schlachter
Venkatesh Srinivasan
Yossi Azar
Zhilin Wu

Random Testing for Distributed Systems with Theoretical Guarantees

Rupak Majumdar
Max Planck Institute for Software Systems, Kaiserslautern, Germany
rupak@mpi-sws.org

Abstract

Random testing has proven to be an effective way to catch bugs in concurrent and distributed systems. This is surprising, as the space of executions is enormous and conventional formal methods intuition would suggest that bad behaviors would only be found by extremely unlikely coincidences.

Empirically, many bugs in distributed systems can be explained by interactions among only a small number of features. Thus, one can attempt to explain the effectiveness of random testing under various "small depth" hypotheses. In particular, it may be possible to test all interactions of k features for a small constant k by executing a family of tests that is exponentially or even doubly-exponentially smaller than the family of all tests. Moreover, under certain conditions, a randomly chosen small set of tests is sufficient to cover all k-wise interactions with high probability.

I will describe two concrete scenarios. First, I will describe bugs in distributed systems caused by network partition faults. In many practical instances, these bugs occur due to two or three key nodes, such as leaders or replicas, not being able to communicate, or because the leading node finds itself in a block of the partition without quorum. In this case, I will show using the probabilistic method that a small set of randomly chosen tests will cover all "small partition" scenarios with high probability.

Second, I will consider bugs that arise due to unexpected schedules (interleavings) of concurrent events. Again, many bugs depend only on the relative ordering of a small number of events (the "bug depth" of the bug). In this case, I will show a testing algorithm that prioritizes low depth interleavings and a randomized testing algorithm that bounds the probability of sampling any behavior of bug depth k for a fixed k. The testing algorithm is based on combinatorial insights from the theory of partial orders, such as the notion of dimension and its generalization to d-hitting families as well as results on online chain partitioning.

Beyond the potential for designing or explaining random testing procedures, the technical arguments show the potential of combining "Theory A" and "Theory B" results to the important domain of software testing.

This is joint work primarily with Filip Niksic [1], and with Dmitry Chistikov, Simin Oraee, Burcu Kulahcioglu Özkan, Mitra Tabaei Befrouei, and Georg Weissenbacher. This work was partially funded by an ERC Synergy Award (ImPACT).

2012 ACM Subject Classification Theory of computation → Generating random combinatorial structures, Software and its engineering → Software testing and debugging

Keywords and phrases Random testing, Hitting families

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.1

Category Invited Paper

References

1 Filip Niksic. *Combinatorial Constructions for Effective Testing.* PhD thesis, University of Kaiserslautern, 2018.

Model Checking Randomized Security Protocols

A. Prasad Sistla
University of Illinois at Chicago, USA
sistla@uic.edu

Abstract

The design of security protocols is extremely subtle and is prone to serious faults. Many tools for automatic analysis of such protocols have been developed. However, none of them have the ability to model protocols that use explicit randomization. Such randomized protocols are being increasingly used in systems to provide privacy and anonymity guarantees. In this talk we consider the problem of automatic verification of randomized security protocols. We consider verification of secrecy and indistinguishability properties under a powerful threat model of Dolev-Yao adversary. We present some complexity bounds on verification of these properties. We also describe practical algorithms for checking indistinguishability. These algorithms have been implemented in the tool SPAN and have been experimentally evaluated. The talk concludes with future challenges.

(Joint work with: Matt Bauer, Rohit Chadha and Mahesh Viswanathan)

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Randomized Protocols, Verification

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.2

Category Invited Paper

38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018).
Editors: Sumit Ganguly and Paritosh Pandya; Article No. 2; pp. 2:1–2:1

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

Algorithms for the Asymmetric Traveling Salesman Problem

Ola Svensson[1]
School of Computer and Communication Sciences, EPFL, Switzerland
ola.svensson@epfl.ch
https://orcid.org/0000-0003-2997-1372

Abstract

The traveling salesman problem is one of the most fundamental optimization problems. Given n cities and pairwise distances, it is the problem of finding a tour of minimum total distance that visits each city once. In spite of significant research efforts, current techniques seem insufficient for settling the approximability of the traveling salesman problem. The gap in our understanding is especially large in the general asymmetric setting where the distance from city i to j is *not* assumed to equal the distance from j to i.

Indeed, until recently, it remained an open problem to design an algorithm with *any* constant approximation guarantee. This status is particularly intriguing as the standard linear programming relaxation is believed to give a constant-factor approximation algorithm, where the constant may in fact be as small as 2.

In this talk, we will give an overview of old and new approaches for settling this question. We shall, in particular, talk about our new approach that gives the first constant-factor approximation algorithm for the asymmetric traveling salesman problem. Our approximation guarantee is analyzed with respect to the standard LP relaxation, and thus our result confirms the conjectured constant integrality gap of that relaxation. The main idea of our approach is to first give a generic reduction to structured instances and on those instances we then solve an easier problem (but equivalent in terms of constant-factor approximation) obtained by relaxing the general connectivity requirements into local connectivity conditions.

This is based on joint work with Jakub Tarnawski and László A. Végh.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Approximation algorithms, combinatorial optimization, linear programming, traveling salesman problem

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.3

Category Invited Paper

[1] Supported by ERC Starting Grant 335288-OptApprox.

Continuous Algorithms

Santosh Vempala
Georgia Institute of Technology, Atlanta, Georgia, USA

Abstract

While the design of algorithms is traditionally a discrete endeavour, in recent years many advances have come from continuous perspectives. Typically, a continuous process, deterministic or randomized, is designed and shown to have desirable properties, such as approaching an optimal solution or a target distribution, and an algorithm is derived from this by appropriate discretization. We will discuss examples of this for optimization (gradient descent, interior-point method) and sampling (Brownian motion, Hamiltonian Monte Carlo), with applications to learning. In some interesting and rather general settings, the current fastest methods have been obtained via this approach.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.4

Category Invited Paper

38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018).
Editors: Sumit Ganguly and Paritosh Pandya; Article No. 4; pp. 4:1–4:1

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

On the Probabilistic Degree of OR over the Reals

Siddharth Bhandari
TIFR, Mumbai, India
siddharth.bhandari@tifr.res.in

Prahladh Harsha
TIFR, Mumbai, India
prahladh@tifr.res.in

Tulasimohan Molli
TIFR, Mumbai, India
tulasi.molli@tifr.res.in

Srikanth Srinivasan
Department of Mathematics, IIT Bombay, Mumbai, India
srikanth@math.iitb.ac.in

Abstract

We study the probabilistic degree over $\mathbb{R}$ of the OR function on n variables. For $\varepsilon \in (0, 1/3)$, the ε-error probabilistic degree of any Boolean function $f : \{0,1\}^n \to \{0,1\}$ over $\mathbb{R}$ is the smallest non-negative integer d such that the following holds: there exists a distribution of polynomials $\mathbf{P} \in \mathbb{R}[x_1, \ldots, x_n]$ entirely supported on polynomials of degree at most d such that for all $z \in \{0,1\}^n$, we have $\Pr_{P \sim \mathbf{P}}[P(z) = f(z)] \geq 1 - \varepsilon$. It is known from the works of Tarui (*Theoret. Comput. Sci.* 1993) and Beigel, Reingold, and Spielman (*Proc. 6th CCC* 1991), that the ε-error probabilistic degree of the OR function is at most $O(\log n \cdot \log(1/\varepsilon))$. Our first observation is that this can be improved to $O\left(\log \binom{n}{\leq \log(1/\varepsilon)}\right)$, which is better for small values of ε.

In all known constructions of probabilistic polynomials for the OR function (including the above improvement), the polynomials P in the support of the distribution $\mathbf{P}$ have the following special structure:

$$P(x_1, \ldots, x_n) = 1 - \prod_{i \in [t]} (1 - L_i(x_1, \ldots, x_n)),$$

where each $L_i(x_1, \ldots, x_n)$ is a linear form in the variables $x_1, \ldots, x_n$, i.e., the polynomial $1 - P(\bar{x})$ is a product of affine forms. We show that the ε-error probabilistic degree of OR when restricted to polynomials of the above form is $\Omega\left(\log \binom{n}{\leq \log(1/\varepsilon)} / \log^2\left(\log \binom{n}{\leq \log(1/\varepsilon)}\right)\right)$, thus matching the above upper bound (up to polylogarithmic factors).

2012 ACM Subject Classification Theory of computation → Probabilistic computation, Theory of computation → Circuit complexity

Keywords and phrases Polynomials over reals, probabilistic polynomials, probabilistic degree, OR polynomial

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.5

Acknowledgements The authors thanks Noga Alon for referring them to the paper on radio-broadcast [1].

38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018).
Editors: Sumit Ganguly and Paritosh Pandya; Article No. 5; pp. 5:1–5:12

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

1 Introduction

Low-degree polynomial approximations of Boolean functions were introduced by Razborov in his celebrated work [11] on proving lower bounds for the class of Boolean functions computed by low-depth circuits. We begin by recalling this notion of approximation over $\mathbb{R}$.

▶ **Definition 1.1** (probabilistic degree)**.** Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$ and $\varepsilon \in (0, 1/3)$, an *ε-error probabilistic polynomial* over $\mathbb{R}$[1] for f is a distribution of polynomials $\mathbf{P}(x_1, \ldots, x_n) \in \mathbb{R}[x_1, \ldots, x_n]$ such that for any $z \in \{0,1\}^n$, we have $\Pr_{P \sim \mathbf{P}}[P(z) \neq f(z)] \leq \varepsilon$. The *$\varepsilon$-error Probabilistic degree of f*, denoted by $\text{P-deg}_\varepsilon(f)$, is the smallest non-negative integer d such that the following holds: there exists an ε-error probabilistic polynomial $\mathbf{P}$ over $\mathbb{R}$ such that $\mathbf{P}$ is entirely supported on polynomials of degree at most d.

Classical results in polynomial approximation of Boolean functions [15, 14, 3] show that the OR function over n variables, denoted by OR_n, has ε-error probabilistic degree at most $O\left(\log n \cdot \log(1/\varepsilon)\right)$. This basic construction for the OR function is then recursively used to show that any function computed by an AC^0 circuit of size s and depth d has ε-error probabilistic degree at most $(\log s)^{O(d)} \cdot \log(1/\varepsilon)$ (see work by the second and last author [6] for recent improvements). These results can then be used to prove, eg. [12], a (slightly weaker) version of Håstad's celebrated theorem [7] that parity does not have subexponential-sized AC^0 circuits. These results were employed more recently by Braverman [4] to prove that polylog-wise independence fools AC^0 functions.

Despite the fact that probabilistic polynomials for the OR function are such a basic primitive, it is surprising that we do not yet have a complete understanding of $\text{P-deg}_\varepsilon(\text{OR}_n)$. As mentioned above, it is known from the works of Beigel, Reingold and Spielman [3] and Tarui [14] that $\text{P-deg}_\varepsilon(\text{OR}_n) = O\left(\log n \cdot \log(1/\varepsilon)\right)$. It can be easily checked via a simple application of the Schwartz-Zippel lemma that a dependence of $\Omega\left(\log(1/\varepsilon)\right)$ is necessary in the above bound. However, till not long ago, it was unclear if *any* dependence on n is required over the reals [2]. In recent papers of Meka, Nguyen and Vu [10] and the second and last author [6], it was shown using anti-concentration of low-degree polynomials that the $\text{P-deg}_{1/4}(\text{OR}_n) = \widetilde{\Omega}(\sqrt{\log n})$. The main objective of this paper is to obtain a better understanding of the ε-error probabilistic degree of OR_n, $\text{P-deg}_\varepsilon(\text{OR}_n)$. Besides being interesting in its own right, this question has bearing on the amount of independence needed to fool AC^0 circuits. Recent improvements due to Tal [13] and [6] of Braverman's result demonstrate that $(\log s)^{2.5d+O(1)} \cdot \log(1/\varepsilon)$-wise independence fools functions computed by AC^0 circuits of size s and depth d. An improvement of the upper bound on $\text{P-deg}_\varepsilon(\text{OR}_n)$ to $O\left(\log n\right) + \log(1/\varepsilon))$ could potentially strengthen this result to $(\log s)^{d+O(1)} \cdot \log(1/\varepsilon)$, nearly matching the lower bound of $(\log s)^{d-1} \cdot \log(1/\varepsilon)$ due to Mansour [9].

The above discussion demonstrates that the current bounds on $\text{P-deg}_\varepsilon(\text{OR}_n)$ fall short of being tight in two aspects: one, the dependence on n in the lower bound is $\widetilde{\Omega}\left(\sqrt{\log n}\right)$ while in the upper bound it is $O\left(\log n\right)$ and two, the joint dependence on ε and n in the upper bound is multiplicative, i.e., $O\left(\log n \cdot \log(1/\varepsilon)\right)$ while the current lower bounds can only show an additive $\widetilde{\Omega}\left(\sqrt{\log n}\right) + \Omega\left(\log(1/\varepsilon)\right)$ bound.

Which of these bounds is tight? A casual observer might suspect that the upper bound is, given the relatively neat expression. However, a closer look tells us that it cannot be, at least when ε is quite small. For example, setting $\varepsilon = 1/2^{\Omega(n)}$, the upper bound yields a

[1] Similar notions over other fields are also studied. Unless otherwise specified, we will be considering probabilistic polynomials over the reals in this paper.

[2] For finite fields of constant size, Razborov [11] showed that the ε-error probabilistic degree of OR_n is $O\left(\log(1/\varepsilon)\right)$, independent of n, the number of the input bits.

degree of $O(n \log n)$, but it is a standard fact that any Boolean function on n variables can be represented exactly (i.e. with no error) as a polynomial of degree n. Hence the upper bound is not tight in this regime.

Our first observation is that the upper bound of Tarui and Beigel et al. [3] can indeed be slightly improved to $O\left(\log\left(\binom{n}{\leq \log(1/\varepsilon)}\right)\right)$;[3] note that this is asymptotically better than $O\left(\log n \cdot \log(1/\varepsilon)\right)$ for very small ε. This interpolates smoothly between the construction of Tarui [14] and Beigel et al. [3] and the exact representation of degree n mentioned above. (See Section 2 for details on this upper-bound construction.)

Given this observation, one might hope to prove a matching lower bound on the ε-error probabilistic degree of OR_n. We can indeed show such a bound (upto polylogarithmic factors) if we suitably restrict the class of polynomials being considered. While restricted, this subclass of polynomials nevertheless includes all polynomials that were used in previous upper bound constructions, including our own. Moreover, this result generalizes a result of Alon, Bar-Noy, Linial and Peleg [1], who prove such a result for a further restricted class of polynomials (mentioned at the end of this section) and for $\log(1/\varepsilon) = O(\log n)$.[4] A careful reworking of their analysis shows that their lower bound extends to even smaller ε to show a lower bound of $\Omega(\log\left(\binom{n}{\leq \log(1/\varepsilon)}\right))$ for this smaller class of polynomials.

To state our result, we first need to describe the class of polynomials for which our bounds hold. To this end, we note that all known upper-bound constructions of probabilistic polynomials for the OR function have the following structure:

$$P(x_1, \ldots, x_n) = 1 - \prod_{i \in [t]} \left(1 - L_i(x_1, \ldots, x_n)\right),$$

where each $L_i(x_1, \ldots, x_n) = a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n$ is a linear form in the variables $x_1, \ldots, x_n$ (here, $a_{ij} \in \mathbb{R}$).

This includes the improved upper-bound construction that achieves an ε-probabilistic degree of $O(\log\left(\binom{n}{\leq \log(1/\varepsilon)}\right))$ mentioned in the preceding paragraph. This motivates the following definition.

▶ **Definition 1.2** (hyperplane covering polynomials)**.** A polynomial $P \in \mathbb{R}[x_1, \ldots, x_n]$ is said to be a hyperplane covering polynomial of degree t if there exist t linear forms $L_1, \ldots, L_t$ over the reals such that

$$P(x_1, \ldots, x_n) = 1 - \prod_{i \in [t]} \left(1 - L_i(x_1, \ldots, x_n)\right).$$

For $\varepsilon \in (0, 1/2)$, the *ε-error hyperplane covering probabilistic degree of f*, denoted by $\mathrm{hcP\text{-}deg}_\varepsilon(f)$, is the smallest non-negative integer d such that the following holds: there exists an ε-error probabilistic polynomial $\mathbf{P}$ over $\mathbb{R}$ such that $\mathbf{P}$ is supported on hyperplane covering polynomials of degree at most d.

We call these polynomials hyperplane covering polynomials as these polynomials have the property that the one's of the polynomials in the Boolean hypercube (i.e, the set $\{z \in \{0,1\}^n \mid P(z) = 1\}$) are a union of hyperplanes not passing through the origin. We further note that all these polynomials satisfy the property that $P(\bar{0}) = 0$. Clearly, $\mathrm{hcP\text{-}deg}_\varepsilon(f) \geq$

[3] Here, $\binom{N}{\leq \alpha}$ denotes $\sum_{i \leq \alpha} \binom{N}{i}$.

[4] The result of [1] is stated in a slightly different language, but is essentially equivalent to a probabilistic degree lower bound for OR_n for a suitable class of polynomials.

P-deg$_\varepsilon(f)$. Also, since all upper-bound constructions for the OR polynomials are hyperplane covering polynomials, we not only have that P-deg$_\varepsilon(\mathrm{OR}_n) = O\left(\log\binom{n}{\leq \log(1/\varepsilon)}\right)$ but also that hcP-deg$_\varepsilon(\mathrm{OR}_n) = O\left(\log\binom{n}{\leq \log(1/\varepsilon)}\right)$. For this class of polynomials, we prove the following (almost) tight result on the ε-error hyperplane covering probabilistic degree of the OR function.

▶ **Theorem 1.3** (hyperplane covering degree of OR_n)**.** *For any any positive integer n and $\varepsilon \in (0, 1/3)$,*

$$\text{hcP-deg}_\varepsilon(\mathrm{OR}_n) = \Omega\left(\frac{\log\binom{n}{\leq \log(1/\varepsilon)}}{\log^2\left(\log\binom{n}{\leq \log(1/\varepsilon)}\right)}\right).$$

It is open if this result can be extended to prove a tighter lower bound on the ε-error probabilistic degree of the OR_n function. The special class of hyperplane covering polynomials for which Alon, Bar-Noy, Peleg and Linial [1] proved a similar bound is the class of hyperplane covering polynomials where the linear forms are sums of variables (i.e., $L_i(\bar{z}) = \sum_{j\in S_i} z_j$ for some $S_i \subseteq [n]$). Ideally, one would have liked to extend their lower bound result for hyperplace covering polynomials where the linear forms are sums of variables to *all* polynomials. Theorem 1.3, is a step in this direction, in that, it shows that their result can be extended to a slightly larger class, the set of all hyperplane covering polynomials (modulo polylogarithmic factors). We remark that though our lower bound works for a larger class of polynomials, our proof technique is nevertheless inspired by their proof.

2 Upper bounds on probabilistic degree of OR

In this section, we describe the construction of a probabilistic polynomial which shows that the hcP-deg$_\varepsilon(\mathrm{OR}_n) = O\left(\log\binom{n}{\leq \log(1/\varepsilon)}\right)$. To begin with, we observe that the following "trivial" hyperplane covering polynomial of degree n exactly computes OR_n everywhere on the Boolean hypercube:

$$P_{\mathrm{OR}}(x) := 1 - \prod_{i=1}^{n}\left(1 - \frac{1}{i}\sum_{j\in[n]} x_j\right).$$

This is a polynomial which covers each Hamming slice of the hypercube with a different hyperplane. We now recall the construction of Beigel, Reingold and Spielman [3] and Tarui [14].

▶ **Claim 2.1.** *For every non-negative integer ℓ, there exists a distribution of linear forms $\mathbf{L}_\ell$ such that if the Hamming weight of $x = (x_1, \ldots, x_n)$ lies in the interval $[2^\ell, 2^{\ell+1}]$, then* $\Pr_{L\sim\mathbf{L}}[L(x) = 1] = \Omega(1)$.

Proof. $\mathbf{L}$ is defined as follows: pick a random set $S \subseteq [n]$ by picking each element of $[n]$ independently with probability $\frac{1}{2^\ell}$ and construct the linear polynomial

$$L_S(x) := \sum_{i\in S} x_i \ .$$

For a non-zero input $x = (x_1, \ldots, x_n)$ such that the Hamming weight of x, denoted as $|x|$, is in $[2^\ell, 2^{\ell+1}]$, we have

$$\begin{aligned}\Pr_S[L_S(x) = 1] &= |x| \left(\frac{1}{2^\ell}\right)\left(1 - \frac{1}{2^\ell}\right)^{|x|-1} && [\text{where } 0^0 = 1]\\ &= \frac{|x|}{2^\ell}\exp(-O(1)) && [\because (1-a)^b \geq \exp(-ab/1-a)]\\ &\geq \Omega(1)\ . \end{aligned}$$

◀

In the above proof we could have set $L_S(x) = \sum_{i\in S} \alpha_i x_i$ where each $\alpha_i \in \pm 1$ u.a.r. and independently. Clearly, even with the new definition $Pr_S[L_S(x) = 1] \geq \Omega(1)$. The idea behind introducing the α's is that even when $\sum_{i\in S} x_i > 1$, it could be that $\sum_{i\in S} \alpha_i x_i = 1$. However, this does not lead to improvements beyond possibly changing the constant hidden in the $\Omega(\cdot)$ notation.

The preceding claim is then used to construct ε-error probabilistic polynomials for OR_n as follows. Divide the set of one's of the OR function in the Boolean hypercube, ie., $\{0,1\}^n \setminus \{\bar{0}\}$, into $\log n$ epochs $[2^0, 2^1], [2^1, 2^2], \ldots, [2^{\log n - 1}, 2^{\log n}]$ where each epoch $[2^\ell, 2^{\ell+1}]$ includes all strings whose Hamming weight is in that range. For each such epoch $[2^\ell, 2^{\ell+1}]$, sample $t := O(\log(1/\varepsilon))$ independent linear forms $L_i^{(\ell)}, i \in [t]$ from $\mathbf{L}_\ell$ and consider the randomized polynomial $P_\ell(x) := 1 - \prod_{i\in[t]}(1 - L_i^{(\ell)}(x))$. Clearly, for x in the epoch $[2^\ell, 2^{\ell+1}]$, we have $\Pr[P_\ell(x) = 1] \geq 1 - \varepsilon$. Now, the randomized polynomial

$$P(x) := 1 - \prod_{\ell\in[\log n]} (1 - P_\ell(x)) = 1 - \prod_{\ell\in[\log n]} \prod_{i\in[t]} (1 - L_i^{(\ell)})\ ,$$

satisfies for all $x \in \{0,1\}^n \setminus \{\bar{0}\}$, $\Pr[P(x) = 1] \geq 1 - \varepsilon$. Also, clearly any such P satisfies $P(\bar{0}) = 0$. This polynomial is a hyperplane covering polynomial of degree at most $O(\log n \cdot \log(1/\varepsilon))$.

Now, suppose ε is very small, eg., $\varepsilon = 2^{-n/10}$, then this construction is wasteful over the trivial construction P_{OR} since $O(\log n \cdot \log(1/\varepsilon)) = O(n \log n)$. The improved bound of $O(\log \binom{n}{\leq \log(1/\varepsilon)})$ is obtained by "interpolating" between the trivial construction P_{OR} and the above construction. Since we know that the $\text{P-deg}_\varepsilon(\mathrm{OR}_n)$ is at least $\log(1/\varepsilon)$, one might as well exactly compute OR_n for the first $O(\log(1/\varepsilon))$ Hamming slices of the hypercube and use the above randomized construction to cover the remaining slices using only $(\log n - \log\log(1/\varepsilon))$ epochs, $[\log(1/\varepsilon), 2\log(1/\varepsilon)], \ldots, [2^{\log n - 1}, 2^{\log n}]$. Another way to view this is that when we focus on the epoch $[2^\ell, 2^{\ell+1}]$ and draw $t = O(\log(1/\varepsilon))$ samples from $\mathbf{L}_\ell$, the trivial polynomial $\prod_{i=2^\ell}^{2^{\ell+1}} \left(1 - \frac{1}{i}\sum_{j\in[n]} x_j\right)$ has degree smaller than $O(\log(1/\varepsilon))$ when $2^\ell < \log(1/\varepsilon)$ or $\ell < \log\log(1/\varepsilon)$.

Formally, we construct the polynomial (where $P_\ell(x)$ and $L^{(\ell)}$ are as defined above)

$$\begin{aligned}P(x) &:= 1 - \left(\prod_{\ell\in[\log\log(1/\varepsilon), \log n]} (1 - P_\ell(x))\right) \times \prod_{i=1}^{\log(1/\varepsilon)} \left(1 - \frac{1}{i}\sum_{j\in[n]} x_j\right)\\ &= 1 - \left(\prod_{\ell\in[\log\log(1/\varepsilon), \log n]} \prod_{i\in[t]} \left(1 - L_i^{(\ell)}\right)\right) \times \prod_{i=1}^{\log(1/\varepsilon)} \left(1 - \frac{1}{i}\sum_{j\in[n]} x_j\right).\end{aligned}$$

Clearly, P is a hyperplane covering polynomial. For an input x such that $|x| \leq \log(1/\varepsilon)$, $P(x) = 1$ as $\prod_{i=1}^{\log(1/\varepsilon)} \left(1 - \frac{1}{i}\sum_{j\in[n]} x_j\right) = 0$. If $|x| \in [2^\ell, 2^{\ell+1}]$ where $\ell \geq \log\log(1/\varepsilon)$, then from our previous argument we have $\Pr[P_l(x) = 1] \geq 1 - \varepsilon$ and hence $\Pr[P(x) = 1] \geq 1 - \varepsilon$. Hence, we have an ε-error probabilistic polynomial of degree $O(\log(1/\varepsilon) + (\log n - \log\log(1/\varepsilon)) \cdot \log(1/\varepsilon))$ which is at most $O(\log \binom{n}{\leq \log(1/\varepsilon)})$.

3 Lower bound on hyperplane covering degree of OR

We now turn to the lower bound. To prove a lower bound of $d_\varepsilon := \widetilde{\Omega}(\log \binom{n}{\leq \log(1/\varepsilon)})$, by Yao's minmax theorem (duality arguments) it suffices (and is neccesary) to demonstrate a "hard" distribution $\mathcal{D}_\varepsilon$ under which it is hard to approximate the OR_n function by any hyperplane covering polynomial of degree at most d_ε.

Similar to previous works [10, 6], our choice of hard distribution is motivated by the polynomial constructions in the upper bound. We first need the following definitions to define the hard distribution $\mathcal{D}_\varepsilon$.

▶ **Definition 3.1** ($(0,1)$-restriction μ_p)**.** The $\mu_p^{[n]}$ distribution on $\{0,1\}^n$ is obtained by setting each variable x_i independently to 1 with probability p and 0 otherwise.

▶ **Definition 3.2** ($(0,*)$-restriction ρ_p)**.** The $\rho_p^{[n]}$ distribution on $\{0,*\}^n$ is obtained by setting each variable to 0 independently with probability $(1-p)$ and leaving it unset with probability p.

If the number of variables is n, we will drop the superscript and refer to the corresponding restrictions as just μ_p and ρ_p respectively.

It will be convenient to view the distribution μ_p as applying a $(0,*)$ restriction ρ_{2p} followed by a $\{0,1\}$ restriction $\mu_{1/2}$ to the unset variables. In short, $\mu_p^{[n]} = \mu_{1/2}^{\rho_{2p}^{-1}(*)} \circ \rho_{2p}^{[n]}$.

▶ **Definition 3.3** (hard distribution)**.** Consider the distribution $\mathcal{D}_\varepsilon$ on the input set $\{0,1\}^n$ defined as follows:

- pick an integer $\ell \in I_\varepsilon := [1, \log n - \log\log(1/\varepsilon)] \cap \mathbb{Z}$ uniformly at random.
- pick $x \in \{0,1\}^n$ according to $\mu_{1/2^\ell}$, i.e., for each $i \in [n]$, independently sets $x_i \leftarrow 1$ with probability $1/2^\ell$ and 0 otherwise.

The hard distribution $\mathcal{D}_\varepsilon$ is a convex combination of the distributions $\mu_{1/2^\ell}$ for $\ell \in I_\varepsilon$. In other words, $\mathcal{D}_\varepsilon := \frac{1}{|I_\varepsilon|} \sum_{\ell \in I_\varepsilon} \mu_{1/2^\ell}$. Each of the distributions $\mu_{1/2^\ell}$ roughly correspond to the epochs used in the upper-bound construction.

Theorem 1.3 follows from the following "distributional" version of the theorem.

▶ **Theorem 3.4.** *Let $\mathcal{D}_\varepsilon$ be the hard distribution defined in Definition 3.3 and $P = 1 - \prod_{i\in[t]}(1 - L_i)$ be a hyperplane covering polynomial of degree t such that*

$$\Pr_{x\sim\mathcal{D}_\varepsilon}[P(x) \neq OR_n(x)] \leq \varepsilon$$

then, $t \geq \Omega\left(\frac{\log\left(\binom{n}{\leq\log(1/\varepsilon)}\right)}{\log^2\left(\log\left(\binom{n}{\leq\log(1/\varepsilon)}\right)\right)}\right)$.

We now introduce some notations that will be useful. For a set S, $|S|$ denotes the cardinality of S, and for an input $x \in 0,1^n$, $|x|$ denotes the Hamming weight of x.

▶ **Definition 3.5** (support of a linear form)**.** For a linear form $L(x_1,\ldots,x_n) = a_1x_1 + a_2x_2 + \cdots + a_nx_n$, we define the support of L, denoted as $\mathrm{supp}(L)$, to be the set of non-zero a_i's, i.e., $|\{i \in [n] \mid a_i \neq 0\}|$.

The proof of Theorem 3.4 requires the following variant of the Schwartz-Zippel Lemma (due to Alon and Füredi [2]) and Littlewood-Offord-Erdös' anti-concentration lemma of linear forms over the reals, which we state below.

▶ **Lemma 3.6** ([2, Theorem 5]). *Let $P \in \mathbb{R}[x_1, \ldots, x_n]$ be a polynomial of degree at most d polynomial over $\mathbb{R}$ computing a non-zero function over $\{0,1\}^n$. Then for x chosen uniformly from $\{0,1\}^n$,*

$$\Pr_{x \in \{0,1\}^n}[P(x) \neq 0] \geq \frac{1}{2^d} .$$

▶ **Lemma 3.7** (anti-concentration of linear forms over $\mathbb{R}$ [8, 5]). *Let $L(x_1, \ldots, x_k) = \sum a_i x_i$ be a linear form which is supported on exactly k variables (i.e., $a_i \neq 0, i = 1, \ldots, k$. Then, for all $a \in \mathbb{R}$ and x chosen uniformly from $\{0,1\}^n$,*

$$\Pr_{x \in \{0,1\}^n}[L_i(x) = a] \leq \frac{1}{\sqrt{k}} .$$

The rest of this section is devoted to proving Theorem 3.4. We begin with a proof outline in Section 3.1 followed by the proof in Section 3.2.

3.1 Proof outline

We would like to show that hyperplane covering polynomial P that approximates OR_n w.r.t distribution $\mathcal{D}_\varepsilon$ (as in Theorem 3.4) must have large degree. Let $\mathcal{L}$ denote the set of linear forms that appear in P, i.e., $\mathcal{L} := \{L_i \mid i \in [t]\}$.

Let us see how P behaves on the distribution $\mu_{1/2^\ell}$ or equivalently $\mu_{1/2} \circ \rho_{1/2^{\ell-1}}$. Let us see what happens to the linear forms $\{L_i, i \in [t]\}$ when the restriction $\rho := \rho_{1/2^{\ell-1}}$ is first applied. We first consider two extreme cases.

Very few linear forms survive: Suppose all but $\log(1/2\varepsilon)$ linear forms trivialize on the restriction ρ (i.e. the corresponding linear form $L_i|_\rho$ becomes 0). Then, $(1-P)|_\rho$ is a polynomial of degree at most $\log(1/2\varepsilon)$ computing a non-zero function (since $1 - P(\bar{0}) = 1$). Hence, by Lemma 3.6, it is not equal to 0 with probability at least 2ε. This implies that the polynomial P errs with probability at least 2ε on the distribution $\mu_{1/2^\ell}$.

All linear forms that survive have large support: Suppose all the linear forms that survive post restriction ρ have large support, say $4t^2$. Then, by the anti-concentration of linear forms over reals (Lemma 3.7), we have that each linear form is 1 with probability at most $1/\sqrt{4t^2} = 1/2t$. Since there are most t linear forms, the probability that any of them is 1 is at most $t/2t = 1/2$. Thus, P errs with probability $1/2$ on the distribution $\mu_{1/2^\ell}$.

Note that the actual situation for each distribution $\mu_{1/2^\ell}$ will most likely be a combination of the above two. We can then show that a combination of the above two arguments will still work if the surviving linear forms have the following nice structure. Let $\mathcal{L}_\rho$ be the set of surviving linear forms subsequent to the restiction ρ, i.e., $\mathcal{L}_\rho := \{L_i|_\rho \mid i \in [t], L_i|_\rho \neq 0\}$. Suppose $\mathcal{L}_\rho$ can be partitioned into 2 sets $\mathcal{L}'_\rho \dot\cup \mathcal{L}''_\rho$ such that the number of linear forms in $\mathcal{L}'_\rho$ is small (less than $O(\log(1/\varepsilon))$) and each of the linear forms in $\mathcal{L}''_\rho$ have large support even after subtracting $\cup_{L \in \mathcal{L}'_\rho} \mathrm{supp}(L)$ from their support. How does one then show that a constant faction of ρ's satisfy that the corresponding linear forms $\mathcal{L}_\rho$ have this nice structure? For this, we draw inspiration from the proof of Alon, Bar-Noy, Linial and Peleg [1], where they prove similar bounds for hyperplane covering polynomials supported entirely on linear forms arising as sums of variables. They construct an appropriate potential function that guarantees a similar property in their lower-bound argument.

We use a slightly different potential function, which has the following nice property. If the total number of linear forms is t, then $\mathbb{E}_\ell[\Phi_\ell(\mathcal{L})] = O(t/(\log n - \log\log(1/\varepsilon)))$ and furthermore, whenever $\Phi_\ell(\mathcal{L})$ is small then the corresponding set $\mathcal{L}_\ell$ of surviving linear forms post restriction $\rho_{1/2^{\ell-1}}$ can be partitioned as indicated above. This shows that for most ℓ, P errs on computing the OR_n function unless t is large.

3.2 Proof of Theorem 3.4

We now turn to defining the potential function $\Phi_\ell(\mathcal{L})$, indicated in the proof outline.

▶ **Definition 3.8** (potential function)**.** The weight of a linear form L, denoted by $w(L)$, is defined as follows:

$$w(L) := \begin{cases} 0 & \text{if } \operatorname{supp}(L) = \emptyset, \\ \frac{1}{\log^2(2|\operatorname{supp}(L)|)} & \text{otherwise.} \end{cases}$$

Given a collection $\mathcal{L} = \{L_1, \ldots, L_t\}$ of linear forms and ℓ a positive integer, the potential function $\Phi_\ell(\mathcal{L})$ is defined as follows

$$\Phi_\ell(\mathcal{L}) := \sum_{i=1}^{t} \mathop{\mathbb{E}}_{\rho_{1/2^{\ell-1}}} \left[w\left(L_i|_{\rho_{1/2^{\ell-1}}} \right) \right],$$

where $\rho_{1/2^{\ell-1}}$ is a $(0,*)$-restriction as defined in Definition 3.2.

The potential function $\Phi_\ell(\mathcal{L})$ satisfies the following two properties, given by Propositions 3.9 and 3.10.

▶ **Proposition 3.9.** *There exists a universal constant C such that the following holds. Let $\mathcal{L} = \{L_1, \ldots, L_t\}$ be any collection of t linear forms, then*

$$\mathop{\mathbb{E}}_{\ell \in I_\varepsilon} [\Phi_\ell(\mathcal{L})] \leq \frac{Ct}{|I_\varepsilon|} .$$

▶ **Proposition 3.10** (partition of linear forms)**.** *Let $\mathcal{L} = \{L_1, \ldots, L_t\}$ be a collection of t non-zero linear forms and K, R be two positive integers such that*

$$\sum_{i=1}^{t} w(L_i) < \frac{R}{\log^2(2RK)} .$$

Then, there exists a partition $\mathcal{L} = \mathcal{L}' \dot\cup \mathcal{L}''$ of the set of linear forms $\mathcal{L}$ such that

- $|\mathcal{L}'| \leq R$,
- *For all $L \in \mathcal{L}''$, $|\operatorname{supp}(L) \setminus \cup_{L' \in \mathcal{L}'} \operatorname{supp}(L')| \geq K$.*

Before proving these two propositions, we first show how they imply Theorem 3.4.

Proof of Theorem 3.4. Let

$$t := \frac{\log(1/8\varepsilon) \cdot (\log n - \log\log(1/\varepsilon))}{2C \log^2\left(1/C^2 \cdot \log^4(1/8\varepsilon) \cdot (\log n - \log\log(1/\varepsilon))^3\right)} ,$$

where C is the universal constant in Proposition 3.9. Clearly, $t = \Omega\left(\frac{\log\left(\leq \log(1/\varepsilon)\right)}{\log^2\left(\log\left(\leq \log(1/\varepsilon)\right)\right)}\right)$. Let $P = 1 - \prod_{i\in[t]}(1 - L_i)$ be any hyperplane covering polynomial of degree t. To prove the theorem, it suffices if we show that $\Pr_{x\sim\mathcal{D}_\varepsilon}[P(x) \neq \mathrm{OR}_n(x)] > \varepsilon$. To this end, we first note that $\Pr_{x\sim\mathcal{D}_\varepsilon}[x = \bar{0}] < \varepsilon$ (since for all $\ell \in I_\varepsilon$, we have $\ell \leq \log n - \log\log(1/\varepsilon)$). Hence, to prove the theorem it suffices to show that $\Pr_{x\sim\mathcal{D}_\varepsilon}[P(x) \neq 1] \geq 2\varepsilon$.

Since $\mathcal{D}_\varepsilon = \frac{1}{|I_\varepsilon|}\sum_{\ell\in I_\varepsilon} \mu_{1/2^\ell}$ and $\mu_p^{[n]} = \mu_{1/2} \circ \rho_{2p}^{[n]}$, this is equivalent to showing

$$\mathop{\mathbb{E}}_{\ell\in I_\varepsilon}\left[\mathop{\mathbb{E}}_{\rho\sim\rho_{1/2^{\ell-1}}}\left[\Pr_{x\sim\mu_{1/2}}[P|_\rho(x) \neq 1]\right]\right] \geq 2\varepsilon . \tag{1}$$

To this end, we first apply Proposition 3.9 to the set $\mathcal{L}$ of t linear forms in the polynomial P to obtain that

$$\mathop{\mathbb{E}}_{\ell \in I_\varepsilon}\left[\mathop{\mathbb{E}}_{\rho \sim \rho_{1/2^{\ell-1}}}\left[\sum_{i\in[t]} w(L_i|_\rho)\right]\right] = \mathop{\mathbb{E}}_{\ell\in I_\varepsilon}\left[\Phi_\ell(\mathcal{L})\right] \leq \frac{Ct}{|I_\varepsilon|}\ .$$

Applying Markov to the above inequality, we have

$$\Pr_{\ell,\ \rho}\left[\sum_{i\in[t]} w(L_i|_\rho) \leq \frac{2Ct}{|I_\varepsilon|}\right] \geq \frac{1}{2}\ .$$

We call an(ℓ,ρ) pair *good* if the above event holds, i.e., $\sum_{i=1}^{t} w(L_i|_\rho) \leq {}^{2Ct}/_{|I_\varepsilon|}$. Thus,

$$\Pr_{\ell,\ \rho}[(\ell,\rho) \text{ is good }] \geq {}^1/_2\ . \tag{2}$$

Now given a good (ℓ,ρ)-pair, let $\mathcal{L}_\rho$ be the set of surviving linear forms subsequent to the restriction ρ, i.e., $\mathcal{L}_\rho := \{L_i|_\rho \mid i \in [t], L_i|_\rho \neq 0\}$. We thus have $\sum_{L\in\mathcal{L}_\rho} w(L) \leq {}^{2Ct}/_{|I_\varepsilon|}$. Let $K := 4t^2$ and $R := \log({}^1/_{8\varepsilon})$. It can be checked that for this choice of parameters we have ${}^{2Ct}/_{|I_\varepsilon|} < {}^R/_{\log^2(2RK)}$. We can now apply Proposition 3.10 to obtain a partition $\mathcal{L}_\rho = \mathcal{L}'_\rho \dot\cup \mathcal{L}''_\rho$ such that

- $|\mathcal{L}'_\rho| \leq R = \log({}^1/_{8\varepsilon})$,
- for all $L \in \mathcal{L}''_\rho$, we have $|\operatorname{supp}(L) \setminus \cup_{L'\in\mathcal{L}'_\rho} \operatorname{supp}(L')| \geq K = 4t^2$.

Consider the polynomial $P|_\rho = 1 - \prod_{i\in[t]}(1 - L_i|_\rho) = 1 - \prod_{L\in\mathcal{L}|_\rho}(1-L)$ subsequent to the restriction ρ. We will rewrite this polynomial as $P|_\rho = 1 - Q'_\rho \cdot Q''_\rho$ where the polynomials Q'_ρ and Q''_ρ are defined as follows (using the sets $\mathcal{L}'_\rho$ and $\mathcal{L}''_\rho$ respectively).

$$Q'_\rho(x) := \prod_{L\in\mathcal{L}'_\rho}(1-L(x)),$$
$$Q''_\rho(x) := \prod_{L\in\mathcal{L}''_\rho}(1-L(x)).$$

Clearly, $P|_\rho = 1 - Q'_\rho \cdot Q''_\rho$.

Since $|\mathcal{L}'_\rho| \leq \log({}^1/_{8\varepsilon})$, we have that the degree of Q'_ρ is at most $\log({}^1/_{8\varepsilon})$. Furthermore $Q'_\rho(x) \not\equiv 0$ (since $Q'_\rho(\bar 0) = 1$). Thus applying Lemma 3.6, we have

$$\Pr_{x\sim\mu_{1/2}}\left[Q'_\rho(x) \neq 0\right] \geq 8\varepsilon.$$

Consider any setting of variables in $\cup_{L\in\mathcal{L}'_\rho}\operatorname{supp}(L)$ such that $Q'_\rho(x) \neq 0$. Even conditioned on setting all these variables, we know that each $L \in \mathcal{L}''_\rho$ still has surviving support of size at least $4t^2$. Thus, by Lemma 3.7, we have for each $L \in \mathcal{L}''_\rho$,

$$\Pr_{x\sim\mu_{1/2}}\left[L(x) = 1 \mid Q'_\rho(x) \neq 0\right] \leq \frac{1}{\sqrt{4t^2}} = \frac{1}{2t}.$$

By a union bound, we have

$$\Pr_{x\sim\mu_{1/2}}\left[Q''_\rho(x) = 0 \mid Q'_\rho(x) \neq 0\right] = \Pr_{x\sim\mu_{1/2}}\left[\exists L \in \mathcal{L}''_\rho, L(x) = 1 \mid Q'_\rho(x) \neq 0\right] \leq \frac{t}{2t} = \frac{1}{2}.$$

Hence,

$$\Pr_{x\sim\mu_{1/2}}\left[P|_\rho(x) \neq 1\right] = \Pr\left[Q'_\rho(x) \neq 0\right] \cdot \Pr\left[Q''_\rho(x) = 0 \mid Q'_\rho(x) \neq 0\right] \geq 8\varepsilon \cdot \frac{1}{2} = 4\varepsilon.$$

Finally averaging over all (ℓ, ρ) we have from above and (2)

$$\Pr_{x\sim\mathcal{D}_\varepsilon}[P(x)\neq 1] \geq \Pr_{\ell,\rho}[(\ell,\rho) \text{ is good }] \cdot \Pr[P|_\rho(x)\neq 1 \mid (\ell,\rho) \text{ is good }] \geq \frac{1}{2}\cdot 4\varepsilon = 2\varepsilon.$$

This proves (1) and thus completes the proof of Theorem 3.4. ◀

We are now left with the proofs of Propositions 3.9 and 3.10. We begin with the proof of Proposition 3.10.

Proof of Proposition 3.10. Consider the following algorithm to obtain the partition $\mathcal{L} = \mathcal{L}' \dot\cup \mathcal{L}''$.

1. Initialize $\mathcal{L}' \leftarrow \emptyset$ and $\mathcal{L}'' \leftarrow \mathcal{L}$.
2. While there exists an $L \in \mathcal{L}''$ such that $|\operatorname{supp}(L) \setminus \cup_{L'\in\mathcal{L}'} \operatorname{supp}(L')| \leq K$,
   - Move such an L from $\mathcal{L}''$ to $\mathcal{L}'$ (i.e., $\mathcal{L}' \leftarrow \mathcal{L}' \cup \{L\}$ and $\mathcal{L}'' \leftarrow \mathcal{L}'' \setminus \{L\}$).

Clearly, when the algorithm terminates, we have $|\operatorname{supp}(L) \setminus \operatorname{supp}(\mathcal{L}')| \geq K$ for all $L \in \mathcal{L}''$.

We now argue that $|\mathcal{L}'| \leq R$. Each iteration of the while loop adds a linear form L to $\mathcal{L}'$ with at most K new variables. If the while loop is performed for T iterations, then the support of each L added to $\mathcal{L}'$ is at most TK. We now argue that $T < R$. If not, then after exactly R iterations of the while loop, we have that

$$\sum_{L\in\mathcal{L}} w(L) \geq \sum_{L\in\mathcal{L}'} w(L) \geq \frac{R}{\log^2(2RK)},$$

contradicting the hypothesis of the proposition. Hence $T < R$. The size of $\mathcal{L}'$ is the number of iterations of the while loop and is thus bounded above by R. This completes the proof of the proposition. ◀

Proof of Proposition 3.9.

$$\begin{aligned}
\mathop{\mathbb{E}}_{\ell\in I_\varepsilon}[\Phi_\ell(\mathcal{L})] &= \mathop{\mathbb{E}}_{\ell\in I_\varepsilon}\left[\mathop{\mathbb{E}}_{\rho\sim\rho_{1/2^{\ell-1}}}\left[\sum_{i\in[t]} w(L_i|_\rho)\right]\right] \\
&= \frac{1}{|I_\varepsilon|}\sum_{i\in[t]}\sum_{\ell\in I_\varepsilon}\mathop{\mathbb{E}}_{\rho}[w(L_i|_\rho)] \\
&\leq \frac{1}{|I_\varepsilon|}\sum_{i\in[t]}\left(\underbrace{\sum_{\ell>\log|\operatorname{supp}(L_i)|}\mathop{\mathbb{E}}_{\rho}[w(L_i|_\rho)]}_{T_1} + \underbrace{\sum_{\ell\leq\log|\operatorname{supp}(L_i)|}\mathop{\mathbb{E}}_{\rho}[w(L_i|_\rho)]}_{T_2}\right).
\end{aligned}$$

T_1 and T_2 are bound using Claim 3.11 and Claim 3.12 respectively. Hence,

$$\mathop{\mathbb{E}}_{\ell\in I_\varepsilon}[\Phi_\ell(\mathcal{L})] \leq \frac{1}{|I_\varepsilon|}\sum_{i=1}^{t}\left(2+\frac{\pi^2}{6}+\frac{e}{e-1}\right) \leq \frac{t}{|I_\varepsilon|}\cdot\left(3+\frac{\pi^2}{6}\right).$$ ◀

▶ **Claim 3.11.** *Let L be a linear form such that $|\operatorname{supp}(L)| = k$. Then*

$$\sum_{\ell:\ell>\log k}\mathop{\mathbb{E}}_{\rho\sim\rho_{1/2^{\ell-1}}}[w(L|_\rho)] \leq 2.$$

Proof.

$$\begin{aligned}
&\sum_{\ell:\ell>\log k} \mathop{\mathbb{E}}_{\rho\sim\rho_{1/2^{\ell-1}}} [w(L|_\rho)] \\
&\leq \sum_{\ell:\ell>\log k} \left(\Pr_\rho [|\operatorname{supp}(L|_\rho)| = 0] \cdot 0 + \Pr_\rho [|\operatorname{supp}(L|_\rho)| \geq 1] \cdot 1 \right) \\
&\leq \sum_{\ell:\ell>\log k} \left(1 - \left(1 - \frac{1}{2^{\ell-1}} \right)^k \right) \\
&\leq \sum_{\ell:\ell>\log k} \frac{k}{2^{\ell-1}} && [\because (1-x)^n \geq 1 - nx, \forall\ 0 < x \leq 1] \\
&\leq 2.
\end{aligned}$$

◀

▶ **Claim 3.12.** *Let L be a linear form such that $|\operatorname{supp}(L)| = k$. Then*

$$\sum_{\ell\leq\log k} \mathop{\mathbb{E}}_{\rho\sim\rho_{1/2^{\ell-1}}} [w(L|_\rho)] \leq \frac{\pi^2}{6} + \frac{e}{e-1}$$

Proof.

$$\begin{aligned}
\mathop{\mathbb{E}}_\rho [w(L|_\rho)] &\leq \Pr_\rho \left[|\operatorname{supp}(L|_\rho)| \geq \frac{k}{2^\ell} \right] \frac{1}{\log^2 (k/2^\ell)} + \Pr_\rho \left[|\operatorname{supp}(L|_\rho)| \leq \frac{1}{2} \cdot \frac{k}{2^{\ell-1}} \right] \\
&\leq \frac{1}{(\log (k) - \ell)^2} + \exp\left(-\frac{1}{4} \cdot \frac{k}{2^{\ell-1}} \right) && \text{[by chernoff bound]}
\end{aligned}$$

$$\begin{aligned}
\sum_{\ell\leq\log k} \mathop{\mathbb{E}}_{\rho\sim\mathcal{R}_\ell} [w(L|_\rho)] &\leq \sum_{\ell\leq\log (k)} \frac{1}{(\log (k) - \ell)^2} + \sum_{\ell\leq\log (k)} \exp\left(-\frac{k}{2^{\ell+1}} \right) \\
&\leq \frac{\pi^2}{6} + \frac{e}{e-1} \,.
\end{aligned}$$

◀

References

1. Noga Alon, Amotz Bar-Noy, Nathan Linial, and David Peleg. A Lower Bound for Radio Broadcast. *J. Comput. Syst. Sci.*, 43(2):290–298, 1991. doi:10.1016/0022-0000(91)90015-W.
2. Noga Alon and Zoltán Füredi. Covering the Cube by Affine Hyperplanes. *Eur. J. Comb.*, 14(2):79–83, 1993. doi:10.1006/eujc.1993.1011.
3. Richard Beigel, Nick Reingold, and Daniel A. Spielman. The Perceptron Strikes Back. In *Proc. 6th IEEE Conf. on Structure in Complexity Theory*, pages 286–291, 1991. doi:10.1109/SCT.1991.160270.
4. Mark Braverman. Polylogarithmic independence fools AC^0 circuits. *J. ACM*, 57(5), 2010. (Preliminary version in *24th IEEE Conference on Computational Complexity*, 2009). eccc:2009/TR09-011, doi:10.1145/1754399.1754401.
5. Paul Erdős. On a lemma of Littlewood and Offord. *Bull. Amer. Math. Soc.*, 51(12):898–902, 1945. doi:10.1090/S0002-9904-1945-08454-7.
6. Prahladh Harsha and Srikanth Srinivasan. On Polynomial Approximations to AC^0. *Random Structures Algorithms*, 2018. (Preliminary version in *20th RANDOM*, 2016). doi:10.1002/rsa.20786.

7 Johan Håstad. Almost optimal lower bounds for small depth circuits. In Silvio Micali, editor, *Randomness and Computation*, volume 5 of *Advances in Computing Research*, pages 143–170. JAI Press, Greenwich, Connecticut, 1989. (Preliminary version in *18th STOC* 1986). URL: http://www.csc.kth.se/~johanh/largesmalldepth.pdf.

8 John Edensor Littlewood and A. Cyril Offord. On the Number of Real Roots of a Random Algebraic Equation. *J. London Math. Soc.*, s1-13(4):288–295, 1938. doi:10.1112/jlms/s1-13.4.288.

9 Michael Luby and Boban Velickovic. On Deterministic Approximation of DNF. *Algorithmica*, 16(4/5):415–433, 1996. (Preliminary version in *23rd STOC*, 1991). doi:10.1007/BF01940873.

10 Raghu Meka, Oanh Nguyen, and Van Vu. Anti-concentration for Polynomials of Independent Random Variables. *Theory Comput.*, 12(11):1–17, 2016. doi:10.4086/toc.2016.v012a011.

11 Alexander A. Razborov. Нжние оценки размера схем ограниченной глубины в полном базисе, содержащем функцию логического сложения (Russian) [Lower bounds on the size of bounded depth circuits over a complete basis with logical addition]. *Mathematicheskie Zametki*, 41(4):598–607, 1987. (English translation in *Mathematical Notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987). doi:10.1007/BF01137685.

12 Roman Smolensky. Algebraic Methods in the Theory of Lower Bounds for Boolean Circuit Complexity. In *Proc. 19th ACM Symp. on Theory of Computing (STOC)*, pages 77–82, 1987. doi:10.1145/28395.28404.

13 Avishay Tal. Tight Bounds on the Fourier Spectrum of AC^0. In *Proc. 32nd Comput. Complexity Conf.*, volume 79 of *LIPIcs*, pages 15:1–15:31. Schloss Dagstuhl, 2017. eccc:2014/TR14-174, doi:10.4230/LIPIcs.CCC.2017.15.

14 Jun Tarui. Probabilistic Polynomials, AC^0 Functions, and the Polynomial-Time Hierarchy. *Theoret. Comput. Sci.*, 113(1):167–183, 1993. (Preliminary version in *8th STACS*, 1991). doi:10.1016/0304-3975(93)90214-E.

15 Seinosuke Toda and Mitsunori Ogiwara. Counting Classes are at Least as Hard as the Polynomial-Time Hierarchy. *SIAM J. Comput.*, 21(2):316–328, 1992. (Preliminary version in *6th Structure in Complexity Theory Conference*, 1991). doi:10.1137/0221023.

Quasipolynomial Hitting Sets for Circuits with Restricted Parse Trees

Ramprasad Saptharishi[1]
Tata Institute of Fundamental Research, Mumbai, India
ramprasad@tifr.res.in

Anamay Tengse[2]
Tata Institute of Fundamental Research, Mumbai, India
tengse.anamay@tifr.res.in

Abstract

We study the class of non-commutative *Unambiguous circuits or Unique-Parse-Tree (UPT) circuits*, and a related model of *Few-Parse-Trees (FewPT)* circuits (which were recently introduced by Lagarde, Malod and Perifel [18] and Lagarde, Limaye and Srinivasan [17]) and give the following constructions:

- An explicit hitting set of *quasipolynomial* size for UPT circuits,
- An explicit hitting set of *quasipolynomial* size for FewPT circuits (circuits with constantly many parse tree shapes),
- An explicit hitting set of *polynomial* size for UPT circuits (of known parse tree shape), when a parameter of *preimage-width* is bounded by a constant.

The above three results are extensions of the results of [2], [10] and [9] to the setting of UPT circuits, and hence also generalize their results in the commutative world from *read-once oblivious algebraic branching programs (ROABPs)* to *UPT-set-multilinear* circuits.

The main idea is to study *shufflings* of non-commutative polynomials, which can then be used to prove suitable depth reduction results for UPT circuits and thereby allow a careful translation of the ideas in [2], [10] and [9].

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory, Theory of computation → Complexity classes

Keywords and phrases Unambiguous Circuits, Read-once Oblivious ABPs, Polynomial Identity Testing, Lower Bounds, Algebraic Circuit Complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.6

Related Version A full version of the paper is available at `https://arxiv.org/abs/1709.03068`.

Acknowledgements We thank the organizers of the NMI Workshop on Arithmetic Complexity 2017 where we learned of the circuit classes that we study in this paper. We thank Nutan Limaye and Srikanth Srinivasan for numerous discussions that eventually led to these results. We thank Rohit Gurjar for pointing out a subtlety in a previous draft of this paper, and also thank Amir Shpilka for inviting RS to Tel Aviv University (where this discussion took place). We also thank the anonymous reviewers for their valuable suggestions.

[1] Research supported by Ramanujan Fellowship of DST.
[2] Supported by a fellowship of the DAE.

38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018).
Editors: Sumit Ganguly and Paritosh Pandya; Article No. 6; pp. 6:1–6:19

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

1 Introduction

The field of algebraic complexity deals with classifying multivariate polynomials based on their hardness. Typically, the complexity of a polynomial is measured by the size of the smallest circuit computing it (an arithmetic circuit is a directed acyclic graph made up of internal nodes that are labeled with $+$ or $\times$ and leaves labelled with variables or constants from the field; size of the circuit is the number of nodes). The central question in this field is to construct an explicit family of polynomials ($\{\text{Perm}_n\}$ is the top candidate) that requires large arithmetic circuits to compute it. This is also called the "VP vs VNP" question (named after Valiant [26]), and thought of as an algebraic analogue of the "P vs NP" question.

So far, the best lower bound we have for general arithmetic circuits computing an n-variate degree d polynomial is a barely super-linear $\Omega(n \log d)$ lower bound by Baur and Strassen [6]. Recent research has focused on proving lower bounds for restricted classes of circuits, either by bounding the depth of such circuits or by focusing on other syntactic restrictions. One such syntactic restriction is to consider *non-commutative circuits*, where we assume that the underlying variables $x_1, \ldots, x_n$ do not commute. In the non-commutative model, there is an inherent order in which elements are multiplied and this adds restrictions on the way monomials can be computed ($xy \neq yx$ here and hence $x^2 + 2xy + y^2 \neq (x+y)^2 = x^2 + xy + yx + y^2$). It is therefore natural to expect that it should be easier to prove lower bounds in this model.

Nisan [20] introduced the non-commutative model, specifically the non-commutative algebraic branching programs (ABP). In his seminal paper, he showed exponential lower bounds against non-commutative ABPs for the non-commutative versions of the determinant and permanent polynomials (among others). In fact, using his technique, one could even reconstruct the smallest non-commutative ABP given just oracle access to that polynomial (cf. [16])! Although we have exponential lower bounds for non-commutative ABPs, we do not have any non-trivial lower bounds for non-commutative circuits. Hrubeš, Wigderson and Yehudayoff [12] presented an approach via *sum-of-squares* lower bounds but we do not have any non-trivial lower bounds for the class of general non-commutative circuits.

Limaye, Malod and Srinivasan [19] extended Nisan's lower bound to non-commutative skew circuits, which are circuits where every multiplication gate has at most one child that is a non-leaf. Lagarde, Malod and Perifel [18] initiated the study of non-commutative *unambiguous circuits*, or *Unique Parse Tree (UPT)* circuits. These circuits and their generalizations are the main models of study in this paper.

Arvind and Raja [5] also studied lower bounds for various subclasses of commutative set-multilinear circuits. Some of the models they studied include analogues of UPT and FewPT circuits. They proved lower bounds for UPT and FewPT set-multilinear circuits, and also for other subclasses of set-multilinear circuits called *narrow* set-multilinear circuits and *interval* set-multilinear circuits, the latter of which assumes the sum-of-squares conjecture of Hrubeš, Wigderson and Yehudayoff [12].

1.1 The model of study

A parse tree of a circuit is obtained by starting at the root, and at every $+$ gate choosing exactly one child, and at every $\times$ gate choosing all its children (formally defined in Theorem 2.1). Informally, a parse tree of a circuit is basically a *certificate* of computation of a monomial in a circuit. Lagarde, Malod and Perifel [18] introduced a subclass of non-commutative circuits called *Unique Parse Tree (UPT) circuits* or *unambiguous circuits* where all parse trees of the circuit have the same shape (formally defined in Theorem 2.2). The class of

non-commutative UPT circuits subsumes the class of non-commutative ABPs as any ABP can be expressed as a left-skew circuit. A related model of *set-depth-Δ formulas* was studied by Agrawal, Saha and Saxena [3] that is a subclass of UPT circuits where the underlying parse trees are extremely regular[3].

Lagarde, Malod and Perifel [18] extended the techniques of Nisan [20] to give exponential lower bounds for UPT circuits. Subsequently, Lagarde, Limaye and Srinivasan [17] extended the lower bounds to the class of circuits with parse trees of not-too-many shapes (at most $2^{o(n)}$ shapes).

1.2 Polynomial identity testing

A *Polynomial Identity Test (PIT)* is an algorithm that, given a circuit as input, checks if the circuit is computing the zero polynomial or not. The standard Ore-DeMillo-Lipton-Schwartz-Zippel lemma [22, 7, 24, 28] provides a simple randomized algorithm but the goal is to construct an efficient deterministic PIT. A stronger test is what is called a *black-box PIT* where we are only provided evaluation access to the circuit. Hence, a black-box PIT is essentially equivalent to constructing a *hitting set*, i.e., a set of points (or matrices, in the non-commutative case) $\mathcal{H}$ such that every non-zero polynomial from the class of interest is guaranteed to evaluate to a nonzero value on some element $\mathbf{a} \in \mathcal{H}$. PITs that use the structure of the circuit are called *white-box* PITs.

The task of constructing efficient PITs is intimately connected to the task of proving lower bounds [11, 15, 1]. Once we have a lower bound for a class $\mathcal{C}$, it is natural to ask if we can also construct efficient PITs for that class. Raz and Shpilka [23] gave the first deterministic polynomial time white-box PIT for the class of non-commutative ABPs. Forbes and Shpilka [8] gave a quasipolynomial ($n^{O(\log n)}$) size hitting set for non-commutative ABPs. This was achieved by studying a natural commutative analogue of non-commutative ABPs, and this was the class of *Read-Once Oblivious Algebraic Branching Programs (ROABPs)* where the variables are read in a "known order".

The class of ROABPs is interesting in its own right owing to the connection with the "RL vs L" question. In fact, much of the hitting set constructions for ROABPs has been inspired by Nisan's [21] pseudorandom generator for RL (which has seed length $O(\log^2 n)$). As mentioned earlier, Forbes and Shpilka gave a hitting set of size $n^{O(\log n)}$ for polynomial sized ROABPs when the order in which variables are read was known. Agrawal, Gurjar, Korwar and Saxena [2] presented a different hitting set for the class of commutative ROABPs that did not need the knowledge of the order in which the variables were read. Subsequently, Gurjar, Korwar, Saxena and Thierauf [10] studied polynomials that can be computed as a sum of constantly many ROABPs (of possibly different orders) and presented a polynomial time white-box PIT, and also a quasipolynomial time black-box PIT for this class.

Lagarde, Malod and Perifel [18], besides presenting lower bounds for non-commutative UPT circuits, also gave a polynomial time white-box PIT for this class. This was extended by Lagarde, Limaye and Srinivasan [17] to a white-box algorithm for non-commutative circuits with constantly many parse tree shapes (analogous to the result of [10]). The question of constructing black-box PITs was left open by them, and we answer this in our paper.

[3] the formula is levelled, and all nodes at a level have the same fan-in

1.3 Our results

Polynomial Identity Testing

Our main results are hitting sets for the class of polynomials computed by UPT circuits and related classes.

▶ **Theorem 1.1** (Hitting sets for UPT circuits). *There is an explicit hitting set $\mathcal{H}_{d,n,s}$ of at most $(snd)^{O(\log d)}$ size for the class of degree d n-variate homogeneous non-commutative polynomials in $\mathbb{F}\langle x_1, \ldots, x_n\rangle$ that are computed by UPT circuits of size at most s.*

This result builds on the technique of *basis isolating weight assignments* introduced by [2] for constructing hitting sets for ROABPs. Furthermore, we can also extend the hitting set to the class of non-commutative circuits that have *few shapes* (analogous to [10]'s hitting set for sums of few ROABPs).

▶ **Theorem 1.2** (Hitting sets for circuits with few parse tree shapes). *There is an explicit hitting set $\mathcal{H}_{d,n,s,k}$ of size at most $(s^{2^k}nd)^{O(\log d)}$ for the class of n-variate degree d homogeneous non-commutative polynomials in $\mathbb{F}\langle x_1, \ldots, x_n\rangle$ that are computed by non-commutative circuits of size at most s consisting of parse trees of at most k shapes.*

Both the above theorems are fully black-box in the sense that it is not required to know the underlying shape(s). For the case of non-commutative ABPs (and more generally, ROABPs in a known order), Gurjar, Korwar and Saxena [9] presented a more efficient hitting set when the width of the ABP is small. For UPT circuits, there is a natural notion of *preimage-width* of a UPT circuit (formally defined in Theorem 2.3) that corresponds to the notion of width of an ABP. We show an analogue of the hitting set of Gurjar, Korwar and Saxena for the class of UPT circuits of small *preimage-width* if the underlying shape of the parse trees is known.

▶ **Theorem 1.3** (Hitting sets for known-shape low-width UPT circuits). *Let $\mathcal{C}_{n,d,T,w}$ be the class of n-variate degree d non-commutative polynomials that are computable by UPT circuits of preimage-width at most w and underlying parse-tree shape as T. Over any field of zero or large characteristic, there is an explicit hitting set $\mathcal{H}_{n,d,T,w}$ of size $w^{O(\log d)}\,\mathrm{poly}(nd)$ for $\mathcal{C}_{n,d,T,w}$.*

These hitting sets also translate to the natural commutative analogues of *UPT set-multilinear circuits* etc. (formally defined in Theorem 5.1).

Structural results

If f is a non-commutative polynomial of degree d and if $\sigma \in S_d$ is a permutation on d letters, we define the *shuffling* of f by σ (denoted by $\Delta_\sigma(f)$) as the natural operation of permuting each *word* of f according to σ.

The three PIT statements stated above begin with the following depth reduction statement about UPT circuits.

▶ **Theorem 1.4** (Depth reduction for UPT circuits). *Let f be an n-variate degree d polynomial that is computable by a UPT circuit of preimage-width w. Then, there is some $\sigma \in S_d$ such that $\Delta_\sigma(f)$ can be computed by a UPT circuit of $O(\log d)$ depth and preimage-width $O(w^2)$.*

The above theorem implies that $\Delta_\sigma(f)$ is computable by an ABP of quasipolynomial size. We also show that this blow-up of quasipolynomial size is tight.

▶ **Theorem 1.5** (Separating UPT circuits and ABPs, under shuffling)**.** *There is an explicit n-variate degree d non-commutative polynomial f that is computable by UPT circuits of preimage-width $w = \text{poly}(n, d)$ such that for every $\sigma \in S_d$, the polynomial $\Delta_\sigma(f)$ requires non-commutative ABPs of size $(nd)^{\Omega(\log nd)}$ to compute it.*

We also extend the lower bound of [18] to give a polynomial computed by a *skew circuit* that requires exponential sized UPT circuits under any shuffling. Details can be found in the full version.

1.4 Proof ideas

As mentioned, the starting point of all these results is the depth reduction. From a result of Nisan [20], the palindrome polynomial Pal_d is known to require ABPs of size $2^{\Omega(d)}$ even though it can be computed by a polynomial sized UPT circuit. Therefore, Pal_d cannot be computed by a circuit of depth $o(d/\log d)$. The key insight here is that even though Pal_d cannot be computed by small depth non-commutative circuits, a shuffling of the palindrome is

$$\sum_{w_1,\ldots,w_d \in [n]} x_{w_1} x_{w_1} x_{w_2} x_{w_2} \cdots x_{w_d} x_{w_d} = \prod_{i=1}^{d} (x_1 x_1 + \cdots + x_n x_n)\,,$$

which is of course computable by an $O(\log d)$ depth UPT formula even. Hence we attempt to reduce the depth under a suitable shuffling.

In order to establish the depth reduction (Theorem 1.4) we follow the strategy of Valiant, Skyum, Berkowitz and Rackoff [27] and Allender, Jiao, Mahajan and Vinay [4] but make use of the UPT structure (work with different *frontier nodes* and *gate quotients*) based on the underlying shape of the parse trees. It was pointed out to us that the key ideas in our proof of depth reduction were used by Arvind and Raja ([5]) for a commutative analogue of UPT circuits.

This depth reduction immediately yields that there is a quasipolynomial sized ABP computing a shuffling of f. We show that this blow-up is tight (Theorem 1.5) by essentially following the proof of Hrubeš and Yehudayoff [13] to separate monotone ABPs and monotone circuits in the commutative world.

In order to obtain hitting sets for UPT circuits, one could potentially just use the fact that there is a quasipolynomial sized ABP computing a shuffling of f and just use the known hitting sets for non-commutative ABPs [8] to obtain a hitting set of $\text{poly}(ndw)^{O(\log^2 d)}$. However, we directly work with the UPT circuit and lift the technique of *basis isolating weight assignments* of Agrawal, Gurjar, Korwar and Saxena [2] to this more general setting to obtain Theorem 1.1. Theorem 1.3 is an easy generalization of the ideas of Gurjar, Korwar and Saxena [9] once we observe that the depth reduction keeps the preimage-width small.

Theorem 1.2 essentially follows the same ideas of Gurjar, Korwar, Saxena and Thierauf [10]. The techniques of [10] are general enough that once a circuit class has a *characterizing set of dependencies* and a *basis isolating weight assignment*, there is a natural method to lift the techniques to work with the sum of few elements from this class. [10] use this for ROABPs and we use this for UPT circuits.

To summarize, once we obtain the depth reduction, much of the results in this paper is a careful translation of prior work of [13], [2], [10], [9] to the setting of UPT (or FewPT) circuits. Consequently, this also generalizes the hitting sets of [2, 10, 9] from ROABPs to *UPT (or FewPT) set-multilinear* circuits. Such a generalization was unknown prior to this work.

2 Preliminaries

2.1 Notation

- We use $\mathbb{F}\langle x_1, \ldots, x_n\rangle$ to refer to the ring of polynomials in non-commuting variables $\{x_1, \ldots, x_n\}$. For a parameter d, we use $\mathbb{F}\langle x_1, \ldots, x_n\rangle_{\deg=d}$ to refer to the set of polynomials in $\mathbb{F}\langle x_1, \ldots, x_n\rangle$ that are homogeneous and of degree d. Similarly, the set of polynomials of degree at most d will be denoted by $\mathbb{F}\langle x_1, \ldots, x_n\rangle_{\deg\leq d}$.
- We use boldface letters $\mathbf{x}$ and $\mathbf{y}$ to denote sets of variables (the number of variables would be clear from context). We shall also use $[d]$ to refer to the set $\{1, 2, \ldots, d\}$.
- The paper will sometimes shift between the commutative and the non-commutative domains. We use $\mathbf{x}$ whenever we are talking about non-commutative variables, and $\mathbf{y}$, $\mathbf{z}$ for variables in the commutative domain.

2.2 Basic definitions

UPT and FewPT circuits

▶ **Definition 2.1** (Parse trees)**.** A parse tree T of a circuit C is a tree obtained as follows:

- the root of C is the root of T,
- if $v \in T$ is a $\times$ gate, then all the children of v in C are the children of v in T in the same order,
- if $v \in T$ is a $+$ gate, then exactly one child of v in C is a child of v in T.

Gates are replicated to ensure that T is a tree. The value of the parse tree T, denoted by $[T]$, is just the product of the leaf labels in T.

Intuitively, a parse tree is a *certificate* that a monomial was produced in the computation of C (though it could potentially be canceled by other parse trees computing the same monomial). Therefore, if f is the polynomial computed by C, then

$$f = \sum_{T \text{ is a parse tree}} [T].$$

▶ **Definition 2.2.** (UPT and FewPT circuits) A circuit C computing a homogeneous polynomial is said to be a *Unique Parse Tree (UPT) circuit* if all parse trees of C have the same shape (that is, they are identical except perhaps for the gate names).

A circuit C that computes a homogeneous polynomial is said to be a FewPT(k) circuit if the parse trees of C have at most k distinct shapes.

▶ **Definition 2.3** (Preimage-width)**.** Suppose C is a UPT circuit and say T is the shape of the underlying parse trees. For a node $\tau \in T$ and a gate $g \in C$, we shall say that g is a *preimage* of τ, denoted by $g \sim \tau$, if and only if there is some parse tree T' of C where the gate g appears in position τ.

The *preimage-width* of a UPT circuit C is the largest size of preimages of any node $\tau \in T$. That is,

$$\text{preimage-width}(C) = \max_{\tau \in T} |\{g \in C \ : \ g \sim \tau\}|.$$

It is clear that if C is a UPT circuit of preimage-width w computing a homogeneous degree d polynomial, then the size of C is at most dw. The preimage-width of a UPT circuit is a more useful measure to study than the size of the circuit. A simple concrete example of this is that the standard conversion of homogeneous ABPs to homogeneous circuits in

fact yields UPT circuits. Furthermore, the width of the ABP is directly related to the preimage-width of the resulting UPT circuit.

▶ **Observation 2.4.** *If f is computable by a width w homogeneous algebraic branching program, then f can be equivalently computed by UPT circuits of preimage-width w^2.*

$\times_p$-products

▶ **Definition 2.5** ($\times_p$-products)**.** For any $d_1, d_2 \geq 0$ and p satisfying $0 \leq p \leq d_2$, define a map $\times_p : \mathbb{F}\langle x_1, \ldots, x_n\rangle_{\deg=d_1} \times \mathbb{F}\langle x_1, \ldots, x_n\rangle_{\deg=d_2} \to \mathbb{F}\langle x_1, \ldots, x_n\rangle_{\deg=d_1+d_2}$ as the unique bilinear that satisfies

$$x_{w_1} \cdots x_{w_{d_1}} \times_p x_{v_1} \cdots x_{v_{d_2}} = x_{v_1} \cdots x_{v_p} x_{w_1} \cdots x_{w_{d_1}} x_{v_{p+1}} \cdots x_{v_{d_2}}.$$

For instance, the usual multiplication (or concatenation) operation is just $\times_0$.

Shuffling of a polynomial

▶ **Definition 2.6** (Shuffling of a non-commutative polynomial)**.** Let $P_d(x_1, \ldots, x_n)$ be a homogeneous degree d non-commutative polynomial from $\mathbb{F}\langle x_1, \ldots, x_n\rangle_{\deg=d}$. Given any permutation $\sigma \in S_d$ over d-letters, we can define the *shuffling of P_d via σ* as the unique linear map $\Delta_\sigma : \mathbb{F}\langle x_1, \ldots, x_n\rangle_{\deg=d} \to \mathbb{F}\langle x_1, \ldots, x_n\rangle_{\deg=d}$ that is obtained by linearly extending

$$\Delta_\sigma(x_{w_1} \cdots x_{w_d}) = x_{w_{\sigma(1)}} \cdots x_{w_{\sigma(d)}}.$$

2.3 Basic lemmas

Canonical UPT circuits, and types of gates

We shall say that a UPT circuit C with underlying parse tree shape T is *canonical* if for every gate $g \in C$ there is some node $\tau \in T$ such that every parse tree of C involving g has g only in position τ. That is, every gate of the circuit has a unique type associated with it.

▶ **Lemma 2.7** ([18])**.** *Suppose if $f \in \mathbb{F}\langle x_1, \ldots, x_n\rangle$ is a homogeneous, degree d, non-commutative polynomial computed by a non-commutative UPT circuit of preimage-width w. Then, f can be equivalently computed by a canonical UPT circuit of preimage-width w as well.*

For a canonical UPT circuit where the parse trees have shape T, we shall say that g has type τ if $\tau \in T$ is the unique node in T such that $g \sim \tau$.

Fix a $\tau \in T$ and let i be the number of leaves of the subtree rooted at τ, and let p be the number of leaves to the left of τ in the inorder traversal of T. We shall then say that τ (or a gate $g \in C$ of type τ) has *position-type* (i, p). The following lemma allows us to write the polynomial computed by the circuit as a small sum of $\times_p$-products.

▶ **Lemma 2.8** ([18])**.** *Let f be a polynomial computed by a canonical UPT circuit C of preimage-width w and say T is the shape of the underlying parse trees. If $\tau \in T$ with position-type (i, p), then we can write f as*

$$f(\mathbf{x}) = \sum_{r=1}^{w} g_r(\mathbf{x}) \times_p h_r(\mathbf{x}),$$

where $\deg g_r = i$ and $\deg h_r = \deg(f) - i$ for all $r = 1, \ldots, w$.

3 Depth reduction for UPT circuits

This section shall address Theorem 1.4, which we recall below.

▶ **Theorem 1.4** (Depth reduction for UPT circuits). *Let f be an n-variate degree d polynomial that is computable by a UPT circuit of preimage-width w. Then, there is some $\sigma \in S_d$ such that $\Delta_\sigma(f)$ can be computed by a UPT circuit of $O(\log d)$ depth and preimage-width $O(w^2)$.*

It was pointed out to us that a similar depth reduction was also proved by Arvind and Raja [5]. They showed that a commutative UPT set-multilinear circuit can be depth-reduced to a corresponding quasipolynomial sized $O(\log d)$ depth UPT set-multilinar formula via Hyafil's [14] depth reduction. Using techniques similar to [27], one can directly obtain a polynomial sized UPT circuit of depth $O(\log d)$. Though this can be inferred from the results in [5], we state and prove it in the form needed for the non-commutative setting.

3.1 UPT $\otimes$-circuits

To prove the depth reduction, we will move to an intermediate model of UPT $\otimes$-circuits.

▶ **Definition 3.1** (UPT $\otimes$-circuits). The class of UPT $\otimes$-circuits is a generalization of homogeneous non-commutative circuits in that the internal gates are $+$ gates and $\times_p$ gates instead of the usual $+$ and $\times$ gates. We shall also say that the circuit is *semi-unbounded* if all $\times_p$ gates have fan-in bounded by 2 (with no restriction on $+$ gates).

A *parse tree* for an $\otimes$-circuit is similar to parse trees in a general non-commutative circuit but the internal nodes of the parse tree are labelled by $+$ and $\times_p$ (with the p specified at each gate).

We shall say that an $\otimes$-circuit C is UPT if every parse tree is of the same shape, i.e. two parse trees in C can differ only in the gate names.

To prove Theorem 1.4, we begin by depth reducing the circuit to get an $\otimes$-circuit computing f of $O(\log d)$ depth. We then convert that to a UPT circuit computing a shuffling of f.

▶ **Lemma 3.2** (Depth reducing to $\otimes$-circuits). *Let $f \in \mathbb{F}\langle x_1, \ldots, x_n\rangle$ be a homogeneous degree d polynomial that is computable by a UPT circuit of size s. Then, f can equivalently be computed by a semi-unbounded UPT $\otimes$-circuit of size $O(s^2)$ and depth $O(\log d)$.*

The rough sketch is to follow a similar process as in [27] by defining a suitable notion of a *gate quotient* $[u : v]$ for this setting. The set of *frontier* nodes is different from the previous depth reduction results but Theorem 2.8 allows us to essentially follow the same strategy to obtain the above depth-reduced UPT circuit.

Proof. Let C be the UPT circuit computing $f(x_1, \ldots, x_n)$ and say T is the shape of the parse trees of C. For any node $\tau \in T$, let $\mathcal{F}_\tau$ be the set of all gates in C whose position in T is τ. For two gates $u, v \in C$, we shall say that $u \succeq v$ if the place of u in T is an ancestor of the place of v in T. We shall abuse notation and use $u \succeq \tau$ to mean that u's position in T is an ancestor of $\tau \in T$. For a gate $u \in C$, let $[u]$ refer to the polynomial computed at that gate. Similar to [27, 4], we define inductively the following notion of a *gate quotient* for any

pair of gates $u, v \in C$:

$$[u : v] = \begin{cases} 0 & \text{if } u \not\succeq v, \\ 1 & \text{if } u = v, \\ [u_1 : v] + [u_2 : v] & \text{if } u = u_1 + u_2, \\ [u_1 : v] \cdot [u_2] & \text{if } u = u_1 \times u_2 \text{ and } u_1 \succeq v, \\ [u_1] \cdot [u_2 : v] & \text{if } u = u_1 \times u_2 \text{ and } u_2 \succeq v. \end{cases}$$

▶ **Claim 3.3.** *For any $u \in C$, if $\tau \in T$ such that $u \succeq \tau$, then*

$$[u] = \sum_{\substack{w \in C \\ w \sim \tau}} [w] \times_p [u : w] \tag{3.1}$$

for a suitable p depending just on τ and the type of u. Furthermore, suppose $u, v \in C$ with v being a multiplication gate and if $\tau \in T$ such that $u \succeq \tau \succeq v$ then

$$[u : v] = \sum_{\substack{w \in C \\ w \sim \tau}} [w : v] \times_p [u : w]. \tag{3.2}$$

for a suitable p depending just on τ and the type of u and v.

We'll defer this proof to later and first finish the proof of Theorem 3.2. With (3.1) and (3.2), we can construct the $\otimes$-circuit C' for f just as in [27, 4]. The circuit C' would have gates computing each $[u]$ and $[u : v]$ for nodes $u, v \in C$ with $u \succeq v$ and v being a multiplication gate. The wirings in C' is built by appropriate applications of (3.1) and (3.2).

Let $u \in C$ and say $\deg[u] = d_u$. The plan would be to set up the computation in C' so that using an $O(1)$ depth computation, we can compute $[u]$ using gates whose degrees are a constant factor smaller than d_u. Consider any parse tree rooted at u, and starting from u follow the *higher degree* child. Let τ be the last point on the path with degree $\geq d_u/2$ (degree of its children will be $< d_u/2$). Applying (3.1),

$$\begin{aligned} [u] &= \sum_{w \sim \tau} [w] \times_p [u : w] \\ &= \sum_{w \sim \tau} ([w_1] \times [w_2]) \times_p [u : w] \qquad \text{where } w = w_1 \times w_2. \end{aligned}$$

Now observe that each of the terms on the RHS, $[u : w], [w_1], [w_2]$ have degree at most $d_u/2$, as we wanted. Furthermore, each coordinate of tuple $([u : w], [w_1], [w_2])$ are all of the same *type* as we run over all $w \sim \tau$.

We now need to show how to compute $[u : v]$ for a pair $u \succ v$. Say $\deg[u] = d_u$ and $\deg[v] = d_v$. For this, start with some parse tree rooted at u and walk down the path leading to the place of v, and let τ be the last point on this path such that $\deg \tau \geq \frac{d_u + d_v}{2}$. Using (3.2),

$$\begin{aligned} [u : v] &= \sum_{w \sim \tau} [w : v] \times_p [u : w] \\ &= \sum_{w \sim \tau} ([w_1] \times [w_2 : v]) \times_p [u : w] \end{aligned}$$

where $w = w_1 \times w_2$ and $w_2 \succeq v$ (the other possibility is identical). By the choice of τ, we have $\deg[u : w], \deg[w_2 : v] \leq \frac{d_u - d_v}{2}$. However, the best bound we can give on $\deg[w_1]$

is $d_u - d_v$. Nevertheless, we can apply (3.1) again on $[w_1]$ by finding a suitable $\tau' \prec w_1$ satisfying $\deg \tau' \geq \frac{\deg w_1}{2}$ and write

$$\begin{aligned}
[u:v] &= \sum_{w \sim \tau} ([w_1] \times [w_2 : v]) \times_p [u : w] \\
&= \sum_{w \sim \tau} \left(\left(\sum_{w' \sim \tau'} [w'] \times_{p'} [w_1 : w'] \right) \times [w_2 : v] \right) \times_p [u : w] \\
&= \sum_{w \sim \tau} \sum_{w' \sim \tau'} ((([w'_1] \times [w'_2]) \times_{p'} [w_1 : w']) \times [w_2 : v]) \times_p [u : w]
\end{aligned}$$

By the choice of τ and τ', each of the *factors* on the RHS have degree at most $\frac{(d_u - d_v)}{2}$ as we wanted. Furthermore, once again, all of the summands consists of similarly typed factors.

This naturally yields an $\otimes$-circuit computing f of depth $O(\log d)$ and size $O(s^2)$. Since all summands consist of similarly typed factors, it follows that the circuit is UPT as well. ◀

Proof of Claim 3.3. The proof is by induction. As a base case, suppose $u \sim \tau$. Then, $[u]$ is just the sum of the values of parse trees. Some of the parse trees use u. Of all nodes $w \in C$ such that $w \sim \tau$, only $[u : u] = 1$ and every other $[u : w] = 0$. Therefore, clearly $[u] = \sum_{w \sim \tau} [w] \cdot [u : w]$.

Now suppose $u \succ \tau$ and say we already know that $[u'] = \sum_{w \sim \tau} [w] \times_p [u' : w]$ for every $u \succ u' \succeq \tau$. If $u = u_1 + u_2$, then

$$\begin{aligned}
[u] &= [u_1] + [u_2] \\
&= \left(\sum_{w \sim \tau} [w] \times_p [u_1 : w] \right) + \left(\sum_{w \sim \tau} [w] \times_p [u_2 : w] \right) \\
&= \sum_{w \sim \tau} [w] \times_p ([u_1 : w] + [u_2 : w]) \\
&= \sum_{w \sim \tau} [w] \times_p [u : w].
\end{aligned}$$

Similarly, suppose $[u] = [u_1] \times [u_2]$. We have two cases depending on whether $u_1 \succeq \tau$ or $u_2 \succeq \tau$.

If $u_1 \succeq \tau$, then

$$\begin{aligned}
[u] &= [u_1] \times [u_2] \\
&= \left(\sum_{w \sim \tau} [w] \times_p [u_1 : w] \right) \times [u_2] \\
&= \sum_{w \sim \tau} [w] \times_p ([u_1 : w] \times [u_2]) \\
&= \sum_{w \sim \tau} [w] \times_p [u : w].
\end{aligned}$$

If $u_2 \succeq \tau$, then

$$\begin{aligned}
[u] &= [u_1] \times [u_2] \\
&= [u_1] \times \left(\sum_{w \sim \tau} [w] \times_p [u_2 : w] \right) \\
&= \sum_{w \sim \tau} [w] \times_{p + \deg u_1} ([u_1] \times [u_2 : w]) \\
&= \sum_{w \sim \tau} [w] \times_{p + d_1} [u : w].
\end{aligned}$$

Essentially the same proof works for (3.2) as well. ◀

▶ **Lemma 3.4** ($\otimes$-circuits to circuits for a shuffling)**.** *Let $f \in \mathbb{F}\langle x_1, \ldots, x_n \rangle$ be a homogeneous degree d polynomial that is computable by a UPT $\otimes$-circuit C' of size s. Consider the circuit C'' obtained by replacing all $\otimes$ gates in C' by $\times$ gates. Then, C'' computes $\Delta_\sigma(f)$ for some $\sigma \in S_d$.*

Proof. We shall prove this by induction. We need a slightly stronger inductive hypothesis which is that the choice of permutation σ depends only on the shape of the parse trees in C'.

Say u is the root of C'. Suppose u is a $+$ gate and say $u = u_1 + u_2 + \cdots + u_r$. If $u' = u'_1 + \cdots + u'_r$ is the resulting computation in C'' then by the inductive hypothesis, we know that there is a $\sigma \in S_d$ such that $[u'_i] = \Delta_\sigma([u_i])$. Therefore,

$$[u'] = \sum_{i=1}^{r} \Delta_\sigma([u_i]) = \Delta_\sigma([u]).$$

Suppose $u = u_1 \times_p u_2$ with $\deg[u_1] = d_1$ and $\deg[u_2] = d_2$. Say $u_1 = \sum_{\alpha \in [n]^{d_1}} a_\alpha x_\alpha$ and $\sum_{\beta \in [n]^{d_2}} b_\beta x_\beta$. Then, $[u] = \sum_{\alpha,\beta} a_\alpha b_\beta \cdot x_\alpha \times_p x_\beta$. If u', u'_1 and u'_2 is the resulting computation in C'', then

$$\begin{aligned}
[u'] &= [u'_1] \times [u'_2] \\
&= \Delta_{\sigma_1}([u_1]) \times \Delta_{\sigma_2}([u_2]) && \text{for some } \sigma_1 \in S_{d_1}, \sigma_2 \in S_{d_2}, \\
&= \sum_{\alpha,\beta} a_\alpha b_\beta \cdot (\Delta_{\sigma_1}(x_\alpha) \times \Delta_{\sigma_2}(x_\beta)) \\
&= \sum_{\alpha,\beta} a_\alpha b_\beta \cdot \Delta_\sigma(x_\alpha \times_p x_\beta) && \text{for some } \sigma \in S_d, \\
&= \Delta_\sigma([u])
\end{aligned}$$

◀

The following corollary is immediate from the fact that any circuit of depth d and size s can be computed by a formula of size $s^{O(d)}$ and hence an ABP of size $s^{O(d)}$.

▶ **Corollary 3.5.** *If $f \in \mathbb{F}\langle x_1, \ldots, x_n \rangle$ is a homogeneous degree d polynomial that is computable by a UPT circuit of size s, then there is some $\sigma \in S_d$ such that $\Delta_\sigma(f)$ is computable by a non-commutative algebraic branching program of size $s^{O(\log d)}$.*

Furthermore, the shuffling σ that permits this can also be efficiently computed given the underlying shape for the circuit computing f.

3.2 UPT circuits of constant width

For a UPT circuit C, recall that we say that its preimage-width is w if for every node τ in the shape T, there are at most w gates of C that have type τ. The following observation is evident from the proof of the above depth reduction.

▶ **Observation 3.6.** *If C is a UPT circuit of width w, then the depth reduced circuit C' as obtained in Theorem 1.4 has width $O(w^2)$.*

This observation would allow us to yield a more efficient hitting set for the class of *small width known shape* UPT circuits. Details are present in the full version.

4 Separating ROABPs and UPT circuits

▶ **Theorem 1.5** (Separating UPT circuits and ABPs, under shuffling)**.** *There is an explicit n-variate degree d non-commutative polynomial f that is computable by UPT circuits of preimage-width $w = \mathrm{poly}(n, d)$ such that for every $\sigma \in S_d$, the polynomial $\Delta_\sigma(f)$ requires non-commutative ABPs of size $(nd)^{\Omega(\log nd)}$ to compute it.*

The polynomial and the proof technique described here were introduced by Hrubeš and Yehudayoff [13] to separate monotone circuits and monotone ABPs in the commutative regime. The polynomial used here is a non-commutative analogue of the polynomial used by [13]. Much of the proof is the argument of [13] tailored to the non-commutative setting.

4.1 The polynomial

Let T_d denote the complete binary tree of depth d (with 2^d leaves) and let $D = 2^{d+1} - 1$ refer to the number of nodes in T_d. We shall say that a colouring $\gamma : T_d \to \mathbb{Z}_m$ is *legal* if for every node $u \in T$, if v and w are the children of u then $\gamma(u) = \gamma(v) + \gamma(w) \bmod m$.

Let $v_1, \ldots, v_D$ be the vertices of T_d listed in an *in-order* manner (left-subtree listed inductively, then the root, and then the right-subtree listed inductively). We now define the non-commutative polynomial $P_d(x_1, \ldots, x_m) \in \mathbb{F}\langle x_1, \ldots, x_m \rangle$ of degree $D = 2^{d+1} - 1$ as

$$P_d(x_1, \ldots, x_m) = \sum_{\substack{\gamma \in [m]^D \\ \gamma \text{ is legal}}} x_{\gamma(v_1)} x_{\gamma(v_2)} \cdots x_{\gamma(v_D)}. \tag{4.1}$$

▶ **Lemma 4.1** (Upper bound). *For every $m, d > 0$, the polynomial $P_d(y_1, \ldots, y_m)$ can be computed by a non-commutative UPT circuit of size $O(m^2 d)$.*

(Refer to the full version for a proof.)

▶ **Theorem 4.2** (Lower bound). *For every permutation $\sigma \in S_D$, any non-commutative ABP computing the polynomial $\Delta_\sigma(P_d)$ has width $m^{\Omega(d)}$.*

Hence for $d = \log m$, we have that $P_d(x_1, \ldots, x_m)$ is computable by a UPT circuit of size $O(m^2 \log m)$ but for every $\sigma \in S_D$ the above theorem tells us that $\Delta_\sigma(P_d)$ requires ABPs of width $m^{\Omega(\log m)}$ to compute it. The lower bound follows on exactly same lines as the [13]. A proof is present in the full version.

5 Hitting sets for non-commutative models

Commutative brethren of non-commutative models

This reduction to an appropriate commutative case was used by Forbes and Shpilka [8] to reduce constructing hitting sets for non-commutative ABPs to hitting sets for commutative ROABPs (more precisely, to set-multilinear ABPs). They studied the image of the non-commutative polynomial under the map $\Psi : \mathbb{F}\langle x_1, \ldots, x_n \rangle_{\deg=d} \to \mathbb{F}[y_{1,1}, \ldots, y_{d,n}]$ which is the unique $\mathbb{F}$-linear map given by $\Psi : x_{w_1} \cdots x_{w_d} \mapsto y_{1,w_1} \cdots y_{d,w_d}$.

For the model of non-commutative UPT circuits, the appropriate commutative model is a restriction of set-multilinear circuits that we call UPT set-multilinear (UPT-SML) circuits.

▶ **Definition 5.1** (Set-multilinear circuits). Let $\mathbf{y} = \mathbf{y}_1 \sqcup \cdots \sqcup \mathbf{y}_d$ be a partition of the variables. A circuit C computing a polynomial $f \in \mathbb{F}[\mathbf{y}]$ is said to be a *set-multilinear circuit* with respect to the above partition if:

- each gate $g \in C$ is labelled by a subset $S_g \subseteq [d]$ and g computes a polynomial over variables $\bigcup_{i \in S_g} \mathbf{y}_i$ where every monomial of $[g]$ is divisible by exactly one variable in $\mathbf{y}_i$ for each $i \in S_g$,
- if g is a $+$ gate, then the subset that labels g also labels each of its children,
- if g is a $\times$ gate with g_1 and g_2 being its children, then the subsets S_{g_1} and S_{g_2} labelling g_1 and g_2 respectively is a partition of S_g, i.e. $S_g = S_{g_1} \sqcup S_{g_2}$.

We shall say the circuit C is *UPT set-multilinear* if every parse tree of C is of the same shape and identically labelled. That is, if g and g' are $\times$ gates labelled by a set $S \subseteq [d]$, and if $g = g_1 \times g_2$ with S_1 and S_2 labelling g_1 and g_2, then the children of g' are also labelled by S_1 and S_2 respectively.

We shall say the set-multilinear circuit C is *FewPT(k) set-multilinear* if the circuit consists of parse trees of at most k different shapes.

A natural generalization that will be useful later is a *multi-output UPT set-multilinear* circuit, which is a UPT set-multilinear circuit that potentially has multiple output gates, which are all labelled with the same subset.

Forbes and Shpilka [8] showed that constructing hitting sets for these commutative models suffices for the non-commutative models by a simple reduction (details in the full version). We shall therefore focus on these commutative models for the hitting set constructions. And since we have already seen that such circuits can be depth reduced[4] to $O(\log d)$ depth, it suffices to construct a hitting set for $O(\log d)$-depth UPT and FewPT set-multilinear circuits.

5.1 Hitting sets for UPT set-multilinear circuits

▶ **Theorem 5.2** (Hitting sets for UPT set-multilinear circuits). *Let $\mathcal{C}$ be the class of n-variate degree d set-multilinear polynomials (with respect to $\mathbf{y} = \mathbf{y}_1 \sqcup \cdots \sqcup \mathbf{y}_d$) that are computable by UPT set-multlinear circuits of preimage-width w and depth r. Then, for $M = \left(\binom{w}{2} n^2 d + 1\right)^2$, the set*

$$\mathcal{H} = \left\{ (b_{11}, \ldots, b_{dn}) \ : \ \mathbf{p} \in [M]^r \ , \ a_k \in A \ , \ b_{ij} = \prod_{k=1}^{r+1} a_k^{2^{(i-1)n+(j-1)} \bmod p_i} \right\}$$

is a hitting set for $\mathcal{C}$ of size $\mathrm{poly}(ndw)^r$.

The proof of this theorem is obtained by constructing what is called a *basis isolating weight assignment* for polynomials simultaneously computed by a multi-output UPT-SML circuit, heavily borrowing from the ideas in [2]. The details of the hitting set construction are present in the full version.

5.2 Poly-sized hitting sets for constant width UPT circuits

▶ **Theorem 1.3** (Hitting sets for known-shape low-width UPT circuits). *Let $\mathcal{C}_{n,d,T,w}$ be the class of n-variate degree d non-commutative polynomials that are computable by UPT circuits of preimage-width at most w and underlying parse-tree shape as T. Over any field of zero or large characteristic, there is an explicit hitting set $\mathcal{H}_{n,d,T,w}$ of size $w^{O(\log d)}\,\mathrm{poly}(nd)$ for $\mathcal{C}_{n,d,T,w}$.*

This is an easy extension of the ideas from [9], a proof can be found in the full version.

6 FewPT circuits

In this section we describe the black-box identity test for FewPT(k) circuits. The following lemma from [17] shows that this class is equivalent to polynomials computed by sum of k UPT circuits (of possibly different shapes).

6.1 Preliminaries

▶ **Lemma 6.1** ([17], Lemma 16). *Let $f(\mathbf{x})$ be a polynomial computed by FewPT(k) circuit of preimage-width w. Then f can be equivalently computed by a sum of k UPT circuits of preimage-width w each.*

[4] the shuffling just reorders the partition of the set-multilinear circuit

Like in [17], we will refer to this class by Σ^k-UPT. We shall further qualify this notation to use Σ^k-UPT(w) to denote the class of circuits that is a sum of k UPT circuits of preimage-width w.

From this lemma, we can focus on constructing hitting sets for Σ^k-UPT-SML circuits. The proof largely follows the ideas of Gurjar, Korwar, Saxena and Thierauf [10][5].

Notation

Let $\mathbf{y} = \mathbf{y}_1 \sqcup \cdots \sqcup \mathbf{y}_d$ be a partition of the variables and let $S = \{s_1, \ldots, s_p\}$ be a subset of $[d]$. Define the set of variables $\mathbf{y}_S = \mathbf{y}_{s_1} \cup \cdots \cup \mathbf{y}_{s_p}$ and the set of monomials $\mathbf{y}^S = \mathbf{y}_{s_1} \times \cdots \times \mathbf{y}_{s_p}$. Also, define $\mathbf{y}_{-S} = \mathbf{y} \setminus \mathbf{y}_S$ and $\mathbf{y}^{-S} = \mathbf{y}^{[d]\setminus S}$.

▶ **Definition 6.2** (Coefficient operator)**.** Let $f = \sum_{m \in \mathbf{y}^{[d]}} \alpha_m m$ be a set-multilinear polynomial of degree d, for $S \subseteq [d]$ and a monomial $m \in \mathbf{y}^S$, define $\mathrm{coeff}_m : \mathbb{F}[\mathbf{y}] \to \mathbb{F}[\mathbf{y}_{-S}]$ to be as follows.

$$\mathrm{coeff}_m(f) = \sum_{m' \in \mathbf{y}^{-S}} \alpha_{(m \cdot m')} m'$$

where $\alpha_{(m \cdot m')}$ is the coefficient of mm' in f.

▶ **Lemma 6.3.** *Let $\mathbf{y} = \mathbf{y}_1 \sqcup \ldots \sqcup \mathbf{y}_d$ be a partition and $f(\mathbf{y})$ be a set-multilinear polynomial (with respect to the above partition) computed by a* UPT-SML *circuit of preimage-width w and underlying parse-tree shape T. Suppose $g(\mathbf{y})$ is another set-multilinear polynomial (under the same partition) that* cannot *be computed by a* UPT-SML *circuit of preimage-width w with the same shape T.*

Then, there exists $S \subseteq [d]$ and $R \in \mathbb{F}[\mathbf{y}_S]^{1 \times w'}$, and $P, Q \in \mathbb{F}[y_{-S}]^{w' \times 1}$ satisfying $f = RP$, $g = RQ$ with $w' \leq w^2$ such that:

- *For each $i \in [w']$, there is a monomial $m_i \in \mathbf{y}^S$ such that the i-th element of P and Q is $\mathrm{coeff}_{m_i}(f)$ and $\mathrm{coeff}_{m_i}(g)$ respectively,*
- *there is a vector $\Gamma \in \mathbb{F}^{1 \times w'}$ of support size at most $w+1$ such that $\Gamma P = 0$ and $\Gamma Q \neq 0$,*
- *the coefficient space of R is full-rank, i.e. if we interpret R as a matrix over $\mathbb{F}$ by listing each of its w' entries as a column vector of coefficients, then this matrix has full column-rank.*
- *the vector of polynomials R is simultaneously computable by a* UPT-SML *circuit of preimage-width at most w'.*

This lemma is a fairly natural and straightforward generalization of [10, Lemma 4.5] and a proof of this is provided in the full version.

▶ **Lemma 6.4.** *Suppose $f(\mathbf{y})$ is a non-zero polynomial computed by a Σ^k-UPT-SML(w) circuit. Suppose* $\mathrm{wt} : \mathbf{y} \to M^r$ *is a weight assignment that satisfies the following properties:*

- wt *is a BIWA for spaces of polynomials simultaneously computed by* UPT-SML *circuits of preimage-width at most $w(w+1)$,*
- *For any g in Σ^{k-1}-UPT-SML$(w(w+1))$, the polynomial $g(\mathbf{y} + \mathbf{t}^{\mathrm{wt}}) \in \mathbb{F}(\mathbf{t})[\mathbf{y}]$ has a monomial with non-zero coefficient that depends on at most ℓ distinct variables in $\mathbf{y}$.*

[5] [10] constructed hitting sets for sums of ROABPs and we use similar techniques for sums of UPT circuits. Roughly speaking, if we have a class $\mathcal{C}$ that has a *characterizing set of dependencies* for which we know how to construct BIWAs, then we can also construct hitting sets for $\Sigma^k\mathcal{C}$.

Then, the polynomial $f(\mathbf{y} + \mathbf{t}^{\mathrm{wt}})$ *has a monomial, depending on at most* $\log(w(w+1)) + \ell$ *distinct variables in* $\mathbf{y}$*, with a non-zero coefficient.*

This is essentially a restatement of [10, Lemma 4.6, Lemma 4.8] and follows from their proof. Unravelling the recursion, we get the following corollary.

▶ **Corollary 6.5.** *Let* $f(\mathbf{y})$ *be a non-zero polynomial that can computed by a* Σ^k-UPT-SML(w) *circuit. Suppose* $\mathrm{wt} : \mathbf{y} \to M^r$ *is a BIWA for the class of polynomials simultaneously computed by* UPT-SML *circuits of preimage-width at most* $w^{2^{O(k)}}$*. Then, the polynomial* $f(\mathbf{y} + \mathbf{t}^{\mathrm{wt}}) \in \mathbb{F}(\mathbf{t})[\mathbf{y}]$ *has a monomial with a non-zero coefficient that depends on at most* $2^{O(k)} \log w$ *variables in* $\mathbf{y}$*.*

Once we are guaranteed to retain a monomial of small-support, we can construct a hitting set by enumerating over all possible supports and applying the Schwartz-Zippel lemma [22, 7, 24, 28] (or apply standard generators such as the Shpilka-Volkovich generator [25]). This completes the proof of Theorem 1.2, which we restate below for convenience.

▶ **Theorem 1.2** (Hitting sets for circuits with few parse tree shapes)**.** *There is an explicit hitting set* $\mathcal{H}_{d,n,s,k}$ *of size at most* $(s^{2^k} nd)^{O(\log d)}$ *for the class of* n*-variate degree* d *homogeneous non-commutative polynomials in* $\mathbb{F}\langle x_1, \ldots, x_n \rangle$ *that are computed by non-commutative circuits of size at most* s *consisting of parse trees of at most* k *shapes.*

7 Open problems

An interesting open problem (at least to us) is whether we can give non-trivial hitting sets for the class of *non-commutative skew circuits*. Lagarde, Limaye and Srinivasan [17] provide a white-box PIT in some restricted settings when the skew circuits are somewhat closer to UPT (with some restriction on what sort of parse trees they can have) but removing this restriction would be a great step forward.

Another issue is that the current construction of hitting sets for FewPT circuits (which build on [10]) incurs quasipolynomial losses at two different places. The first is in the construction of the *basis isolating weight assignment (BIWA)*, and we only know to construct that using quasipolynomially large weights. The other is in a brute-force enumeration of all monomials of support $O(\log s)$. As a result, even if at a later date we have a construction of a BIWA with polynomially large weights, this proof would still only yield a quasipolynomially large hitting set for FewPT circuits. It would be interesting to see if this brute-force enumeration could be circumvented.

References

1 Manindra Agrawal. Proving Lower Bounds Via Pseudo-random Generators. In *Proceedings of the 25th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2005)*, pages 92–105, 2005. doi:10.1007/11590156_6.

2 Manindra Agrawal, Rohit Gurjar, Arpita Korwar, and Nitin Saxena. Hitting-Sets for ROABP and Sum of Set-Multilinear Circuits. *SIAM Journal of Computing*, 44(3):669–697, 2015. doi:10.1137/140975103.

3 Manindra Agrawal, Chandan Saha, and Nitin Saxena. Quasi-polynomial hitting-set for set-depth-Δ formulas. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC 2013)*, pages 321–330, 2013. eccc:TR12-113. doi:10.1145/2488608.2488649.

4 Eric Allender, Jia Jiao, Meena Mahajan, and V. Vinay. Non-Commutative Arithmetic Circuits: Depth Reduction and Size Lower Bounds. *Theoretical Computer Science*, 209(1-2):47–86, 1998. doi:10.1016/S0304-3975(97)00227-2.

5 Vikraman Arvind and S. Raja. Some Lower Bound Results for Set-Multilinear Arithmetic Computations. *Chicago Journal of Theoretical Computer Science*, 2016. URL: http://cjtcs.cs.uchicago.edu/articles/2016/6/contents.html.

6 Walter Baur and Volker Strassen. The Complexity of Partial Derivatives. *Theoretical Computer Science*, 22:317–330, 1983. doi:10.1016/0304-3975(83)90110-X.

7 Richard A. DeMillo and Richard J. Lipton. A Probabilistic Remark on Algebraic Program Testing. *Information Processing Letters*, 7(4):193–195, 1978. doi:10.1016/0020-0190(78)90067-4.

8 Michael A. Forbes and Amir Shpilka. Quasipolynomial-Time Identity Testing of Non-commutative and Read-Once Oblivious Algebraic Branching Programs. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2013)*, pages 243–252, 2013. Full version at arXiv:1209.2408. doi:10.1109/FOCS.2013.34.

9 Rohit Gurjar, Arpita Korwar, and Nitin Saxena. Identity Testing for Constant-Width, and Commutative, Read-Once Oblivious ABPs. In *Proceedings of the 31st Annual Computational Complexity Conference (CCC 2016)*, pages 29:1–29:16, 2016. arXiv:1601.08031. doi:10.4230/LIPIcs.CCC.2016.29.

10 Rohit Gurjar, Arpita Korwar, Nitin Saxena, and Thomas Thierauf. Deterministic Identity Testing for Sum of Read-once Oblivious Arithmetic Branching Programs. In *Proceedings of the 30th Annual Computational Complexity Conference (CCC 2015)*, pages 323–346, 2015. arXiv:1411.7341. doi:10.4230/LIPIcs.CCC.2015.323.

11 Joos Heintz and Claus-Peter Schnorr. Testing Polynomials which Are Easy to Compute (Extended Abstract). In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing (STOC 1980)*, pages 262–272, 1980. doi:10.1145/800141.804674.

12 Pavel Hrubes, Avi Wigderson, and Amir Yehudayoff. Non-commutative circuits and the sum-of-squares problem. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing (STOC 2010)*, pages 667–676, 2010. doi:10.1145/1806689.1806781.

13 Pavel Hrubeš and Amir Yehudayoff. On Isoperimetric Profiles and Computational Complexity. In *Proceedings of the 43rd International Colloquium on Automata, Languages and Programming (ICALP 2016)*, pages 89:1–89:12, 2016. eccc:TR15-164. doi:10.4230/LIPIcs.ICALP.2016.89.

14 Laurent Hyafil. On the Parallel Evaluation of Multivariate Polynomials. *SIAM Journal of Computing*, 8(2):120–123, 1979. Preliminary version in the *10th Annual ACM Symposium on Theory of Computing (STOC 1978)*. doi:10.1137/0208010.

15 Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004. Preliminary version in the *35th Annual ACM Symposium on Theory of Computing (STOC 2003)*. doi:10.1007/s00037-004-0182-6.

16 Adam R. Klivans and Amir Shpilka. Learning Restricted Models of Arithmetic Circuits. *Theory of Computing*, 2(10):185–206, 2006. Preliminary version in the *16th Annual Conference on Computational Learning Theory (COLT 2003)*. doi:10.4086/toc.2006.v002a010.

17 Guillaume Lagarde, Nutan Limaye, and Srikanth Srinivasan. Lower Bounds and PIT for Non-Commutative Arithmetic circuits with Restricted Parse Trees. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:77, 2017. eccc:TR17-077. URL: https://eccc.weizmann.ac.il/report/2017/077.

18 Guillaume Lagarde, Guillaume Malod, and Sylvain Perifel. Non-commutative computations: lower bounds and polynomial identity testing. *Electronic Colloquium on Computa-*

tional Complexity (ECCC), 23:94, 2016. URL: http://eccc.hpi-web.de/report/2016/094.

19 Nutan Limaye, Guillaume Malod, and Srikanth Srinivasan. Lower Bounds for Non-Commutative Skew Circuits. *Theory of Computing*, 12(1):1–38, 2016. eccc:TR15-22. doi:10.4086/toc.2016.v012a012.

20 Noam Nisan. Lower bounds for non-commutative computation. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC 1991)*, pages 410–418, 1991. Available on citeseer:10.1.1.17.5067. doi:10.1145/103418.103462.

21 Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992. doi:10.1007/BF01305237.

22 Øystein Ore. Über höhere kongruenzen. *Norsk Mat. Forenings Skrifter*, 1(7):15, 1922.

23 Ran Raz and Amir Shpilka. Deterministic polynomial identity testing in non-commutative models. *Computational Complexity*, 14(1):1–19, 2005. Preliminary version in the *19th Annual IEEE Conference on Computational Complexity (CCC 2004)*. doi:10.1007/s00037-005-0188-8.

24 Jacob T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *Journal of the ACM*, 27(4):701–717, 1980. doi:10.1145/322217.322225.

25 Amir Shpilka and Ilya Volkovich. Read-once polynomial identity testing. *Computational Complexity*, 24(3):477–532, 2015. Preliminary version in the *40th Annual ACM Symposium on Theory of Computing (STOC 2008)*. doi:10.1007/s00037-015-0105-8.

26 Leslie G. Valiant. Completeness Classes in Algebra. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing (STOC 1979)*, pages 249–261, 1979. doi:10.1145/800135.804419.

27 Leslie G. Valiant, Sven Skyum, S. Berkowitz, and Charles Rackoff. Fast Parallel Computation of Polynomials Using Few Processors. *SIAM Journal of Computing*, 12(4):641–644, 1983. Preliminary version in the *6th Internationl Symposium on the Mathematical Foundations of Computer Science (MFCS 1981)*. doi:10.1137/0212043.

28 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation, EUROSAM '79, An International Symposiumon Symbolic and Algebraic Computation*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 1979. doi:10.1007/3-540-09519-5_73.

A Relative computational power of restricted parse tree models

In this section we discuss the relative computational power of the models studied in the paper and their algebraic branching program counterparts.

We begin by recalling that Lagarde, Malod and Perifel [18] show that the computational power of non-commutative UPT circuits lies *strictly* between that of ABPs and circuits. In other words their work refines the strict separation between ABPs and circuits given by the seminal work of Nisan [20]. Each of these results use a generalization of the notion of *partial derivative matrix* which was first introduced by Nisan. We skip defining the generalized partial derivative matrix formally for brevity. A formal definition can be found in the proof of Theorem 1.5 in the full version. The following statements will help in understanding the rest of this section.

- Nisan [20] showed that for any homogeneous non-commutative polynomial f the *width* of a (homogeneous) ABP computing it, in the layer i, is exactly equal to the rank of the partial derivative matrix of f for degree i.
- Lagarde et al. [18] show that in the smallest UPT circuit of shape T computing a polynomial f, the number of gates of a type $\tau \in T$ is equal to the rank of the generalized partial derivative matrix for the type τ.

Thus, the ranks of the appropriate generalized partial derivative matrices for f characterize the ABP complexity and the UPT complexity of f. As discussed in section 1.3, we have strict separations between ABPs, UPT circuits and general circuits, even under shufflings. We now provide an informal discussion about constant width (or preimage-width) models for the sake of completeness.

A.1 Constant width models

For ease of exposition, we will use the term *width* to refer to both the width of an ABP and the preimage-width of a UPT circuit. The intended meaning will be clear from the context.

We can obtain a strict separation between constant width ABPs and constant width UPT circuits using the proof of Theorem 1.5. This is done by working with a constant variate version of the polynomial described in the proof. We skip the details to avoid repeating the proof. A more interesting comparison is that of ABPs of unrestricted (poly(n)) width and constant width UPT circuits, which we discuss now.

A.1.1 ABPs vs constant width UPT circuits

Let f_n be the following non-commutative variant of the elementary symmetric polynomial on n-variables of degree $d = n/2$.

$$f_n := \sum_{1 \le i_1 < \cdots < i_d \le n} x_{i_1} x_{i_2} \cdots x_{i_d}$$

Note that the generalized partial derivative matrix of f_n for any interval of positions $I \subset [d]$ will have rank $\geq n - d = \Omega(n)$. Moreover, any shuffling of f_n has the same property, since the rank does not depend on how we "order" the indices in $[n]$. Hence, any UPT circuit computing f_n or even a shuffling of f_n, requires poly(n) width. However, it is easy to see that f_n has a poly(n) sized ABP.

▶ **Fact A.1.** *(Informal) There is a polynomial f_n that is computable by a* poly(n) *sized ABP, but any shuffling of f_n requires UPT circuits of width $\Omega(n)$.*

Consider the bivariate palindrome polynomial of degree $2n$, for a growing parameter n.

$$P_n(x_1, x_2) = \sum_{(i_1,\ldots,i_n) \in [2]^n} (x_{i_1} x_{i_2} \cdots x_{i_n}) \cdot (x_{i_n} \cdots x_{i_2} x_{i_1})$$

It is easy to verify that the rank of the partial derivative matrix of $P_n(x_1, x_2)$ for degree n, is exactly 2^n. Also note that the polynomial has a UPT circuit of constant width. However, as remarked before in the paper, there is a shuffling of the palindrome polynomial that makes it simple for ABPs. Therefore we get the following fact, *when shufflings are not allowed.*

▶ **Fact A.2.** *(Informal) The classes of constant width UPT circuits and ABPs are incomparable.*

Let us now look at the case when shufflings are allowed.

Say $f_n(\mathbf{x})$ is an n-variate homogeneous polynomial of degree poly(n). If f has a UPT circuit C of width w, then Theorem 1.4 gives us that a shuffling of f_n, say f'_n, has a UPT circuit C' of depth $O(\log n)$ and width $O(w^2)$. Let T be the shape of C' and let $g \in C'$ be a gate with type $\tau \in T$. Note that any path in C', from g to the root of C', goes through $O(\log n)$ gates,

each of which is one out of the $O(w^2)$ gates of its type. Therefore even a trivial conversion of C' to a formula replicates any gate $g \in C'$ at most $w^{O(\log n)}$ times. We therefore have the following.

▶ **Fact A.3.** *(Informal) If $f_n(\mathbf{x})$ is computable by a constant width UPT circuit of size* $\text{poly}(n)$*, then a shuffling of f_n is computable by an ABP of size* $\text{poly}(n)$*.*

Univariate Ideal Membership Parameterized by Rank, Degree, and Number of Generators

V. Arvind
Institute of Mathematical Sciences (HBNI), Chennai, India
arvind@imsc.res.in

Abhranil Chatterjee
Institute of Mathematical Sciences (HBNI), Chennai, India
abhranilc@imsc.res.in

Rajit Datta
Chennai Mathematical Institute, Chennai, India
rajit@cmi.ac.in

Partha Mukhopadhyay
Chennai Mathematical Institute, Chennai, India
partham@cmi.ac.in

Abstract

Let $\mathbb{F}[\mathrm{X}]$ be the polynomial ring over the variables $\mathrm{X} = \{x_1, x_2, \ldots, x_n\}$. An ideal $I = \langle p_1(x_1), \ldots, p_n(x_n)\rangle$ generated by univariate polynomials $\{p_i(x_i)\}_{i=1}^n$ is a *univariate ideal*. We study the ideal membership problem for the univariate ideals and show the following results.

- Let $f(\mathrm{X}) \in \mathbb{F}[\ell_1, \ldots, \ell_r]$ be a (low rank) polynomial given by an arithmetic circuit where $\ell_i : 1 \leq i \leq r$ are linear forms, and $I = \langle p_1(x_1), \ldots, p_n(x_n)\rangle$ be a univariate ideal. Given $\vec{\alpha} \in \mathbb{F}^n$, the (unique) remainder $f(\mathrm{X}) \pmod I$ can be evaluated at $\vec{\alpha}$ in deterministic time $d^{O(r)} \cdot \mathrm{poly}(n)$, where $d = \max\{\deg(f), \deg(p_1) \ldots, \deg(p_n)\}$. This yields a randomized $n^{O(r)}$ algorithm for minimum vertex cover in graphs with rank-r adjacency matrices. It also yields an $n^{O(r)}$ algorithm for evaluating the permanent of a $n \times n$ matrix of rank r, over any field $\mathbb{F}$. Over $\mathbb{Q}$, an algorithm of similar run time for low rank permanent is due to Barvinok [5] via a different technique.
- Let $f(\mathrm{X}) \in \mathbb{F}[\mathrm{X}]$ be given by an arithmetic circuit of degree k (k treated as fixed parameter) and $I = \langle p_1(x_1), \ldots, p_n(x_n)\rangle$. We show that in the special case when $I = \langle x_1^{e_1}, \ldots, x_n^{e_n}\rangle$, we obtain a randomized $O^*(4.08^k)$ algorithm that uses $\mathrm{poly}(n, k)$ space.
- Given $f(\mathrm{X}) \in \mathbb{F}[\mathrm{X}]$ by an arithmetic circuit and $I = \langle p_1(x_1), \ldots, p_k(x_k)\rangle$, membership testing is W[1]-hard, parameterized by k. The problem is MINI[1]-hard in the special case when $I = \langle x_1^{e_1}, \ldots, x_k^{e_k}\rangle$.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Combinatorial Nullstellensatz, Ideal Membership, Parametric Hardness, Low Rank Permanent

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.7

Acknowledgements We thank the anonymous reviewers for their useful comments. The third author acknowledges partial support from Infosys Foundation. The fourth author acknowledges partial support from Infosys Foundation and Tata Trust.

38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018).
Editors: Sumit Ganguly and Paritosh Pandya; Article No. 7; pp. 7:1–7:18

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

1 Introduction

Let $R = \mathbb{F}[x_1, x_2, \ldots, x_n]$[1] be the ring of polynomials over the variables $\mathrm{X} = \{x_1, x_2, \ldots, x_n\}$. A subring $I \subseteq R$ is an ideal if $IR \subseteq I$. Computationally, an ideal I is often given by generators: $I = \langle f_1, f_2, \ldots, f_\ell \rangle$. Given $f \in R$ and $I = \langle f_1, \ldots, f_\ell \rangle$, the *Ideal Membership problem* is to decide whether $f \in I$ or not. In general, this is computationally highly intractable. In fact, it is EXPSPACE-complete even if f and the generators $f_i, i \in [\ell]$ are given explicitly by sum of monomials [21]. Nevertheless, special cases of ideal membership problem have played important roles in several results in arithmetic complexity. For example, the polynomial identity testing algorithm for depth three $\Sigma\Pi\Sigma$ circuits with bounded top fan-in; the structure theorem for $\Sigma\Pi\Sigma(k, d)$ identities use ideal membership very crucially [4, 13, 24].

In this paper, our study of ideal membership is motivated by a basic result in algebraic complexity: the Combinatorial Nullstellensatz of Alon [1], and we recall a basic result in that paper.

▶ **Theorem 1.** *Let $\mathbb{F}$ be any field, and $f(\mathrm{X}) \in \mathbb{F}[\mathrm{X}]$. Define polynomials $g_i(x_i) = \prod_{s \in S_i}(x_i - s)$ for non-empty subsets $S_i, 1 \leq i \leq n$ of $\mathbb{F}$. If f vanishes on all the common zeros of $g_1, \ldots, g_n$, then there are polynomials $h_1, \ldots, h_n$ satisfying $\deg(h_i) \leq \deg(f) - \deg(g_i)$ such that $f = \sum_{i=1}^{n} h_i g_i$.*

The theorem can be restated in terms of ideal membership: Let $f(\mathrm{X}) \in \mathbb{F}[\mathrm{X}]$ be a given polynomial, and $I = \langle g_1(x_1), g_2(x_2), \ldots, g_n(x_n) \rangle$ be an ideal generated by univariate polynomials g_i *without repeated roots*. Let $Z(g_i)$ denote the zero set of $g_i, 1 \leq i \leq n$. By Theorem 1, if $f \notin I$ then there is a $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n) \in Z(g_1) \times \cdots \times Z(g_n)$ such that $f(\vec{\alpha}) \neq 0$. Of course, if $f \in I$ then $f|_{Z(g_1) \times \cdots \times Z(g_n)} = 0$.

Ideals I generated by univariate polynomials are called *univariate ideals*. For any univariate ideal I and any polynomial f, by repeated application of the division algorithm, we can write $f(\mathrm{X}) = \sum_{i=1}^{n} h_i(\mathrm{X}) g_i(x_i) + R(\mathrm{X})$ where R is unique and for each $i \in [n]$: $\deg_{x_i}(R) < \deg(g_i(x_i))$. Since the remainder is unique, it is convenient to write $R = f \mod I$. By Alon's theorem, if $f \notin I$ then there is a $\vec{\alpha} \in Z(g_1) \times \cdots \times Z(g_n)$ such that $R(\vec{\alpha}) \neq 0$.

As an application of the theorem, Alon and Tarsi showed that checking k-colorability of a graph G is polynomial-time equivalent to testing whether the graph polynomial f_G is in the ideal $\langle x_1^k - 1, \ldots, x_n^k - 1 \rangle$ [1]. It follows that univariate ideal membership problem coNP-hard.

Univariate ideal membership is further motivated by its connection with two well-studied problems. Computing the permanent of a $n \times n$ matrix over any field $\mathbb{F}$ can be cast in terms of univariate ideal membership. Given a matrix $A = (a_{i,j})_{1 \leq i,j \leq n} \in \mathbb{F}^{n \times n}$, consider the product of linear forms $P_A(\mathrm{X}) = \prod_{i=1}^{n} (\sum_{j=1}^{n} a_{ij} x_j)$. The following observation is well known.

▶ **Fact 2.** *The permanent of the matrix A is given by the coefficient of the monomial $x_1 x_2 \cdots x_n$ in P_A.*

It follows immediately that $P_A(\mathrm{X}) \pmod{\langle x_1^2, \ldots, x_n^2 \rangle} = \mathrm{Perm}(A)\, x_1 x_2 \cdots x_n$. I.e., the remainder $P_A \pmod{\langle x_1^2, \ldots, x_n^2 \rangle}$ evaluates to $\mathrm{Perm}(A)$ at the point $\vec{1} \in \mathbb{F}^n$.

[1] We often use the shorthand notation $\mathbb{F}[\mathrm{X}]$.

Next, we briefly mention the connection of univariate ideal membership with the multilinear monomial detection problem, a benchmark problem that is useful in designing fast parameterized algorithms for a host of problems [16, 17, 18, 28].

Notice that, given an arithmetic circuit C computing a polynomial $f \in \mathbb{F}[\mathrm{X}]$ of degree k, checking if f has a non-zero multilinear monomial of degree k is equivalent to checking if $f \pmod{\langle x_1^2, \ldots, x_n^2\rangle}$ is non-zero. Moreover, the constrained multilinear detection problem studied in [6, 17] can also be viewed as a problem of deciding membership in a univariate ideal.

Our Results. A contribution of this paper is to consider several parameterized problems in arithmetic complexity as instances of univariate ideal membership. One parameter of interest is the rank of a multivariate polynomial: We say $f \in \mathbb{F}[\mathrm{X}]$ is a *rank* r polynomial if $f \in \mathbb{F}[\ell_1, \ell_2, \ldots, \ell_r]$ for linear forms $\ell_j : 1 \leq j \leq r$. This concept has found application in algorithms for depth-3 polynomial identity testing [24]. Given a univariate ideal I, a point $\vec{\alpha} \in \mathbb{F}^n$, and an arithmetic circuit computing a polynomial f of rank r, we obtain an efficient algorithm to compute $f \pmod{I}$ at $\vec{\alpha}$.

▶ **Theorem 3.** *Let $\mathbb{F}$ be an arbitrary field where the field arithmetic can be done efficiently, and C be a polynomial-size arithmetic circuit computing a polynomial f in $\mathbb{F}[\ell_1, \ell_2, \ldots, \ell_r]$, where $\ell_1, \ell_2, \ldots, \ell_r$ are given linear forms in $\{x_1, x_2, \ldots, x_n\}$. Let $I = \langle p_1, \ldots, p_n\rangle$ be a univariate ideal generated by $p_i(x_i) \in \mathbb{F}[x_i], 1 \leq i \leq n$. Given $\vec{\alpha} \in \mathbb{F}^n$, we can evaluate the remainder $f \pmod{I}$ at the point $\vec{\alpha}$ in time $d^{O(r)}\mathrm{poly}(n)$, where $d = \max\{\deg(f), \deg(p_i) : 1 \leq i \leq n\}$.*

This also allows us to check whether $f \in I$ by picking a point $\vec{\alpha}$ at random and checking whether $f \pmod{I}$ evaluated at $\vec{\alpha}$ is zero or not. The intuitive idea behind the proof of Theorem 3 is as follows. Given a polynomial $f(\mathrm{X}) \in \mathbb{F}[\ell_1, \ldots, \ell_r]$, a univariate ideal $I = \langle p_1(x_1), \ldots, p_n(x_n)\rangle$, and a point $\vec{\alpha} \in \mathbb{F}^n$, we first find an invertible linear transformation T such that the polynomial $T(f)$ becomes a polynomial over at most $2r$ variables. Additionally T has the property that T fixes the variables $x_1, \ldots, x_r$. Then we recover the polynomial (call it $\tilde{f}$) over at most $2r$ variables explicitly and perform division algorithm with respect to the ideal $I_{[r]} = \langle p_1(x_1), \ldots, p_r(x_r)\rangle$. For notational convenience, call $\tilde{f}$ be the polynomial obtained over at most $2r$ variables. It turns out $T^{-1}(\tilde{f})$ is the *true remainder* $f \pmod{I_{[r]}}$. Since the variables $x_1, \ldots, x_r$ do not play role in the subsequent stages of division, we can eliminate them by substituting $x_i \leftarrow \alpha_i$ for each $1 \leq i \leq r$. Then we apply the division algorithm on $T^{-1}(\tilde{f})|_{x_i \leftarrow \alpha_i : 1 \leq i \leq r}$ recursively with respect to the ideal $I_{[n]\setminus[r]}$ to compute the final remainder at the point $\vec{\alpha}$.

Our next result is an efficient algorithm to detect vertex cover in low rank graphs. A graph G is said to be of rank r if the rank of the adjacency matrix A_G is of rank r. Graphs of low rank were studied by Lovasz and Kotlov [2, 15] in the context of graph coloring. Our idea is to construct a low rank polynomial from the graph and check its membership in an appropriate univariate ideal.

▶ **Theorem 4.** *Given a graph $G = (V, E)$ on n vertices such that the rank of the adjacency matrix A_G is at most r, and a parameter k, there is a randomized $n^{O(r)}$ algorithm to decide if the graph G has vertex cover of size k or not.*

Theorem 3 also yields an $n^{O(r)}$ algorithm to compute the permanent of rank-r matrices over any field. Barvinok had given [5] an algorithm of same running time for the permanent of low rank matrices (over $\mathbb{Q}$) using apolar bilinear forms. By Fact 2, if matrix A is rank r then P_A is a rank-r polynomial, and for the univariate ideal $I = \langle x_1^2, \ldots, x_n^2\rangle$ computing

$P_A \pmod I$ at the point $\vec{1}$ yields the permanent. Theorem 3 works more generally for all univariate ideals. In particular, the ideal in the proof of Theorem 4 is generated by polynomials that are not powers of variables. Thus, Theorem 3 can potentially have more algorithmic consequences than the technique in [5].

If k is the degree of the input polynomial and the ideal is given by the powers of variables as generators, we have a randomized FPT algorithm for the problem.

▶ **Theorem 5.** *Given an arithmetic circuit C computing a polynomial $f(\mathrm{X}) \in \mathbb{Z}[\mathrm{X}]$ of degree k and integers $e_1, e_2, \ldots, e_n$, there is a randomized algorithm to decide whether $f \in \langle x_1^{e_1}, x_2^{e_2}, \ldots, x_n^{e_n} \rangle$ in $O^*(4.08^k)$ time.*

Note that this generalizes the well-known problem of *multilinear monomial detection* for which the ideal of interest would be $I = \langle x_1^2, x_2^2, \ldots, x_n^2 \rangle$. Surprisingly, the run time of the algorithm in Theorem 5 is independent of the e_i. Brand et al. have given the first FPT algorithm for multilinear monomial detection in the case of general circuit with run time randomized $O^*(4.32^k)$ [7]. Recently, this problem has also been studied using the Hadamard product [3] of the given polynomial with the elementary symmetric polynomial (and differently using apolar bilinear forms [22]). When the number of generators in the ideal is treated as the fixed parameter, the problem is W[1]-hard.

▶ **Theorem 6.** *Given a polynomial $f(\mathrm{X}) \in \mathbb{F}[\mathrm{X}]$ by an arithmetic circuit C and univariate polynomials $p_1(x_1), p_2(x_2), \ldots, p_k(x_k)$, checking if $f \notin \langle p_1(x_1), p_2(x_2), \ldots, p_k(x_k) \rangle$ is* W[1]*-hard with k as the parameter.*

Theorem 6 is shown by a suitable reduction from independent set problem to ideal membership. To find an independent set of size k, the reduction produces an ideal with k univariates and the polynomial created from the graph has k variables. Unlike Theorem 5, the above parameterization of the problem remains MINI[1]-hard even if the ideal is generated by powers of variables. More precisely, we show the following result.

▶ **Theorem 7.** *Let C be a polynomial-size arithmetic circuit computing a polynomial $f \in \mathbb{F}[\mathrm{X}]$. Let $I = \langle {x_1}^{e_1}, {x_2}^{e_2}, \ldots, {x_k}^{e_k} \rangle$ be the given ideal where $e_1, \ldots, e_k$ are given in unary, checking if $f \notin I$ is* MINI[1]*-hard with k as parameter.*

It turns out that the complement of the ideal membership problem can be easily reduced from k-Lin-Eq problem which asks if there is a $\vec{x} \in \{0,1\}^n$ satisfying $A\vec{x} = \vec{b}$, where $A \in \mathbb{F}^{k \times n}$ and $\vec{b} \in \mathbb{F}^k$.

We can show k-Lin-Eq is hard for the parameterized complexity class MINI[1] by reducing the miniature version of 1-in-3 POSITIVE 3-SAT to it.

As already mentioned, the result of Alon and Tarsi [1] shows that the membership of f_G in $\langle x_1^k - 1, \ldots, x_n^k - 1 \rangle$ is coNP-hard and the proof crucially uses the fact that the roots of the generator polynomials are all distinct. This naturally raises the question if univariate ideal membership is in coNP when each generator polynomial has distinct roots. We show membership in coNP.

▶ **Theorem 8.** *Let $f \in \mathbb{Q}[\mathrm{X}]$ be a polynomial of degree at most d given by a black-box. Let $I = \langle p_1(x_1), \ldots, p_n(x_n) \rangle$ be an ideal given explicitly by a set of univariate polynomials $p_1, p_2, \ldots, p_n$ as generators of maximum degree bounded by d. Let L be the bit-size upper bound for any coefficient in $f, p_1, p_2, \ldots, p_n$. Moreover, assume that p_is have distinct roots over $\mathbb{C}$. Then there is a non-deterministic algorithm running in time* $\mathrm{poly}(n, d, L)$ *that decides the non-membership of f in the ideal I.*

▶ Remark. The distinct roots case discussed in Theorem 8 is in stark contrast to the complexity of testing membership of $P_A(\mathrm{X})$ in the ideal $\langle x_1^2, \ldots, x_n^2 \rangle$. That problem is equivalent to checking if $\mathrm{Perm}(A)$ is nonzero for a rational matrix A, which is hard for the exact counting class $\mathrm{C_{=}P}$. Hence it cannot be in coNP unless the polynomial-time hierarchy collapses.

Recall from Alon's Nullstellensatz that if $f \notin I$, then there is always a point $\vec{\alpha} \in Z(p_1) \times \ldots \times Z(p_n)$ such that $f(\vec{\alpha}) \neq 0$. Notice that in general the roots $\alpha_i \in \mathbb{C}$ and in the standard *Turing Machine* model the NP machine can not guess the roots directly with only finite precision. But we are able to prove that the NP machine can guess the tuple of roots $\vec{\alpha} \in \mathbb{Q}^n$ using only polynomial bits of precision and still can decide the non-membership. The main technical idea is to compute efficiently a parameter M only from the input parameters such that $|f(\vec{\alpha})| \leq M$ if $f \in I$, and $|f(\vec{\alpha})| \geq 2M$ if $f \notin I$. The NP machine decides the non-membership according to the final value of $|f(\vec{\alpha})|$. We remark that Koiran has considered the weak version of Hilbert Nullstellensatz (HN) problem [14]. The input is a set of multivariate polynomials $f_1, f_2, \ldots, f_m \in \mathbb{Z}[\mathrm{X}]$ and the problem is to decide whether $1 \in \langle f_1, \ldots, f_m \rangle$. The result of Koiran shows that $\overline{\mathrm{HN}} \in \mathrm{AM}$ (under GRH), and it is an outstanding open problem problem to decide whether $\overline{\mathrm{HN}} \in \mathrm{NP}$.

Organization. In Section 2 we give some background results. We prove Theorem 3 and Theorem 4 in Section 3. In Section 4, we explore the parameterized complexity of univariate ideal membership. In the first subsection, we prove 5, and in the second subsection we prove Theorems 6 and 7. Finally, in Section 5, we prove Theorem 8.

2 Preliminaries

Basics of Ideal Membership. Let $\mathbb{F}[\mathrm{X}]$ be the ring of polynomials $\mathbb{F}[x_1, x_2, \ldots, x_n]$. Let $I \subseteq \mathbb{F}[\mathrm{X}]$ be an ideal given by a set of generators $I = \langle g_1, \ldots, g_\ell \rangle$. Then for any polynomial $f \in \mathbb{F}[\mathrm{X}]$, it is a member of the ideal if and only if $f = \sum_{i=1}^{\ell} h_i g_i$ where $\forall i : h_i \in \mathbb{F}[\mathrm{X}]$. Dividing f by the g_i by applying the standard division algorithm does not work in general to check if $f \in I$. Indeed, the remainder is not even uniquely defined. However, if the leading monomials of the generators are already pairwise relatively prime, then we can apply the division algorithm to compute the unique remainder.

▶ **Theorem 9** (See[9], Theorem 3, proposition 4, pp.101)**.** *Let I be a polynomial ideal given by a basis $G = \{g_1, g_2, \cdots, g_s\}$ such that all pairs $i \neq j$ $LM(g_i)$ and $LM(g_j)$ are relatively prime. Then G is a Gröbner basis for I.*

In particular, if the ideal I is a univariate ideal given by $I = \langle p_1(x_1), \ldots, p_n(x_n) \rangle$, we can apply the division algorithm to compute the unique remainder $f \pmod I$. To bound the run time of this procedure we note the following: Let $\bar{p}$ denote the ordered list $\{p_1, p_2, \ldots, p_n\}$. Let $\mathrm{Divide}(f; \bar{p})$ be the procedure that divides f by p_1 to obtain remainder f_1, then divides f_1 by p_2 to obtain remainder f_2, and so on to obtain the final remainder f_n after dividing by p_n. We note the following time bound for $\mathrm{Divide}(f; \bar{p})$.

▶ **Fact 10** (See [27], Section 6, pp.5-12)**.** *Let $f \in \mathbb{F}[\mathrm{X}]$ be given by a size s arithmetic circuit and $p_i(x_i) \in \mathbb{F}[x_i]$ be given univariate polynomials. The running time of $\mathrm{Divide}(f; \bar{p})$ is bounded by $O(s \cdot \prod_{i=1}^{n} (d_i + 1)^{O(1)})$, where $d_i = \max\{\deg_{x_i}(f), \deg(p_i(x_i))\}$.*

On Roots of Univariate Polynomials. The following lemma shows that the absolute value of any root of a univariate polynomial can be bounded in terms of the degree and the coefficients. The result is folklore.

▶ **Lemma 11.** *Let $f(x) = \sum_{i=0}^{d} a_i x^i \in \mathbb{Q}[x]$ be a univariate polynomial and α be a root of f. Then, either $\frac{|a_0|}{\sum_{i=1}^{d} |a_i|} \leq |\alpha| < 1$ or $1 \leq |\alpha| \leq d \cdot \frac{\max_i |a_i|}{|a_d|}$.*

Proof. Since α is a root of f, we have that, $0 = f(\alpha) = \sum_{i=0}^{d} a_i \alpha^i = 0$, and $\sum_{i=1}^{d} a_i \alpha^i = -a_0$. Then by an application of triangle inequality, we get that $\sum_{i=1}^{d} |a_i||\alpha|^i \geq |a_0|$. Now we analyse two different cases. In the first case assume that $|\alpha| < 1$. Observe that $|\alpha| \cdot (\sum_{i=1}^{d} |a_i|) \geq |a_0|$, and hence $|\alpha| \geq \frac{|a_0|}{\sum_{i=1}^{d} |a_i|}$. In the second case $|\alpha| \geq 1$. Observe that $-a_d \alpha^d = \sum_{i=0}^{d-1} a_i \alpha^i$. Then use triangle inequality to get that $|a_d||\alpha|^d \leq |\alpha|^{d-1} \cdot (\sum_{i=0}^{d-1} |a_i|)$. Now we get the following, $|\alpha| \leq \frac{\sum_{i=0}^{d-1} |a_i|}{|a_d|} \leq d \cdot \frac{\max_i |a_i|}{|a_d|}$. The lemma follows by combining the two cases. ◀

The next lemma shows that the separation between two distinct roots of any univariate polynomial can be lower bounded in terms of degree and the size of the coefficients. This was shown by Mahler [20].

▶ **Lemma 12.** *Let $g(x) = \sum_{i=0}^{d} a_i x^i \in \mathbb{Q}[x]$ and $2^{-L} \leq |a_i| \leq 2^L$ (if $a_i \neq 0$). Let α, β are two distinct roots of g. Then $|\alpha - \beta| \geq \frac{1}{2^{O(dL)}}$.*

The following lemma states that any univariate polynomial can not get a very small value (in absolute sense) on any point which is far from every root.

▶ **Lemma 13.** *Let $f = \sum_{i=1}^{d} a_i x^i$ be a univariate polynomial with $2^{-L} \leq |a_i| \leq 2^L$ (if $a_i \neq 0$). Let $\tilde{\alpha}$ be a point such that $|\tilde{\alpha} - \beta_i| \geq \delta$ for every root β_i of f then $|f(\tilde{\alpha})| \geq 2^{-L}\delta^d$.*

Proof. We observe that, $f(\tilde{\alpha}) = c \prod_{i=1}^{d} (\tilde{\alpha} - \beta_i)$. Since $|\tilde{\alpha} - \beta_i| \geq \delta$ we get, $|f(\tilde{\alpha})| = |c| \prod_{i=1}^{d} |\tilde{\alpha} - \beta_i| \geq 2^{-L}\delta^d$. This completes the proof. ◀

Parameterized Complexity Classes. We recall some standard definitions in parameterized Complexity [10, ch.1,pp. 7-14]. We only state them informally. For a parameterized input problem (x, k) with k be the parameter of interest, we say that the problem is in FPT if it has an algorithm with run time $f(k)|(x, k)|^{O(1)}$ for some computable function f. A parameterized reduction [10, def. 13.1] between two problems should be computable in time $f(k)|(x, k)|^{O(1)}$, and if the reduction outputs (x', k') then $k' \leq f(k)$.

The complexity class MINI[1] consists of parameterized problems that are miniature versions of NP problems: For $L \in$ NP, its miniature version mini(L) has instances of the form $(0^n, x)$, where $|x| \leq k \log n$, k is the fixed parameter, and x is an instance of L. Showing mini(L) to be MINI[1]-hard under parameterized reductions is evidence of its parameterized intractability, for it cannot be in FPT assuming the Exponential Time Hypothesis [12].

Hadamard Product. We recall the definition of Hadamard product of two polynomials.

▶ **Definition 14.** Given two polynomials $f, g \in \mathbb{F}[X]$, the Hadamard product $f \circ g$ is defined as $f \circ g = \sum_m [m]f \cdot [m]g \cdot m$.

In this paper we adapt the notion of Hadamard product suitably and define a scaled version of Hadamard Product of two polynomials.

▶ **Definition 15.** Given two polynomials $f, g \in \mathbb{F}[X]$, their *scaled Hadamard Product* $f \circ^s g$, is defined as $f \circ^s g = \sum_m m! \cdot [m]f \cdot [m]g \cdot m$, where $m = x_{i_1}^{e_1} x_{i_2}^{e_2} \dots x_{i_r}^{e_r}$ and $m! = e_1! \cdot e_2! \cdots e_r!$ abusing the notation.

▶ Remark. Given two polynomials $f \in \mathbb{F}[X]$ and $g \in \mathbb{F}[X]$, if one of these two is a multilinear polynomial then scaled Hadamard product $f \circ^s g$ is same as Hadamard product $f \circ g$.

Symmetric Polynomial and Weakly Equivalence of Polynomials. The symmetric polynomial of degree k over n variables $\{x_1, x_2, \dots, x_n\}$, denoted by $S_{n,k}$, is defined as follows:

$$S_{n,k}(x_1, x_2, \dots, x_n) = \sum_{T \subseteq [n], |T|=k} \prod_{i \in T} x_i.$$

Notice that, $S_{n,k}$ contains all the degree k multillinear terms. A recent result of Lee gives the following homogeneous diagonal circuit for $S_{n,k}$ [19].

▶ **Lemma 16.** *The symmetric polynomial $S_{n,k}$ can be computed by a homogenous $\Sigma^{[s]} \wedge^{[k]} \Sigma$ circuit where $s \leq \sum_{i=0}^{k/2} \binom{n}{i}$.*

A polynomial $f \in \mathbb{F}[X]$ is said to be *weakly equivalent* to a polynomial $g \in \mathbb{F}[X]$, if the following is true. For each monomial m, $[m]f = 0$ if and only if $[m]g = 0$. In this paper, we will use polynomials weakly equivalent to $S_{n,k}$.

3 Ideal Membership for Low Rank Polynomials

In this section we prove Theorem 3. Given a r-rank polynomial f by an arithmetic circuit, a univariate ideal I, and a point $\vec{\alpha} \in \mathbb{F}^n$, we give an $d^{O(r)}$ time algorithm to evaluate the remainder polynomial $f \pmod I$ at $\vec{\alpha}$ where d is the degree of the polynomial f. As mentioned in Section 1, an application of our result yields an $n^{O(r)}$ time algorithm for computing the permanent of rank-r matrices over any field. Barvinok [5], via a different method, had obtained an $n^{O(r)}$ time algorithm for this problem over $\mathbb{Q}$. We also obtain a randomized $n^{O(r)}$ time algorithm for minimum vertex cover of low rank graphs. We first define the notion *rank* of a polynomial in $\mathbb{F}$[X].

▶ **Definition 17.** A polynomial $f(\mathrm{X}) \in \mathbb{F}[\mathrm{X}]$ is a *rank-r* polynomial if there are linear forms $\ell_1, \ell_2, \dots, \ell_r$ such that $f(\mathrm{X})$ is in the sub-algebra $\mathbb{F}[\ell_1, \dots, \ell_r]$.

For an unspecified fixed parameter r, we refer to rank-r polynomials as *low rank polynomials*.

Given $\vec{\alpha} \in \mathbb{F}^n$, a univariate ideal $I = \langle p_1(x_1), \dots, p_n(x_n) \rangle$, and a rank r polynomial $f(\ell_1, \dots, \ell_r)$ we show how to compute $f(\ell_1, \dots, \ell_r) \pmod I$ at $\vec{\alpha}$ using a recursive procedure $\mathrm{REM}(f(\ell_1, \dots, \ell_r), I, \vec{\alpha})$ efficiently. We introduce the following notation. For $S \subseteq [n]$, the ideal $I_S = \langle p_i(x_i) : i \in [S] \rangle$.

We first observe the following lemma which shows how to remove the redundant variables from a low rank polynomial.

▶ **Lemma 18.** *Given a polynomial $f(\ell_1, \dots, \ell_r)$ where $\ell_1, \dots, \ell_r$ are linear forms in $\mathbb{F}$[X], there is an invertible linear transform $T : \mathbb{F}^n \mapsto \mathbb{F}^n$ that fixes $x_1, \dots, x_r$ and the transformed polynomial $T(f)$ is over at most $2r$ variables.*

Proof. Write each linear form ℓ_i in two parts: $\ell_i = \ell_{i,1} + \ell_{i,2}$, where $\ell_{i,1}$ is the part over variables $x_1, \dots, x_r$ and $\ell_{i,2}$ is over variables $x_{r+1}, \dots, x_n$. W.l.o.g, assume that $\{\ell_{i,2}\}_{i=1}^{r'}$ is a maximum linearly independent subset of linear forms in $\{\ell_{i,2}\}_{i=1}^{r}$. Let $T : \mathbb{F}^n \to \mathbb{F}^n$ be the

invertible linear map that fixes $x_1, \ldots, x_r$, maps the independent linear forms $\{\ell_{i,2}\}_{i=1}^{r'}$ to variables $x_{r+1}, \ldots, x_{r+r'}$, and suitably extends T to an invertible map. This completes the proof. ◀

The following lemma shows that the univariate division and evaluating the remainder at the end can be achieved by division and evaluation partially.

▶ **Lemma 19.** *Let $f(\mathrm{X}) \in \mathbb{F}[\mathrm{X}]$ and $I = \langle p_1(x_1), \ldots, p_n(x_n)\rangle$ be a univariate ideal. Let $R(\mathrm{X})$ be the unique remainder $f \pmod I$. Let $\vec{\alpha} \in \mathbb{F}^r, r \leq n$ and $R_r(\mathrm{X}) = f \pmod{I_{[r]}}$. Then $R(\alpha_1, \ldots, \alpha_r, x_{r+1}, \ldots, x_n) = R_r(\alpha_1, \ldots, \alpha_r, x_{r+1}, \ldots, x_n) \pmod{I_{[n]\setminus[r]}}$.*

Proof. From the uniqueness of the remainder for the univariate ideals, we get that $R(\mathrm{X}) = R_r(\mathrm{X}) \pmod{I_{[n]\setminus[r]}}$. Now we write explicitly the polynomial $R_r(\mathrm{X})$ as $R_r = \sum_{\bar{u}} r_{\bar{u}} \cdot x_{r+1}^{u_1} \ldots, x_n^{u_{n-r}}$ where $r_u \in \mathbb{F}[\mathrm{X}_{[r]}]$. So we get that,

$$R_r \pmod{I_{[n]\setminus[r]}} = \sum_{\bar{u}} r_{\bar{u}} \prod_{j=1}^{n-r} q(x_{r+j})$$

where $q(x_{r+j}) = x_{r+j}^{u_j} \pmod{p(x_{r+j})}$. Then the lemma follows by substituting $x_1 = \alpha_1, \ldots, x_r = \alpha_r$ in the relation $R = R_r \pmod{I_{[n]\setminus[r]}}$. ◀

We require the following lemma in the proof of the main result of this section.

▶ **Lemma 20.** *Let $f \in \mathbb{F}[\mathrm{X}]$, and $T : \mathbb{F}^n \to \mathbb{F}^n$ be an invertible linear transformation fixing $x_1, \ldots, x_r$ and mapping $x_{r+1}, \ldots, x_n$ to linearly independent linear forms over $x_{r+1}, \ldots, x_n$. Write $R = f \pmod{I_{[r]}}$ and $R' = T(f) \pmod{I_{[r]}}$. Then $R' = T(R)$.*

Proof. Let $f = \sum_{i=1}^{r} h_i(\mathrm{X}) \cdot p_i(x_i) + R(\mathrm{X})$ and $T(f) = \sum_{i=1}^{r} h_i'(\mathrm{X}) \cdot p_i(x_i) + R'(\mathrm{X})$. Note that $\deg_{x_i} R, \deg_{x_i} R' < \deg(p_i(x_i))$ for $1 \leq i \leq r$. Since T is invertible and also fixes $x_1, \ldots, x_r$, we can write $f = \sum_{i=1}^{r} T^{-1}(h_i'(\mathrm{X})) \cdot p_i(x_i) + T^{-1}(R'(\mathrm{X}))$. By the property of T it is clear that $\deg_{x_i}(T^{-1}(R'(\mathrm{X}))) < \deg(p_i(x_i))$ for $1 \leq i \leq r$. Combining two expression for f, we immediately conclude that $(R - T^{-1}(R')) = 0 \pmod{I_{[r]}}$ which forces that $R = T^{-1}(R')$. ◀

3.1 Proof of Theorem 3

Proof. We now describe a recursive procedure REM to solve the problem. The initial call to it is $\mathrm{REM}(f(\ell_1, \ldots, \ell_r), I_{[n]}, \vec{\alpha})$. We apply the invertible linear transformation obtained in Lemma 18 to get the polynomial $T(f)$ over the variables $x_1, \ldots, x_r, x_{r+1}, \ldots, x_{r+r'}$ where $r' \leq r$.[2] The polynomial $T(f)$ can be explicitly computed in time $\mathrm{poly}(L, s, n, d^{O(r)})$. Then we compute the remainder polynomial $f'(x_1, \ldots, x_{r+r'}) = T(f) \pmod{I_{[r]}}$ by applying the division algorithm which runs in time $\mathrm{poly}(L, s, n, d^{O(r)})$. Next we compute the polynomial $g = f'(\alpha_1, \ldots, \alpha_r, x_{r+1}, \ldots, x_{r+r'})$. Notice from Lemma 18 that $T^{-1}(x_{r+i}) = \ell_{i,2}$ for $1 \leq i \leq r'$, thus we are interested in the polynomial $g(\ell_{1,2}, \ldots, \ell_{r',2})$. Now we recursively compute $\mathrm{REM}(g(\ell_{1,2}, \ldots, \ell_{r',2}), I_{[n]\setminus[r]}, \vec{\alpha}')$ where $\vec{\alpha}' = (\alpha_{r+1}, \ldots, \alpha_n)$.

[2] We use f to denote $f(\ell_1, \ldots, \ell_r)$.

Correctness of the algorithm. Let $R(\mathrm{X}) = f \pmod{I_{[n]}}$ be the unique remainder polynomial. Let $R_r(\mathrm{X}) = f \pmod{I_{[r]}}$ and we know that $R_r \pmod{I_{[n]\setminus[r]}} = R$. So by Lemma 19, to show the correctness of the algorithm, it is enough to show that $g(\ell_{1,2}, \ldots, \ell_{r',2}) = R_r(\alpha_1, \ldots, \alpha_r, x_{r+1}, \ldots, x_n)$.

Following Lemma 20, write $R' = f'(x_1, \ldots, x_r, x_{r+1}, \ldots, x_n) = T(f) \pmod{I_{[r]}}$. Then, by Lemma 20 we conclude that $R' = T(R_r)$. It immediately follows that $R_r = T^{-1}(R') = f'(x_1, \ldots, x_r, T^{-1}(x_{r+1}), \ldots, T^{-1}(x_n))$. Now by definition the polynomial $g(\ell_{1,2}, \ldots, \ell_{r',2})$ is $f'(\alpha_1, \ldots, \alpha_r, T^{-1}(x_{r+1}), \ldots, T^{-1}(x_{r+r'}))$ which is simply $R_r(\alpha_1, \ldots, \alpha_r, x_{r+1}, \ldots, x_n)$.

Time complexity. First, suppose that the field arithmetic over $\mathbb{F}$ can be implemented using polynomial bits and L be the bit-size upper bound for any coefficient in $f, p_1, \ldots, p_n$. This covers all the finite fields where the field is given by an explicit irreducible polynomial. Also, over any such field the polynomial $T(f)$ can be explicitly computed from the input arithmetic circuit deterministically in time $\mathrm{poly}(L, s, n, d^{O(r)})$.

Notice that in each recursive application the number of generators in the ideal is reduced by at least one. Furthermore, in each recursive step we need time $\mathrm{poly}(L, s, n, d^{O(r)})$ to run the division algorithm. This gives us a recurrence of $t(n) \leq t(n-1) + \mathrm{poly}(L, s, n, d^{O(r)})$ which solves to $t(n) \leq \mathrm{poly}(L, s, n, d^{O(r)})$.

Bit-size growth over $\mathbb{Q}$: Over $\mathbb{Q}$, we only need to argue that the intermediate bit-size complexity growth is only polynomial in the input size.Let $\tilde{L}$ be the maximum bit size of any coefficient appearing in $f(z_1, \ldots, z_r)$, and let L be an upper bound on the bit sizes of the other inputs, i.e. bit sizes of coefficients of $\ell_1, \ldots, \ell_r, p_1, \ldots, p_n$ and $\alpha_1, \ldots, \alpha_n$. We will show that the circuit that we use in the next recursive step has coefficients of bit size at most $\tilde{L} + \mathrm{poly}(n, d, L)$.

Let $|c(h)|$ denote the maximum coefficient (in absolute value) appearing in any polynomial h. Then by direct expansion we can see that $|c(f(\ell_1, \ldots, \ell_r))| \leq 2^{\tilde{L}+\mathrm{poly}(n,d,L)}$. Also the linear transformation from lemma 18 can be implemented using poly-bit size entries. Together, we get that that $c(T(f(\ell_1, \ldots, \ell_r)) \leq 2^{\tilde{L}+\mathrm{poly}(n,d,L)}$. At this point, we expand the circuit and obtain $T(f)$ explicitly as a sum of $d^{O(r)}$ monomials. Then divide $T(f)$ by $p_1(x_1), \ldots, p_r(x_r)$ one-by-one, and substitute $x_1 = \alpha_1, \ldots, x_r = \alpha_r$ giving us the remainder $g(x_{r+1}, \ldots, x_{r+r'})$. We note that $|c(g)| \leq 2^{\tilde{L}+\mathrm{poly}(n,d,L)}$ [3]. Now the algorithm passes the $d^{O(r)}$ size $\Sigma\Pi\Sigma$ circuit $g(\ell_{1,2}, \ldots, \ell_{r',2})$ (We note that $T^{-1}(x_{r+1}) = \ell_{1,2}, \ldots, T^{-1}(x_{r+r'}) = \ell_{r',2}$), univariates $p_{r+1}(x_{r+1}), \ldots, p_n(x_n)$ and the point $(\alpha_{r+1}, \ldots, \alpha_n)$ for the next recursive call. We note that the bit-size upper bound L does not change for the input linear forms, and the coefficient bit-size of f grows from $\tilde{L}$ to $\tilde{L} + \mathrm{poly}(n, d, L)$ in one step of the recursion. This gives us the recurrence $S(n) \leq S(n-1) + \mathrm{poly}(n, d, L)$ with $S(1) = \tilde{L}$, which solves to $S(n) = O(\tilde{L} + \mathrm{poly}(n, d, L))$. ◀

▶ Remark. Given a rank r polynomial $f(\ell_1, \ldots, \ell_r)$ and a univariate ideal $I = \langle p_1(x_1), \ldots, p_n(x_n) \rangle$, we can decide the membership of f in I by testing identity of $f \pmod{I}$ i.e. by evaluating $f \pmod{I}$ at some $\alpha \in \mathbb{F}^n$ chosen randomly [11, 29, 26]. Hence, the membership can be decided in randomized $d^{O(r)} \cdot \mathrm{poly}(n)$ time where $d = \max\{\deg(f), \deg(p_i) : 1 \leq i \leq n\}$ using Theorem 3.

[3] We tackle a similar situation in Section 5, and Lemma 33 gives further explanation on the bit-complexity growth when we divide by univariate polynomials.

3.2 Vertex Cover Detection in Low Rank Graphs

In the Vertex Cover problem, we are given a graph $G = (V, E)$ on n vertices and an integer k and the question is to decide whether there is a vertex cover of size k in G. This is a classical NP-complete problem. In this section we show an efficient algorithm to detect vertex cover in a graph whose adjacency matrix is of low rank.

Proof of Theorem 4. We present a reduction from Vertex Cover problem to Univariate Ideal Membership problem that produces a polynomial whose rank is almost same as the rank of A_G. Consider the ideal $I = \langle x_1^2 - x_1, x_2^2 - x_2, \ldots, x_n^2 - x_n \rangle$ and the polynomial

$$f = \prod_{s=1}^{\binom{n}{2}} (\vec{x} A_G \vec{x}^T - s) \cdot \prod_{t=0}^{n-k-1} \left(\sum_{i=1}^{n} x_i - t \right),$$

where A_G is the adjacency matrix of the graph G and $\vec{x} = (x_1, x_2, \ldots, x_n)$ is row-vector.

▶ **Lemma 21.** *The rank of the polynomial f is at most $r + 1$.*

Proof. We note that A_G is symmetric since it encodes an undirected graph. Let Q be an invertible $n \times n$ matrix that diagonalizes A_G. So we have $QA_GQ^T = D$ where D is a diagonal matrix with only the first r diagonal elements being non-zero. Let $\vec{y} = (y_1, y_2, \ldots, y_n)$ be another row-vector of variables. Now, we show the effect of the transform $\vec{x} \mapsto \vec{y}Q$ on the polynomial $\vec{x} A_G \vec{x}^T$. Clearly, $\vec{y}QA_GQ^T\vec{y}^T = \vec{y}D\vec{y}^T$ and since there are only r non-zero entries on the diagonal, the polynomial $\vec{y}D\vec{y}^T$ is over the variables $y_1, y_2, \ldots, y_r$. Thus $g = \prod_{s=1}^{\binom{n}{2}} (\vec{x} A_G \vec{x}^T - s)$ is a rank r polynomial. Also $h = \prod_{t=0}^{n-k-1} (\sum_{i=1}^{n} x_i - t)$ is a rank 1 polynomial as there is only one linear form $\sum_{i=1}^{n} x_i$. Since $f = gh$, we conclude that f is a rank $r + 1$ polynomial. ◀

Now the proof of Theorem 4 follows from the next claim.

▶ **Claim 22.** *The graph G has a Vertex Cover of size k if and only if $f \notin I$.*

Proof. First, observe that the set of common zeroes of the generators of the ideal I is the set $\{0, 1\}^n$. Let S be a vertex cover in G such that $|S| \leq k$. We will exhibit a point $\vec{\alpha} \in \{0, 1\}^n$ such that $f(\vec{\alpha}) \neq 0$. This will imply that $f \notin I$. Identify the vertices of G with $\{1, 2, \ldots, n\}$. Define $\vec{\alpha}(i) = 0$ if and only if $i \in S$. Since $\vec{x} A_G \vec{x}^T = \sum_{(i,j) \in E_G} x_i x_j$ and S is a vertex cover for G, it is clear that $\vec{x} A_G \vec{x}^T(\vec{\alpha}) = 0$. Also $(\sum_{i=1}^{n} x_i)(\vec{\alpha}) \geq n - k$. Then clearly $f(\vec{\alpha}) \neq 0$.

For the other direction, suppose that $f \notin I$. Then by Theorem 1, there exists $\vec{\alpha} \in \{0, 1\}^n$ such that $f(\vec{\alpha}) \neq 0$. Define the set $S \subseteq [n]$ as follows. Include $i \in S$ if and only if $\vec{\alpha}(i) = 0$. Since $f(\vec{\alpha}) \neq 0$, and the range of values that $\vec{x} A_G \vec{x}^T$ can take is $\{0, 1, \ldots, |E|\}$, it must be the case that $\vec{x} A_G \vec{x}^T(\vec{\alpha}) = 0$. It implies that the set S is a vertex cover for G. Moreover, $\prod_{t=0}^{n-k-1} (\sum_{i=1}^{n} x_i - t)(\vec{\alpha}) \neq 0$ implies that $|S| \leq k$. ◀

The degree of the polynomial f is bounded by $n^2 + n$ and from Claim 22 we know that $f \pmod I$ is a non-zero polynomial if and only if G has a vertex cover of size k. By Schwartz-Zippel-Demillo-Lipton [11, 29, 26] lemma $(f \pmod I)(\vec{\beta})$ is non-zero with high probability when $\vec{\beta}$ is chosen randomly from a small domain. Now, we need to just compute $(f \pmod I)(\vec{\beta})$ where f is a rank $r+1$ polynomial with $\ell_i = (\vec{x}Q^{-1})_i$ for each $1 \leq i \leq r$ and $\ell_{r+1} = \sum_{i=1}^{n} x_i$ which can be performed in $(n, k)^{O(r)}$ time using Theorem 3. ◀

4 Parameterized Complexity of Univariate Ideals

We have already mentioned in Fact 2, that checking if the integer permanent is zero is reducible to testing membership of a polynomial $f(\mathrm{X})$ in the ideal $\langle x_1^2, \ldots, x_n^2 \rangle$. So univariate ideal membership is hard for the complexity class $\mathrm{C_=P}$ even when the ideal is generated by powers of variables [23]. In this section we study the univariate ideal membership with the lens of parametrized complexity. The parameters we consider are either polynomial degree or number of the generators for the ideal.

4.1 Parameterized by the Degree of the Polynomial

We consider the following: Let I be a univarite ideal given by generators and $f \in \mathbb{F}[\mathrm{X}]$ a degree k polynomial. Is checking whether f is in I fixed parameter tractable (with k as the fixed parameter)?

We show that it admits an FPT algorithm for the special case when $I = \langle x_1^{e_1}, x_2^{e_2}, \ldots, x_n^{e_n} \rangle$ and we work over either $\mathbb{Z}$ or any finite field of large characteristic.

4.1.1 Proof of Theorem 5

Proof. The proof consists of following three lemmas. Firstly, given an input instance a degree-k $f(\mathrm{X})$ and ideal $I = \langle x_1^{e_1}, x_2^{e_2}, \ldots, x_n^{e_n} \rangle$ of ideal membership, we reduce it to computing the (scaled) Hadamard product of $f(\mathrm{X})$ and a polynomial $g(\mathrm{X})$, where $g(\mathrm{X})$ is a weighted sum of all degree k monomials that are not in I. Then we show that we can evaluate Hadamard product(defined in Section 2) of any two polynomials at a point in time roughly linear in the product of the size of the circuits when one of the polynomials is given by a diagonal circuit as input. Finally the last part of the proof is a randomized construction of a homogeneous degree k diagonal circuit of top fain-in roughly $O^*(4.08^k)$ that computes a polynomial weakly equivalent to the polynomial g with constant probability. Recall that, two polynomials f and g are said to be *weakly equivalent* if they share same set of monomials.

To define the polynomial $g(\mathrm{X})$, let $S_{m,k}$ be the elementary symmetric polynomial of degree k over m variables. Set $m = \sum_{i=1}^{n}(e_i - 1)$. Let $S_{m,k}$ is defined over the variable set $\{z_{1,1}, \ldots, z_{1,e_1-1}, \ldots, z_{n,1}, \ldots, z_{n,e_n-1}\}$. We define $g(\mathrm{X})$ as the polynomial obtained from $S_{m,k}$ replacing each $z_{i,j}$ by x_i.

▶ **Lemma 23.** *Given integers $e_1, e_2, \ldots, e_n$, and a polynomial $f(\mathrm{X})$ of degree k, $f \in \langle x_1^{e_1}, x_2^{e_2}, \ldots, x_n^{e_n} \rangle$ if and only if $f \circ^s g \equiv 0$.*

Proof. Suppose, $f \notin \langle x_1^{e_1}, x_2^{e_2}, \ldots, x_n^{e_n} \rangle$, then f must contain a degree k monomial $M = x_1^{f_1} x_2^{f_2} \ldots x_n^{f_n}$ such that $f_i < e_i$ for each $1 \leq i \leq n$. From the construction, it is clear that $g(\mathrm{X})$ contains M. Therefore, the polynomial $f \circ^s g$ is not identically zero. The converse is also true for the similar reason. ◀

▶ **Lemma 24.** *Given a circuit C of size s computing a polynomial $g \in \mathbb{F}[\mathrm{X}]$ and a homogeneous degree k diagonal circuit $\Sigma^{[s']}\wedge^{[k]}\Sigma$ circuit D of top fan-in s' computing $f \in \mathbb{Q}[\mathrm{X}]$ and $\vec{a} \in \mathbb{Q}^n$, we can evaluate $(f \circ^s g)(\vec{a})$ in deterministic $ss' \cdot \mathrm{poly}(n,k)$ time using $\mathrm{poly}(n,k)$ space.*

Proof. Let M be a degree d monomial over X in f and $M = x_1^{e_1} \cdots x_n^{e_n}$, it follows from the definition that

$$\left(M \circ^s (b_1 x_1 + \ldots + b_n x_n)^d\right)(\vec{a}) = \left(M! \cdot \frac{d!}{M!} \cdot b_1^{e_1} \cdots b_n^{e_n} \cdot M\right)(\vec{a}) = d! \cdot M(a_1 b_1, \ldots, a_n b_n).$$

Recall that, $M!$ is used for $e_1! \cdots e_n!$. As $\circ^s$ distributes over addition, we can write

$$\left(f \circ^s \sum_{i=1}^{s'} (b_{i1}x_1 + \ldots + b_{in}x_n)^d \right)(\vec{a}) = d! \cdot \sum_{i=1}^{s'} f(a_1 b_{i1}, \ldots, a_n b_{in}).$$

The computation can be done in deterministic $ss' \cdot \text{poly}(n,k)$ time using $\text{poly}(n,k)$ space. ◀

▶ **Lemma 25.** *There is an efficient randomized algorithm that constructs with constant probability a homogeneous degree k diagonal circuit D of top fan-in $O^*(4.08^k)$ which computes a polynomial weakly equivalent to the polynomial g (defined before Lemma 23).*

Proof. To construct such a diagonal circuit D, we use the idea of [22]. We pick a collection of colourings $\{\zeta : [m] \to [1.5 \cdot k]\}$ of size roughly $O^*((\frac{e}{\sqrt{3}})^k)$ uniformly at random. For each such colouring ζ_i, we define a $\Pi^{[1.5 \cdot k]}\Sigma$ formula $P_i = \prod_{j=1}^{1.5k}(L_j + 1)$, where $L_j = \sum_{\ell:\zeta_i(\ell)=j} x_\ell$. We say that a monomial is *covered* by a coloring ζ_i if the monomial is in P_i. It is easy to see that, given any multilinear monomial of degree k, the probability that a random coloring will cover the monomial is roughly $(\frac{\sqrt{3}}{e})^k$. Hence, going over such a collection of colorings of size $O^*((\frac{e}{\sqrt{3}})^k)$ chosen uniformly at random, with a constant probability all the multilinear terms of degree k will be covered. To take the Hadamard product with a polynomial of degree k, we need to extract out the degree k homogeneous part (say P_i') from each P_i. Notice that, using elementary symmetric polynomial over $1.5k$ many variables $S_{1.5k,k}$, we can write $P_i' = S_{1.5k,k}(L_1, \ldots, L_{1.5k})$. Now we use Lemma 16 to get a diagonal $\Sigma \wedge^{[k]} \Sigma$ circuit of top fan-in roughly $\binom{1.5k}{0.5k}$ for each P_i'. Define $D = \sum_{i=1}^{O^*((\frac{e}{\sqrt{3}})^k)} P_i'$. By a direct calculation, one can obtain a diagonal circuit D of top fan-in $O^*(4.08^k)$ which is weakly equivalent to the polynomial $S_{m,k}$. The construction of the polynomial $g(\mathrm{X})$ from $S_{m,k}$ is already explained before Lemma 23. ◀

Now, given a circuit C computing $f \in \mathbb{Z}[\mathrm{X}]$ and integers $e_1, \ldots, e_n$, to decide the membership of f in the ideal $I = \langle x_1^{e_1}, \ldots, x_n^{e_n} \rangle$, we construct a diagonal circuit D from Lemma 25. Following Lemma 23, we can decide the membership of f in the ideal checking the polynomial $C \circ^s D$ is identically zero or not which can be performed by randomly picking $\vec{a} \in \mathbb{Z}^n$ using Schwartz-Zippel-Demillo-Lipton Lemma [26, 29, 11] and evaluating $(C \circ^s D)(\vec{a})$ using Lemma 24. Over $\mathbb{Z}$ the given circuit can compute numbers as large as $2^{2^{n^{O(1)}}}$. To handle this, a standard idea is to evaluate the circuit modulo a random polynomial bit prime. ◀

4.2 Parameterized by Number of Generators

In this section, we consider the univariate ideal membership parameterized on the number of generators of the ideal. More precisely, given a polynomial $f(\mathrm{X})$, can we obtain an FPT algorithm for testing membership in the univariate ideal $\langle p_1(x_1), \ldots, p_k(x_k) \rangle$ parameterized by k? We show that the problem is W[1]-hard. Moreover, in contrast to the previous case, we obtain MINI[1]-hardness for a special case of the problem when the univariate generators are just power of variables.

Proof of Theorem 6. We show a reduction from *k-independent set*, a well known W[1]-hard problem [10], to this problem. Let $G = (V, E)$ be a graph on n vertices and k be the size of the independent set. We identify its vertex set with the numbers $\{1, 2, \ldots, n\}$ and the edges are tuples over $[n] \times [n]$. Define the univariate ideal $I = \langle p_1(x_1), \ldots, p_k(x_k) \rangle$ where for each

$1 \leq i \leq k$, we define $p_i(x_i) = \prod_{j=1}^{n}(x_i - j)$. Now we are going to define a polynomial f that uses only k variables which will be used for the ideal membership problem. First consider the polynomial $D = \prod_{1 \leq i \neq j \leq k}(x_i - x_j)$.

Now we define the polynomial,

$$f = \prod_{1 \leq i \neq j \leq k} \prod_{(u,v) \in E \subseteq [n] \times [n]} [(x_i - u)^2 + (x_j - v)^2] \cdot [(x_j - u)^2 + (x_i - v)^2].$$

The proof follows from the following claim.

▶ **Claim 26.** *$f \cdot D \notin \langle p_1(x_1), p_2(x_2), \ldots, p_k(x_k) \rangle$ if and only if G has an independent set of size k.*

Proof. We use Theorem 1 to prove the claim. Let $\{j_1, j_2, \ldots, j_k\}$ be an independent set in G. Notice that $(j_1, \ldots, j_k)$ is a common zero of the generators $p_1, \ldots, p_k$. Now notice that $f \cdot D$ does not vanish at the point $(j_1, \ldots, j_k)$ as all the edges $(j_\ell, j_{\ell'}) : 1 \leq \ell, \ell' \leq k$ are absent in the edge set E. Thus there is a common root of the ideal on which $f \cdot D$ does not vanish and hence $f \cdot D \notin \langle p_1(x_1), p_2(x_2), \ldots, p_k(x_k) \rangle$.

Now if $f \cdot D \notin \langle p_1(x_1), p_2(x_2), \ldots, p_k(x_k) \rangle$ then there is a common zero $(j_1, \ldots, j_k)$ of the ideal on which $f \cdot D$ does not vanish. Using the same argument one can easily see that $\{j_1, \ldots, j_k\}$ is an independent set in G. ◀

◀

4.2.1 Proof of Theorem 7

We first relate our univariate ideal membership problem with a linear algebraic problem k-Lin-Eq. It turns that k-Lin-Eq problem is more amenable to the MINI[1]-hardness proof. Finally we show a reduction from MINI-1-in-3 POSITIVE 3-SAT to k-Lin-Eq to complete the proof.

▶ **Definition 27.** k-Lin-Eq
Input: Integers k, n in unary, a $k \times n$ matrix A with all the entries given in unary and a k dimensional vector $\vec{b}$ with all entries in unary.
Parameter: k.
Question: Does there exist an $\vec{x} \in \{0,1\}^n$ such that $A\vec{x} = \vec{b}$?

▶ **Lemma 28.** *There is a parameterized reduction from k-Lin-Eq to the univariate ideal membership problem when the ideal is given by the powers of variables as generators.*

Proof. We introduce $2k$ variables $x_1, x_2, \ldots, x_k, y_1, y_2, \ldots, y_k$ where two variables will be used for each row. For each $i \in [n]$, let $\mu_i = \sum_{j=1}^{n} a_{ij}$. For each column $c_i = (a_{1i}, a_{2i}, \ldots, a_{ki})$ we construct the polynomial $P_i = ({y_1}^{a_{1i}} {y_2}^{a_{2i}} \ldots {y_k}^{a_{ki}} + {x_1}^{a_{1i}} {x_2}^{a_{2i}} \ldots {x_k}^{a_{ki}})$. We let $P_A = \prod_{i=1}^{n} P_i$ and we choose the ideal to be $\langle x_1^{b_1+1}, y_1^{\mu_1 - b_1 + 1}, \ldots, x_k^{b_k+1}, y_1^{\mu_k - b_k + 1} \rangle$. Notice that P_A has a small arithmetic circuit which is polynomial time computable.

▶ **Claim 29.** *An instance $(A, \vec{b})$ is an YES instance for k-Lin-Eq iff $P_A \notin \langle x_1^{b_1+1}, y_1^{\mu_1 - b_1 + 1}, \ldots, x_k^{b_k+1}, y_k^{\mu_k - b_k + 1} \rangle$.*

Proof of Claim. Suppose $(A, \vec{b})$ is an YES instance. Then there is an $\vec{x} \in \{0,1\}^n$ such that $A\vec{x} = \vec{b}$. Define $S := \{i \in [n] : \vec{x}_i = 1\}$ where $\mathbf{x_i}$ is the ith co-ordinate of $\vec{x}$. Think of the monomial where ${x_1}^{a_{1i}} {x_2}^{a_{2i}} \ldots {x_k}^{a_{ki}}$ is picked from P_i for each $i \in S$ and ${y_1}^{a_{1i}} {y_2}^{a_{2i}} \ldots {y_k}^{a_{ki}}$

is picked from reaming P_j's where $j \in \bar{S}$. This gives us the monomial $x_1^{b_1} y_1^{\mu_1 - b_1} \dots x_k^{b_k} y_1^{\mu_k - b_k}$ in the polynomial P_A. Thus $P_A \notin \langle x_1^{b_1+1}, y_1^{\mu_1 - b_1 + 1}, \dots, x_k^{b_k+1}, y_k^{\mu_k - b_k + 1} \rangle$.

Now we show the other direction. Now suppose $P_A \notin \langle x_1^{b_1+1}, y_1^{\mu_1 - b_1 + 1}, \dots, x_k^{b_k+1}, y_k^{\mu_k - b_k + 1} \rangle$. Let $S := \{i \in [n] : {x_1}^{a_{1i}} {x_2}^{a_{2i}} \dots {x_k}^{a_{ki}}$ is picked from $P_i\}$. There must be a monomial ${x_1}^{c_1} {x_2}^{c_2} \dots {x_k}^{c_k} {y_1}^{d_1} {y_2}^{d_2} \dots {y_k}^{d_k}$ in P_A such that for each i, $\sum_{j \in S} a_{ij} = c_i \leq b_i$, $\sum_{j \notin S} a_{ij} = d_i \leq (\mu_i - b_i)$. As, $\mu_i = \sum_{j \in S} a_{ij} + \sum_{i \notin S} a_{ij}$, we get $b_i \leq \sum_{j \in S} a_{ij}$. Hence, $\sum_{j \in S} a_{ij} = b_i$ for each i. Define $\vec{x} \in \{0,1\}^n$ where $\vec{x}_i = 1$ if $i \in S$ else $\vec{x}_i = 0$. This shows $(A, \vec{b})$ is an YES instance. ◀

◀

Before we prove the MINI[1]-hardness of k-Lin-Eq, we show that the following problem is MINI[1]-hard.

▶ **Definition 30.** MINI-1-in-3 POSITIVE 3-SAT
Input: Integers k, n in unary, a 3-SAT instance $\mathcal{E}$ consisting of only positive literals where $\mathcal{E}$ has at most $k \log n$ variables and atmost $k \log n$ clauses.
Parameter: k.
Question: Does there exist a satisfiable assignment for $\mathcal{E}$ such that every clause has exactly one TRUE literal?

▶ **Claim 31.** MINI-1-in-3 POSITIVE 3-SAT *is* MINI[1]*-hard.*

To prove the claim we only need to observe that the standard *Schaefer Reduction* [25] from 3-SAT to 1-in-3 POSITIVE 3-SAT is in fact a linear size reduction, that directly gives us an FPT reduction from MINI-3SAT to MINI-1-in-3 POSITIVE 3-SAT.

Proof of Theorem 7. Given a MINI-1-in-3 POSITIVE 3-SAT instance $\mathcal{E}$, order the variables $v_1, \dots, v_{k \log n}$ and the clauses $C_1, \dots, C_{k \log n}$. Construct the following $k \log n \times k \log n$ matrix M where the rows are indexed by the clauses and the columns are indexed by the variables. $M[i][j]$ is set to 1 if v_j appears in C_i, otherwise set it to 0. Make M a $2k \log n \times n$ matrix by adding an all zero row between every rows and appending all zero columns at the end. Now, define $\vec{e}$ as a $2k \log n$ dimensional vector where ith co-ordinate of e, $e_i = 1$ when i is odd and $e_i = 0$ when i is even. We want to find $\vec{y} \in \{0,1\}^n$ such that $M\vec{y} = \vec{e}$.

However this is not an instance of k-Lin-Eq. To make it so, we observe that M is a bit matrix and $\vec{e}$ is a bit vector, hence we can modify them to a $k \times n$ matrix A and k dimensional vector $\vec{b}$ in the following way. For each column j, think of the ith consecutive $2 \log n$ bits as the binary expansion of a single entry, call it N and set $A[i][j]$ to N. Similarly, we modify $\vec{e}$ to a k dimensional vector $\vec{b}$ by considering $2 \log n$ bits as a binary expansion of a single entry. Now the proof follows from the following claim.

▶ **Claim 32.** *$\mathcal{E}$ is an YES instance for* MINI-1-in-3 POSITIVE 3-SAT *if and only if there exists an $\vec{x} \in \{0,1\}^n$ such that $A\vec{x} = \vec{b}$.*

Proof. Suppose there is such a satisfiable assignment for $\mathcal{E}$. Define $S := \{j \in [k \log n] \mid v_j = \text{TRUE}\}$. Define $\vec{z} \in \{0,1\}^n$ such that $z_j = 1$ where $j \in S$ else $z_j = 0$. For each i, as C_i contains exactly one TRUE literal, hence $e_{2i+1} = \sum_{j=1}^{n} M[i][j] \cdot z_j = 1$ and $e_{2i} = 0$. Therefore $\vec{z}$ is a solution for $M\vec{y} = \vec{e}$. As every integer has a unique binary expansion, hence $\vec{z}$ is also a solution for $A\vec{x} = \vec{b}$.

Now we prove the other direction. Suppose $A\vec{z} = \vec{b}$ for some $\vec{z} \in \{0,1\}^n$. From the construction of the matrix M, it is sufficient to show that $\vec{z}$ is a satisfying assignment for $M\vec{y} = \vec{e}$. First we note that the numbers $A[i][j], b[i]$ in their binary expansion have

bits 1 in the odd location and 0 in the even locations. Let $A[i][j] = \sum_{t=1}^{2\log n} a_{ijt}2^{t-1}$ and $b[i] = \sum_{t=1}^{2\log n} e_t 2^{t-1}$. Since $A\vec{z} = \vec{b}$ we have $\sum_{j=1}^{n} A[i][j] \cdot z_j = b[i]$. This shows that

$$\sum_{j=1}^{n} A[i][j] \cdot z_j = \sum_{j=1}^{n} \left(\sum_{t=1}^{2\log n} a_{ijt}2^{t-1} \right) \cdot z_j = \sum_{t=1}^{2\log n} \left(\sum_{j=1}^{n} a_{ijt} \cdot z_j \right) 2^{t-1}.$$

Since $\mathcal{E}$ is a 3-CNF formula we have $(\sum_{j=1}^{n} a_{ijt} \cdot z_j) \in \{0,1,2,3\}$. Now we compare $(\sum_{j=1}^{n} a_{ijt} \cdot z_j)$ with the binary expansion of $b[i]$. When t is odd the bit e_t is 1 and so there must be a 1 in the corresponding bit of $(\sum_{j=1}^{n} a_{ijt} \cdot z_j)$. This shows that $(\sum_{j=1}^{n} a_{ijt} \cdot z_j) \neq 0$ when t is odd. Now if $(\sum_{j=1}^{n} a_{ijt} \cdot z_j) \in \{2,3\}$ for any odd t then the term 2^{t+1} will be produced and this will not match the expansion of $b[i]$ as the $e_{t+1} = 0$. Thus by the uniqueness of binary expansion we conclude that $(\sum_{j=1}^{n} a_{ijt} \cdot z_j) = 1$ if t is odd and 0 otherwise. Thus $M\vec{y} = \vec{e}$ has a solution with $y_i = z_i$. ◀

◀

5 Non-deterministic Algorithm for Univariate Ideal Membership

In this section we prove Theorem 8. Given a polynomial $f(\mathrm{X}) \in \mathbb{Q}[\mathrm{X}]$ and a univariate ideal $I = \langle p_1(x_1), \ldots, p_n(x_n) \rangle$ where the generators are $p_1, \ldots, p_n$, we show a non-deterministic algorithm to decide the (non)-membership of f in I. By Theorem 1, it suffices to show that there is a common zero $\vec{\alpha}$ of the generators $p_1, p_2, \ldots, p_n$ such that $f(\alpha) \neq 0$. Since in general $\vec{\alpha} \in \mathbb{C}^n$, it is not immediately clear how to guess such a common zero by a NP machine. However, we are able to show that for the NP machine it suffices to guess such an $\vec{\alpha}$ upto polynomially many bits of approximation.

We begin by proving a few technical facts which are useful for the main proof. Write $f(\mathrm{X}) = \sum_{i=1}^{n} h_i(\mathrm{X})\ p_i(x_i) + R(\mathrm{X})$ where for all $i \in [n]$, $\deg_{x_i}(R) < \deg(p_i)$. For any polynomial g, let $|c(g)|$ be the maximum coefficient (in absolute value) appearing in g. The following lemma gives an estimate for the coefficients of the polynomials $h_1, \ldots, h_n, R$.

▶ **Lemma 33.** *Let $2^{-L} \leq |c(f)|, |c(p_i)| \leq 2^{L}$. Then there is $L' = \mathrm{poly}(L, d, n)$ such that $2^{-L'} \leq |c(h_i)|, |c(R)| \leq 2^{L'}$ where d is the degree upper bound for f, and $\{p_i : 1 \leq i \leq n\}$.*

Proof. The estimate on L' follows implicitly from the known results [8]. It can be also seen by direct computation. Write $f(\mathrm{X}) = \sum_i f_i(x_2, \ldots, x_n)\ x_1^i$ and then divide $x_1^i \pmod{p_1(x_1)}$ for each i. The modulo computation can be done by writing $x_1^i = q_1(x_1)p(x_1) + r_1(x_1)$ with the coefficients of q_1 and r_1 are unknown. We can then solve it using standard linear algebra. In particular, one can use the Cramer's rule for system of linear equation solution. The growth of the bit-size is only $\mathrm{poly}(L, d)$. More precisely, if $c_{\max}$ is the maximum among $|c(f)|, |c(p_1)|$, any final coefficient is at most $c_{\max} \cdot 2^{\mathrm{poly}(L,d)}$. We repeat the procedure for the other univariate polynomials one by one. The final growth on the coefficients size is at most $\mathrm{poly}(n, L, d)$. ◀

Let $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{C}^n$ be such that $p_i(\alpha_i) = 0$, $1 \leq i \leq n$. From Lemma 11, we get that $\frac{1}{2^{\hat{L}}} \leq |\alpha_i| \leq 2^{\hat{L}}$ where $\hat{L} = \mathrm{poly}(L, d)$. Let $\tilde{\alpha}_i \in \mathbb{Q}[i]$ be an ϵ-approximation of α_i, e.g. $|\alpha_i - \tilde{\alpha}_i| \leq \epsilon$. Then we show that the absolute value of $p_i(\tilde{\alpha}_i)$ is not too far from zero.

▶ **Observation 34.** *For $1 \leq i \leq n$ we have that $|p_i(\tilde{\alpha}_i)| \leq \epsilon \cdot 2^{(dL)^{O(1)}}$.*

Proof. Let $p_i(x_i) = c \cdot \prod_{j=1}^{d}(x_i - \beta_{i,j})$ and w.l.o.g assume that $\tilde{\alpha}_i$ is the approximation of the root $\beta_{i,1}$. Then $|p_i(\tilde{\alpha}_i)| \leq \epsilon \cdot |c| \cdot \prod_{j=2}^{d} |\tilde{\alpha}_i - \beta_{i,j}| \leq \epsilon \cdot |c| \cdot \prod_{j=2}^{d} (|\beta_{i,1} - \beta_{i,j}| + \epsilon) \leq \epsilon \cdot 2^{\mathrm{poly}(d,L)}$. The final bound follows from the bound on the roots given in Lemma 11. ◀

Since we have an upper bound on the coefficients of the polynomials $\{h_i : 1 \leq i \leq n\}$ from Lemma 33, it follows that for $1 \leq i \leq n$ we have that $|h_i(\tilde{\alpha})| \leq 2^{(ndL)^{O(1)}}$. Here we use the fact that the approximate root α_i can be trivially bounded by $2^{\hat{L}+1}$.

5.1 Proof of Theorem 8

Proof. If f is not in the ideal I, by Alon's Nullstellensatz, we know that there exists a tuple $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n) \in Z(p_1) \times \ldots \times Z(p_n)$ such that $R(\vec{\alpha}) \neq 0$. Suppose that the NP Machine guess the tuple $\vec{\tilde{\alpha}} = (\tilde{\alpha}_1, \ldots, \tilde{\alpha}_n)$ which is the ϵ-approximation of the tuple $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n)$ [4]. Using the black-box for f, obtain the value for $f(\vec{\tilde{\alpha}})$. Next, we show that the value $|f(\vec{\tilde{\alpha}})|$ distinguishes between the cases $f \in I$ and $f \notin I$.

Case 1: $f \in I$. $|f(\vec{\tilde{\alpha}})| = |\sum_{i=1}^{n} h_i(\vec{\tilde{\alpha}}) p_i(\tilde{\alpha}_i)| \leq (\sum_{i=1}^{n} |h_i(\vec{\tilde{\alpha}})|) \cdot \epsilon \cdot 2^{(dL)^{c_1}} \leq \epsilon \cdot 2^{(ndL)^{c_2}}$. where the constant c_2 is fixed by Observation 34 and the bounds on $|h_i(\vec{\tilde{\alpha}})|$.

Case 2: $f \notin I$. Recall the inequality for complex numbers : $|Z_1 + Z_2| \geq |Z_2| - |Z_1|$. Using this write $|f(\vec{\tilde{\alpha}})| \geq |R(\vec{\tilde{\alpha}})| - \sum_{i=1}^{n} |h_i(\vec{\tilde{\alpha}})|\ |p_i(\vec{\tilde{\alpha}})|$. Notice that $|R(\vec{\tilde{\alpha}})| \geq |R(\vec{\alpha})| - |R(\vec{\tilde{\alpha}}) - R(\vec{\alpha})|$. Combining we get the following : $|f(\vec{\tilde{\alpha}}) \geq |R(\vec{\alpha})| - |R(\vec{\tilde{\alpha}}) - R(\vec{\alpha})| - \epsilon \cdot 2^{(ndL)^{c_2}}$.

Now to complete the proof, we show a lower bound on $|R(\vec{\alpha})|$ and an upper bound for $|R(\vec{\tilde{\alpha}}) - R(\vec{\alpha})|$.

▶ **Claim 35.** $|R(\vec{\alpha})| \geq \frac{1}{2^{(ndL)^{c_3}}}$ *for some constant* c_3.

Proof. Define the polynomial $\hat{R}(x_n) = R(\alpha_1, \ldots, \alpha_{n-1}, x_n) = c \cdot \prod_{j=1}^{d'} (x_n - \beta_j)$ where c is some constant and $d' \leq d$. Note that α_n is not a zero for $\hat{R}(x_n)$. Consider the polynomial $Q(x_n) = p_n(x_n)\hat{R}(x_n)$. The set $\{\alpha_n, \beta_1, \ldots, \beta_{d'}\} \subseteq Z(Q)$ and $\alpha_n \neq \beta_j : 1 \leq j \leq d'$. Using the root separation bound for $|\alpha_n - \beta_j|$ obtained in Lemma 12, we can easily lower bound that $|\hat{R}(\alpha_n)| \geq \frac{1}{2^{(ndL)^{c_3}}}$. ◀

▶ **Claim 36.** $|R(\vec{\tilde{\alpha}}) - R(\vec{\alpha})| \leq 2^{(ndL)^{c_4}}$ *for some constant* c_4.

Proof. Define $R^0(\vec{\tilde{\alpha}}) = R(\vec{\alpha})$ and $R^i(\vec{\tilde{\alpha}}) = R(\tilde{\alpha}_1, \ldots, \tilde{\alpha}_i, \alpha_{i+1}, \ldots, \alpha_n)$. Then we use triangle inequality to notice that $|R(\vec{\alpha}) - R(\vec{\tilde{\alpha}})| \leq \sum_{i=1}^{n} |R^{i-1}(\vec{\tilde{\alpha}}) - R^i(\vec{\tilde{\alpha}})|$. Write explicitly $R^{i-1}(\vec{\tilde{\alpha}}) - R^i(\vec{\tilde{\alpha}}) = \sum_{\vec{e}} c_{\vec{e}} \tilde{\alpha}_1^{e_1} \ldots \tilde{\alpha}_{i-1}^{e_{i-1}} (\alpha_i^{e_i} - \tilde{\alpha}_i^{e_i}) \alpha_i^{e_{i+1}} \ldots \alpha_n^{e_n}$. Notice the upper bounds on $|\alpha_i| \leq 2^{(ndL)^{O(1)}}$, and $|\alpha_i - \tilde{\alpha}_i| \leq \epsilon$. We apply these bounds and use triangle inequality to get that $|R(\vec{\tilde{\alpha}}) - R(\vec{\alpha})| \leq \epsilon \cdot 2^{(ndL)^{c_4}}$. ◀

Combining Claim 35, and Claim 36, we get the lower bound $|f(\vec{\tilde{\alpha}})| \geq \frac{1}{2^{(ndL)^{c_3}}} - \epsilon \cdot (2^{(ndL)^{c_4}} + 2^{(ndL)^{c_2}})$. To make the calculation precise, let $3M = \frac{1}{2^{(ndL)^{c_3}}}$ and choose ϵ such that $\epsilon \cdot (2^{(ndL)^{c_4}} + 2^{(ndL)^{c_2}}) \leq M$.

The final implication will be $|f(\vec{\tilde{\alpha}})| \leq M$ when $f \in I$ and $|f(\vec{\tilde{\alpha}})| \geq 2M$ when $f \notin I$. It is important to note that the parameter M can be pre-computed from the input parameters efficiently.

Now we show how to verify that the guessed point $\vec{\tilde{\alpha}}$ is a good approximation of the roots for the univariate polynomials. We need to only verify that for each i, $\tilde{\alpha}_i$ is a good approximation for *some root* of the univariate polynomial $p_i(x_i)$. The fact that it is also a good approximation for the non-zero of R is already verified above. The NP machine, given

[4] Later we fix ϵ suitably and use Lemma 13 to verify in polynomial time that $\vec{\tilde{\alpha}}$ is indeed ϵ-approximation of $\vec{\alpha}$.

$p_1, \ldots, p_n$ guesses $\tilde{\alpha_i}$ using b bits and verifies that $|p_i(\tilde{\alpha_i})| < 2^{-L}\epsilon^d$ which, by lemma 13, shows that the guessed $\tilde{\alpha_i}$ is ϵ-close to some root of p_i.

We note that such a guess always exists. Indeed, invoking Observation 34 with $|\alpha_i - \tilde{\alpha_i}| \leq \delta$ we can conclude that $|p_i(\tilde{\alpha_i})| \leq \delta \cdot 2^{(dL)^{O(1)}}$. Now, the NP machine can guess b bits such that $|\alpha_i - \tilde{\alpha_i}| \leq 2^{-b}$. We require $2^{-b} \cdot 2^{(dL)^{O(1)}} < 2^{-L}\epsilon^d$, simplifying we get, $2^{-b} < 2^{-(dL)^{O(1)}} \cdot \epsilon^d$. Hence $b > (dL)^{O(1)} \log \frac{1}{\epsilon}$. Thus using $\text{poly}(d, L, \log \frac{1}{\epsilon})$ bits there is always a guess $\tilde{\alpha_i}$ for which $|p_i(\tilde{\alpha_i})| < 2^{-L}\epsilon^d$. ◀

References

1. Noga Alon. Combinatorial Nullstellensatz. *Comb. Probab. Comput.*, 8(1-2):7–29, January 1999. URL: `http://dl.acm.org/citation.cfm?id=971651.971653`.
2. Kotlov Andrew and Lovász László. The rank and size of graphs. *Journal of Graph Theory*, 23(2):185–189, 1996.
3. Vikraman Arvind, Abhranil Chatterjee, Rajit Datta, and Partha Mukhopadhyay. Fast Exact Algorithms Using Hadamard Product of Polynomials. *CoRR*, abs/1807.04496, 2018. `arXiv:1807.04496`.
4. Vikraman Arvind and Partha Mukhopadhyay. The ideal membership problem and polynomial identity testing. *Inf. Comput.*, 208(4):351–363, 2010. `doi:10.1016/j.ic.2009.06.003`.
5. Alexander I. Barvinok. Two Algorithmic Results for the Traveling Salesman Problem. *Math. Oper. Res.*, 21(1):65–84, February 1996. `doi:10.1287/moor.21.1.65`.
6. Andreas Björklund, Petteri Kaski, and Lukasz Kowalik. Constrained Multilinear Detection and Generalized Graph Motifs. *Algorithmica*, 74(2):947–967, 2016. `doi:10.1007/s00453-015-9981-1`.
7. Cornelius Brand, Holger Dell, and Thore Husfeldt. Extensor-coding. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 151–164, 2018. `doi:10.1145/3188745.3188902`.
8. George E. Collins. Subresultants and Reduced Polynomial Remainder Sequences. *J. ACM*, 14(1):128–142, 1967. `doi:10.1145/321371.321381`.
9. David A. Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra, 3/e (Undergraduate Texts in Mathematics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.
10. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. `doi:10.1007/978-3-319-21275-3`.
11. Richard A. Demillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193–195, 1978. `doi:10.1016/0020-0190(78)90067-4`.
12. Rodney G. Downey, Vladimir Estivill-Castro, Michael R. Fellows, Elena Prieto-Rodriguez, and Frances A. Rosamond. Cutting Up is Hard to Do: the Parameterized Complexity of k-Cut and Related Problems. *Electr. Notes Theor. Comput. Sci.*, 78:209–222, 2003. `doi:10.1016/S1571-0661(04)81014-4`.
13. Neeraj Kayal and Nitin Saxena. Polynomial Identity Testing for Depth 3 Circuits. *Computational Complexity*, 16(2):115–138, 2007. `doi:10.1007/s00037-007-0226-9`.
14. Pascal Koiran. Hilbert's Nullstellensatz Is in the Polynomial Hierarchy. *J. Complexity*, 12(4):273–286, 1996. `doi:10.1006/jcom.1996.0019`.
15. Andrevı Kotlov. Rank and Chromatic Number of a Graph. *J. Graph Theory*, 26(1):1–8, September 1997.

16 Ioannis Koutis. Faster Algebraic Algorithms for Path and Packing Problems. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, pages 575–586, 2008. doi:10.1007/978-3-540-70575-8_47.

17 Ioannis Koutis. Constrained multilinear detection for faster functional motif discovery. *Inf. Process. Lett.*, 112(22):889–892, 2012. doi:10.1016/j.ipl.2012.08.008.

18 Ioannis Koutis and Ryan Williams. LIMITS and applications of group algebras for parameterized problems. *ACM Trans. Algorithms*, 12(3):31:1–31:18, 2016. doi:10.1145/2885499.

19 Hwangrae Lee. Power Sum Decompositions of Elementary Symmetric Polynomials. *Linear Algebra and its Applications*, 492:89 – 97, 2016. doi:10.1016/j.laa.2015.11.018.

20 K. Mahler. An inequality for the discriminant of a polynomial. *Michigan Math. J.*, 11(3):257–262, September 1964. doi:10.1307/mmj/1028999140.

21 E. Mayr and A. Meyer. The complexity of word problem for commutative semigroups and polynomial ideals. *Adv. Math*, 46:305–329, 1982.

22 Kevin Pratt. Faster Algorithms via Waring Decompositions. *CoRR*, abs/1807.06194, 2018. arXiv:1807.06194.

23 Sanjeev Saluja. A note on the permanent value problem. *Information Processing Letters*, 43(1):1–5, 1992. doi:10.1016/0020-0190(92)90021-M.

24 Nitin Saxena and C. Seshadhri. From sylvester-gallai configurations to rank bounds: Improved blackbox identity test for depth-3 circuits. *J. ACM*, 60(5):33:1–33:33, 2013. doi:10.1145/2528403.

25 Thomas J. Schaefer. The Complexity of Satisfiability Problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pages 216–226, New York, NY, USA, 1978. ACM.

26 Jacob T. Schwartz. Fast Probabilistic algorithm for verification of polynomial identities. *J. ACM.*, 27(4):701–717, 1980.

27 Madhu Sudan. Lectures on Algebra and Computation: Lecture Notes 6,12,13,14, 1998.

28 Ryan Williams. Finding paths of length k in $O^*(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, 2009. doi:10.1016/j.ipl.2008.11.004.

29 R. Zippel. Probabilistic algorithms for sparse polynomials. In *Proc. of the Int. Sym. on Symbolic and Algebraic Computation*, pages 216–226, 1979.

Verification of Timed Asynchronous Programs

Parosh Aziz Abdulla
Uppsala University, Sweden
parosh@it.uu.se

Mohamed Faouzi Atig
Uppsala University, Sweden
mohamed_faouzi.atig@it.uu.se

Shankara Narayanan Krishna
IIT Bombay, India
krishnas@cse.iitb.ac.in

Shaan Vaidya
IIT Bombay, India
shaan@cse.iitb.ac.in

Abstract

In this paper, we address the verification problem for timed asynchronous programs. We associate to each task, a deadline for its execution. We first show that the control state reachability problem for such class of systems is decidable while the configuration reachability problem is undecidable. Then, we consider the subclass of timed asynchronous programs where tasks are always being executed from the same state. For this subclass, we show that the control state reachability problem is PSPACE-complete. Furthermore, we show the decidability for the configuration reachability problem for the subclass.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Reachability, Timed Automata, Asynchronous programs

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.8

Related Version An extended version of this paper is available at [1], `http://www.cse.iitb.ac.in/~krishnas/fsttcs2018.pdf`.

1 Introduction

One of the well-known design paradigms in concurrent programs is to break a problem into smaller subproblems which are solved asynchronously and concurrently. Each process or thread in the program can then dispatch tasks to other processes, expecting them to be completed by a certain deadline. Each process has a potentially unbounded bag where its pending tasks are stored. In the asynchronous paradigm, one need not wait for time-consuming tasks to be completed to proceed; asynchronous procedure calls are stored in a task buffer, which are executed later, rather than right away. The tasks which are posted asynchronously have deadlines attached to them, and the process or thread, in whose bag the task has been posted, must execute the task within the deadline. In addition to asynchronous procedure calls, one can also make use of synchronous procedure calls where the caller of the procedure blocks until the callee returns. To summarize, an asynchronous program is one that contains procedure calls which are not immediately executed from the calling site, but stored and dispatched in a non-deterministic order by some scheduler(s) at a later point.

As an example for timed asynchronous programs, we look at *SwingWorker*, an abstract class developed for the Swing library of Java, and is used to perform lengthy GUI interaction tasks in a background thread. While developing applications, sometimes the GUI hangs when it is trying to do some lengthy task. For such purposes, the *SwingWorker* class schedules the execution of this lengthy task on a different thread while the GUI still remains responsive. There are deadlines associated with the background tasks, and if the worker thread which is handling the background task does not finish by the given deadline, then an interrupt is created. To update the user (and GUI) regarding the progress of background tasks, inter-thread communication is allowed.

Writing correct asynchronous programs and reasoning about their correctness is very difficult, since the creation and execution of tasks within deadlines leads to unpredictable behaviours. The verification of asynchronous programs is hence a very challenging topic. A formal model of mutiset pushdown systems for asynchronous recursive programs was presented in [16]. This model consists of a pushdown automaton equipped with a multiset or bag. The automaton adds pending asynchronous method calls to the bag, and the stack executes synchronous recursive method calls. A task can be taken from the bag for execution when the stack is empty. The control state reachability problem was shown to be decidable with an EXPSPACE lower bound under this model. This shows that the case of single-thread asynchronous programs, the reachability problem is very difficult. Subsequently, [8] showed that control state reachability for single-thread asynchronous recursive programs is EXPSPACE-complete. In all these models, time constraints do not play a role in the execution of the asynchronous methods. In the timed setting, [7] considers asynchronous calls of the form `future(p,t)` posted to the task buffer, where p is a handler and $t \in \mathbb{N}$. The idea is that the handler p will execute the task in t time units from now. The execution of the program is controlled by logical ticks of a clock. The model proposed in [7] is a generalization of the models in [16] and [11]. [7] shows that safety checking for such programs is undecidable.

The goal of this paper is to investigate the decidability and complexity of the reachability problem for asynchronous non-recursive programs under dense time. We propose a formalism called multiset timed automata (MTA) where each process is modeled as a timed automaton [2]. Each timed automaton is equipped with a bag or multiset. To handle asynchronous method calls, each timed automaton can post a task to the bag of another automaton. These tasks have deadlines attached to them. The deadline is either a natural number $d \in \mathbb{N}$ or ∞. When a task is posted to a bag, its age is considered to be 0, and with elapse of time, the age also grows. A task can be executed by the process in whose bag it lies, before the age of the task exceeds the deadline; tasks whose ages have exceeded the deadline will be forever pending. While a main process picks up pending tasks depending on their ages in [7], in our model, a process can execute a pending task in its bag at its will. There are 2 sources of infinity in our model: one coming from dense-time, and the second coming from the unbounded size of the bags of each process. We investigate control state reachability as well as configuration reachability of this model, and show that control state reachability is decidable and EXPSPACE-hard, while configuration reachability is undecidable. We then identify a practically relevant class of MTA where the task execution happens from the same state in each process, and give a PSPACE-complete decision procedure for control state reachability. The configuration reachability also turns out to be decidable for this class.

Related Work

Most of the existing work (e.g., [3,4,6,8,11,13,14,16]) on the formal verification of asynchronous programs considers the untimed version. In [7], the authors consider timed constraints on tasks; however, this model is different from the formal model studied in this paper. In

fact, in [7], the authors assume that a task should always be executed by its deadline and the execution of each task is done in logical zero time. In our model, a task whose age has exceeded the deadline will be forever pending. Furthermore, the control state reachability for the model presented in [7] is undecidable while it is decidable for our model. In [5], the authors consider a model similar to the one considered in this paper and show that the coverability problem is decidable using a different technique than ours.

2 Preliminaries

In this section, we introduce some notations and definitions that will be used throughout the paper.

Notations

We use standard notation $\mathbb{N}$ for the set of naturals, along with ∞. $\mathbb{R}$ represents the set of non-negative real numbers. Let $\mathcal{X}$ be a finite set of variables called *clocks*, taking values from $\mathbb{R}$. A *valuation* on $\mathcal{X}$ is a function $\nu : \mathcal{X} \to \mathbb{R}$. We assume an arbitrary but fixed ordering on the clocks and write x_i for the i-th clock. This allows us to treat a valuation ν as a vector $(\nu(x_1), \nu(x_2), \ldots, \nu(x_n))$ in $\mathbb{R}^{|\mathcal{X}|}$. For a subset of clocks $X \in 2^{\mathcal{X}}$ and valuation $\nu \in \mathbb{R}^{|\mathcal{X}|}$, we write $\nu[X{:=}0]$ for the valuation where $\nu[X{:=}0](x) = 0$ if $x \in X$, and $\nu[X{:=}0](x) = \nu(x)$ otherwise. For $t \in \mathbb{R}$, write $\nu + t$ for the valuation defined by $\nu(x) + t$ for all $x \in \mathcal{X}$. The valuation $\mathbf{0} \in \mathbb{R}^{|\mathcal{X}|}$ is a special valuation such that $\mathbf{0}(x) = 0$ for all $x \in \mathcal{X}$. For $a, b \in \mathbb{N}$ and $a < b$, the set $\mathcal{I}$ of time intervals is defined by $\mathcal{I} := [a, b] \mid [a, a] \mid (a, b] \mid [a, b) \mid (a, b) \mid [a, \infty) \mid (a, \infty)$. The set of clock constraints, denoted $\varphi(\mathcal{X})$, is the set of Boolean formulae over $\{x \in I \mid x \in \mathcal{X}, I \in \mathcal{I}\}$. For a constraint $g \in \varphi(\mathcal{X})$, and a valuation $\nu \in \mathbb{R}^{|\mathcal{X}|}$, we write $\nu \models g$ to represent the fact that valuation ν satisfies the constraint g. For example, $(1.1, 0, 10) \models (x_1 \in (0, 2)) \wedge (x_2 \in [0, 0]) \wedge (x_3 \in (1, \infty))$.

Timed Automata

Let *Act* denote a finite set called actions. A timed automaton (TA) [2] is a tuple $\mathcal{A} = (L, L^0, Act, \mathcal{X}, E)$ such that (i) L is a finite set of locations, (ii) $\mathcal{X}$ is a finite set of clocks, (iii) *Act* is a finite alphabet called an action set, (iv) $E \subseteq L \times \varphi(\mathcal{X}) \times Act \times 2^{\mathcal{X}} \times L$ is a finite set of transitions, and (v) $L^0 \subseteq L$ is the set of initial locations. A state s of a timed automaton is a pair $s = (\ell, \nu) \in L \times \mathbb{R}^{|\mathcal{X}|}$. A time elapse transition from $s = (\ell, \nu)$ to $s' = (\ell', \nu')$ denoted $s \xrightarrow{t} s'$ is defined iff $\ell' = \ell$ and $\nu' = \nu + t$. Given $e = (\ell, g, a, Y, \ell') \in E$, a discrete transition from s to s' on e is written as $s \xrightarrow{e} s'$, such that $\nu \models g$ and $\nu' = \nu[Y{:=}0]$. A run is a finite sequence $\rho = s_0 \xrightarrow{t_1} s'_0 \xrightarrow{e_1} s_1 \xrightarrow{t_2} s'_1 \xrightarrow{e_2} s_2 \ldots s_{n-1} \xrightarrow{t_n} s'_{n-1} \xrightarrow{e_n} s_n$ of states with alternating time elapse transitions and discrete transitions.

Multisets or Bags

A *multiset* or *bag* over an alphabet Σ is a mapping $M : \Sigma \mapsto \mathbb{N}$. For an element $a \in \Sigma$, we use $a \in M$ to denote that $M(a) \geq 1$. We use $\emptyset$ to denote the empty multiset. Given two multisets M_1, M_2 over Σ, we write $M_1 \leq M_2$ iff $M_1(a) \leq M_2(a)$ for all $a \in \Sigma$. $M_1 + M_2$ denotes the multiset M such that $M(a) = M_1(a) + M_2(a)$ for all $a \in \Sigma$. Likewise, $M_1 - M_2$

denotes, when it is defined (i.e., $M_1 \geq M_2$), the multiset M such that $M(a) = M_1(a) - M_2(a)$ for all $a \in \Sigma$. The notation $M_1 + a$ denotes a multiset M_2 such that $M_2(a) = M_1(a) + 1$ and $M_2(b) = M_1(b)$ for all $b \neq a$. Likewise, $M_1 - a$ denotes, when it is defined (i.e. $M_1(a) \geq 1$), a multiset M_2 such that $M_2(a) = M_1(a) - 1$ and $M_2(b) = M_1(b)$ for all $b \neq a$. The terms multiset and bag will be used interchangeably.

Timed Petri Nets

A Timed Petri Net (TPN) [17] is a tuple $\mathcal{N} = (P, T, F, c)$ where P is a finite set of places, T is a finite set of transitions, $T \cap P = \emptyset$ and $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation, $c : F \cap (P \times T) \to \mathcal{I}$ is a time constraint relation assigning a time interval to every arc from a place to a transition. A marking M of $\mathcal{N}$ is a mapping that associates to each place p a multiset over $\mathbb{R}$. A marked TPN is a pair $(\mathcal{N}, M_0)$ where M_0 is an initial marking, which assigns to each place in P, an initial multiset of tokens annotated with 0 (the initial age). The dynamics of a TPN consists of two types of transitions rules: firing of a transition and time elapsing. Given $\mathcal{N}$, along with a marking M, (denoted $(\mathcal{N}, M)$) a transition t is enabled at M iff for all places p such that $(p, t) \in F$, there exists some $x \in M(p)$, and $x \in c(p, t)$. If t is enabled by M, then it can be fired, producing a marking M' obtained from M by (i) removing a token from $M(p)$ for all places p such that $(p, t) \in F$ and whose age satisfies $c(p, t)$, and (ii) adding a token with age 0 to $M(q)$ for all places q such that $(t, q) \in F$. In a time elapse transition, with an elapsing time $r \in \mathbb{R}$, the age of all tokens increases by r. A marked TPN $(\mathcal{N}, M_0)$ induces a transition system with states are the markings of $\mathcal{N}$, and the transition relation consists of time elapsing and firing transitions.

A read arc in a TPN facilitates firing a transition without removing the token. We use $F* \subseteq P \times T$ to denote the set of read arcs and $c* : F* \to \mathcal{I}$ to denote a function that assigns a time interval to each read arc. A transition t is enabled iff for all places p such that $(p, t)* \in F*$, there exists some $x \in M(p)$ and $x \in c*(p, t)$. The transition system induced by a marked TPN with read arcs can be defined in a similar manner as for marked TPN. A 1-safe marking is one where $|M(p)| \leq 1$ for all $p \in P$. A 1-safe TPN is a marked TPN $(\mathcal{N}, M_0)$, with $F \cap F* = \emptyset$, where all markings which are reachable from M_0 are 1-safe.

Coverability problem. For markings M_1 and M_2 in a TPN $\mathcal{N}$, define $M_1 \leq M_2$ iff for all $p \in P$, $M_1(p) \leq M_2(p)$. The coverability problem for $\mathcal{N}$ asks whether, given a marking M, it is possible to reach a marking M' in $\mathcal{N}$ from the initial marking M_0 such that $M \leq M'$.

3 Multiset Timed Automata

Let $\mathcal{T}=\{T_1, \ldots, T_N\}$ be a set consisting of $N \geq 1$ timed automata $T_i=(L_i, L_i^0, Act_i, \mathcal{X}_i, E_i)$. A Multiset Timed Automata (MTA) is defined as $\mathcal{M} = (\Sigma, \mathcal{T}, \mathcal{X}, St)$, where Σ is a finite alphabet called *tasks*, $\mathcal{X} = \biguplus_{i=1}^{N} \mathcal{X}_i$ is the finite disjoint union of clocks in T_i, St is a function that assigns a finite multiset $St(i)$ over Σ (possibly empty) to the timed automaton T_i. This is the initial set of tasks assigned to T_i. The actions Act_i are defined as $Act_i = \{i!j(a[d]), i?a \mid a \in \Sigma, j \in \{1, \ldots, N\}, d \in \mathbb{N} \cup \{\infty\}\} \cup \{\mathsf{nop}_i\}$. The number d is the *deadline* for the task a. The action $i!j(a[d])$ represents T_i adding the task a to the bag of automaton T_j, and the task a has an associated deadline d. Likewise, the action $i?a$ represents automaton T_i picking up the task a from its bag, provided its age has not exceeded the deadline. For readability reasons, we assume that any outgoing transition from any initial location is labeled by an action of the form $i?a$. We use the notation N-MTA whenever we need to clarify the number of timed automata T_i which are used in the definition.

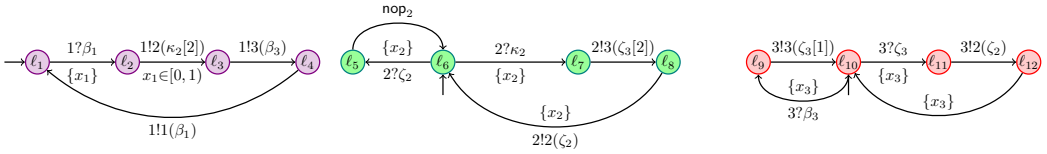


Figure 1 A stateless and time independent 3-MTA consisting of timed automata T_1, T_2, T_3 from left to right. When the deadline of a task is ∞, we do not mention it.

Let $\overline{q} = (q_1, \ldots, q_N)$ be a tuple of states, where $q_i = (s_i, \nu_i)$ is the current state of T_i. Let $\overline{m} = (M_1, \ldots, M_N)$ be a tuple of multisets. Each element in M_i has the form $\alpha = (a, r, d) \in \Sigma \times \mathbb{R} \times \mathbb{N}$ consisting of – *pending tasks*, their *ages*, and their *deadlines* in T_i. The age of a task in a bag is the time elapse since it has been added to the bag. For $t \in \mathbb{R}$, let $\overline{q}+t$ represent the tuple $(q'_1, \ldots, q'_n)$ where $q'_i = (s_i, \nu_i + t)$. For an element $\alpha = (a, r, d) \in M_i$, $\alpha + t = (a, r + t, d)$; $M_i + t$ is the multiset obtained by replacing each $\alpha \in M_i$ with $\alpha + t$. We define $\overline{m} + t$ as the tuple $(M_1 + t, \ldots, M_N + t)$.

A configuration $\mathfrak{c}$ of an N-MTA is the tuple $(\overline{q}, \overline{m})$ consisting of the current states of all the N timed automata, along with the multisets of pending tasks corresponding to each T_i. An initial configuration is defined as $\mathfrak{c}_0 = (\overline{q}_0, \overline{m}_0)$, where $\overline{q}_0$ is the tuple $((\ell_1^0, \mathbf{0}), \ldots, (\ell_N^0, \mathbf{0}))$ of initial states of all T_i ($\ell_i^0 \in L_i^0$) and $\overline{m}_0 = (M_1, \ldots, M_N)$ where $M_i((a, r, d)) = St(i)(a)$, for all $a \in \Sigma$, $r = 0$ and $d = \infty$, and $M_i((a, r, d)) = 0$ otherwise. Given two configurations $\mathfrak{c} = (\overline{q}, \overline{m})$, and $\mathfrak{c}' = (\overline{q'}, \overline{m'})$, we have:

- For $t \in \mathbb{R}$, $\mathfrak{c} \xrightarrow{t} \mathfrak{c}'$ is a time elapse transition iff $\overline{q'} = \overline{q} + t, \overline{m'} = \overline{m} + t$.
- Let $e_i = (\ell_i, g_i, act_i, Y_i, \ell'_i) \in E_i$. Then, $\mathfrak{c} \xrightarrow{e_i} \mathfrak{c}'$ iff
  - $q_i = (\ell_i, \nu_i)$, $\nu_i \models g_i$, $q'_i = (\ell'_i, \nu'_i)$, $\nu'_i = \nu_i[Y_i := 0]$, and for all $k \neq i$, $q'_k = q_k$, and,
  - If $act_i = i!j(a[d])$, then $M'_j = M_j + (a, 0, d)$, and $M'_k = M_k$ for all $k \neq j$,
  - If $act_i = i?a$, then $\exists c, d$, such that $(a, c, d) \in M_i$, $M'_i = M_i - (a, c, d)$, and $c \leq d$ (i.e. the age of the task has not yet exceeded the deadline) and $M'_k = M_k$ for all $k \neq i$,
  - If $act_i = \mathsf{nop}_i$, then $M'_k = M_k$ for all $1 \leq k \leq N$.

Starting with an initial configuration $\mathfrak{c}_0$, a run ρ is defined as a finite sequence of alternating time elapse and discrete transitions of the form $\mathfrak{c}_0 \xrightarrow{t_0} \mathfrak{c}'_0 \xrightarrow{e_1} \mathfrak{c}_1 \xrightarrow{t_1} \mathfrak{c}'_1 \xrightarrow{e_2} \mathfrak{c}_2 \cdots \xrightarrow{e_j} \mathfrak{c}_j$ or $\mathfrak{c}_0 \xrightarrow{t_0} \mathfrak{c}'_0 \xrightarrow{e_1} \mathfrak{c}_1 \xrightarrow{t_1} \mathfrak{c}'_1 \xrightarrow{e_2} \mathfrak{c}_2 \cdots \xrightarrow{t_j} \mathfrak{c}'_j$. In that case we say the configuration $\mathfrak{c}_j$ is reachable from the initial configuration $\mathfrak{c}_0$ by the run ρ.

In this paper, we consider the following problems. Let $\overline{s} = (s_1, \ldots, s_N) \in L_1 \times \cdots \times L_N$.

P1 Control State Reachability. Given a particular tuple of locations $\overline{s} = (s_1, \ldots, s_N)$ of an N-MTA $\mathcal{M}$, the control state reachability problem asks if starting from the initial configuration $\mathfrak{c}_0$ of $\mathcal{M}$, there is a run reaching a configuration $\mathfrak{c} = (\overline{q}, \overline{m})$ such that $q_i = (s_i, \nu_i)$ for some $\overline{m}$, and for some ν_i, for all $1 \leq i \leq N$.

P2 Configuration Reachability. Given a particular tuple of locations $\overline{s} = (s_1, \ldots, s_N)$ of an N-MTA $\mathcal{M}$, the configuration reachability problem asks if starting from the initial configuration $\mathfrak{c}_0$ of $\mathcal{M}$, there is a run reaching a configuration $\mathfrak{c} = (\overline{q}, \overline{m})$ such that $\overline{m} = (\emptyset, \ldots, \emptyset)$ and $q_i = (s_i, \mathbf{0})$, for all $1 \leq i \leq N$.

Stateless and Time-Independent MTA

An N-MTA is said to be *stateless* if $E_i \cap (L_i \backslash \{\ell_i^0\} \times \varphi(\mathcal{X}_i) \times \{i?a | a \in \Sigma\} \times 2^{\mathcal{X}_i} \times L_i) = \emptyset$ for all $1 \leq i \leq N$, and some $\ell_0^i \in L_0^i$. The stateless condition ensures that a new task can be picked by an automaton only from a unique initial location. An N-MTA is said to be

time-independent iff, in each T_i, all clocks are reset on picking a task from the multiset, and no clock constraints are checked (i.e. $E_i \cap (L_i \times \varphi(\mathcal{X}_i) \times \{i?a | a \in \Sigma\} \times (2^{\mathcal{X}_i} \setminus \{\mathcal{X}_i\}) \times L_i) = \emptyset$ and $E_i \cap (L_i \times (\varphi(\mathcal{X}_i) \setminus \{true\}) \times \{i?a | a \in \Sigma\} \times 2^{\mathcal{X}_i} \times L_i) = \emptyset$ for all $1 \leq i \leq N$).

Figure 1 describes a stateless and time-independent MTA $\mathcal{M}$ consisting of 3 timed automata T_1, T_2, T_3. The following is a run in $\mathcal{M}$. The initial configuration $\mathfrak{c}_0 = (\overline{q}_0, \overline{m}_0)$ where $\overline{q}_0 = ((\ell_1, 0), (\ell_6, 0), (\ell_{10}, 0))$ and $\overline{m}_0 = (M_1, M_2, M_3)$ with multisets $M_1 = \{(\beta_1, 0, \infty)\}$, $M_2 = \{(\beta_2, 0, \infty)\}$, and $M_3 = \{(\beta_3, 0, \infty)\}$. Let $e_{i,j}$ denote the transition from location ℓ_i to ℓ_j (in the example, we have at most one transition between any pair of locations ℓ_i, ℓ_j). For example, $e_{23} = (\ell_2, x_1 \in [0,1), 1!2(\kappa_2[2]), \emptyset, \ell_3)$. Consider the run σ $\mathfrak{c}_0 \xrightarrow{0.5} \mathfrak{c}'_0 \xrightarrow{e_{1,2}} \mathfrak{c}_1 \xrightarrow{0.3} \mathfrak{c}'_1 \xrightarrow{e_{2,3}} \mathfrak{c}_2 \xrightarrow{0.5} \mathfrak{c}'_2 \xrightarrow{e_{6,7}} \mathfrak{c}_3 \xrightarrow{0.2} \mathfrak{c}'_3 \xrightarrow{e_{7,8}} \mathfrak{c}_4 \xrightarrow{0} \mathfrak{c}'_4 \xrightarrow{e_{10,9}} \mathfrak{c}_5 \xrightarrow{0.4} \mathfrak{c}'_5 \xrightarrow{e_{3,4}} \mathfrak{c}_6 \xrightarrow{0.1} \mathfrak{c}'_6 \xrightarrow{e_{4,1}} \mathfrak{c}_7 \xrightarrow{0.1} \mathfrak{c}'_7 \xrightarrow{e_{1,2}} \mathfrak{c}_8 \xrightarrow{0.1} \mathfrak{c}'_8 \xrightarrow{e_{8,6}} \mathfrak{c}_9 \xrightarrow{0.2} \mathfrak{c}'_9 \xrightarrow{e_{9,10}} \mathfrak{c}_{10} \xrightarrow{0.6} \mathfrak{c}'_{10} \xrightarrow{e_{6,5}} \mathfrak{c}_{11} \xrightarrow{0.5} \mathfrak{c}'_{11} \xrightarrow{e_{10,11}} \mathfrak{c}_{12} \xrightarrow{0.9} \mathfrak{c}'_{12} \xrightarrow{e_{11,12}} \mathfrak{c}_{13}$ which reaches locations $(\ell_2, \ell_5, \ell_{12})$ in T_1, T_2, T_3 respectively.

4 Control State Reachability

In the following, we first prove that the control reachability is decidable with a non-primitive complexity (at the level $F_{\omega^{\omega^\omega}}$ in the fast growing hierarchy [9]). Then, we show that the control state reachability for stateless and time independent MTA is PSPACE-complete.

▶ **Theorem 1.** *The control state reachability problem for N-MTA is reducible to the coverability problem for timed Petri nets with read-arcs.*

Proof. We give a translation from an N-MTA $\mathcal{M}$ to a TPN with read arcs $\mathcal{N}$ such that the control state reachability of $\mathcal{M}$ reduces to the coverability of $\mathcal{N}$.

Let $\mathcal{M} = (\Sigma, \mathcal{T}, \mathcal{X}, St)$ be an N-MTA. Without loss of generality, assume that we are interested in reaching $\overline{f} = (f_1, \ldots, f_N) \in L_1 \times \cdots \times L_N$. Given the N-MTA $\mathcal{M}$, we construct a timed Petri net $\mathcal{N}$ as follows. There is a place p_ℓ in the net corresponding to each location $\ell \in L_i$ in T_i for each $i \in \{1, \ldots, N\}$. For each T_i, there is one and only one marked place p_ℓ such that $\ell \in L_i$, to denote that the control of T_i is at a certain location ℓ. For each clock x in $\mathcal{X}$, we have a place p_x in the net. Next, we model the multisets M_i of each T_i. Let $d_{max} \in \mathbb{N}$ be the maximal value used for any deadline in $\mathcal{M}$. The possible task, deadline combinations are in the set $\Sigma \times \{0, 1, \ldots, d_{max}, \infty\}$. Therefore, corresponding to each T_i, we have $|\Sigma| \times (d_{max} + 2)$ places in the net. We need to have these many places so as to distinguish between the tokens. Thus, for each pair $(a, d) \in \Sigma \times \{0, 1, \ldots, d_{max}, \infty\}$, we have the places $p^1_{(a,d)}, \ldots, p^N_{(a,d)}$.

A transition of the form $(\ell, g, i?a, Y, \ell')$ in automaton T_i is simulated by a transition in $\mathcal{N}$ as follows. A token from the place p_ℓ corresponding to the location ℓ, and a token from one of the places $p^i_{(a,d)}, d \in \{0, 1, \ldots, d_{max}, \infty\}$ are removed. The deadline is checked on the arc via a constraint $[0, z]$ from the place $p^i_{(a,z)}$ containing the token. A token is added to the place $p_{\ell'}$ corresponding to ℓ'. A transition of the form $(\ell, g, i!j(a[d]), Y, \ell')$ in automaton T_i is simulated in a similar way. The tokens corresponding to locations are removed and added as in the previous case and a token is added to the place $p^j_{(a,d)}$. The clock constraints corresponding to any transition are checked using read arcs from the places simulating the clocks. Clock resets are simulated by removing a token and putting back a token in the place corresponding to the clock.

The details of the formal construction of $\mathcal{N}$ and the correctness proof can be found in the extended version of the paper [1]. ◀

As a corollary of Theorem 1, we get:

▶ **Corollary 2.** *The control state reachability problem for (time-independent) N-MTA is decidable.*

Observe that we can easily show that the coverability of Petri nets is reducible to the control state reachability problem for time-independent (resp. stateless) N-MTA (in the same way as the proof of EXPSPACE lower bound for the model multiset pushdown systems presented in [16]). Therefore, the control state reachability problem for time-independent (resp. stateless) N-MTA is EXPSPACE-hard.

In the rest of this section, we consider the case of stateless and time-independent N-MTA.

▶ **Theorem 3.** *The control state reachability problem for stateless and time-independent N-MTA is PSPACE-complete (for $N \geq 1$).*

Proof. Since MTA subsume timed automata [2], the PSPACE-hardness of the control state reachability of MTA follows directly from the PSPACE-hardness of reachability of timed automata. The rest of the proof is devoted to proving the PSPACE-membership of the problem.

Let $\mathcal{M} = (\Sigma, \mathcal{T}, \mathcal{X}, St)$ be a stateless, time-independent N-MTA, with $\mathcal{T} = \{T_1, \ldots, T_N\}$ $\mathcal{X} = \biguplus_{i=1}^{N} \mathcal{X}_i$ and St, the function that assigns an initial multiset $St(i)$ to each timed automaton T_i. Incurring a polynomial blowup in the size, we give a reduction from the control state reachability of $\mathcal{M}$ to the coverability in 1-safe timed Petri net with read arcs. The coverability of 1-safe timed Petri nets with read arcs is known to be PSPACE-complete [17] and our result follows from this.

Without loss of generality, assume that we are interested in reaching $\overline{f} = (f_1, \ldots, f_N) \in L_1 \times \cdots \times L_N$. Let σ be any run from the initial configuration $\mathfrak{c}_0$ of $\mathcal{M}$ which leads into a configuration with locations $\overline{f}$. Let $\mathfrak{c} = (\overline{q}, \overline{m})$ be any configuration that appears in σ. Our proof is divided into two parts.

1. We show that the number of *relevant* task tuples along σ is bounded by N. Intuitively, A task tuple $(a, r, d) \in \Sigma \times \mathbb{R} \times \mathbb{N}$ is relevant for an automaton T_i if $(a, r, d) \in M_j$, for some j, $(r \leq d)$ and the task (a, r, d) must be executed by T_j in order to reach the location f_i. The irrelevant task tuples can hence be ignored from each M_i, as they do not affect the control state reachability.
2. The bound on the number of relevant task tuples obtained in the previous step is used in constructing a reachability preserving 1-safe timed Petri net with read arcs.

Bounding the number of relevant task tuples

Consider the run σ as described above. Starting from $\mathfrak{c}_0$, let σ_i denote the sequence of transitions (in the order they appear in σ), pertaining only to T_i. In the run σ pertaining to the example in figure 1, σ_1 consists of all the violet discrete transitions separated by time elapses: $\mathfrak{c}_0 \xrightarrow{0.5} \mathfrak{c}'_0 \xrightarrow{e_{1,2}} \mathfrak{c}_1 \xrightarrow{0.3} \mathfrak{c}'_1 \xrightarrow{e_{2,3}} \mathfrak{c}_2 \xrightarrow{1.1} \mathfrak{c}'_5 \xrightarrow{e_{3,4}} \mathfrak{c}_6 \xrightarrow{0.1} \mathfrak{c}'_6 \xrightarrow{e_{4,1}} \mathfrak{c}_7 \xrightarrow{0.1} \mathfrak{c}'_7 \xrightarrow{e_{1,2}} \mathfrak{c}_8$. We now define a block.

A *block* in σ_i begins with a discrete transition of the form $\xrightarrow{i?a}$ (for some task a) and extends until the next transition of the form $\xrightarrow{i?b}$ (for some task b) is encountered. Thus, a block is a sequence of transitions between two executions of tasks by some T_i, and begins with some task execution. σ_1 has two blocks: the sequence of transitions from $\mathfrak{c}'_0$ till $\mathfrak{c}'_7$ forms a block, and the second block is the transition from $\mathfrak{c}'_7$ to $\mathfrak{c}_8$. Omitting the time elapse transitions in σ_i, let us label each transition in σ_i with a unique name. Doing this for all σ_i gives us a unique label for each discrete transition in σ. Let $\mathcal{L} = \{\alpha_1, \ldots, \alpha_m\}$ be the set of block labels occurring in σ. In our running example, using labels $\{\alpha_1, \ldots, \alpha_6\}$, we can label the blocks in σ as $\mathfrak{c}_0 \xrightarrow{0.5} \mathfrak{c}'_0 \xrightarrow{e_{1,2},\alpha_1} \mathfrak{c}_1 \xrightarrow{0.3} \mathfrak{c}'_1 \xrightarrow{e_{2,3},\alpha_1} \mathfrak{c}_2 \xrightarrow{0.5} \mathfrak{c}'_2 \xrightarrow{e_{6,7},\alpha_2} \mathfrak{c}_3 \xrightarrow{0.2} \mathfrak{c}'_3 \xrightarrow{e_{7,8},\alpha_2} \mathfrak{c}_4 \xrightarrow{0} \mathfrak{c}'_4 \xrightarrow{e_{10,9},\alpha_3}$ $\mathfrak{c}_5 \xrightarrow{0.4} \mathfrak{c}'_5 \xrightarrow{e_{3,4},\alpha_1} \mathfrak{c}_6 \xrightarrow{0.1} \mathfrak{c}'_6 \xrightarrow{e_{4,1},\alpha_1} \mathfrak{c}_7 \xrightarrow{0.1} \mathfrak{c}'_7 \xrightarrow{e_{1,2},\alpha_4} \mathfrak{c}_8 \xrightarrow{0.1} \mathfrak{c}'_8 \xrightarrow{e_{8,6},\alpha_2} \mathfrak{c}_9 \xrightarrow{0.2} \mathfrak{c}'_9 \xrightarrow{e_{9,10},\alpha_3} \mathfrak{c}_{10} \xrightarrow{0.6} \mathfrak{c}'_{10} \xrightarrow{e_{6,5},\alpha_5}$ $\mathfrak{c}_{11} \xrightarrow{0.5} \mathfrak{c}'_{11} \xrightarrow{e_{10,11},\alpha_6} \mathfrak{c}_{12} \xrightarrow{0.9} \mathfrak{c}'_{12} \xrightarrow{e_{11,12},\alpha_6} \mathfrak{c}_{13}$. From here on, we refer to the blocks using the block labels.

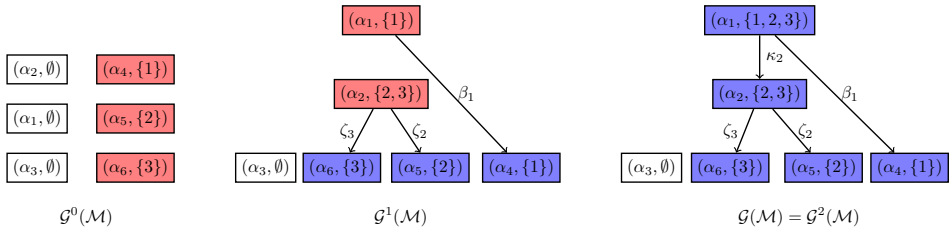


Figure 2 The dependency graph in stages. $\mathcal{G}^0(\mathcal{M})$ is the initial graph with no edges. $\mathcal{G}^{i+1}(\mathcal{M})$ is obtained from $\mathcal{G}^i(\mathcal{M})$ by changing the color of all the red vertices in $\mathcal{G}^i(\mathcal{M})$. The graph stabilizes when there are no red vertices.

For each timed automaton T_i, we now analyze the blocks which contribute in reaching the desired location f_i. The last block α of σ_i, which contains the last task tuple (a, r, d) executed by T_i definitely contributes to T_i reaching f_i. Likewise, the block α' which added this last task a to the bag of T_i also contributes to T_i reaching f_i (note that block α' may start with a task b which is executed by T_j, $j \neq i$). We can continue backwards in this manner and say that the block α'' which added the task b to the bag of T_j also contributes to T_i reaching f_i and so on. Given a block label α, let $\mathsf{dep}(\alpha)$ denote the set of timed automata T_i such that α contributes to T_i reaching f_i. Thus, if α is the last block in T_i, then $i \in \mathsf{dep}(\alpha)$ (we just write the indices i rather than T_i). Likewise, if $i \in \mathsf{dep}(\alpha)$ and if α' is the block which added the task a which was executed at the beginning of α, then $i \in \mathsf{dep}(\alpha')$, and so on. $\mathsf{dep}(\alpha)$ is called the dependency set of α. In our running example above, $3 \in \mathsf{dep}(\alpha_6)$ since α_6 is the last block for T_3; however the task ζ_3 which was executed in block α_6 was added in block α_2 ($e_{7,8}$), and the task κ_2 executed in block α_2 was added in block α_1. Thus, $3 \in \mathsf{dep}(\alpha_6), \mathsf{dep}(\alpha_2), \mathsf{dep}(\alpha_1)$.

We construct a *dependency graph* $\mathcal{G}(\mathcal{M})$ which keeps track of the dependencies between blocks. Define a function $g : \mathcal{L} \to (\Sigma \times \{1, \ldots, N\} \times \mathcal{L}) \cup \{\perp\}$ which maps a block label α to the triple (a, i, α') if block α begins with $(i?a)$ the execution of task a, which was added to the bag of T_i by block α'. If a is part of the initial multiset ($a \in St(i)$) then $g(\alpha) = \perp$. The vertex set of $\mathcal{G}(\mathcal{M})$ is the set of pairs $(\alpha, \mathsf{dep}(\alpha))$ where α is a block label and $\mathsf{dep}(\alpha)$ is its dependency set. $\mathcal{G}(\mathcal{M})$ is a graph with colored vertices, and is built inductively. To begin, there are no edges, and we have the following vertices.

- Vertices $(\alpha, \{i\})$ and α is the last block of T_i. To begin, we are sure of $i \in \mathsf{dep}(\alpha)$. We color these vertices red.
- Vertices $(\alpha, \emptyset)$, and α is not the last block for any T_i. To begin, we have not yet discovered whether α contributes to any T_i, so we keep $\mathsf{dep}(\alpha) = \emptyset$. The information with respect to $\mathsf{dep}(\alpha)$ will be updated when we discover that α contributed to some T_i. We color these vertices white.

To add the edges, we repeat the following procedure until no red vertices remain. In each step, we choose a red vertex $(\alpha, \mathsf{dep}(\alpha))$ and do the following.

1. If $g(\alpha) = \perp$, then color $(\alpha, \mathsf{dep}(\alpha))$ blue,
2. If $g(\alpha) = (a, i, \alpha')$ and $(\alpha', \mathsf{dep}(\alpha'))$ is white, then color $(\alpha, \mathsf{dep}(\alpha))$ blue and color $(\alpha', \mathsf{dep}(\alpha'))$ red. Update $\mathsf{dep}(\alpha')$ to be $\mathsf{dep}(\alpha') \cup \mathsf{dep}(\alpha)$, and add an edge $\xrightarrow{a}$ from $(\alpha', \mathsf{dep}(\alpha'))$ to $(\alpha, \mathsf{dep}(\alpha))$.
3. If $g(\alpha) = (a, i, \alpha')$ and $(\alpha', \mathsf{dep}(\alpha'))$ is not white, then color $(\alpha, \mathsf{dep}(\alpha))$ blue, update $\mathsf{dep}(\alpha')$ to be $\mathsf{dep}(\alpha') \cup \mathsf{dep}(\alpha)$, and add an edge $\xrightarrow{a}$ from $(\alpha', \mathsf{dep}(\alpha'))$ to $(\alpha, \mathsf{dep}(\alpha))$.

Finally, we update the dependency relation dep of the vertices as follows: If $(\alpha, \mathsf{dep}(\alpha))$ and $(\alpha', \mathsf{dep}(\alpha'))$ are blue with $g(\alpha) = (a, i, \alpha')$, then update $\mathsf{dep}(\alpha')$ to be $\mathsf{dep}(\alpha') \cup \mathsf{dep}(\alpha)$. Note that the above procedure terminates, since the number of blue vertices in each step increases. The final graph obtained as result is $\mathcal{G}(\mathcal{M})$. Figure 2 describes constructing $\mathcal{G}(\mathcal{M})$ for the run ρ discussed above. Consider any vertex $(\alpha, \mathsf{dep}(\alpha))$ colored blue in $\mathcal{G}(\mathcal{M})$. Clearly, this vertex contributes to all T_i such that $i \in \mathsf{dep}(\alpha)$. Consider any path in $\mathcal{G}(\mathcal{M})$ from a vertex with no incoming edges to a vertex with no outgoing edges. There is at least one such path since the last task executed along σ corresponds to the last block of some T_i which has not contributed to any T_j. A path $v_1 \dots v_s$ in $\mathcal{G}(\mathcal{M})$ is a dependency path for automaton T_i if the vertex $v_s = (\alpha, \mathsf{dep}(\alpha))$, and α is the last block for T_i. Let us go back to our running example run σ using Figure 2. The tasks appearing on the edges of $\mathcal{G}(\mathcal{M})$ are the relevant tasks. From $\mathcal{G}(\mathcal{M})$, the relevant task in the bags when the second block α_2 started is κ_2. κ_2 is executed at the beginning of block α_2. Relevant tasks ζ_2, ζ_3 are added to the bag in block α_2, and α_1 adds β_1. β_1 is executed in block α_4 while ζ_2, ζ_3 respectively are executed in blocks α_5, α_6. The relevant tasks along run σ are $\beta_1, \kappa_2, \zeta_2, \zeta_3$, of which at most 3 are stored across bags at any point of time. Thus, we can obtain another run σ' which is reachability equivalent to σ as follows. The block α_3 is useless as it is not contributing to any of the automata. Each block begins at a unique initial location of some automaton, and, on the transition which executes the task, it does not check any constraints, and resets all clocks on the transition. Due to this, we can "prune away" a block from a run, and reconnect the run at a later block if we maintain the time elapse in the interim. Hence, removing a useless block of some automaton T_i does not affect the control reachability, since the next useful block of T_i again starts from the same initial location of T_i. Accounting for the time elapse in the useless block is sufficient to ensure that the ages of the pending tasks are accurate. σ' can be constructed with the rest of the blocks, using only the relevant tasks: $\mathfrak{c}_0 \xrightarrow{0.5} \mathfrak{c}'_0 \xrightarrow{e_{1,2},\alpha_1} \mathfrak{c}_1 \xrightarrow{0.3} \mathfrak{c}'_1 \xrightarrow{e_{2,3},\alpha_1} \mathfrak{c}_2 \xrightarrow{0.5} \mathfrak{c}'_2 \xrightarrow{e_{6,7},\alpha_2} \mathfrak{c}_3 \xrightarrow{0.2} \mathfrak{c}'_3 \xrightarrow{e_{7,8},\alpha_2} \mathfrak{c}_4 \xrightarrow{0} \mathfrak{c}'_4 \xrightarrow{0.4} \mathfrak{c}'_5 \xrightarrow{e_{3,4},\alpha_1} \mathfrak{c}_6 \xrightarrow{0.1} \mathfrak{c}'_6 \xrightarrow{e_{4,1},\alpha_1} \mathfrak{c}_7 \xrightarrow{0.1} \mathfrak{c}'_7 \xrightarrow{e_{1,2},\alpha_4} \mathfrak{c}_8 \xrightarrow{0.1} \mathfrak{c}'_8 \xrightarrow{e_{8,6},\alpha_2} \mathfrak{c}_9 \xrightarrow{0.2} \mathfrak{c}'_9 \xrightarrow{0.6} \mathfrak{c}'_{10} \xrightarrow{e_{6,5},\alpha_5} \mathfrak{c}_{11} \xrightarrow{0.5} \mathfrak{c}'_{11} \xrightarrow{e_{10,11},\alpha_6} \mathfrak{c}_{12} \xrightarrow{0.9} \mathfrak{c}'_{12} \xrightarrow{e_{11,12},\alpha_6} \mathfrak{c}_{13}$.

We want to prove that in any configuration $\mathfrak{c} = (\overline{q}, \overline{m})$ appearing in the run σ, the number of pending tasks maintained in $\overline{m} = (M_1, \dots, M_N)$ which contribute, in reaching the desired control states, in σ is $\leq N$. These are the relevant tasks, and each one is part of a block α, and the corresponding vertex $(\alpha, \mathsf{dep}(\alpha))$ in $\mathcal{G}(\mathcal{M})$ is colored blue. If we attach the color of the vertex $(\alpha, \mathsf{dep}(\alpha))$ to the task a in $g(\alpha)$, then we want to prove that in any configuration appearing in σ, the number of blue tasks is $\leq N$. Assume that there is some configuration $\mathfrak{c} = (\overline{q}, \overline{m})$ in σ such that the number of blue tasks in $\overline{m}$ is $p > N$. Let $a_1, \dots, a_p$ be the tasks in $\overline{m}$, and let $\alpha_1, \dots, \alpha_p$ be the blocks where these are executed. Since $p > N$, and there are only N multisets in $\overline{m}$, there are at least two tasks a_i, a_j such that $\mathsf{dep}(\alpha_i) \cap \mathsf{dep}(\alpha_j) \neq \emptyset$. Observe that, by definition, we have $\mathsf{dep}(\alpha_k) \neq \emptyset$ for all $k \in \{1, \dots, p\}$. Let us assume that $k \in \mathsf{dep}(\alpha_i) \cap \mathsf{dep}(\alpha_j)$. Since both are blue, both get executed in σ, and both lie in the dependency path of the last block of the automaton T_k. Clearly, one must come before the other, and the earlier block has contributed to the creation of the later block. Hence, they cannot be pending at the same time. Thus, the number of blue pending tasks in any configuration is bounded above by N.

Construction of 1-safe TPN with read arcs

Now, we are ready to propose a 1-safe timed Petri net (with read-arcs) whose coverability problem is equivalent to the control state reachability problem of the given N-MTA.

Given the N-MTA $\mathcal{M}$ consisting of timed automata $T_1, \ldots, T_N$, we construct a 1-safe TPN $\mathcal{N}$. There is a place p_ℓ corresponding to each location $\ell \in L_i$ in T_i. For each T_i, there is one and only one marked place p_ℓ at any point in the execution, such that $\ell \in L_i$, to denote that the control of T_i is at a certain location ℓ. For each clock x in X, there is a place p_x. Next, we model the multisets M_i of each T_i. Let $d_{max} \in \mathbb{N}$ be the maximal value used for any deadline in $\mathcal{M}$. For each task $a \in \Sigma$, we have $|\Sigma| \times (2 + d_{max})$ possible combinations of tasks and associated deadlines. The bound established above tells us that there are at most N pending tasks in any configuration i.e. at any point we will have to keep track of N tasks but they can be distributed in any of the multisets. There are $|\Sigma| \times (d_{max} + 2)$ possibilities for task, deadline pairs. Tasks will be modeled as tokens in the net. So to be able to distinguish between them, for each T_i, we need $N \times |\Sigma| \times (d_{max} + 2)$ places (N, because 1-safe). For each T_i and for each pair $(a, d) \in \Sigma \times \{0, 1, \ldots, d_{max}, \infty\}$, we have N places $p^i_{(a,d,1)}, \ldots, p^i_{(a,d,N)}$.

A transition of the form $(\ell^0_i, g, i?a, Y, \ell')$ in automaton T_i is simulated by $N \times (d_{max} + 2)$ transitions in $\mathcal{N}$ as follows. For each $(z, j) \in \{0, 1, \ldots, d_{max}, \infty\} \times \{1, \ldots, N\}$, a transition removes a token from the place $p_{\ell^0_i}$ corresponding to the unique initial location ℓ^0_i, a token from $p^i_{(a,z,j)}$ and adds a token to the place $p_{\ell'}$ corresponding to ℓ'. The deadline is checked on the arc from the place $p^i_{(a,z,j)}$ by a constraint which checks the age of the token to be in the interval $[0, z]$. As any deadline value is possible, and any of the N places can be filled, one of the $N \times (d_{max} + 2)$ transitions is non-deterministically chosen.

A transition of the form $(\ell, g, i!j(a[d]), Y, \ell')$ in automaton T_i is simulated in a similar way by $N + 1$ transitions. In each of the N of these transitions, tokens for control locations are added and removed as in the previous case. For each $k \in \{1, \ldots, N\}$, one of the N transitions adds a token to the place $p^j_{a,d,k}$ if it is empty. The $(N+1)$-th transition simulates the possibility that the task a is not relevant (only N are relevant at any point) and so it simulates only the change in control location and adds no other tokens. One of these $N + 1$ transitions is chosen non-deterministically. Observe that the first N transitions add a token only to an empty place $p^j_{a,d,k}$ by definition of an 1-safe Petri net.

Clock resets are simulated by adding and removing a token from the corresponding place p_x for the clock. Clock constraints are simulated by read arcs. These arcs are connected with the corresponding transitions that are described above.

The formal construction is in [1]. Thus, the control state reachability in $\mathcal{M}$ to reach $(f_1, \ldots, f_N) \in L_1 \times \cdots \times L_N$ reduces to the coverability problem of the marking M given by $M(p_{f_i}) = 1$ for all $1 \leq i \leq N$ (and hence $M(p_\ell) = 0$ for all $\ell \notin \{f_1 \ldots, f_N\}$). The control state reachability of $\mathcal{M}$ thus reduces to the coverability of the constructed 1-safe timed Petri net with read arcs. Since the coverability of 1-safe timed Petri nets with read arcs is PSPACE-complete [17], the control state reachability of $\mathcal{M}$ is also PSPACE-complete. ◀

5 Configuration Reachability

In this section, we explore the general question of the configuration reachability problem for N-MTA. We first show (theorem 4) that the configuration reachability problem for N-MTA is undecidable.

▶ **Theorem 4.** *The configuration reachability problem for N-MTA is undecidable. This undecidability holds even in the case of time-independent N-MTA.*

Proof. The proof is done by a reduction from the reachability problem for a 2-counter machine (which is known to be undecidable [15]). The main idea is to construct an 1-MTA whose set of states contains the states of the two counter machine plus some auxiliary states

that are used to simulate the zero tests as we will see later on. The 1-MTA has two types of tasks a and b. The number of pending tasks of type a (resp. b) corresponds to the value of the counter c_1 (resp. c_2). Furthermore, the 1-MTA has one clock x that is used to check that no time elapsed when simulating some transitions of the two counter machine.

To simulate an increment transition of the form $(q, c_1{+}{+}, q')$ (resp.$(q, c_2{+}{+}, q')$) of the two counter machine, the 1-MTA proceeds as follows: first it checks that the value of the clock $x = 0$, then it will change its state from q to q' and finally adds a pending task of type a (resp. b) with zero as its deadline. Observe that we need only one transition to perform all these steps of the simulation of an increment operation.

To simulate a decrement transition of the form $(q, c_1{-}{-}, q')$ (resp. $(q, c_2{-}{-}, q')$) of the two counter machine, the 1-MTA proceeds as follows: first it checks that the value of the clock $x = 0$, then it will change its state from q to q' and finally consumes a pending task of type a (resp. b). Observe that we need only one transition to perform all these steps of the simulation of an increment operation.

To simulate a zero test transition of the form $(q, c_1 == 0, q')$ (resp. $(q, c_2 == 0, q')$) of the two counter machine, the 1-MTA proceeds as follows: (i) it checks that the value of the clock $x = 0$, (ii) it enters to a loop where it consumes a task of type b (resp. a) and creates a task of the same type but its deadline is now set to one time unit, (iii) it will change its state from q to q', (iv) it checks that the value of the clock x is still zero, (v) it checks that one time unit has elapsed (ie., checking whether $x \in [1, 1]$) and resets the clock x, (vi) it enters to a loop where it consumes a task of type b (resp. a) and creates a task of the same type but its deadline is now set to zero, and (vii) it checks that the value of $x = 0$. Here the auxiliary states are needed in the simulation of these steps.

Observe that if the 1-MTA reaches the final state with empty set of pending tasks, then all the simulation of the zero tests are performed correctly. Finally, note that the constructed 1-MTA is time independent. ◀

We now focus on the class of stateless and time-independent N-MTA.

▶ **Theorem 5.** *The configuration reachability problem for stateless and time-independent* N-MTA *is decidable.*

We begin by setting up some notations for the proof.

Well-quasi-orders and Higman's Lemma

Given a set $\mathcal{Q}$, a *quasi-order* on $\mathcal{Q}$ is a reflexive and transitive relation $\preceq \subseteq \mathcal{Q} \times \mathcal{Q}$. An infinite sequence $(q_1, q_2 \dots)$ in $\mathcal{Q}$ is said to be *saturating* if there exists indices $i < j$ such that $q_i \preceq q_j$. A quasi-order $\preceq$ is a *well-quasi-order* (*wqo*) [12] on $\mathcal{Q}$ if every infinite sequence in $\mathcal{Q}$ is saturating. Let $\sqsubseteq$ be a quasi-order on $\mathcal{Q}$. The *induced monotone domination order* $\preceq$ on $\mathcal{Q}^*$, (i.e., the set of finite words over $\mathcal{Q}$) is defined as follows: $a_1 a_2 \dots a_m \preceq b_1 b_2 \dots b_n$ if there exists a strictly increasing function $g : \{1, 2, \dots, m\} \to \{1, 2, \dots, n\}$ such that, for all $1 \leq i \leq m$, $a_i \sqsubseteq b_{g(i)}$. It is well-known by *Higman's Lemma* [10] that if $\sqsubseteq$ is a wqo on $\mathcal{Q}$, then the induced domination order $\preceq$ is also a wqo on $\mathcal{Q}^*$. As an example, let $\Sigma = \{1, 2, \dots, 12\}$ and let $\mathcal{Q}$ be the power set of Σ. Define $\sqsubseteq$ on $\mathcal{Q}$ to be the set inclusion relation. $\sqsubseteq$ is clearly a wqo since $\mathcal{Q}$ is finite. The induced monotone domination order $\preceq$ on $\mathcal{Q}^*$ is the subword order: for example, $\{1, 2\}\{3\}\{5, 6, 7\} \preceq \{1, 2, 9\}\{1\}\{3, 11\}\{12\}\{4, 5, 6, 7\}$.

Encoding Configurations

We have seen in section 3 that a configuration of an N-MTA $\mathcal{M}$ is a tuple $(\overline{q}, \overline{m})$ where $\overline{q}$ is the sequence of states in each T_i, $1 \leq i \leq N$, and $\overline{m}$ is the tuple of multisets $(M_1, \ldots, M_N)$ corresponding to each T_i. Given $(\ell_1, \ldots, \ell_N) \in L_1 \times \cdots \times L_N$, we are interested in finding whether the configuration $\mathfrak{c}_{goal} = (\overline{q}, \overline{m})$ is reachable, where $\overline{q} = (q_1, \ldots, q_N)$, $q_i = (\ell_i, \mathbf{0})$ and $\overline{m} = (\emptyset, \ldots, \emptyset)$. A configuration $\mathfrak{c}$ is called *good* if $\mathfrak{c}_{goal}$ is reachable from $\mathfrak{c}$. A configuration is *bad* if it is not good. Clearly, $\mathfrak{c}_{goal}$ is reachable in $\mathcal{M}$ iff some initial configuration $\mathfrak{c}_0$ is good.

We now construct an equivalence relation on $\mathcal{M}$ by encoding the configurations of $\mathcal{M}$ as words over a certain alphabet. This will enable us to define a wqo on the resulting transition system. Let K be the maximal constant used in the clock constraints and deadlines in $\mathcal{M}$. Let $[K] = \{0, 1, \ldots, K, \infty\}$. Let $\mathsf{reg} = \{r_0, r_1, \ldots, r_{2K}\}$ be a finite set of *regions*, where for $0 \leq i \leq K$, r_{2i} is defined as the singleton $\{i\}$, while r_{2i+1} is defined as the interval $(i, i+1)$ for $0 \leq i \leq K-1$. We also define the region r_{2K+1} as (K, ∞). Let Γ_1 be the set $\mathcal{X} \times \mathsf{reg}$, and let Γ_2 be a multiset over $\{(a, r, j)_i \mid a \in \Sigma, r \in \mathsf{reg}, j \in [K], 1 \leq i \leq N\}$. Let Γ_3 be the set $\mathcal{X} \times r_{2K+1}$, and let Γ_4 be a multiset over $\{(a, r_{2K+1}, j)_i \mid a \in \Sigma, 1 \leq i \leq N, j \in [K]\}$.

Let Υ, Δ respectively be the power sets of $\Gamma_1 \cup \Gamma_2$ and $\Gamma_3 \cup \Gamma_4$. Let $\mathcal{L} = L_1 \times \cdots \times L_N$. We consider words of the form $\alpha w (P + \epsilon)$ where $\alpha \in \mathcal{L}$, $w \in \Upsilon^*$ and $P \in \Delta$. Since $\Upsilon, \Delta, \mathcal{L}$ are finite, they are all clearly well-quasi-ordered by set inclusion, and the set of words of the form $\alpha w (P + \epsilon)$ is well-quasi-ordered by the induced monotone domination order $\preceq$: $\alpha_1 \rho_1 \ldots \rho_m P_1 \preceq \alpha_2 \gamma_1 \ldots \gamma_n P_2$ if $\alpha_1 = \alpha_2$, $P_1 \subseteq P_2$, and there exists a strictly increasing function $g : \{1, 2 \ldots, m\} \to \{1, 2, \ldots, n\}$ such that for all $1 \leq i \leq m$, $\rho_i \subseteq \gamma_{g(i)}$.

We next associate to any configuration $\mathfrak{c}$ of $\mathcal{M}$, a canonical word $\mathsf{W}(\mathfrak{c}) \in \mathcal{L} \cdot \Upsilon^* \cdot (\Delta + \epsilon)$. Let $y_{i,1}, \ldots, y_{i,|\mathcal{X}_i|}$ be the set of clocks in T_i. Given a configuration $\mathfrak{c} = (\overline{q}, \overline{m})$ with $\overline{q} = ((\ell_1, \nu_1), \ldots, (\ell_N, \nu_N))$ and $\overline{m} = (M_1, \ldots, M_N)$, $\overline{q}$ is completely specified by describing for each $1 \leq i \leq N$, (i) the locations ℓ_i, (ii) the tuples $(\alpha_{i,j}, \mathsf{frac}(y_{i,j}))$ (resp, α_{ij}) if $\alpha_{ij} = ((y_{i,j}, \mathsf{reg}(\nu(y_{i,j}))))$ is in Γ_1 (resp. Γ_3) and $1 \leq j \leq |\mathcal{X}_i|$. Observe that here we use $\mathsf{frac}(y_{i,j})$ (resp. $\mathsf{reg}(\nu(y_{i,j}))$) to denote the fractional part (resp. the corresponding region) of $\nu(y_{i,j}))$. The former case keeps track of clocks, their regions as well as the fractional parts of the clock valuations, while in the latter, the value of clock $y_{i,j}$ is more than K, (iii) the multi set consisting of tuples $(\beta_i, \mathsf{frac}(age(a)))$ (resp. β_i) if $\beta_i = (a, \mathsf{reg}(age(a)), d)$ is in Γ_2 (resp. Γ_4). The former keeps track of tasks, the region of their ages, and their deadlines, along with the fractional parts of the ages, while in the latter, the age of the task is more than K. Observe that $age(a)$ returns the age of the task a.

Next, we group together the symbols $\alpha_h \in \Gamma_1, \beta_g \in \Gamma_2$ having the same fractional parts. Notice that the fractional parts are retained only for clocks (tasks) whose value (age) has not yet exceeded K. This yields a new set of $\Gamma_1 \cup \Gamma_2$ letters paired with their fractional parts $\{(\zeta_i, \mathsf{frac}_i) \mid 1 \leq i \leq p\}$ where ζ_i is a (multi)set of symbols from $\Gamma_1 \cup \Gamma_2$ and frac_i is the fractional part of those symbols. p is the number of distinct fractional parts in $\mathfrak{c}$. We then form the word $w = \rho_{i_{z_1}} \ldots \rho_{i_{z_p}} \in \Upsilon^+$ where $z_1 \ldots z_p$ is a permutation of $1 \ldots p$ that puts $\mathsf{frac}_{z_1} \ldots \mathsf{frac}_{z_p}$ in ascending order. Let $P \in \Delta$ be the set obtained (if any) by grouping all the symbols $\alpha_h \in \Gamma_3$ and $\beta_g \in \Gamma_4$. We then define $\mathsf{W}(\mathfrak{c}) = \alpha.w.P \in \mathcal{L}.\Upsilon^*(\Delta + \epsilon)$ as the canonical word encoding $\mathfrak{c}$.

▶ **Example 6.** Consider a 2-MTA $\mathcal{M}$. Let x_1, x_2 be the clocks of T_1 and y_1, y_2 be the clocks of T_2. Let $K = 3$ be the maximal constant used in $\mathcal{M}$. Consider the configurations
$\mathfrak{c}_1 = ((s_1, 0.5, 2.1), (s_2, 1.7, 2.5), (\{(a, 1.1, 2), (b, 2.3, \infty), (c, 3.5, \infty)\}, \{(d, 1.9, 2), (e, 0.7, 1)\}))$
and
$\mathfrak{c}_2 = ((s_1, 0.5, 2.4), (s_2, 1.9, 2.5), (\{(a, 1.4, 2), (b, 2.45, \infty), (c, 3.9, \infty)\}, \{(d, 1.99, 2), (e, 0.9, 1)\}))$.

Then $\mathsf{W}(\mathfrak{c}_1) = \mathsf{W}(\mathfrak{c}_2) = \alpha w P$ where $\alpha = (s_1, s_2)$, $P = \{(c, r_7, \infty)_1\}$, and
$w = \{(x_2, r_5), (a, r_3, 2)_1\}\{(b, r_5, \infty)_1\}\{(x_1, r_1), (y_2, r_5)\}\{(y_1, r_3), (e, r_1, 1)_2\}\{(d, r_3, 2)_2\}$.

Two configurations $\mathfrak{c}_1, \mathfrak{c}_2$ are equivalent ($\mathfrak{c} \sim \mathfrak{c}'$) if $\mathsf{W}(\mathfrak{c}_1) = \mathsf{W}(\mathfrak{c}_2)$. A configuration $\mathfrak{c}_1$ is dominated by a configuration $\mathfrak{c}_2$ (written $\mathfrak{c}_1 \preceq \mathfrak{c}_2$) if writing $\mathfrak{c}_2 = (\overline{q}_2, \overline{m}_2)$, there exists $\overline{q}_1, \overline{m}_1$ such that and $\overline{m}_1 = (M'_1, \ldots, M'_N)$ with $M'_i \subseteq M_i$ for all i, and $\mathfrak{c}_1 \sim (\overline{q}_1, \overline{m}_1)$. It can be easily seen that $\mathfrak{c}_1 \preceq \mathfrak{c}_2$ iff $\mathsf{W}(\mathfrak{c}_1) \preceq \mathsf{W}(\mathfrak{c}_2)$. In fact, the following lemma shows that $\sim$ is a bisimulation relation.

▶ **Lemma 7.** *Let $\mathfrak{c}_1, \mathfrak{c}_2$ be two configurations of an N-MTA. Let $e \in E_i$ be a transition, $1 \leq i \leq N$, and let $t \in \mathbb{R}$. If $\mathfrak{c}_1 \sim \mathfrak{c}_2$, then*
(1) If $\mathfrak{c}_1 \xrightarrow{e} \mathfrak{c}'_1$, there exists $\mathfrak{c}'_2$ such that $\mathfrak{c}_2 \xrightarrow{e} \mathfrak{c}'_2$ and $\mathfrak{c}'_1 \sim \mathfrak{c}'_2$. If $\mathfrak{c}_2 \xrightarrow{e} \mathfrak{c}'_2$, there exists $\mathfrak{c}'_1$ such that $\mathfrak{c}_1 \xrightarrow{e} \mathfrak{c}'_1$ and $\mathfrak{c}'_1 \sim \mathfrak{c}'_2$.
(2) If $\mathfrak{c}_1 \xrightarrow{t} \mathfrak{c}'_1$, there exists $\mathfrak{c}'_2$ and $t' \in \mathbb{R}$ such that $\mathfrak{c}_2 \xrightarrow{t'} \mathfrak{c}'_2$ and $\mathfrak{c}'_1 \sim \mathfrak{c}'_2$. If $\mathfrak{c}_2 \xrightarrow{t} \mathfrak{c}'_2$, there exists $\mathfrak{c}'_1$ and $t' \in \mathbb{R}$ such that $\mathfrak{c}_1 \xrightarrow{t'} \mathfrak{c}'_1$ and $\mathfrak{c}'_1 \sim \mathfrak{c}'_2$.

As an easy corollary of the above, we see that $\sim$ preserves goodness and badness: For any configurations $\mathfrak{c} \sim \mathfrak{c}'$, $\mathfrak{c}$ is good iff $\mathfrak{c}'$ is good. The proof follows from the definition of goodness and Lemma 7, whose proof can be found in the extended version of the paper [1].

It is hence sufficient to only consider configurations upto $\sim$-equivalence, and we define the quotient labeled transition system $\mathcal{M}/\sim$ to consist of all the words $\mathsf{W}(\mathfrak{c})$ whenever $\mathfrak{c}$ is a configuration of $\mathcal{M}$. Call $\mathcal{M}/\sim$ as $\mathcal{W}$. $\mathcal{W} = \{\mathsf{W}(\mathfrak{c}) \mid \mathfrak{c} \text{ is a configuration in } \mathcal{M}\}$. For $W_1, W_2 \in \mathcal{W}$, and a transition $e \in E_i$ for $1 \leq i \leq N$, we define a transition $W_1 \xrightarrow{e} W_2$ if for all $\mathfrak{c}_1 \in \mathsf{W}^{-1}(W_1)$, there is some configuration $\mathfrak{c}_2 \in \mathsf{W}^{-1}(W_2)$ such that $\mathfrak{c}_1 \xrightarrow{e} \mathfrak{c}_2$. The timed transition is defined similarly. Corresponding to each initial configuration $\mathfrak{c}_0$ in $\mathcal{M}$, we consider $W_0 = \mathsf{W}(\mathfrak{c}_0)$ to be an initial word in $\mathcal{W}$. Let $\mathcal{W}_0$ be the set of initial words corresponding to initial configurations $\mathfrak{c}_0$. It can be seen that for any $W_1, W_2 \in \mathcal{W}$, and a transition $e \in E_i$ or $t \in \mathbb{R}$, $W_1 \xrightarrow{\alpha} W_2$ ($\alpha \in \{e, t\}$) iff there exist configurations $\mathfrak{c}_1 \in \mathsf{W}^{-1}(W_1)$ and $\mathfrak{c}_2 \in \mathsf{W}^{-1}(W_2)$ such that $\mathfrak{c}_1 \xrightarrow{\alpha} \mathfrak{c}_2$. Given a word $W \in \mathcal{W}$, and $\alpha \in \{e, t\}$ for some transition e and time $t \in \mathbb{R}$, let $\mathsf{succ}(W) = \{W' \in \mathcal{W} \mid W \xrightarrow{\alpha} W'\}$ denote the successors of W in $\mathcal{W}$.

▶ **Lemma 8.** *For any word W, the set $\mathsf{succ}(W)$ is finite and effectively computable.*

Let $\mathcal{W}_0$ be the set of initial words corresponding to $\mathsf{W}(\mathfrak{c}_0)$ for initial configurations $\mathfrak{c}_0$. Let $W_\emptyset = \mathsf{W}(\mathfrak{c}_{goal})$. Algorithm 1 decides whether the configuration $\mathfrak{c}_{goal}$ can be reached. In this algorithm, the function $\mathsf{Minimize}(R)$ is used, where $R \subseteq \mathcal{W}$ is a set of words. It does the following: it chooses a word $W_1 \in R$ and removes W_1 from R if there exists a word $W_2 \in R$ such that $W_2 \preceq W_1$, and then repeats the procedure until all words in R are processed. Overall, the algorithm works as follows. Till the set Next of words waiting to be processed is non-empty, the algorithm chooses one word from Next, and moves it to the Processed set. It also generates all successors of the chosen word, minimizes them, and adds them to Next unless there is already some $\preceq$-smaller word in Next or Processed. If a new word is added to Next, the algorithm removes at the same time all $\preceq$-bigger words from both Next and Processed. The correctness of the algorithm is discussed next.

A set of words R is good (denoted $Good(R)$) iff there exists some word $W \in R$ which is good. A word W is good iff there exists a good configuration $\mathfrak{c}$ such that $\mathsf{W}(\mathfrak{c}) = W$. If W is a good word, and if $i \in \mathbb{N}$ is the length of the shortest path (excluding time elapse transitions) from W to $W_\emptyset$, then we say that $\mathsf{dist}(W)$ is i. Given a set R of words, $\mathsf{dist}(R) \in \mathbb{N} \cup \{\infty\}$ is defined as the length of the shortest path (excluding time elapse transitions) from some $W \in R$ to $W_\emptyset$. More precisely, if $R = \emptyset$, then $\mathsf{dist}(R) = \infty$, otherwise, $\mathsf{dist}(R) = min_{W \in R}\mathsf{dist}(W)$.

Algorithm 1 REACH EMPTY.

Input: A stateless, time-independent N-MTA, and configuration $\mathfrak{c}_{goal} = (\overline{q}, \overline{m})$ as above.
Output: TRUE if $\mathfrak{c}_{goal}$ is reachable. Otherwise, FALSE.
if $W_\emptyset \in \mathcal{W}_0$, **then return** TRUE;
Processed $= \emptyset$;
Next $=$ Minimize($\mathcal{W}_0$);
while Next $\neq \emptyset$ **do**
  leftmirgin=0.5in Pick and remove a word W from Next and move it to Processed,
  leftmiirgiin=0.5iin **foreach** $U \in$ Minimize(succ(W))
    leftmirgin=0.5in **if** $U = W_\emptyset$, **then return** TRUE,
    leftmiirgiin=0.5iin **else if** $\nexists V \in$ Processed $\cup$ Next s.t. $V \preceq U$, **then**
      leftmiiirgiiin=0.5iiin Remove all V from Processed$\cup$Next s.t $U \preceq V$
      leftmivrgivn=0.5ivn Add U to Next
return FALSE

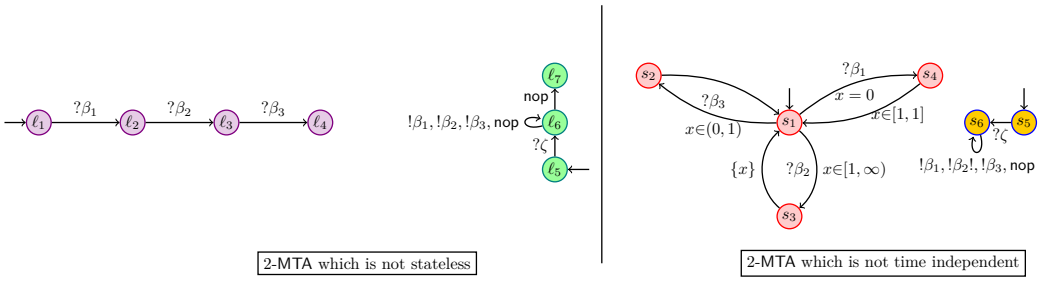


Figure 3

▶ **Lemma 9.**
1. $Good(\text{Processed} \cup \text{Next}) \rightarrow Good(\mathcal{W}_0)$
2. $Good(\mathcal{W}_0) \rightarrow \text{dist(Processed)} > \text{dist(Next)}$

To prove the invariants, we use the following lemma.

▶ **Lemma 10.** *If $W \preceq W'$ and $\text{dist}(W') = i$, then $\text{dist}(W) = j$ for some $j \leq i$.*

Due to the well-quasi ordering, the algorithm terminates: if not, over a period of time, there will be an infinite sequence of words in Next, each new word added having the property that it does not dominate any of its predecessors. This would constitute an infinite non saturating sequence, directly contradicting Higman's Lemma. The algorithm returns FALSE only when Next is empty. Then, dist(Processed) $>$ dist(Next) is not true. Therefore, by invariant 2 in lemma 9, $\mathcal{W}_0$ is not good. The algorithm returns TRUE only if either $W_\emptyset$ is already in $\mathcal{W}_0$, or if $W_\emptyset$ is a member of Minimize(succ(W)) for some $W \in$ Next. In either case, Next is good. Then, by invariant 1 of lemma 9, $\mathcal{W}_0$ is good. This gives the following lemma.

▶ **Lemma 11.** *Algorithm* REACH EMPTY *terminates and returns true iff starting from the initial configuration $\mathfrak{c}_0$ in $\mathcal{M}$, $\mathfrak{c}_{goal}$ is reachable.*

This concludes the proof of theorem 5. A detailed discussion with proofs for the lemmas can be found in the extended version of the paper [1].

Notice that the stateless, and time-independent properties of $\mathcal{M}$ are crucial in Lemma 10. The example below shows that relaxing either condition violates Lemma 10.

To the left is a 2-MTA which is not stateless. It can be seen that $\mathfrak{c}_1=(\ell_1,\ell_6,\{\beta_1,\beta_3\},\emptyset) \preceq (\ell_1,\ell_6,\{\beta_1,\beta_2,\beta_3\},\emptyset)=\mathfrak{c}_2$. Hence, $\mathsf{W}(\mathfrak{c}_1) \preceq \mathsf{W}(\mathfrak{c}_2)$. Indeed from $\mathfrak{c}_2$, one can reach $(\ell_4,\ell_6,\emptyset,\emptyset)$, but not from $\mathfrak{c}_1$. To the right is a 2-MTA which is not time independent. It can be seen that $\mathfrak{c}_1=(((s_1,0),s_6),\{(\beta_1,0,\infty),(\beta_3,0,\infty)\},\emptyset) \preceq (((s_1,0),s_6),\{(\beta_1,0,\infty),(\beta_2,0,\infty),(\beta_3,0,\infty)\},\emptyset)=\mathfrak{c}_2$. However, $(((s_1,0),s_6),\emptyset,\emptyset)$ is reachable from $\mathfrak{c}_2$ but not from $\mathfrak{c}_1$.

6 Conclusion

We proposed a model to address the verification problem for timed asynchronous programs. We identified a special subclass (stateless and time-independent) for which the reachability problem is decidable and control reachability is PSPACE-complete. There are multiple avenues for further work. The first question is to check the tightness of the EXPSPACE lower bound provided. Another question would be to consider the model where we use *priority bags* instead of bags. In a priority bag, tasks have associated deadlines and priorities. The process, while picking up a task for execution, is expected to pick up a task with the highest priority. Queues are yet another interesting data structure in place of bags: in this set up, the tasks which require a processor's attention are picked up in the order in which they were assigned by various processes. One can also look at mutiset timed pushdown systems, which extend the model of [16] with time, and multiple processes. Finally, we can move from the one player setting to two players, where the environment chooses a task for the process to execute. Under this two player setting, the question would be if the system has a strategy to execute all the pending tasks.

References

1. P.A. Abdulla, M. Faouzi Atig, S. Krishna, and S. Vaidya. *Verification of Timed Asynchronous Programs.* URL: `http://www.cse.iitb.ac.in/~krishnas/fsttcs2018.pdf`.
2. Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theor. Comput. Sci.*, 126(2):183–235, April 1994. `doi:10.1016/0304-3975(94)90010-8`.
3. Mohamed Faouzi Atig, Ahmed Bouajjani, K. Narayan Kumar, and Prakash Saivasan. Verification of Asynchronous Programs with Nested Locks. In Satya V. Lokam and R. Ramanujam, editors, *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017, December 11-15, 2017, Kanpur, India*, volume 93 of *LIPIcs*, pages 11:1–11:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. `doi:10.4230/LIPIcs.FSTTCS.2017.11`.
4. Mohamed Faouzi Atig, Ahmed Bouajjani, and Tayssir Touili. Analyzing Asynchronous Programs with Preemption. In Ramesh Hariharan, Madhavan Mukund, and V. Vinay, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2008, December 9-11, 2008, Bangalore, India*, volume 2 of *LIPIcs*, pages 37–48. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2008. `doi:10.4230/LIPIcs.FSTTCS.2008.1739`.
5. Rohit Chadha and Mahesh Viswanathan. Decidability Results for Well-Structured Transition Systems with Auxiliary Storage. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *CONCUR 2007 - Concurrency Theory, 18th International Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007, Proceedings*, volume 4703 of *Lecture Notes in Computer Science*, pages 136–150. Springer, 2007. `doi:10.1007/978-3-540-74407-8_10`.
6. Michael Emmi, Shaz Qadeer, and Zvonimir Rakamaric. Delay-bounded scheduling. In Thomas Ball and Mooly Sagiv, editors, *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 411–422. ACM, 2011. `doi:10.1145/1926385.1926432`.

7 Pierre Ganty and Rupak Majumdar. Analyzing Real-Time Event-Driven Programs. In Joël Ouaknine and Frits W. Vaandrager, editors, *Formal Modeling and Analysis of Timed Systems, 7th International Conference, FORMATS 2009, Budapest, Hungary, September 14-16, 2009. Proceedings*, volume 5813 of *Lecture Notes in Computer Science*, pages 164–178. Springer, 2009. doi:10.1007/978-3-642-04368-0_14.

8 Pierre Ganty and Rupak Majumdar. Algorithmic verification of asynchronous programs. *ACM Trans. Program. Lang. Syst.*, 34(1):6:1–6:48, 2012. doi:10.1145/2160910.2160915.

9 Serge Haddad, Sylvain Schmitz, and Philippe Schnoebelen. The Ordinal-Recursive Complexity of Timed-arc Petri Nets, Data Nets, and Other Enriched Nets. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 355–364. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.46.

10 Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, 3(1):326–336, 1952.

11 Ranjit Jhala and Rupak Majumdar. Interprocedural analysis of asynchronous programs. In Martin Hofmann and Matthias Felleisen, editors, *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, Nice, France, January 17-19, 2007*, pages 339–350. ACM, 2007. doi:10.1145/1190216.1190266.

12 Joseph B Kruskal. The theory of well-quasi-ordering: A frequently discovered concept. *Journal of Combinatorial Theory, Series A*, 13(3):297–305, 1972.

13 Pallavi Maiya, Rahul Gupta, Aditya Kanade, and Rupak Majumdar. Partial Order Reduction for Event-Driven Multi-threaded Programs. In *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS*, volume 9636 of *Lecture Notes in Computer Science*, pages 680–697. Springer, 2016.

14 Rupak Majumdar and Zilong Wang. Bbs: A Phase-Bounded Model Checker for Asynchronous Programs. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, volume 9206 of *Lecture Notes in Computer Science*, pages 496–503. Springer, 2015. doi:10.1007/978-3-319-21690-4_33.

15 M. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall International, 1967.

16 Koushik Sen and Mahesh Viswanathan. Model Checking Multithreaded Programs with Asynchronous Atomic Methods. In Thomas Ball and Robert B. Jones, editors, *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4144 of *Lecture Notes in Computer Science*, pages 300–314. Springer, 2006. doi:10.1007/11817963_29.

17 Jiří Srba. Timed-Arc Petri Nets vs. Networks of Timed Automata. *In Proceedings of the 26th International Conference on Application and Theory of Petri Nets (ICATPN 2005). Netherlands: Springer-Verlag, 2005*, pages 385–402, 2005.

The Cayley-Graph of the Queue Monoid: Logic and Decidability

Faried Abu Zaid[1]
Camelot Management Consultants, CoE Artificial Intelligence for Information Management, Munich, Germany
faza@camelot-mc.com

Chris Köcher
Technische Universität Ilmenau, Automata and Logics Group, Ilmenau, Germany
chris.koecher@tu-ilmenau.de
https://orcid.org/0000-0003-4575-9339

Abstract

We investigate the decidability of logical aspects of graphs that arise as Cayley-graphs of the so-called queue monoids. These monoids model the behavior of the classical (reliable) fifo-queues. We answer a question raised by Huschenbett, Kuske, and Zetzsche and prove the decidability of the first-order theory of these graphs with the help of an – at least for the authors – new combination of the well-known method from Ferrante and Rackoff and an automata-based approach. On the other hand, we prove that the monadic second-order of the queue monoid's Cayley-graph is undecidable.

2012 ACM Subject Classification Theory of computation → Logic, Theory of computation → Models of computation, Information systems → Data structures

Keywords and phrases Queues, Transformation Monoid, Cayley-Graph, Logic, First-Order Theory, MSO Theory, Model Checking

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.9

1 Introduction

Data structures are one of the most important concepts in nearly all areas of computer science. Important data structures are, e.g., finite memories, counters, and (theoretically) infinite Turing-tapes. But the most fundamental ones are stacks and queues. And although these two data structures look very similar as they have got the same set of operations on them (i.e. writing and reading of a letter), they differ from the computability's point of view: if we equip finite automata with both data structures, then the ones with stacks compute exactly the context-free languages (these are the well-known pushdown automata). But if we equip a finite automaton with queues (in literature they are called queue automata, communicating automata, or channel systems) then we obtain a Turing-complete computation model (cf. [2, 3]). This strong model can be weakened with various extensions, e.g., if the queue is allowed to forget some of its contents (cf. [1, 5, 22]) or if letters of low priority can be superseded by letters with higher priority (cf. [12]).

One possible approach to analyze the difference of the behavior of the data structures is to model them as a monoid of transformations. Then, finite memories induce finite monoids, counters induce the integers with addition, stacks induce the polycyclic monoids (cf. [14, 27]),

[1] The presented work was conducted while the first author was affiliated with the Technische Universität Ilmenau.

and queues induce the so-called queue monoids which were first introduced in [13]. And while the transformation monoids of the other data structures are very well-understood, we still do not know much about the queue monoid. Further results on the queue monoid (with and without lossiness) can be found in [17, 18]. Here, we only consider the reliable queue monoids. Concretely, we study the Cayley-graph of this monoid.

Cayley-graphs are a natural translation of finitely generated groups and monoids into graph theory and is a fundamental tool to handle these algebraic constructs in combinatorics, topology, and automata theory. Concretely, these are labeled, directed graphs with labels from a fixed generating set Γ of the monoid $\mathcal{M}$. Thereby, the elements from $\mathcal{M}$ are the graph's nodes and there is an a-labeled edge (where $a \in \Gamma$) from $x \in \mathcal{M}$ to $y \in \mathcal{M}$ iff $xa = y$ holds in $\mathcal{M}$. For groups, we already know many results on their Cayley-graphs. For example, the group's Cayley-graph has decidable first-order theory if, and only if, its existential first-order theory is decidable and if, and only if, the group's word problem is decidable [19]. Moreover, a group's Cayley-graph has decidable monadic second-order theory if, and only if, the group is context-free (that is, if the group's word problem is context-free) [19,23]. Besides these results, Kharlampovich et al. considered in [15] so-called Cayley-graph automatic groups (these are the groups having an automatic Cayley-graph in the sense of [16]) which links to the rich theory of automatic structures.

Unfortunately, there are not that many studies on Cayley-graphs of monoids. In particular, there are monoids with decidable word problem but undecidable existential first-order theory of their Cayley-graph [20, 24]. For finite monoids the Cayley-graphs are finite and, hence, the first- and second-order theories are complete for polynomial space and exponential space, respectively [10]. For polycyclic monoids the Cayley-graphs are automatic, complete $|A|$-ary trees (where A is the underlying alphabet) with an additional node every other node is connected with (this is the zero element resp. error state). Therefore, due to [6, 20] the Cayley-graphs monadic second-order theory is decidable (the first-order theory is even in 2EXPSPACE by [21]).

In this paper we want to consider logics on the Cayley-graph of the queue monoid. Concretely, we will see that this graph's first-order theory is decidable by giving a primitive recursive (but non-elementary) algorithm which combines two well-known methods from model theory in a (at least for the authors) new way: the method of Ferrante and Rackoff [8] and an automata-based approach. This gives an answer on a question raised by Huschenbett, Kuske, and Zetzsche [13]. There, they conjectured the undecidability of its first-order logic implying that the graph is not automatic in the sense of [16]. Moreover, we will prove the undecidability of the monadic second-order theory with the help of a well-known result from Seese [28].

2 Preliminaries

Let A be an alphabet. We use $\preceq$ to denote the *prefix-relation* and $\sqsubseteq$ for the *suffix-relation* on A^*. If $u = vw$ we write $v^{-1}u = w$ and $uw^{-1} = v$. Thereby, v is the *complementary prefix* of w wrt. u and w the *complementary suffix* of v wrt. u. For $u, v \in A^*$ let $u \sqcap v$ denote the largest suffix of u that is also a prefix of v.

For $m, n, r \in \mathbb{N}$ we write $m =_r n$ iff $m = n$ or $m, n > r$. The function $\exp_r(n)$ is inductively defined by $\exp_0(n) = n$ and $\exp_{r+1}(n) = 2^{\exp_r(n)}$.

Logic on Graphs and Words

Let A be a finite set of labels. An *edge-labeled graph* is a tuple $G = (V^G, (E_a^G)_{a\in A})$ where V is the set of vertices and $E_a^G \subseteq V \times V$ is the set of a-labeled edges. A *word-structure* over A is a tuple $W = (\{0,\dots,n-1\}, \leqslant^W, (P_a^W)_{a\in A})$ where $\leqslant^W$ is the usual order on $\{0,\dots,n-1\}$, and $(P_a^W)_{a\in A}$ is a partition of $\{0,\dots,n-1\}$ (some of the sets P_a^W may be empty). Whenever we use logic to describe properties of a word w then the formula is evaluated on the corresponding word structure W.

Let $\tau = \{R_1,\dots,R_m,c_1,\dots,c_n\}$ where R_i is a relation symbol of arity r_i and c_j is a constant symbol. *First-order formulas* (over the vocabulary τ) are build up from variables and constant symbols $\{x_i \mid i \in \mathbb{N}\} \cup \{c_1,\dots,c_n\}$, the edge relation symbols $\{R_1,\dots,R_m\}$, the equality symbol $=$, the Boolean connectives $\{\neg,\vee,\wedge,\rightarrow\}$, quantifiers $\{\forall,\exists\}$, and the bracket symbols $\{(,)\}$. We write $G \models \varphi$ to denote that the formula φ is satisfied by the structure G. The *quantifier rank* $\mathrm{qr}(\varphi)$ of a formula φ is the maximal nesting depth of quantifiers within φ. Two structures G and H are *r-equivalent* (denoted $G \equiv_r H$) if they cannot be distinguished by any formula of quantifier rank $\leqslant r$. For a structure G and two tuples $\vec{p},\vec{q} \in (V^G)^m$ we write $\vec{p} \equiv_r^G \vec{q}$ or say that $\vec{p}$ and $\vec{q}$ are r-equivalent in G whenever $G \models \varphi(\vec{p}) \Leftrightarrow G \models \varphi(\vec{q})$ for all first-order formulas φ with m free variables and quantifier rank at most r. For all the above notations we adopt the convention that we omit superscripts whenever this should not lead to any confusion. For instance we write $\vec{p} \equiv_r \vec{q}$ when the underlying structure G is clear from the context.

The *r-type* of a structure G is the set of all first-order sentences φ of quantifier rank at most r such that $G \models \varphi$. It is well known that there are up to equivalence only finitely many sentences of quantifier rank at most r. Hence the r-type of a structure can be characterized by a sentence, which has also quantifier rank r.

Ehrenfeucht-Fraïssé-relations (resp. *EF-relations*) for a graph $G = (V, (E_a)_{a\in A})$ are a system $(\mathcal{E}_m^r)_{r,m\in\mathbb{N}}$ where $\mathcal{E}_m^r$ is an equivalence relation on V^m and the following is true for all $r,m \in \mathbb{N}$ and $\vec{p},\vec{q} \in V^m$:

- If $(p_1,\dots,p_m)\mathcal{E}_m^0(q_1,\dots,q_m)$ then the mapping $p_i \mapsto q_i$ is a partial isomorphism, that is $p_i = p_j \Leftrightarrow q_i = q_j$ and $(p_i,p_j) \in E_a \Leftrightarrow (q_i,q_j) \in E_a$ for all $1 \leqslant i,j \leqslant m$ and all $a \in A$.
- If $\vec{p}\mathcal{E}_m^{r+1}\vec{q}$ then for every $p \in V$ there exists a $q \in V$ such that $(\vec{p},p)\mathcal{E}_{m+1}^r(\vec{q},q)$.

Ehrenfeucht-Fraïssé-relations are useful to identify r-equivalent tuples in a graph. This is formalized in the following theorem.

▶ **Theorem 2.1** ([7, 9]). *Let G be a graph, $(\mathcal{E}_m^r)_{r,m\in\mathbb{N}}$ Ehrenfeucht-Fraïssé-relations for G, and $\vec{p},\vec{q}$ m-tuples of nodes from G. If $\vec{p}\mathcal{E}_m^r\vec{q}$ then $\vec{p} \equiv_r \vec{q}$.*

3 Queue Monoid and its Cayley-Graph

Definition of the Monoid

The queue monoid models the behavior of a (reliable) fifo-queue whose entries come from an alphabet A. Consequently, the state of a queue is a word from A^*. The basic actions of our queue are writing of the symbol $a \in A$ of the queue (denoted by a) and reading the symbol $a \in A$ from the queue (denoted by $\overline{a}$). Thereby, $\overline{A}$ is a disjoint copy of A containing all reading actions $\overline{a}$ and $\Sigma := A \uplus \overline{A}$ is the set of all basic actions. To simplify notation, for a word $u = a_1a_2\dots a_n \in A^*$ we write $\overline{u}$ for the word $\overline{a_1}\,\overline{a_2}\dots\overline{a_n}$.

Formally, the action $a \in A$ appends the letter a to the state of the queue and the action $\overline{a} \in \overline{A}$ tries to cancel the letter a from the beginning of the current state of the queue. Thereby, if the state does not start with this symbol, the queue will end up in an error state

which we denote by $\bot$. Note that in contrast to (partially) lossy queues which we considered in [17, 18], these queues cannot forget any part of their content. Hence, these ideas lead to the following definition:

▶ **Definition 3.1.** Let $\bot \notin A^*$. The function $\circ\colon (A^* \cup \{\bot\}) \times \Sigma^* \to (A^* \cup \{\bot\})$ is defined for each $s \in A^*$, $a, b \in A$, and $u \in \Sigma^*$ as follows:

(1) $s \circ \varepsilon = s$

(2) $s \circ au = sa \circ u$

(3) $bs \circ \overline{a}u = \begin{cases} s \circ u & \text{if } a = b \\ \bot & \text{otherwise} \end{cases}$

(4) $\varepsilon \circ \overline{a}u = \bot \circ u = \bot$

With the help of this function we may now identify sequences of actions that are acting equally. This is finally used to define the monoid of queue actions.

▶ **Definition 3.2.** Let $u, v \in \Sigma^*$. Then u and v *act equally* (denoted by $u \equiv v$) if $s \circ u = s \circ v$ holds for each $s \in A^*$. Since $s \circ uv = (s \circ u) \circ v$, the resulting relation $\equiv$ is a congruence on the free monoid Σ. Hence, the quotient $\mathcal{Q}(A) := \Sigma^*/_{\equiv}$ is a monoid which we call the *monoid of queue actions* or for short *queue monoid*. The neutral element of $\mathcal{Q}(A)$ is $[\varepsilon]_{\equiv} = \{\varepsilon\}$, which we will denote simply by ε.

Note that the queue monoids $\mathcal{Q}(A)$ for alphabets A of different size are not isomorphic. Though, all of the following results hold for any alphabet A with $|A| \geqslant 2$. Hence, we may fix an arbitrary alphabet A from now on and write $\mathcal{Q}$ instead of $\mathcal{Q}(A)$.

▶ Remark. Let $A = \{a\}$ be a singleton. Then a queue on this alphabet acts like a partially blind counter since $a^n \circ a = a^{n+1}$ and $a^{n+1} \circ \overline{a} = a^n$. In other words, $\mathcal{Q}(\{a\})$ is the bicyclic semigroup.

Basic Properties

Now, we want to recall some basic properties considering the equivalence relation $\equiv$. The first important fact expresses the equivalence in terms of some commutations of write and read actions under certain contexts.

▶ **Theorem 3.3** ([13, Theorem 4.3])**.** *The equivalence relation $\equiv$ is the least congruence on the free monoid Σ^* satisfying the following equations for all $a, b \in A$:*

(1) $a\overline{b} \equiv \overline{b}a$ *if* $a \neq b$

(2) $a\overline{a}\overline{b} \equiv \overline{a}a\overline{b}$

(3) $ba\overline{a} \equiv b\overline{a}a$ ◀

A very frequently used notation is the following: the *projections to write and read actions*, resp., are defined as $\mathsf{wrt}, \mathsf{rd}\colon \Sigma^* \to A^*$ by $\mathsf{wrt}(a) = \mathsf{rd}(\overline{a}) = a$ and $\mathsf{wrt}(\overline{a}) = \mathsf{rd}(a) = \varepsilon$ for all $a \in A$. In other words, $\mathsf{wrt}(u)$ can be derived from u by deletion of all read actions and $\mathsf{rd}(u)$ can be obtained from u by deletion of all the write actions and by suppression of the overlines. Due to Theorem 3.3 all words contained in a single equivalence class of $\equiv$ have the same projections. Hence we use them for equivalence classes as well. Though, equality of these projections of two words does not imply equivalence of these words. For example, $u = \overline{a}a$ and $v = a\overline{a}$ have the same projections $\mathsf{wrt}(u) = \mathsf{rd}(u) = a = \mathsf{wrt}(v) = \mathsf{rd}(v)$ but are not equivalent since we have

$$\varepsilon \circ a\overline{a} = \varepsilon \neq \bot = \varepsilon \circ \overline{a}a\,.$$

The non-equivalence of the two words above is very easy to prove. Also (non-)equivalence of two arbitrary words is decidable in polynomial time: for this purpose we compute normal forms of the equivalence classes of $\equiv$. We do this by ordering the equations from Theorem 3.3 from left to right resulting in a terminating and confluent semi-Thue system $\mathcal{R}$ [13, Lemma 4.1]. Then, for any word $u \in \Sigma^*$ there is a unique, irreducible word $\mathsf{nf}(u)$ with $u \to^* \mathsf{nf}(u)$, the so-called *normal form* of u resp. of its equivalence class $[u]_\equiv$. In this word $\mathsf{nf}(u)$ the read actions from u are moved to the left as far as the equations from above allow.

▶ **Example 3.4.** Let $a, b \in A$ with $a \neq b$ and $u = abb\overline{ab}$. Then we have

$$abb\overline{ab} \xrightarrow{(1)} ab\bar{a}b\bar{b} \xrightarrow{(1)} a\bar{a}bb\bar{b} \xrightarrow{(3)} a\bar{a}b\bar{b}b\,.$$

Since we cannot apply any rule from Theorem 3.3 anymore, we have $\mathsf{nf}(u) = a\bar{a}b\bar{b}b$.

From the definition of $\mathcal{R}$ we obtain that a word is in normal form if it starts with a sequence of read operations followed by an alternating sequence of write and read actions, where all of the read actions $\bar{a}$ appear straight behind the write action a. Finally, the normal form ends with a sequence of write actions. Concretely, the set of all normal forms is

$$\mathsf{NF} := \{\mathsf{nf}(u) \mid u \in \Sigma^*\} = \overline{A}^*\{a\bar{a} \mid a \in A\}^*A^*\,.$$

Let $u \in \Sigma^*$. Then the normal form $\mathsf{nf}(u)$ is uniquely defined by three words $u_1, u_2, u_3 \in A^*$ such that $\mathsf{nf}(u) = \overline{u_1}a_1\overline{a_1}\dots a_n\overline{a_n}u_3$ where $u_2 = a_1 \dots a_n$. Thereby, we denote the word u_1 by $\lambda(u)$, the word u_2 by $\mu(u)$, and u_3 by $\varrho(u)$. Hence, we can define the *characteristics* of u ($[u]_\equiv$, resp.) by the triple $\chi(u) := (\lambda(u), \mu(u), \varrho(u))$. Hence, from these characteristics $\chi(u)$ we can obtain the projections of u on its write and read actions as well: $\mathsf{wrt}(u) = \mu(u)\varrho(u)$ and $\mathsf{rd}(u) = \lambda(u)\mu(u)$.

From now on, we will use these characteristics to represent the elements of $\mathcal{Q}$. In other words, we may understand $\mathcal{Q}$ as a triple of words (i.e., $(A^*)^3$) with a special type of concatenation. The concatenation of any transformation $u \in \Sigma^*$ with a single letter is described in the lemma below.

▶ **Lemma 3.5.** *Let $u \in \Sigma^*$ and $a \in A$. Then we have*

$$\chi(ua) = (\lambda(u), \mu(u), \varrho(u)a) \quad \text{and} \quad \chi(u\bar{a}) = (\mathsf{rd}(u)as^{-1}, s, s^{-1}\mathsf{wrt}(u))$$

where $s = \mu(u)a \sqcap \mathsf{wrt}(u)$.

Iterating Lemma 3.5 we obtain the following Theorem:

▶ **Theorem 3.6** ([13, Theorem 5.3])**.** *Let $u, v \in \Sigma^*$. Then $\chi(uv) = (\mathsf{rd}(uv)s^{-1}, s, s^{-1}\mathsf{wrt}(uv))$ where $s = \mu(u)\mathsf{rd}(v) \sqcap \mathsf{wrt}(u)\mu(v)$.* ◀

In other words, the multiplication of two words $u, v \in \Sigma^*$ can be understood as follows: at first we move the read actions from $\overline{\mathsf{rd}(v)}$ to the left such that each of its letters is directly preceded by exactly one write action. If this is not possible (because $\lambda(v)$ is longer than $\varrho(u)$) we move the letters from $\overline{\mu(u)\lambda(v)}$ to the left until there is an alternating word of write and read actions. Now, if there is an infix $a\bar{b}$ with $a \neq b$ all of these read actions move one position to the left. We iterate this last step until there is no such infix. It is easy to see, that the new alternating word contains equal subsequences of write and read actions, respectively. Thereby, the read actions are the longest suffix of $\overline{\mu(u)\mathsf{rd}(v)}$ and the write actions the longest prefix of $\mathsf{wrt}(u)\mu(v)$ such that the equality of these subsequences holds (this is $\mu(u)\mathsf{rd}(v) \sqcap \mathsf{wrt}(u)\mu(v)$).

The Monoid's Cayley-Graph

In this subsection we first recall the definition of Cayley-graphs for arbitrary, finitely generated monoids. Afterwards, we give some common properties as well as some special characteristics of the queue monoid's Cayley-graph.

▶ **Definition 3.7.** Let $\mathcal{M}$ be a monoid generated by a finite set $\Gamma \subseteq \mathcal{M}$. The *(right) Cayley-graph* of $\mathcal{M}$ is the edge-labeled, directed graph $\mathfrak{C}(\mathcal{M}, \Gamma) := (\mathcal{M}, (E_a)_{a\in\Gamma})$ with $E_a = \{(x, y) \in \mathcal{M} \mid y = xa\}$ for each $a \in \Gamma$.

Similar to the right Cayley-graph, we may define the *left Cayley-graph* of $\mathcal{M}$ as the edge-labeled, directed graph $\mathfrak{LC}(\mathcal{M}, \Gamma) = (\mathcal{M}, (F_a)_{a\in\Gamma})$ with $F_a = \{(x, y) \in \mathcal{M} \mid y = ax\}$ for all $a \in \Gamma$.

▶ Remark. There is a strong relation between left and right Cayley-graphs of a monoid and Green's relations which are first introduced and studied in [11]. Recall that $x\mathcal{R}y$ iff $x\mathcal{M} = y\mathcal{M}$ for every $x, y \in \mathcal{M}$ and, similarly, $x\mathcal{L}y$ iff $\mathcal{M}x = \mathcal{M}y$. Then by [25, Proposition V.1.1] we have $x\mathcal{R}y$ ($x\mathcal{L}y$) if, and only if, x is strongly connected to y in $\mathfrak{C}(\mathcal{M}, \Gamma)$ ($\mathfrak{LC}(\mathcal{M}, \Gamma)$, resp.).

The concrete shape of the Cayley-graph of a monoid heavily depends on the chosen set of generators. For example, $\{-1, 1\}$ and $\{-2, 3\}$ are generating sets of $(\mathbb{Z}, +)$, but the resulting Cayley-graphs are not isomorphic (even if we remove the labels). Though, the chosen generating set has no influence on decidability and complexity of the FO and MSO theory of the Cayley-graph since the both problems are logspace reducible on each other (which we denote by $\approx_{\log}$):

▶ **Proposition 3.8** ([20, Proposition 3.1])**.** *Let Γ_1 and Γ_2 be two finite generating sets of the monoid $\mathcal{M}$. Then*
(1) $\mathsf{FOTh}(\mathfrak{C}(\mathcal{M}, \Gamma_1)) \approx_{\log} \mathsf{FOTh}(\mathfrak{C}(\mathcal{M}, \Gamma_2))$ *and*
(2) $\mathsf{MSOTh}(\mathfrak{C}(\mathcal{M}, \Gamma_1)) \approx_{\log} \mathsf{MSOTh}(\mathfrak{C}(\mathcal{M}, \Gamma_2))$. ◀

From now on we only consider the Cayley-graph of the queue monoid $\mathcal{Q}$. To simplify notation we write $\mathfrak{C}$ instead of $\mathfrak{C}(\mathcal{Q}, \Sigma)$ and $\mathfrak{LC}$ instead of $\mathfrak{LC}(\mathcal{Q}, \Sigma)$. First we prove some properties of $\mathfrak{C}$ and $\mathfrak{LC}$.

▶ **Proposition 3.9.** *The following statements hold:*
(1) $\mathsf{FOTh}(\mathfrak{C}) \approx_{\log} \mathsf{FOTh}(\mathfrak{LC})$ *and* $\mathsf{MSOTh}(\mathfrak{C}) \approx_{\log} \mathsf{MSOTh}(\mathfrak{LC})$.
(2) $\mathfrak{C}$ *is an acyclic graph with root ε.*
(3) $\mathfrak{C}$ *has unbounded (in-)degree.*

Proof. At first, we prove (1). Let the duality function $\delta\colon \Sigma^* \to \Sigma^*$ be defined as follows:

$$\delta(\varepsilon) = \varepsilon,\ \delta(au) = \delta(u)\overline{a}\,,\ \text{and}\ \delta(\overline{a}u) = \delta(u)a$$

for all $u \in \Sigma^*$ and $a \in A$. In other words, δ reverses the order of the actions and inverts writing and reading of a letter a. From [13, Proposition 3.4] we know $u \equiv v$ iff $\delta(u) \equiv \delta(v)$. Hence, δ is an anti-morphism on $\mathcal{Q}$ and $(p, q) \in E_\alpha$ iff $(\delta(p), \delta(q)) \in F_{\delta(\alpha)}$ for all $p, q \in \mathcal{Q}$ and $\alpha \in \Sigma$. Let $\varphi \in \mathsf{FO}[(E_\alpha)_{\alpha\in\Sigma}]$ ($\varphi \in \mathsf{MSO}[(E_\alpha)_{\alpha\in\Sigma}]$, resp.). We construct φ' by replacing any atom "$E_\alpha(x, y)$" in φ by "$F_{\delta(\alpha)}(x, y)$". Then $\mathfrak{C} \models \varphi(q_1, \ldots, q_k) \iff \mathfrak{LC} \models \varphi'(\delta(q_1), \ldots, \delta(q_k))$ for any $q_1, \ldots, q_k \in \mathcal{Q}$. In particular, $\varphi \in \mathsf{FOTh}(\mathfrak{C})$ iff $\varphi' \in \mathsf{FOTh}(\mathfrak{LC})$ (resp. $\varphi \in \mathsf{MSOTh}(\mathfrak{C})$ iff $\varphi' \in \mathsf{MSOTh}(\mathfrak{LC})$). Finally, the converse reduction is symmetric to the one described above.

Now, we prove (2). Due to [13, Corollary 4.7] we have $p\mathcal{R}q$ iff $p = q$ for all $p, q \in \mathcal{Q}$. Then, by the remark above $p, q \in \mathcal{Q}$ are strongly connected iff $p = q$, i.e., there are no cycles in $\mathfrak{C}$.

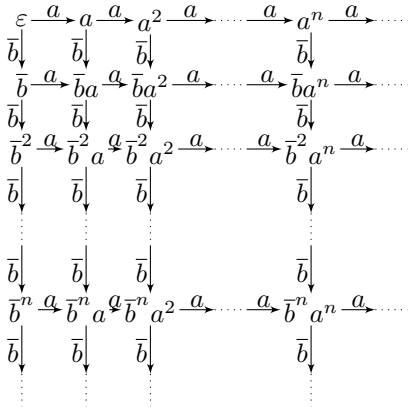


Figure 1 $\mathfrak{C}$ restricted to the nodes reachable by a- and $\overline{b}$-edges, only.

Next, to prove (3) let $n \in \mathbb{N}$ and $a, b \in A$ with $a \neq b$. Set $w_k = \overline{a^k}(a\overline{a})^{n-k}a^k$ for any $0 \leqslant k \leqslant n$. Then $w_k \equiv w_\ell$ (i.e. $[w_k] = [w_\ell]$) iff $k = \ell$ for any $0 \leqslant k, \ell \leqslant n$. By Theorem 3.6 we have $\chi(w_k\overline{b}) = (a^n b, \varepsilon, a^n)$, i.e. $w_k\overline{b} \equiv w_\ell\overline{b}$ for any $0 \leqslant k, \ell \leqslant n$. Hence, we have $([w_k], [\overline{a^n}ba^n]) \in E_{\overline{b}}$ for all $0 \leqslant k \leqslant n$, i.e., the node $[\overline{a^n}ba^n]$ has in-degree $> n$. ◀

By $\mathfrak{G}_n$ we denote the $n \times n$-*grid* for $n \in \mathbb{N}$. This is an undirected graph with n^2 many nodes which we denote by $v_{i,j}$ for any $1 \leqslant i, j \leqslant n$. Thereby, we have an edge between $v_{i,j}$ and $v_{k,\ell}$ if, and only if, $|j - \ell| + |i - k| = 1$ holds. Additionally, for a Γ-labeled, directed graph $\mathfrak{G} = (V, (E_a)_{a\in\Gamma})$ we denote the unlabeled and undirected version by $\mathsf{ud}(\mathfrak{G}) = (V, E)$. Here, we have an edge $(v, w) \in E$ if, and only if, there is an $a \in \Gamma$ such that $(v, w) \in E_a$ or $(w, v) \in E_a$. Then, in $\mathsf{ud}(\mathfrak{C})$ we can find $\mathfrak{G}_n$ for any $n \in \mathbb{N}$:

▶ **Proposition 3.10.** *$\mathfrak{G}_n$ is an induced subgraph of $\mathsf{ud}(\mathfrak{C})$ for any $n \in \mathbb{N}$.*

Proof. Let $a, b \in A$ be distinct. Then the submonoid $\mathcal{M}$ of $\mathcal{Q}$ generated by a and $\overline{b}$ is the free commutative monoid on $\{a, \overline{b}\}$ by Theorem 3.3(1). Its Cayley-graph $\mathfrak{C}(\mathcal{M}, \{a, \overline{b}\})$ is an infinite grid with labeled, directed edges. Then, $\mathfrak{G}_n$ is an induced subgraph of $\mathsf{ud}(\mathfrak{C}(\mathcal{M}, \{a, \overline{b}\}))$. Since in $\mathfrak{C}$ there are no edges with labels other than a or $\overline{b}$ between the nodes from $\mathcal{M}$, $\mathsf{ud}(\mathfrak{C}(\mathcal{M}, \{a, \overline{b}\}))$ is an induced subgraph of $\mathsf{ud}(\mathfrak{C})$ as well implying our claim. ◀

With the help of a famous result from Seese (cf. [28]), we may now prove the undecidability of the monadic second-order theory of the queue monoid's Cayley-graph.

▶ **Corollary 3.11.** $\mathsf{MSOTh}(\mathfrak{C})$ *is undecidable.*

Proof. Due to [26] each planar graph is a minor of some grid $\mathfrak{G}_n$. Since each $\mathfrak{G}_n$ is an induced subgraph of $\mathsf{ud}(\mathfrak{C})$ by Proposition 3.10, each planar graph is minor of an induced subgraph of $\mathsf{ud}(\mathfrak{C})$. Hence, by [28, Theorem 5] $\mathsf{MSOTh}(\mathsf{ud}(\mathfrak{C}))$ is undecidable. Since $\mathsf{ud}(\mathfrak{C})$ is first-order interpretable in $\mathfrak{C}$, $\mathsf{MSOTh}(\mathfrak{C})$ is undecidable as well. ◀

4 Combinatorics on Words

Before diving into the proof of the Cayley-graph's first-order theory we have to prove some combinatorial statements concerning words.

Let $\mathrm{pref}_r(u)$ denote the maximal prefix of u of length at most r. In a first lemma we prove that the complementary prefix and suffix of u resp. v wrt. $u \sqcap v$ can be shortened to words of length at most $2r$ having the same prefixes and suffixes. In terms of $\mathfrak{C}$'s first-order theory we only have to consider words $u \in \Sigma^*$ having "short" $\lambda(u)$ and $\varrho(u)$.

▶ **Lemma 4.1.** *Let $r \in \mathbb{N}$ and $u, v, w \in A^*$ with $uw \sqcap wv = w$. Then there are words u', v' of length $\leqslant 2r$ such that*

- $\mathrm{suf}_r(uw) = \mathrm{suf}_r(u'w)$,
- $\mathrm{suf}_r(wv) = \mathrm{suf}_r(wv')$,
- $\mathrm{pref}_r(wv) = \mathrm{pref}_r(wv')$, *and*
- $u'w \sqcap wv' = w$.

Proof. Set $u' = \mathrm{suf}_r(u)$. Additionally, if $|v| \leqslant 2r$ set $v' := v$, and otherwise, set $v' := \mathrm{pref}_r(v)\,\mathrm{suf}_r(v)$. Then the first three equations are obviously satisfied. Now assume $u'w \sqcap wv' \neq w$, i.e., there is $w' \in A^*$ with $|w'| > |w|$, $w' \preceq wv'$, and $w' \sqsubseteq u'w$. Since $|u'w| \leqslant r + |w|$ we have $w' \preceq w\,\mathrm{pref}_r(v) \preceq wv$. Additionally, we have $w' \sqsubseteq u'w \sqsubseteq uw$ implying $|uw \sqcap wv| \geqslant |w'| > |w|$. This is a contradiction to the definition of w. ◀

▶ Remark. The condition $uw \sqcap wv = w$ in Lemma 4.1 cannot be simplified to $u \sqcap v = \varepsilon$. For example, let $u = v = a$ and $w = baa$. Then only the first equation is satisfied.

A *period* of a word u is a word v such that $u \preceq v^\omega$. Obviously every word u has a unique smallest period, which we denote by $\sqrt{u}$. The *left-exponent* of $u \neq \varepsilon$ in v is the largest number n such that $v = u^n w$, and it is denoted by $\mathrm{lexp}(u, v)$. The *right-remainder*, $v \mod u$, of v with respect to u is defined as $(u^{\mathrm{lexp}(u,v)})^{-1}v$, that is the unique w such that $v = u^{\mathrm{lexp}(u,v)}w$. In particular we have $v = \sqrt{v}^{\mathrm{lexp}(\sqrt{v},v)}(v \mod \sqrt{v})$ for every $v \in A^*$. A word u is *primitive* if there is no v with $|v| < |u|$ and $u = v^n$ for some $n \in \mathbb{N}$. For $v, w \in A^*$ let $v\Delta w = (y, z)$, where y, z are minimal such that there exists an x with $v = xy$ and $w = xz$. For $\vec{v}, \vec{w} \in (A^*)^k$ let $\vec{v}\Delta\vec{w} = (v_1\Delta w_1, \ldots, v_k\Delta w_k) \in ((A^*)^2)^k$ and $|\vec{w}| \coloneqq \sum_{i=1}^k |w_i|$.

▶ **Definition 4.2.** Let $u \in A^*$ be a word. A word $v \in A^*$ is a *border* of u (denoted by $v \overset{\leq}{\sqsubseteq} u$) if $v \preceq u$ and $v \sqsubseteq u$. A *border-decomposition* of u is a sequence of words $\varepsilon = u_0, u_1, \ldots, u_n = u$ such that for all $0 \leqslant i < n$ it holds that $u_i \overset{\leq}{\sqsubsetneq} u_{i+1}$. A border-decomposition $u_0, u_1, \ldots, u_n$ is *complete* if there is no $1 \leqslant i < n$ and $v \in A^*$ with $u_i \overset{\leq}{\sqsubsetneq} v \overset{\leq}{\sqsubsetneq} u_{i+1}$.

Hence, a complete border-decomposition of $u \in A^*$ is the sequence of all borders of u ordered by word length. So, it is easy to observe that each word $u \in A^*$ has exactly one complete border-decomposition.

▶ **Example 4.3.** The complete border-decomposition of $ababa$ is $(\varepsilon, a, aba, ababa)$.

Let $u \in \Sigma^*$ be any element from the $\mathfrak{C}$ and $(u_0, \ldots, u_n)$ be the complete border-decomposition of $\mathsf{rd}(u) \sqcap \mathsf{wrt}(u)$. Then the characteristics $(\mathsf{rd}(u)u_i^{-1}, u_i, u_i^{-1}\mathsf{wrt}(u))$ describe all the words having the same projections to write and read actions, resp., as u. In the decidability proof of $\mathsf{FOTh}(\mathfrak{C})$ we consider these words since these are all close to each other in $\mathfrak{C}$.

From the complete border-decomposition of a word w we derive the so called skeleton of w containing the inner words v of all bordered words uvu in w.

▶ **Definition 4.4.** Let $w \in A^*$ and $\vec{w} = (w_0, \ldots, w_n)$ be the complete border-decomposition of w. The *r-skeleton* of w, denoted by $\mathcal{S}_r(w)$, is the word of length n over the alphabet $\Gamma = A^{\leqslant r}$ with $\mathcal{S}_r(w)[i] = \mathrm{pref}_r(w_i^{-1}w)$ for each $0 \leqslant i \leqslant n-1$. Note that $w_i^{-1}w$ is always defined since $w_i \preceq w$.

Note that it is convenient for our purpose to consider $\mathcal{S}_r(w)$ to be a word over an alphabet, which in itself consists of words of bounded length rather than to consider $\mathcal{S}_r(w)$ as a sequence of words.

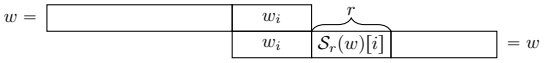


Figure 2 Definition of $\mathcal{S}_r(w)$.

▶ **Example 4.5.** Let $u = bababa$ and $v = ababab$. Then $u \sqcap v = ababa$ and the complete border-decomposition of $u \sqcap v$ is $(\varepsilon, a, aba, ababa)$. The 2-skeleton of $u \sqcap v$ is the word depicted below.

$$ab \longrightarrow ba \longrightarrow ba$$

Skeletons will play a crucial role in Section 5. We will prove the decidability of the Cayley-graph of a queue-monoid by translating back and forth between an Ehrenfeucht-Fraïssé game played on the Cayley-graph (presented as EF-relations) and games played on certain skeletons which are derived from the game played on the Cayley-graph.

▶ **Lemma 4.6.** *Let $r \in \mathbb{N}$, $w \in A^*$ and $n \in \mathbb{N}$ be the length of $\mathcal{S}_r(w)$. Then a word $v \in A^*$ can be constructed from w such that $|v| = \mathcal{O}(2^{nr})$ and $\mathcal{S}_r(w) = \mathcal{S}_r(v)$.*

Proof. Let $\vec{w} = (w_0, \ldots, w_n)$ be the complete border-decomposition of w. At first, assume $|\mathcal{S}_r(w)[n-1]| < r$ (i.e., the last component is small). Then there are two possibilities: on the one hand $w = w_{n-1}xw_{n-1}$ and $|xw_{n-1}| < r$. In this case we have $|w| < 2r = \mathcal{O}(2^{nr})$. On the other hand we have $w = xw_{n-1} = w_{n-1}y$ where $|x| = |y| < \min\{|w_{n-1}|, r\}$, i.e., the prefix and the suffix w_{n-1} overlap in w_n. Then it is easy to see that x is a period of w_{n-1} and of w_n. Concretely, there is a prefix p of x and a number $k \in \mathbb{N}$ such that $w = x^k p$ and $w_{n-1} = x^{k-1}p$. In particular, all word $x^i p$ with $1 \leqslant i \leqslant k$ are borders of w which implies $k \leqslant n$. Hence we have $|w| \leqslant |x| \cdot (k+1) \leqslant r \cdot (n+1) = \mathcal{O}(2^{nr})$. Therefore, in both cases we are ready and we can assume $|\mathcal{S}_r(w)[n-1]|$ from now on.

We construct v inductively as follows: We set $v_0 := \varepsilon$. Now let $a, b \in A$ be distinct with $\mathcal{S}_r(w)[0] \in aA^*$. Then $x \precneqq \mathcal{S}_r(w)[0]b^{2n+r}$ implies $x = \varepsilon$. Hence, we set, for $0 \leqslant i < n$, $v_{i+1} := v_i x_i v_i$ where $x_i = \mathcal{S}_r(w)[i]\, b^{n-i}a^i b^{n+r}$. Finally, we set $v := v_n$.

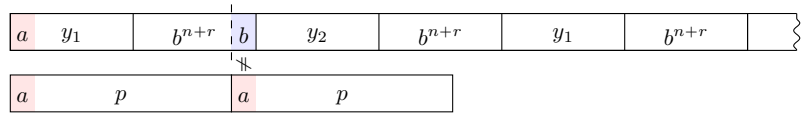
Before we can prove $\mathcal{S}_r(w) = \mathcal{S}_r(v)$ we need to prove the following two properties of $(v_0, \ldots, v_n)$:

(a) For each $0 \leqslant i \leqslant n$ $\sqrt{v_{i+1}} = v_i x_i$ and

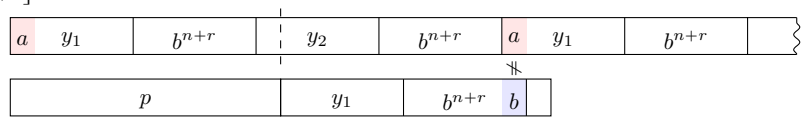
(b) $\vec{v} = (v_0, \ldots, v_n)$ is a complete border-decomposition of v.

Proof of (a). We observe that $v_i x_i$ is a period of v_{i+1} and we prove by induction on $0 \leqslant i \leqslant n$ that this period is minimal. For $i = 0$ this is trivial since $v_1 \in aA^{r-1}b^{2n+r}$ and $a \neq b$. So now let $i > 0$. We suppose that there is a period p of v_{i+1} with $|p| < |v_i x_i|$. Then, for $y_j := x_j(b^{n+r})^{-1}$ for $0 \leqslant j \leqslant i$, the word v_{i+1} is an alternation of words y_j and b^{r+n} which are all of length $r+n$. Note that by construction we have $y_j \neq b^{n+r}$ (since each y_j contains at least one a) as well as $y_j \neq y_k$ if $j \neq k$ for each $0 \leqslant j, k \leqslant i$. Additionally, each second occurrence of a y_j-block is y_1. We now consider two cases:

First, assume that $|p|$ is not a divisor of $n+r$. If $|p| < n+r$ then the distance between each two occurrences of a in p^ω is at most $|p| < n+r$ but v_{i+1} contains at least one b^{n+r}-block. Hence, we have $|p| > n+r$. If $\lfloor \frac{|p|}{n+r} \rfloor$ is odd (cf. Fig. 3a), p starts with a and ends in a block of the form b^{n+r}, but does not contain all of these $n+r$ many b's. Since p start with an a, a first repetition of p this first a is different from the b at this position in v_{i+1}, i.e., p is not a period of v_{i+1}. Otherwise, if $\lfloor \frac{|p|}{n+r} \rfloor$ is even (cf. Fig. 3b), then the prefix of $p^{-1}v_{i+1}$ of length $|p|$ contains at most one y_1-block and this overlaps with a b^{n+r}-block. Hence, there



(a) Case $\left\lfloor \frac{|p|}{n+r} \right\rfloor$ is odd.



(b) Case $\left\lfloor \frac{|p|}{n+r} \right\rfloor$ is even.

Figure 3

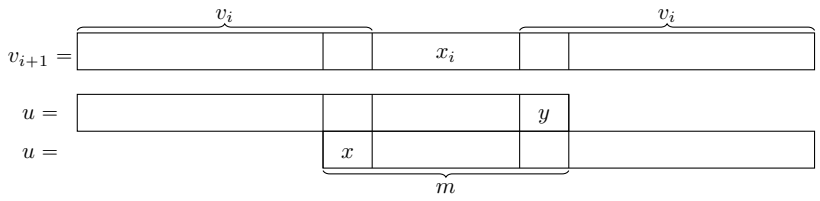


Figure 4

is a position in the first repetition of p containing a b which is different from the a at this position in v_{i+1}.

Now, assume $|p|$ is a divisor of $n+r$. Then we can understand the blocks of length $n+r$ as letters of the alphabet $\{b^{n+r}, y_1, \ldots, y_i\}$. Since there is no y_i-block in v_i we have $|p| \geqslant |v_i y_i|$. Since p starts with y_1 and y_i is followed by b^{n+r}, p has length at least $|v_i x_i|$.

Proof of (b). By construction, it is easy to see that $\vec{v} = (v_0, \ldots, v_n)$ is a border-decomposition of $v = v_n$. We prove now by induction on $0 \leqslant i < n$ that $(v_0, \ldots, v_{i+1})$ is a complete border-decomposition of v_i. The case $i = 0$ is easy to verify since $v_1 \in aA^{r-1}b^{2n+r}$. So, let $i \geqslant 1$. Assume there is $u \in A^*$ with $v_i \lessdot u \lessdot v_{i+1}$. Let u be of minimal length satisfying this inequality. Then there are two possible cases:

First, suppose $|u| \geqslant |v_i x_i|$ holds, i.e., the prefix and suffix u overlap in v_i and the overlap contains at most x_i (cf. Fig. 4). Let $x, y \in A^*$ such that $u = xx_iv_i = y$. Then we have $|x| = |y|$ and $m := xx_iy \lessdot u$. Hence, by minimality of u we have $|m| \leqslant |v_i|$ and therefore, by induction hypothesis, $m = v_k$ for some $0 < k \leqslant i$. This implies

$$v_{k-1}x_{k-1}v_{k-1} = v_k = m = xx_iy\,.$$

Since $|x| = |y|$ and $|x_i| = |x_{k-1}|$ we have $x_i = x_{k-1}$, which is a contradiction to the construction of the x_i's.

Now, suppose $|u| < |v_i x_i|$. If $|u| \geqslant \frac{|v_{i+1}|}{2}$ (i.e., the prefix and suffix u in v_i overlap) then there is a word $m \in A^*$ such that $m \lessdot u$ holds. Hence, by minimality of u and by induction hypothesis we have $m = v_k$ for some $0 \leqslant k \leqslant i$. Since $|m| < |x_i| = |x_1|$ we have $m = \varepsilon$, i.e., we have $|u| = \frac{|v_{i+1}|}{2}$.

Suppose $|u| \leqslant \frac{|v_{i+1}|}{2}$ (i.e., the prefix and suffix u in v_i do not overlap). Then there is a word $p \in A^*$ such that $v_{i+1} = pu$. Since u is a prefix of v_{i+1} and $|p| > \frac{|v_{i+1}|}{2}$, u also is a prefix of p. Hence, p is a period of v_{i+1} and we have

$$|p| = |v_{i+1}| - |u| < |v_{i+1}| - |v_i| = |v_i x_i|\,.$$

This is a contradiction to property a stating that $v_i x_i$ is the minimal period of v_{i+1}.

So, in both cases we have seen that there is no $v_i \lneq u \lneq v_{i+1}$, i.e., $(v_0, \ldots, v_{i+1})$ is a complete border-decomposition.

Finally, let $0 \leqslant i < n$. Then we have

$$\mathcal{S}_r(v)[i] = \mathrm{pref}_r(v_i^{-1} v) = \mathrm{pref}_r(\mathcal{S}_r(w)[i]\, s) = \mathcal{S}_r(w)[i]$$

for some $s \in A^*$, i.e., $\mathcal{S}_r(v) = \mathcal{S}_r(w)$. Additionally, we have $|v_i| = 2|v_{i-1} + 2n + 2r$ for $1 \leqslant i \leqslant n$ and $|v_0| = 0$ which results in $|v| = |v_n| = (2^n - 1)(2n + 2r) = \mathcal{O}(2^{nr})$. ◀

Let $V \in (A^{\leqslant r})^*$ be the r-skeleton of some word $w \in A^*$. We call the word $v \in A^*$ constructed in the proof of Lemma 4.6 the *r-instantiation of V*.

5 Decidability of the FO-Theory

Recall that the Cayley-graph of the queue monoid $\mathcal{Q}$ induced by A is denoted by $\mathfrak{C} = (\mathcal{Q}, (E_\alpha)_{\alpha \in \Sigma})$. In order to ease the notation we let elements of $\mathfrak{C}$ inherit some properties from their projections to the read and write actions. For $p, q \in \mathcal{Q}$ let $|p| = |(\mathsf{rd}(p), \mathsf{wrt}(p))|$, $p \Delta q = (\mathsf{rd}(p), \mathsf{wrt}(p)) \Delta (\mathsf{rd}(q), \mathsf{wrt}(q))$, and we call $|p \Delta q|$ the *(Δ-)distance* of p and q. Note that Δ defines a metric on $\mathfrak{C}$. Further for $\vec{p} = (p_1, \ldots, p_k) \in \mathcal{Q}^k$ let $\mathcal{N}_r(\vec{p}) = \{q \in \mathcal{Q} \mid \exists 1 \leqslant i \leqslant k \colon |p_i \Delta q| \leqslant r \vee |q| \leqslant r\}$ be the *(Δ-)neighborhood* of $\vec{p}$ of radius r (*r-neighborhood*). Note that we implicitly add the origin of $\mathfrak{C}$ to $\vec{p}$ when we compute the neighborhood. Moreover we define the notion of a border-decomposition and an r-skeleton for an element $p \in \mathcal{Q}$ as the border-decomposition and the r-skeleton of $\mathsf{rd}(p) \sqcap \mathsf{wrt}(p)$.

Let us first give an intuitive outline of our decidability proof. We follow a classical proof strategy due to Ferrante and Rackoff [8]. Roughly speaking we show that there is some fixed primitive recursive function $f : \mathbb{N} \to \mathbb{N}$ such that for every two $(r+1)$-equivalent tuples $\vec{p}, \vec{q} \in \mathcal{Q}^n$ and every $p \in \mathcal{Q}$ there is a q in the $f(r+1)$-neighborhood of the tuple $\vec{q}$ such that $(\vec{p}, p) \equiv_r (\vec{q}, q)$. This implies that in order to evaluate a formula $Qx\varphi(\vec{p})$ where φ has quantifier rank r and $Q \in \{\exists, \forall\}$ we can restrict the quantification of x to the $f(r+1)$-neighborhood of $\vec{p}$. Since the r-neighborhood of each element $p \in \mathcal{Q}$ is finite and effectively computable for every radius r, we can use the above observation to implement a decision procedure for the theory of $\mathfrak{C}$. In order to achieve this goal we exploit the fact that first-order logic cannot measure distances between two nodes that are more than exponentially far away in the quantifier rank. Therefore our task for a given quantifier rank $r > 0$ is to find for every p that is far away from a tuple $\vec{p}$ an element p' that is closer (but not yet too close) to $\vec{p}$ such that the neighborhoods of p and p' of a suitably chosen radius are not distinguishable with the remaining quantifier rank $r - 1$. What makes this task more complex than for most other examples of Cayley-graphs with decidable first-order theory that can be found in the literature is that the Cayley-graph of the queue monoid is in some sense less local. In fact, the neighborhood-structure of an element p does not only depend on suffixes of bounded length of $\mathsf{rd}(p)$ and $\mathsf{wrt}(p)$ (as it would be the case for instance for the direct product of two free monoids). We solve this problem via the notion of skeletons. Our proof reveals that the r-type of the 2^{r+1}-neighborhood of an element p is basically determined by the $(r+1)$-type of the $3 \cdot 2^{r+1}$-skeleton of $\mathsf{rd}(p) \sqcap \mathsf{wrt}(p)$. This will be the core of our proof.

Let us start off by making some technical preparations in order to formulate the core idea precisely.

▶ **Definition 5.1.** Let V be an r-skeleton. We say that $q \in \mathcal{Q}$ is *compatible* with V if V has an instantiation v such that $\mathsf{rd}(q) \sqcap \mathsf{wrt}(q) = vx$ for some $x \in A^{\leqslant r}$ and $|\mathsf{wrt}(q) \Delta v| \leqslant r$.

Intuitively, q being compatible to an r-skeleton V means that we can obtain an element q' with r-skeleton V by deleting up to r many read actions and modifying the write actions arbitrarily up to distance r. We use this notion in order to translate elements of the Cayley-graph into positions of an r-skeleton. Next we describe how we translate back and forth between elements of the Cayley-graph and positions in a skeleton. However we can not guarantee that every element in close proximity to a given element p can be associated with a position in the r-skeleton of p because small changes to the read and write actions might change the border-decomposition dramatically. But we can modify r and p slightly to circumvent this problem.

▶ **Definition 5.2.** For $q \in \mathcal{Q}$ with $|\mathsf{rd}(q)| \geqslant r$ let $\mathrm{rc}_r(q)$ be the element q' with $\mathsf{wrt}(q') = \mathsf{wrt}(q)$, $\mathsf{rd}(q') = \mathsf{rd}(q)\,\mathrm{suf}_r(\mathsf{rd}(q))^{-1}$, and $\mu(q') = \mathsf{rd}(q') \sqcap \mathsf{wrt}(q')$. In other words, rc_r just cuts the last r read actions and pushes read and write actions as far together as possible.

▶ **Definition 5.3.** Let $p, q \in \mathcal{Q}$ and let U and V be the $3r$-skeletons of $\mathrm{rc}_{2r}(p)$ and $\mathrm{rc}_{2r}(q)$, respectively. If we suppose that $(m_1, \ldots, m_k)$ are positions in V and $(n_1, \ldots, n_k)$ are positions in U such that $(U, m_1, \ldots, m_k) \equiv_\ell (V, n_1, \ldots, n_k)$ for some $\ell \geqslant 1$. For $p' \in \mathcal{Q}$ with $|p'\Delta p| \leqslant r$ and $|\mu(p')| \geqslant 2r$ we associate a position m_{k+1} in U as follows: Let $(u_1, \ldots, u_m)$ be the complete border-decomposition of $\mathsf{rd}(\mathrm{rc}_{2r}(p))$ and $(v_1, \ldots, v_n)$ be the complete border-decomposition of $\mathsf{rd}(\mathrm{rc}_{2r}(q))$. As p' has distance at most r from p we have that $\mathsf{rd}(p') = \mathsf{rd}(\mathrm{rc}_{2r}(p))x$ for some $x \in A^{\leqslant 2r}$. Therefore there is an $i \leqslant m$ such that $\mu(p') = u_i x$. Then i is the position that is associated with p'.

Now let n_{k+1} be such that $(U, m_1, \ldots, m_{k+1}) \equiv_{\ell-1} (V, n_1, \ldots, n_{k+1})$ we associate an element q' with n_{k+1} as follows: Let q' be the element with $\mathsf{rd}(q') = \mathsf{rd}(\mathrm{rc}_{2r}(q))u_{m_{k+1}}^{-1}\mu(p')$, $\mathsf{wrt}(q')\Delta\mathsf{wrt}(\mathrm{rc}_r(q)) = \mathsf{wrt}(p')\Delta\mathsf{wrt}(\mathrm{rc}_{2r}(p))$, and $\mu(q') = v_{m_{k+1}}u_{n_{k+1}}^{-1}\mu(p')$. Note that q' is well defined since $V[j]$ is labeled by $\mathrm{pref}_{2^{r+2}}(u_i^{-1}\mu(p))$. Therefore $v_j\,\mathrm{pref}_{2^{r+1}}(v_i^{-1}\mu(p))$ is a prefix of $\mathsf{wrt}(q')$ by construction.

Another important ingredient of our proof is to construct "small" r-equivalent words from a given word w. This is routine since it can be achieved by a simple automata-theoretic approach.

▶ **Lemma 5.4** ([29])**.** *From a given alphabet Γ, a word $v \in \Gamma^*$, and $r \in \mathbb{N}$ one can compute an automaton $\mathcal{A}$ in time $\exp_{r+1}(f(r))$ with $L(\mathcal{A}) = \{w \in \Gamma^* \mid w \equiv_r v\}$ for some primitive recursive function f.*

Proof sketch. Construct a first-order formula φ that characterizes the r-type of v. From φ compute an automaton $\mathcal{A}_\varphi$ with $L(\mathcal{A}_\varphi) = \{w \in \Gamma^* \mid w \equiv_r v\}$. One easily show via induction on r that the size of the automaton $\mathcal{A}$ is at most $\exp_{r+1}(2, f(r))$ where $f(r)$ is an upper bound for the size of the formula φ (which can be chosen to be primitive recursive). ◀

We use this idea to define a family of equivalence relations $(\mathcal{E}_m^r)_{r,m\in\mathbb{N}}$. For $r, m \in \mathbb{N}$ and $\vec{p}, \vec{q} \in \mathcal{Q}^m$ let $\vec{p}\,\mathcal{E}_m^r\,\vec{q}$ iff

(1) If $|p_i\Delta\varepsilon| \leqslant 4\exp_{r+2}(2, f(r))$ then $p_i = q_i$ where f is the function from Lemma 5.4.

(2) $|p_i\Delta p_j| =_{2^r} |q_i\Delta q_j|$ for all $1 \leqslant i, j \leqslant m$ and if $|p_i\Delta p_j| \leqslant 2^r$ then also $p_i\Delta p_j = q_i\Delta q_j$.

(3) There is a partition $X_1, \ldots, X_k$ of $\{1, \ldots, m\}$ such that for $X \neq X' \in \{X_1, \ldots, X_k\}$ it holds that with $\min = \min X$:

- **(a)** If $i \in X$, $j \in X'$ it holds that $|p_i\Delta p_j| > 2^r$ (and therefore $|q_i\Delta q_j| > 2^r$).
- **(b)** $\mathrm{suf}_{2^{r+m+2}}(\mathsf{rd}(p_i)) = \mathrm{suf}_{2^{r+m+2}}(\mathsf{rd}(q_i))$ and $\mathrm{suf}_{2^{r+m+2}}(\mathsf{wrt}(p_i)) = \mathrm{suf}_{2^{r+m+2}}(\mathsf{wrt}(q_i))$ for all $i \in X$.

(c) For all $j \in X$ it holds that $|p_{\min}\Delta p_j| \leqslant \sum_{s=r}^{r+m} 2^s$ (and therefore also $|q_{\min}\Delta q_j| \leqslant \sum_{s=r}^{r+m} 2^s$).

(d) Let U be the $3 \cdot 2^{r+m+1}$-skeleton of $\mathrm{rc}_{2^{r+m+2}}(p_{\min})$ and V be the $3 \cdot 2^{r+m+1}$-skeleton $\mathrm{rc}_{2^{r+m+2}}(q_{\min})$. Then for all $j \in X$ we have that either $\mu(p_j) = \mu(q_j)$ or $|\mu(p_j)| \geqslant 2^{r+m+2}$ and p_j is compatible with U and q_j is compatible with V. Further if $m_1, \ldots, m_k$ are the positions in U that are associated with $\{p_j \mid j \in X\}$ and $n_1, \ldots, n_k$ are the positions in U that are associated with $\{q_j \mid j \in X\}$ then $(V, m_1, \ldots, m_k) \equiv_{r+1} (U, n_1, \ldots, n_k)$.

We show that $(\mathcal{E}^r_m)_{r,m\in\mathbb{N}}$ are indeed EF-relations for $\mathfrak{C}$.

▶ **Lemma 5.5.** *For all $m \in \mathbb{N}_{>0}$ and all $\vec{p}, \vec{q} \in \mathcal{Q}^m$: If $\vec{p}\mathcal{E}^0_m\vec{q}$ then the mapping $p_i \mapsto q_i$ is a partial isomorphism.*

Proof. We need to show that $(p_i, p_j) \in E_a \Rightarrow (q_i, q_j) \in E_a$ for all $i, j \leqslant m$ and all $a \in \Sigma$. Let $\vec{p}, \vec{q} \in \mathcal{Q}^m$ with $\vec{p}\mathcal{E}^0_m\vec{q}$. Suppose $(p_i, p_j) \in E_a$ for some $a \in \Sigma$. Then $|p_i\Delta p_j| = 1$. Hence $p_i\Delta p_j = q_i\Delta q_j$ by (2). Let $X_1, \ldots, X_k$ be the partition from Property 3. Since the distance between p_i and p_j and between q_i and q_j is 1 we derive from Property (3a) that i and j belong to the same $X \in \{X_1, \ldots, X_k\}$. Let $\ell = \min X$. If $|\mu(p_i)| < 2^{m+2}$ then, by Property (3d) and (3b), $\mu(p_i) = \mu(q_i)$. In this case $(p_i, p_j) \in E_a \Leftrightarrow (q_i, q_j) \in E_a$ obviously holds. Otherwise there are $3 \cdot 2^{m+1}$-skeletons U, V such that p_i and p_j can be translated into positions m_1, m_2 in U and q_i and q_j can be translated into position n_1, n_2 in V such that $(U, m_1, m_2) \equiv_1 (V, n_1, n_2)$. There are two possible types of configurations for p_i and p_j such that they can be connected by an edge. First, it might be the case that $\mathsf{rd}(p_i) = \mathsf{rd}(p_j)$, $\mathsf{wrt}(p_i)a = \mathsf{wrt}(p_j)$, and $\mu(p_i) = \mu(p_j)$. In this case $m_1 = m_2$ and therefore $n_1 = n_2$, which implies that $\mathsf{rd}(q_i) = \mathsf{rd}(q_j)$, $\mathsf{wrt}(q_i)a = \mathsf{wrt}(q_j)$, and $\mu(q_i) = \mu(q_j)$. Therefore $(q_i, q_j) \in E_a$.

Second, it might be that $\mathsf{rd}(p_i)a = \mathsf{rd}(p_j)$ (where $a = \overline{b}$), $\mathsf{wrt}(p_i) = \mathsf{wrt}(p_j)$, and $\mu(p_j)a^{-1}$ is the largest suffix w of $\mu(p_i)$ such that wa is a prefix of $\mathsf{wrt}(p_i)$. This property can be translated into the formula of quantifier rank 1. Let $(w_0, \ldots, w_n)$ be the complete border-decomposition of $\mathrm{rc}_{2^{m+2}}(p_\ell)$ and $v := w_{m_1}^{-1}\mu(p_i) \in A^{\leqslant 3\cdot 2^{m+1}}$. Then

$$\begin{aligned}&\varphi(x_1, x_2)\\ &:= x_2 \leqslant x_1 \wedge \bigvee_{s\in A^{\leqslant 3\cdot 2^{m+1}}:(va)\preceq s} P_s(x_2) \wedge \forall y : \left(x_2 < y < x_1 \rightarrow \bigwedge_{s\in A^{\leqslant 3\cdot 2^{m+1}}:va\preceq s} \neg P_s(y)\right).\end{aligned}$$

Hence $U \models \varphi(m_1, m_2)$ and since $(U, m_1, m_2) \equiv_1 (V, n_1, n_2)$ also $V \models \varphi(n_1, n_2)$ and therefore $(q_i, q_j) \in E_a$. ◀

▶ **Lemma 5.6.** *For all $m, r \in \mathbb{N}$ and all $\vec{p}, \vec{q} \in \mathcal{Q}^m$:*

$$\vec{p}\mathcal{E}^{r+1}_m\vec{q} \Rightarrow \forall p \in \mathcal{Q}\exists q \in \mathcal{N}_{\exp_{r+3}(g(r+m))}(\vec{q}) : (\vec{p}, p)\mathcal{E}^r_{m+1}(\vec{q}, q)$$

for some primitive recursive function g.

Proof. Let f be the primitive recursive function from Lemma 5.4. Let $\vec{p}, \vec{q} \in \mathcal{Q}^m$ with $(\vec{p}, \vec{q}) \in \mathcal{E}^{r+1}_m$ and let $X_1, \ldots, X_k$ be a partition of $\{1, \ldots, m\}$ with the properties described in (3). Consider $p \in \mathcal{Q}$. We distinguish three cases. If p has distance $\leqslant 4\exp_{r+2}(2, f(r))$ from ε then we choose $q = p$.

From now on suppose p has distance $> 4\exp_{r+2}(2, f(r))$ from ε. We consider the case that p has distance $> 2^r$ from every p_i. Since the distance from ε is exactly $|\overline{\pi}_1(p)|+2|\mu(p)|+|\varrho(p)|$ it follows that $|\overline{\pi}_1(p)| > \exp_{r+2}(f(r))$ or $|\mu(p)| > \exp_{r+2}(f(r))$ or $|\varrho(p)| > \exp_{r+2}(f(r))$. Let

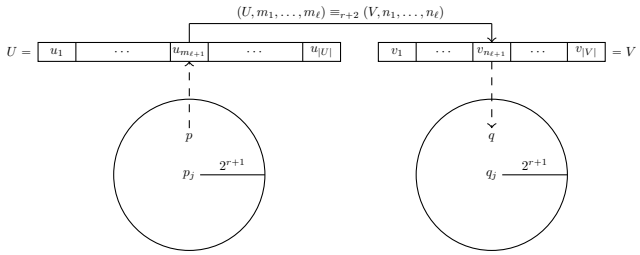


Figure 5 Construction of q from p using U and V.

$p' = \mathrm{rc}_{2^{r+m+2}}(p)$. Consider the $3 \cdot 2^{r+m+1}$-skeleton $V = \mathcal{S}_{3 \cdot 2^{r+m+1}}(p')$. By Lemma 5.4 we can find a $3 \cdot 2^{r+m+1}$-skeleton W of length at most $(m+1)\exp_{r+2}(f(r+1))$ with $V \equiv_{r+1} W$ and $3 \cdot 2^{r+m+1}$-instantiation w with $|w| \leqslant c \cdot 2^{(m+1)\exp_{r+2}(f(r+1)) \cdot 3 \cdot 2^{r+m+1}} \leqslant \exp_{r+3}(g(r+m))$ (for a suitable primitive recursive function g). Using Lemma 4.1, words u, v of length at most $(m+1)2^{r+m+3}$ such that

1. $\mathrm{suf}_{2^{r+m+2}}(uw) = \mathrm{suf}_{2^{r+m+2}}(\mathsf{rd}(p)\,\mathrm{suf}_{2^{r+m+2}}(\mathsf{rd}(p))^{-1})$
2. $\mathrm{suf}_{2^{r+m+2}}(wv) = \mathrm{suf}_{2^{r+m+2}}(\mathsf{wrt}(p))$
3. $\mathrm{pref}_{2^{r+m+2}}(wv) = \mathrm{pref}_{2^{r+m+2}}(\mathsf{wrt}(p))$
4. $uw \sqcap wv = w$

such that every element x with $\mathsf{rd}(x) = uw$ and $\mathsf{wrt}(x) = wv$ has distance $> 2^r$ from every q_i. We choose to q to be such an element x. It remains to specify $\mu(x)$. if $|\mu(p)| \leqslant 2^{r+m+2}$ then choose $\mu(q) = \mu(p)$. Otherwise let $(v_0, v_1, \ldots, v_m)$ be the complete border-decomposition of p' and let $(w_0, w_1, \ldots, w_n)$ be the complete border-decomposition of w. Let i be the index of $\mu(p')$ in $(v_0, v_1, \ldots, v_m)$. Because $\mathcal{S}_{3 \cdot 2^{r+m+1}}(p') \equiv_{r+1} W$ there is a $j \in \{0, \ldots, n\}$ such that $(\mathcal{S}_{3 \cdot 2^{r+m+1}}(p'), i) \equiv_r (W, j)$. Now choose $\mu(q) = w_j$. Finally extend the partition by $X_{k+1} = \{m+1\}$.

If p has distance $\leqslant 2^r$ from some p_i then let $Y \in \{X_1, \ldots, X_k\}$ be such that $i \in Y$ and let $j = \min Y$. Let U be the $3 \cdot 2^{r+m+1}$-skeleton of $\mathrm{rc}_{2^{r+m+2}}(p_j)$ and V be the $3 \cdot 2^{r+m+1}$-skeleton of $\mathrm{rc}_{2^{r+m+2}}(q_j)$. Since $|p_i \Delta p_j| \leqslant \sum_{s=r+1}^{r+m} 2^s$ and $|p \Delta p_i| \leqslant 2^r$ we conclude that $|p \Delta p_j| \leqslant \sum_{s=r}^{r+m} 2^s \leqslant 2^{r+m+1}$. Hence, p is compatible with U. Let $m_1, \ldots, m_\ell$ be the positions in U that are associated with the elements $\{q_s \mid s \in Y\}$, $m_{\ell+1}$ the position in U that is associated with p, and $n_1, \ldots, n_\ell$ be the positions associated with $\{q_s \mid s \in Y\}$ in V. Since $(U, m_1, \ldots, m_\ell) \equiv_{r+2} (V, n_1, \ldots, n_\ell)$ by Property (3d) there exists a $n_{\ell+1}$ with $(U, m_1, \ldots, m_{\ell+1}) \equiv_{r+1} (V, n_1, \ldots, n_{\ell+1})$. From $n_{\ell+1}$ we compute the associated element q in the $(\sum_{s=r+m}^{r} 2^s)$-neighborhood of q_j. The construction of q ensures that Properties (3b) to (3) are fulfilled for $(\vec{p}, p)$ and $(\vec{q}, q)$ by adding $\ell + 1$ to Y. Hence $(\vec{p}, p)\mathcal{E}_m^r(\vec{q}, q)$. ◀

The Lemmata 5.5 and 5.6 ensure that $\mathcal{E}_m^r$-equivalent tuples are also r-equivalent.

▶ **Corollary 5.7.** *For all $\vec{p} \in \mathcal{Q}^m$, $p \in \mathcal{Q}$, and $r \in \mathbb{N}$ there exists an element $q \in \mathcal{N}_{\exp_{r+3}(g(r+m))}(\vec{p})$ with $(\mathfrak{C}, \vec{p}, p) \equiv_r (\mathfrak{C}, \vec{p}, q)$ for some polynomial f.*

▶ **Lemma 5.8.** *For every $p \in \mathcal{Q}$ and every r there are at most $|A|^{4r}(\min\{|\mathsf{rd}(p)|, |\mathsf{wrt}(p)|\} + r)$ many elements in the r-neighborhood of a node $p \in \mathcal{Q}$.*

Proof. Every element q in the r-neighborhood of p can be characterized by the tuple $p\Delta q = (u, v, w, x) \in (A^{\leqslant r})^4$ and $\mu(q)$. Once we have fixed $p\Delta q \in (A^{\leqslant r})^4$ (and therefore fixed $\mathsf{rd}(q)$ and $\mathsf{wrt}(q)$) there are at most $\min\{|\mathsf{rd}(q)|, |\mathsf{wrt}(q)|\} \leqslant \min\{|\mathsf{rd}(p)|, |\mathsf{wrt}(p)|\} + r$ possible values for $\mu(q)$. ◀

With this lemma we obtain our main result.

Table 1 Comparison of the decidability of logics on Cayley-graphs of fundamental data structures.

Data Structure	Transformation Monoid $\mathcal{M}$	$\mathsf{FOTh}(\mathfrak{C}(\mathcal{M}, \Gamma))$	$\mathsf{MSOTh}(\mathfrak{C}(\mathcal{M}, \Gamma))$
finite monoid	finite monoid	PSPACE [10]	PSPACE [10]
counter	$(\mathbb{Z}, +)$	2EXPSPACE [21]	decidable [20]
stack	polycyclic monoid	2EXPSPACE [21]	decidable [6, 20]
queue	queue monoid	primitive recursive	undecidable

▶ **Theorem 5.9.** $\mathsf{FOTh}(\mathfrak{C})$ *is primitive recursive.*

Proof. We use the standard model-checking algorithm for first-order logic but restrict quantification to the $\exp_{r+1}(2, f(r))$-neighborhood of the current variable assignment. The correctness of this procedure is guaranteed by Corollary 5.7. We see that the values $|\mathsf{rd}(p)|$ and $|\mathsf{wrt}(p)|$ are bounded by $\exp_{r+3}(g(r+m))$ Hence, by Lemma 5.8 the algorithm needs to consider at most $|A|^{4r}(\exp_{r+3}(g(r+m))+1)$ many Elements, which leads to a runtime of $|\varphi| \cdot (|A|^{4r}(\exp_{r+3}(g(r+m))+1))^r$, which is obviously a primitive recursive function. ◀

6 Conclusion and Open Problems

We studied the Cayley-graph of the queue monoid and the logics of these graphs. Concretely, we have shown the decidability of the Cayley-graph's first order theory and the undecidability of the monadic second-order theory. This answers a question from Huschenbett et al. in [13].

In Table 1 is a comparison of our results compared to other fundamental data structures.

There are still some questions open relating to the queue monoid: in this paper we have given a primitive recursive but non-elementary upper bound on the complexity of the first-order theory of the queue monoid's Cayley-graph. So, one may ask for tight upper and lower bounds. Another open question concern the automaticity of the queue monoid. While it is neither automatic in the sense of Khoussainov and Nerode [16] nor automatic in the sense of Thurston et al. [4] due to [13], we still do not know whether the Cayley-graph of the queue monoid is automatic. Finally, the decidability of the first-order theory of the (partially) lossy queue monoid's (cf. [17, 18]) Cayley-graph is left open as well and is worth to be studied.

References

1 Parosh A. Abdulla and Bengt Jonsson. Verifying Programs with Unreliable Channels. *Information and Computation*, 127(2):91–101, 1996. doi:10.1006/inco.1996.0053.

2 Benedikt Bollig. *Formal Models of Communicating Systems: Languages, Automata, and Monadic Second-Order Logic*. Springer, 2006. doi:10.1007/3-540-32923-4.

3 Daniel Brand and Pitro Zafiropulo. On Communicating Finite-State Machines. *Journal of the ACM*, 30(2), 1983. doi:10.1145/322374.322380.

4 James W Cannon, David BA Epstein, Derek F Holt, Silvio VF Levy, Michael S Paterson, and William P Thurston. Word Processing in Groups. *Jones and Barlett Publ., Boston, MA*, 1992.

5 Gérard Cécé, Alain Finkel, and S. Purushotaman Iyer. Unreliable Channels Are Easier to Verify than Perfect Channels. *Information and Computation*, 124(1):20–31, 1996. doi:10.1006/inco.1996.0003.

6 Christian Delhommé, Teodor Knapik, and D. Gnanaraj Thomas. Using Transitive–Closure Logic for Deciding Linear Properties of Monoids. In *Mathematical Foundations of Computer*

Science 2003, volume 2747 of *Lecture Notes in Computer Science*, pages 378–387. Springer, 2003. doi:10.1007/978-3-540-45138-9_32.

7 Andrzej Ehrenfeucht. An Application of Games to the Completeness Problem for Formalized Theories. *Fundamenta Mathematicae*, 49(129-141):13, 1961.

8 Jeanne Ferrante and Charles W. Rackoff. *The Computational Complexity of Logical Theories*. Number 718 in Lecture Notes in Mathematics. Springer, 1979. doi:10.1007/BFb0062837.

9 Roland Fraïssé. Sur Quelques Classifications Des Systèmes de Relations. *Université d'Alger, Publications Scientifiques, Série A*, 1:35–182, 1954.

10 Erich Grädel. Finite Model Theory and Descriptive Complexity. In *Finite Model Theory and Its Applications*, Texts in Theoretical Computer Science an EATCS Series, pages 125–230. Springer, 2007. doi:10.1007/3-540-68804-8_3.

11 James A. Green. On the Structure of Semigroups. *Annals of Mathematics*, pages 163–172, 1951.

12 Christoph Haase, Sylvain Schmitz, and Philippe Schnoebelen. The Power of Priority Channel Systems. *Logical Methods in Computer Science*, 10(4:4), 2014. doi:10.2168/LMCS-10(4:4)2014.

13 Martin Huschenbett, Dietrich Kuske, and Georg Zetzsche. The Monoid of Queue Actions. *Semigroup forum*, 95(3):475–508, 2017. doi:10.1007/s00233-016-9835-4.

14 Mark Kambites. Formal Languages and Groups as Memory. *Communications in Algebra*, 37(1):193–208, 2009. doi:10.1080/00927870802243580.

15 Olga Kharlampovich, Bakhadyr Khoussainov, and Alexei Miasnikov. From Automatic Structures to Automatic Groups. *Groups, Geometry, and Dynamics*, 8(1):157–198, 2014. doi:10.4171/GGD/221.

16 Bakhadyr Khoussainov and Anil Nerode. Automatic Presentations of Structures. In *Logic and Computational Complexity*, volume 960 of *Lecture Notes in Computer Science*, pages 367–392. Springer, 1995. doi:10.1007/3-540-60178-3_93.

17 Chris Köcher. Rational, Recognizable, and Aperiodic Sets in the Partially Lossy Queue Monoid. In *STACS'18*, volume 96 of *LIPIcs*, pages 45:1–45:14. Dagstuhl Publishing, 2018. doi:10.4230/LIPIcs.STACS.2018.45.

18 Chris Köcher, Dietrich Kuske, and Olena Prianychnykova. The Inclusion Structure of Partially Lossy Queue Monoids and Their Trace Submonoids. *RAIRO - Theoretical Informatics and Applications*, 52(1):55–86, 2018. doi:10.1051/ita/2018003.

19 Dietrich Kuske and Markus Lohrey. Logical Aspects of Cayley-Graphs: The Group Case. *Annals of Pure and Applied Logic*, 131(1):263–286, 2005. doi:10.1016/j.apal.2004.06.002.

20 Dietrich Kuske and Markus Lohrey. Logical Aspects of Cayley-Graphs: The Monoid Case. *International Journal of Algebra and Computation*, 16(02):307–340, 2006. doi:10.1142/S0218196706003001.

21 Dietrich Kuske and Markus Lohrey. Automatic Structures of Bounded Degree Revisited. *The Journal of Symbolic Logic*, 76(4):1352–1380, 2011. doi:10.2178/jsl/1318338854.

22 Benoît Masson and Philippe Schnoebelen. On Verifying Fair Lossy Channel Systems. In *MFCS'02*, volume 2420 of *Lecture Notes in Computer Science*, pages 543–555. Springer, 2002. doi:10.1007/3-540-45687-2_45.

23 David E. Muller and Paul E. Schupp. The Theory of Ends, Pushdown Automata, and Second-Order Logic. *Theoretical Computer Science*, 37:51–75, January 1985. doi:10.1016/0304-3975(85)90087-8.

24 Paliath Narendran and Friedrich Otto. Some Results on Equational Unification. In *10th International Conference on Automated Deduction*, volume 449 of *Lecture Notes in Computer Science*, pages 276–291. Springer, 1990. doi:10.1007/3-540-52885-7_94.

25 Jean-Éric Pin. Mathematical Foundations of Automata Theory. *Lecture notes LIAFA, Université Paris*, 7, 2010.

26 Neil Robertson and Paul D. Seymour. Graph Minors, Part III: Planar Tree-Width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984. doi:10.1016/0095-8956(84)90013-3.

27 Jacques Sakarovitch. Kleene's Theorem Revisited. In *Trends, Techniques, and Problems in Theoretical Computer Science*, volume 281 of *Lecture Notes in Computer Science*, pages 39–50. Springer, 1986. doi:10.1007/3540185356_29.

28 D. Seese. The Structure of the Models of Decidable Monadic Theories of Graphs. *Annals of Pure and Applied Logic*, 53(2):169–195, 1991. doi:10.1016/0168-0072(91)90054-P.

29 Wolfgang Thomas. Languages, Automata, and Logic. In Grzegorz Rozenberg and Arto Salomaa, editors, *Beyond Words*, volume 3 of *Handbook of Formal Languages*, pages 389–455. Springer, 1997. doi:10.1007/978-3-642-59126-6_7.

Uniformly Automatic Classes of Finite Structures

Faried Abu Zaid[1]
Camelot Management Consultants, CoE Artificial Intellegence for Information Management, Munich, Germany
faza@camelot-mc.com

Abstract

We investigate the recently introduced concept of uniformly tree-automatic classes in the realm of parameterized complexity theory. Roughly speaking, a class of finite structures is uniformly tree-automatic if it can be presented by a set of finite trees and a tuple of automata. A tree t encodes a structure and an element of this structure is encoded by a labeling of t. The automata are used to present the relations of the structure. We use this formalism to obtain algorithmic meta-theorems for first-order logic and in some cases also monadic second-order logic on classes of finite Boolean algebras, finite groups, and graphs of bounded tree-depth. Our main concern is the efficiency of this approach with respect to the hidden parameter dependence (size of the formula). We develop a method to analyze the complexity of uniformly tree-automatic presentations, which allows us to give upper bounds for the runtime of the automata-based model checking algorithm on the presented class. It turns out that the parameter dependence is elementary for all the above mentioned classes. Additionally we show that one can lift the FPT results, which are obtained by our method, from a class $\mathcal{C}$ to the closure of $\mathcal{C}$ under direct products with only a singly exponential blow-up in the parameter dependence.

2012 ACM Subject Classification Theory of computation → Finite Model Theory, Theory of computation → Fixed parameter tractability, Theory of computation → Formal languages and automata theory

Keywords and phrases Automatic Structures, Model Checking, Fixed-Parameter Tractability, Algorithmic Meta Theorems

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.10

1 Introduction

In this paper we investigate the use of automata in *algorithmic meta-theorems*. Algorithmic meta-theorems are general algorithmic results stating that a class of problems $\mathcal{P}$ can be efficiently solved on a class of instances $\mathcal{C}$. In many cases $\mathcal{P}$ is the class of problems definable in a certain logic $\mathcal{L}$. *Parameterised complexity theory* provides one of the key notions to establish algorithmic meta-theorems: we say that the model checking problem for a logic $\mathcal{L}$ on a class of structures $\mathcal{C}$ is fixed-parameter tractable (FPT) (in the size of the formula) if there is a computable function f and a constant c such that we can decide for every $\varphi \in \mathcal{L}$ and every $\mathfrak{A} \in \mathcal{C}$ in time $f(|\varphi|) \cdot |\mathfrak{A}|^c$ whether $\mathfrak{A} \models \varphi$.

Prototypical examples of automata-based algorithmic meta-theorems are the theorem of Courcelle [5] for MSO-definable problems (actually MSO_2, which has the additional capability to quantify over subsets of the edge relation) on graphs of bounded treewidth and the result of Courcelle, Makowsky, and Rotics [4] for MSO-definable problems on graphs of bounded cliquewidth. The basic idea is in both cases to compute from a graph $\mathfrak{G}$ a tree-like

[1] The presented work was conducted while the author was affiliated with the Technische Universität Ilmenau.

38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018).
Editors: Sumit Ganguly and Paritosh Pandya; Article No. 10; pp. 10:1–10:21

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

decomposition $t_{\mathfrak{G}}$ and from an MSO-formula φ a tree-automaton $\mathcal{A}_\varphi$ that accepts exactly the tree-like decompositions of graphs that model φ. Since the construction of $t_{\mathfrak{G}}$ from $\mathfrak{G}$ can be performed efficiently, we can efficiently check if $\mathfrak{G} \models \varphi$ by checking if $\mathcal{A}_\varphi$ accepts $t_{\mathfrak{G}}$. Note that many NP-complete problems, such as 3-Colourability, are definable in MSO and hence efficiently solvable on the above mentioned classes.

The idea to present structures by automata is also the basis for the field of automatic structures. Roughly speaking, a structure is called automatic if its domain can be represented as a regular set in such a way that its relations become recognisable by synchronous multi-tape automata. However, it is not very interesting to study automatic presentations on the class of all finite structures since every finite structure has an automatic presentation (since all finite languages are regular). Recently the concept of uniformly automatic classes was introduced in [1]. In this setting the automata obtain an aditional input (called advice) which encodes the structure that should be presented. Therefore it is possible to present a whole class of structures by a single presentation and a set of advices. Contrary to the classical case without advice it is indeed very interesting to ask which classes of finite structures have a uniformly automatic presentation and which algorithmic consequences can be derived from the existence of such a presentation for a given class.

From a logical point of view it is worthwhile to mention that the presentations which build the core of the FPT algorithms for bounded treewidth and bounded cliquewidth graphs are obtained from MSO-interpretations on trees. Uniformly automatic presentations, however, correspond to so called set-interpretations, which are strictly more powerful than MSO-interpretations. In fact, it is not hard to construct even uniformly word-automatic classes of graphs which have unbounded tree- and cliquewidth. The power to present more complex classes of structures comes with the trade-off that we have to restrict our consideration to FO model checking instead of MSO model checking. Still havening a fixed parameter tractable model checking problem for a class of structures directly leads to FPT results for many other interesting algorithmic problems. For instance, if FO model checking is FPT on a class of graphs $\mathcal{C}$ then Independent Set is FPT on $\mathcal{C}$ in the size of the independent set because for every $k \in \mathbb{N}$ and every graph $\mathfrak{G}$ it holds that $\mathfrak{G} \models \exists x_1, \dots, x_k(\bigwedge_{1 \leq i < j \leq k} x_i \neq x_j \wedge \bigwedge_{i<j} \neg E x_i x_j)$ if and only if $\mathfrak{G}$ contains an independent set of size k.

Meta-theorems for first-order logic have been studied extensively on classes of sparse graphs. The first result in this direction is due to Seese for graphs of bounded degree [23]. Over the past decades larger and larger classes of sparse graphs have been identified for which FO model checking is FPT. This development has recently found its climax in the result of Grohe, Kreutzer, and Siebertz for nowhere dense graphs [15]. They proved that under certain complexity theoretic assumptions this is the largest possible subgraph-closed class of graphs where FO model checking is FPT.

We investigate automaticity as a generic notion of simplicity which might bring up new and interesting classes of structures for which FO model checking is FPT. Towards the theory, we are concerned with the efficiency of this approach. The concept of fixed parameter tractability is often criticized since there are no constraints on the complexity of the parameter. Note that in general the non-elementary worst-case runtime of the automaton construction process leads to a non-elementary parameter dependence in the algorithmic meta-theorems. Frick and Grohe [12] showed, unless PTIME = NP, there is no algorithm that solves the model checking problem for MSO on words or trees in time $f(|\varphi|) \cdot \text{poly}(|t|)$ for any elementary function $f\colon \mathbb{N} \to \mathbb{N}$. A similar statement holds for FO on words. As trees have treewidth one, this renders Courcelle's approach to model checking of graphs with bounded treewidth optimal. Moreover, the efficiency of the automata theoretic approach has

also been confirmed in practice. For instance, Langer et al. [20] implemented Courcelle's technique and found that their implementation can compete with other approaches for specific problems such as Dominating Set.

Even more interestingly, the automata-based approach also tends to behave tamely when applied to interpretations of structures whose theory is elementary. Eisinger [8] gave a triply-exponential upper bound on the size of the minimal automaton for formulae of integer and mixed-real addition. In [6] Durand-Gasselin and Habermehl showed for word-automatic structures that the runtime of the generic algorithm can be bounded by a function which estimates how well the presentation goes along with the Ehrenfeucht-Fraïssé relations of the structure and gave runtime bounds for integer addition matching Eisinger's bound. Additionally they gave a triply-exponential bound for automatic graphs of bounded degree complementing a result by Kuske and Lohrey who proved, using a specialised algorithm, that model checking for automatic graphs of bounded degree is solvable in doubly-exponential space [19].

We adopt Durand-Gasselin's and Habermehl's technique and generalise their result to uniformly tree-automatic presentations. We apply this technique to the presentations that arise as the presentations of the direct product closures of uniformly tree-automatic classes. We prove that the bound of the runtime of the model checking algorithm is at most exponential in the bound of the runtime for the primal classes. Further we apply these findings in the context of FPT model checking for first order logic. We demonstrate the efficiency of the automata-theoretic approach by analysing the runtime in terms of the parameter dependence on structurally rather simple classes. Our results are as follows:

- FO model checking is FPT on the class of all finite Boolean algebras that are succinctly encoded by the number of atoms and can be performed in $\exp_2(\text{poly}(|\varphi|)) \cdot \log|\mathfrak{B}|$. Unless $\text{NEXP} = \bigcup_{c\in\mathbb{N}} \text{STA}(*, 2^{cn}, n)$, this parameter dependence is optimal.
- FO model checking is FPT on the class of all finite abelian groups that are succinctly encoded by the orders of the direct product factors and can be performed in $\exp_4(\text{poly}(|\varphi|)) \cdot \log|\mathfrak{G}|$. We generalise this result to finite groups of *bounded non-abelian decomposition width*, that is groups whose non-abelian direct product factors are of bounded size. We obtain the same asymptotic runtime on these classes.
  This provides some first results towards Grohe's question on which classes of algebraic structures FO model checking is FPT [14]. The mere FPT result for FO model checking on abelian groups was independently also discovered by Bova and Martin [2]. Their algorithm assumes that the groups are encoded by their multiplication tables and yields a non-elementary parameter dependence. Therefore our approach has the two advantages that it works for succinct encodings and yields an elementary parameter dependence.
- MSO model checking is FPT on every class of graphs with tree-depth at most h and can be performed in $\exp_{h+2}(\text{poly}(|\varphi|)) \cdot \text{poly}(|\mathfrak{G}|)$. This matches the runtime of the best known algorithm for these classes, which is due to Gajarsky and Hliněný [13]. Their algorithm uses a kernelisation procedure. Our proof makes use of their analysis.

2 Preliminaries

For natural numbers ℓ, m, n we write $m =_\ell n$ if $m = n$ or $m, n \geq \ell$. We assume that the reader is familiar with first-order logic (FO) as well as with the connection between monadic second-order logic (MSO) and tree-automata. Therefore we use this section mainly to fix our notation.

A **signature** is a finite set of relation symbols $\tau = \{R_1, \dots, R_k\}$, where every symbol $R_i \in \tau$ has an assigned arity r_i. A **τ-structure** is a tuple $\mathfrak{A} = (A, R_1^{\mathfrak{A}}, \dots, R_k^{\mathfrak{A}})$, where A is a set and $R_i^{\mathfrak{A}} \subseteq A^{r_i}$ for all $i \in \{1, \dots, k\}$. From now on we will tacitly assume that all structures under consideration are finite. The class of all finite τ-structures is denoted by $\mathrm{Str}(\tau)$.

Let Σ, Γ be alphabets. A **(labeled binary) tree** is a function $t\colon \mathrm{dom}_t \to \Sigma$, where $\mathrm{dom}_t \subseteq \{0,1\}^*$ is a finite prefix-closed set. The set of all trees with labels from Σ is denoted by T_Σ. Let $t_1 \in T_\Sigma$, $t_2 \in T_\Gamma$ with $\mathrm{dom}_{t_1} = \mathrm{dom}_{t_2} =: \mathrm{dom}$. The **convolution** $t_1 \otimes t_2 \in T_{\Sigma\times\Gamma}$ is defined by $(t_1 \otimes t_2)(w) = (t_1(w), t_2(w))$ for all $w \in \mathrm{dom}$. When we apply the convolution to several trees at once we will often write $\langle t_1, t_2, \dots, t_k\rangle$ instead of $t_1 \otimes t_2 \otimes \cdots \otimes t_k$. A reader that is familiar with automatic presentations might notice that we define the convolution only for trees with the same domain. This allows us to circumvent the introduction of a padding symbol. For trees $t, t_1, \dots, t_n$ and pairwise distinct $w_1, \dots, w_n \in \mathrm{dom}_t$ we define $t[w_1/t_1, \dots, w_n/t_n]$ to be the tree which is obtained by replacing the subtree rooted in w_i by t_i for all $i \leq n$.

A **Σ-context** is a tree $c \in T_{\Sigma\uplus\{x\}}$ such that all nodes except for exactly one *leaf* w are labeled with letters from Σ and $c(w) = x$. The unique leaf w with label x is denoted by $c^{-1}(x)$. For a Σ-context c and a tree $t \in T_\Sigma$ the **composition** $(c \circ t) \in T_\Sigma$ is defined as $c[c^{-1}(x)/t]$.

Let us now introduce tree-automata with advice. Formally, these are just ordinary tree-automata which read letters from a composed alphabet. But since our automatic presentations will assign special semantics to the first component of such a letter it makes sense to handle these components differently in our notation.

▶ **Definition 2.1.** A **(deterministic bottom-up) tree-automaton with advice** is a finite state tree-automaton $\mathcal{A} = (Q, \Sigma\times\Gamma, \delta, F)$. The language that $\mathcal{A}$ recognizes with advice $\alpha \in T_\Sigma$ is $L(\mathcal{A}[\alpha]) = \{t \in T_\Gamma \mid \mathrm{dom}_t = \mathrm{dom}_\alpha \wedge \alpha \otimes t \in L(\mathcal{A})\}$. A tree-language T is called regular with advice α if there is a tree-automaton $\mathcal{A}$ with advice such that $T = L(\mathcal{A}[\alpha])$.

For the sake of brevity we usually just speak of an automaton instead of a tree-automaton with advice. The complement automaton of $\mathcal{A}$ is denoted by $\overline{\mathcal{A}} = (Q, \Sigma\times\Gamma, \delta, Q\setminus F)$. Finally we define uniformly tree-automatic presentations.

▶ **Definition 2.2.** Let $\tau = \{R_1, \dots, R_k\}$ be a finite relational signature. A **uniformly tree-automatic presentation** of a class of τ-structures is a tuple $\mathfrak{c} = (\mathcal{A}, \mathcal{A}_{R_1}, \dots, \mathcal{A}_{R_k})$ of tree-automata with advice such that $L(\mathcal{A}[\alpha]) \subseteq T_{\{0,1\}}$ and $L(\mathcal{A}_{R_i}[\alpha]) \subseteq \{\langle t_1, \dots, t_{r_i}\rangle \mid t_1, \dots, t_{r_i} \in L(\mathcal{A}[\alpha])\}$ for all $\alpha \in T_\Sigma$ and all $i \in \{1, \dots, k\}$. Each $\alpha \in T_\Sigma$ with $L(\mathcal{A}[\alpha]) \neq \emptyset$ presents (the isomorphism type of) a structure $\mathcal{S}(\mathfrak{c}[\alpha]) := \left(L(\mathcal{A}[\alpha]), \left(R_i^{\mathcal{S}(\mathfrak{c}[\alpha])}\right)_{1\leq i\leq k}\right)$, where $R_i := \{(t_1, \dots, t_{r_i}) \mid \langle t_1, \dots, t_{r_i}\rangle \in L(\mathcal{A}_{R_i}[\alpha])\}$. The set $\{\alpha \in T_\Sigma \mid L(\alpha) \neq \emptyset\}$ of all advices that present a structure with respect to $\mathfrak{c}$ is denoted by $P^{\mathfrak{c}}$. The class that is presented by $\mathfrak{c}$ is $\{\mathcal{S}(\mathfrak{c}[\alpha]) \mid \alpha \in P^{\mathfrak{c}}\}$.

3 Model Checking Revisited

Since the class of all words is uniformly tree-automatic it is clear by the previously mentioned result of Frick and Grohe [12] that every algorithm that solves the model checking problem for structures given by a uniformly tree-automatic presentation has an unavoidable non-elementary worst-case runtime behaviour. On the other hand, for many important examples of automatic structures the situation is much better. For instance it is known that the

Algorithm 1 Model Checking on Uniformly Tree-Automatic Classes.

Input: Tree-automatic presentation $\mathfrak{c} = (\mathcal{A}, (\mathcal{A}_R)_{R\in\tau})$, FO-formula $\varphi(x_1, \ldots, x_m)$
Output: Tree-automaton $\mathcal{A}_\varphi$

procedure COMPOSE($\mathfrak{c}$, φ)
  if $\varphi(x_1, \ldots, x_m) = R(x_{i_1}, \ldots, x_{i_k}), R \in \tau \cup \{=\}$ **then**
    $\mathcal{A}'_R \leftarrow$ EXTEND$(\mathcal{A}_R, m, i_1, \ldots, i_k)$
    $\mathcal{A}_D \leftarrow$ DOMAIN$(\mathcal{A}, m)$
    $\mathcal{A}_\varphi \leftarrow$ INTERSECT$(\mathcal{A}'_R, \mathcal{A}_D)$
    minimise $\mathcal{A}_\varphi$
    return $\mathcal{A}_\varphi$
  else if $\varphi(x_1, \ldots, x_m) = \psi(x_1, \ldots, x_m) \wedge \theta(x_1, \ldots, x_m)$ **then**
    $\mathcal{A}_\psi \leftarrow$ COMPOSE$((\mathcal{A}, (\mathcal{A}_R)_{R\in\tau}), \psi)$
    $\mathcal{A}_\theta \leftarrow$ COMPOSE$((\mathcal{A}, (\mathcal{A}_R)_{R\in\tau}), \theta)$
    return INTERSECT$(\mathcal{A}_\psi, \mathcal{A}_\theta)$
  else if $\varphi(x_1, \ldots, x_m) = \neg\psi(x_1, \ldots, x_m)$ **then**
    $\mathcal{A}_\psi \leftarrow$ COMPOSE$((\mathcal{A}, (\mathcal{A}_R)_{R\in\tau}), \psi(x_1, \ldots, x_m))$
    $\mathcal{A}_D \leftarrow$ DOMAIN$(\mathcal{A}, r)$
    return INTERSECT$(\overline{\mathcal{A}_\psi}, \mathcal{A}_D)$
  else if $\varphi(x_1, \ldots, x_m) = \exists x_{m+1} : \psi(x_1, \ldots, x_{m+1})$ **then**
    $\mathcal{A}_\psi \leftarrow$ COMPOSE$((\mathcal{A}, (\mathcal{A}_R)_{R\in\tau}), \psi(x_1, \ldots, x_m))$
    $\mathcal{A}'_\varphi \leftarrow$ PROJECT$(\mathcal{A}_\psi)$
    $\mathcal{A}_\varphi \leftarrow$ DETERMINIZE$(\mathcal{A}'_\varphi)$
    return $\mathcal{A}_\varphi$
  end if
end procedure

first-order theory of Presburger Arithmetic can be decided in three-fold exponential time [21]. It is therefore very natural to analyse the runtime of a given model checking algorithm for automatic structures with respect to some fixed presentation.

In [6] Durand-Gasselin and Habermehl proposed a method to estimate the time that the generic automata based model checking algorithm for structures given by a *word-automatic* presentation needs when it is used to solve the first order theory of a single structure. They showed that for certain presentations of $(\mathbb{Z}, +)$ the running time of the algorithm is only triply exponential in the formula. Similar bounds where established for arbitrary word-automatic presentations of structures of bounded degree.

In the following we want to extend their method to *uniformly tree-automatic* presentations of *classes of structures*. Fortunately this generalization goes through very well because of the nice analogue of the Myhill-Nerode congruence for regular tree-languages.

We start with a detailed description of the model checking algorithm on structures given by an advice α from a uniform tree-automatic presentation $\mathfrak{c}$. Up to small optimizations it resembles the standard algorithm that constructs from $\mathfrak{c}, \alpha$, and $\varphi(x_1, \ldots, x_m)$ an automaton $\mathcal{A}_\varphi$ with $L(\mathcal{A}_\varphi) = \{\langle \alpha, t_1, \ldots, t_m\rangle \mid \mathcal{S}(\mathfrak{c}[\alpha]) \models \varphi(t_1, \ldots, t_m)\}$ by recursion over the structure of φ. The exact procedure is given by Algorithm 1.

The subroutine EXTEND$(\mathcal{A}_R, m, i_1, \ldots, i_k)$ computes the minimal automaton that checks on input $\langle \alpha, t_1, \ldots, t_m\rangle$ if $\langle \alpha, t_{i_1}, \ldots, t_{i_k}\rangle \in L(\mathcal{A}_R)$, that is if $(t_{i_1}, \ldots, t_{i_k}) \in R^{\mathcal{S}(\mathfrak{c}[\alpha])}$. The subroutine DOMAIN constructs the minimal tree-automaton that recognises exactly those trees in $T_{\Sigma\times\Gamma^m}$ that are convolutions of trees $t_0 \in T_\Sigma$ and $t_1, \ldots, t_m \in T_\Gamma$ such that $t_1, \ldots, t_m \in$

$\mathcal{S}(\mathfrak{c}[t_0])$. The subroutine $\textsc{Intersect}(\mathcal{A}_1, \mathcal{A}_2)$ uses the standard product construction to obtain an automaton $\mathcal{A}_\cap$ with $L(\mathcal{A}_\cap) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$. Note that it is crucial for the runtime analysis in the following section that we only construct the reachable states of the product automaton. Finally $\textsc{Project}(\mathcal{A})$ applies the projection $(\sigma, \gamma_1, \ldots, \gamma_{m+1}) \mapsto (\sigma, \gamma_1, \ldots, \gamma_m)$ to the input alphabet of $\mathcal{A}$, which yields a non-deterministic automaton, and $\textsc{Determinize}(\mathcal{A})$ uses the standart determinization preocedure for tree-automata (again omitting non-reachable states).

4 A Presentation Aware Runtime Analysis

The main ingredient for the runtime analysis of Algorithm 1 is the marriage of the Ehrenfeucht-Fraïssé relations (EF-relations) on the presented class of structures and the Myhill-Nerode congruences on the languages which form the presentation. Ehrenfeucht-Fraïssé relations were introduced by Fraïssé in his seminal work [11] as a purely combinatorial characterisation of elementary equivalence. His ideas were later popularised by the appealing game-theoretic presentation given by Ehrenfeucht in [7]. Even the possibility to bound the complexity of certain logical theories using EF-relations was already present in these early works. This technique was later systematically studied by Ferrante and Rackoff (see [9]). They used EF-relations to give upper bounds on the complexity of first-order theories like Presburger Arithmetic or the theory of one-to-one functions.

Klaetke used in [18] the ideas of Ferrante and Rackoff to bound the size of the automata for linear arithmetic $(\mathbb{R}, +, <)$. Eisinger picked up the techniques and showed in [8] similar bounds for a certain automata based presentation of mixed integer and mixed real addition, respectively (we remark here that his way of presenting the structures by automata differs slightly from our definition of an automatic presentation). Durand-Gasselin and Habermehl recently showed that if a refinement of the EF-relations for a structure $\mathfrak{A}$ is compatible with an automatic presentation of $\mathfrak{A}$ in the sense that these relations are congruences on the encodings of the elements (with respect to concatenation) then the runtime of the standard algorithm for solving the theory of an automatic structure can be bounded in terms of the index of these relations. In this section we build upon their work and generalise their result to classes with a uniform tree-automatic presentation. Therefore it is necessary to develop a suitable notion of EF-congruences for our purposes. Besides switching from automatic presentations to uniform tree-automatic presentations, there are a few subtle differences to the definition in [6] in order to make the technique applicable for more presentations.

Let Σ be an advice alphabet and Γ be an input alphabet. In the following we write $\hat{\Sigma}_m$ for $\Sigma \times \Gamma^m$

▶ **Definition 4.1.** Let $\mathfrak{c} = (\mathcal{A}, (\mathcal{A}_R)_{R \in \tau})$ be a uniformly tree-automatic presentation of a class $\mathcal{C} \subseteq \mathrm{Str}(\tau)$. An Ehrenfeucht-Fraïssé congruence (EF-congruence) for $\mathfrak{c}$ is a collection of equivalence relations $(E^r_m)_{r,m \in \mathbb{N}}$, where $E^r_m \subseteq T_{\hat{\Sigma}_m} \times T_{\hat{\Sigma}_m}$ and for all $r, m \in \mathbb{N}$:

1. The relation E^r_m separates the trees in $T_{\hat{\Sigma}_m}$ that are a convolution of a tuple $(\alpha, t_1, \ldots, t_m)$ such that $(t_1, \ldots, t_m)$ represents a tuple of elements in $\mathcal{S}(\mathfrak{c}[\alpha])$ from those trees in $T_{\hat{\Sigma}_m}$ that are not the convolution of such a tuple.
2. If $t_1, \ldots, t_m \in \mathcal{S}(\mathfrak{c}[\alpha])$, $t'_1, \ldots, t'_m \in \mathcal{S}(\mathfrak{c}[\beta])$, and $\langle \alpha, \bar{t} \rangle E^0_m \langle \beta, \bar{t}' \rangle$ then $(t_1, \ldots, t_m)$ and $(t'_1, \ldots, t'_m)$ satisfy the same atomic formulas in $\mathcal{S}(\mathfrak{c}[\alpha])$ and $\mathcal{S}(\mathfrak{c}[\beta])$, respectively.
3. If $s E^{r+1}_m s'$ for some $s, s' \in T_{\hat{\Sigma}_m}$ then for all $t \in T_\Gamma$ there exists a $t' \in T_\Gamma$ such that $\langle s, t \rangle E^r_{m+1} \langle s', t' \rangle$.
4. The relation E^r_m respects contexts, i.e. if $t E^r_m t'$ for some $t, t' \in T_{\hat{\Sigma}_m}$ then for all $\hat{\Sigma}_m$-contexts c the trees $c \circ t$ and $c \circ t'$ are also related by E^r_m.

For a function $f : \mathbb{N} \to \mathbb{N}$ we say that an EF-congruence $(E^r_m)_{r,m\in\mathbb{N}}$ is $f(r+m)$ bounded if the index of E^r_m is bounded by $f(r+m)$ for all $r, m \in \mathbb{N}$.

The EF-congruence $(E^r_m)_{r,m\in\mathbb{N}}$ for a presentation $\mathfrak{c}$ refines the first-order indistinguishably relations $(\equiv_r)_{r\in\mathbb{N}}$ on the presented class $\mathcal{C}$ (recall that $\equiv_r$ means indistinguishable by formulas up to quantifier rank r). This can be shown using standard game theoretic arguments.

▶ **Lemma 4.2.** *Let $\mathfrak{c}$ be a uniform tree-automatic presentation of a class $\mathcal{C}$ and $(E^r_m)_{r,m\in\mathbb{N}}$ an EF-congruence with respect to $\mathfrak{c}$. Then for all $\alpha, \alpha' \in P^{\mathfrak{c}}$ and $t_1, t'_1 \ldots, t_m, t'_m$ with $t_1, \ldots, t_m \in \mathcal{S}(\mathfrak{c}[\alpha])$ and $t'_1, \ldots, t'_m \in \mathcal{S}(\mathfrak{c}[\alpha'])$ the following is true:*

$$\langle \alpha, t_1, \ldots, t_m\rangle E^r_m \langle \alpha', t'_1, \ldots, t'_m\rangle \Rightarrow (\mathcal{S}(\mathfrak{c}[\alpha]), t_1, \ldots, t_m) \equiv_r (\mathcal{S}(\mathfrak{c}[\alpha']), t'_1, \ldots, t'_m).$$

As mentioned before, an EF-congruence with respect to some parametrized tree-automatic presentation connects the Myhill-Nerode-congruences of the languages involved in the presentation with the EF-relations on the presented class. We want to show that the runtime of Algorithm 1 largely depends on how well these relations play along with each other.

▶ **Theorem 4.3.** *Let $\mathfrak{c} = (\mathcal{A}, (\mathcal{A}_R)_{R\in\tau})$ be a uniformly tree-automatic presentation of a class of τ-structures. Suppose there is an $f(r+m)$ bounded EF-congruence $(E^r_m)_{r,m\in\mathbb{N}}$ for $\mathfrak{c}$. Then for every $\varphi(x_1, \ldots, x_m) \in \mathrm{FO}$ of quantifier rank r Algorithm 2 computes the automaton $\mathcal{A}_\varphi$ in time $\mathcal{O}(|\varphi|(|\mathfrak{c}|^{m+r} \cdot f(m+r))^c)$ for some constant c.*

The proof is similar to [6]. We omit it here due to space constraints. In the following section we will be concerned with classes of finite structures that arise as the closure under direct products of a certain prime class. It ist not hard to see that if a class $\mathcal{C}$ is uniformly tree-automatic then the same holds for the closure of $\mathcal{C}$ under direct products (see also [1]). We close this section by showing that also the EF-congruences can (with a certain blow up of the index) be lifted from the original presentation to a certain presentation of the direct product closure.

▶ **Definition 4.4.** Let $\mathcal{C}$ be a class of τ-structures. Then $\mathcal{C}^\times$ denotes the closure of $\mathcal{C}$ under direct products That is $\mathcal{C}^\times = \{\mathfrak{A}_1 \times \cdots \times \mathfrak{A}_n \mid n \geq 1, \mathfrak{A}_1, \ldots, \mathfrak{A}_n \in \mathcal{C}\}$.

▶ **Lemma 4.5** (Abu Zaid, Grädel, Reinhardt [1])**.** *Let $\mathcal{C}$ be a uniformly tree-automatic class of structures. From a given tree-automatic presentation $\mathfrak{c}$ of $\mathcal{C}$ one can effectively construct tree-automatic presentations $\mathfrak{c}^\times$ of $\mathcal{C}^\times$.*

Proof. Construction of $(P^\times, \mathfrak{c}^\times)$: Suppose $\mathcal{C}$ is presented by the uniform tree-automatic presentation $\mathfrak{c}$ over the advice set P. As the construction is rather straightforward we only give the parameter set for the presentation and the idea for the encoding. The parameter set consists of all trees where the right child of every node in the left-most branch induces a subtree which is in P. This is depicted in Figure 1. Such an advice presents the structure $\mathcal{S}(\mathfrak{c}[\alpha_1]) \times \mathcal{S}(\mathfrak{c}[\alpha_2]) \times \cdots \times \mathcal{S}(\mathfrak{c}[\alpha_n])$. Let $t_1, \ldots, t_n$ be elements of $\mathcal{S}(\mathfrak{c}[\alpha_1]), \ldots, \mathcal{S}(\mathfrak{c}[\alpha_n])$, respectively. Then the element $(t_1, \ldots, t_n)$ is put together in the same way as the advices. ◀

In order to ease the process of analyzing the complexity of these presentations, we introduce some notations. Let Γ be an alphabet with $\# \notin \Gamma$. The n-context-tree $t^\#_n$ is the tree with domain $\mathrm{dom}(t^\#_n) = \{0^k \mid k < n\} \cup \{0^k 1 \mid k+1 < n\}$ and labeling

$$t^\#_n(w) = \begin{cases} \# & ;\text{if } w \in \{0\}^{<n} \\ c_{i+1} & ;\text{if } w = 0^i 1, \text{ with } 0 \leq i < n-1 \\ c_n & ;\text{if } w = 0^{n-1}. \end{cases}$$

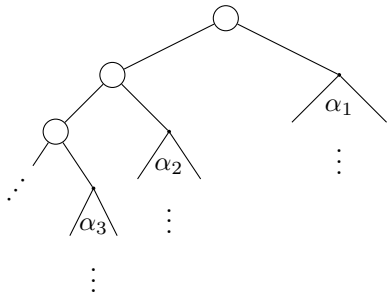


Figure 1 The Parameters for the class $\mathcal{C}^\times$.

With $T_\Gamma^{\#,n}$ we denote the set of all trees that are obtained from $t_n^\#$ by replacing all contexts with trees from T_Γ, that is $T_\Gamma^{\#,n} = \{t_n^\#[c_1/t_1, \ldots, c_n/t_n] \mid t_1, \ldots, t_n \in T_\Gamma\}$. Finally let $T_\Gamma^\#$ be the union of all sets $T_\Gamma^{\#,n}$ with $n \geq 1$.

▶ **Theorem 4.6.** *Let $\mathfrak{c}$ be a uniformly tree-automatic presentation of a class $\mathcal{C}$ with associated $f(r+m)$ bounded EF-congruences $(E_m^r)_{r,m\in\mathbb{N}}$. Then there is a uniformly tree-automatic presentation of $\mathcal{C}^\times$ with associated $2^{\mathcal{O}((r+m)f(r+m)\log(f(r+m)))}$ bounded EF-congruences.*

Proof. Let $\mathfrak{c}^\times$ be the presentation of $\mathcal{C}^\times$ that is derived from $\mathfrak{c}$ by the construction from Lemma 4.5. Recall that if P is the set of advice trees for the presentation $\mathfrak{c}$ and $\alpha_1, \ldots, \alpha_n \in P$, then the structure $\mathcal{S}(\mathfrak{c}[\alpha_1]) \times \cdots \times \mathcal{S}(\mathfrak{c}[\alpha_n])$ is presented by the advice $t_n^\#[c_1/\alpha_1, \ldots, c_n/\alpha_n]$ and an element $(t_1, \ldots, t_n) \in \mathcal{S}(\mathfrak{c}[\alpha_1]) \times \cdots \times \mathcal{S}(\mathfrak{c}[\alpha_n])$ is represented by the tree $t_n^\#[c_1/t_1, \ldots, c_n/t_n]$, where $\#$ is a newly introduced letter.

For all $r, m \in \mathbb{N}$ we define a relation $\sim_m^r$ on $T_{(\Sigma\cup\{\#\})\times(\Gamma\cup\{\#\})^m}$, where $t \sim_m^r t'$ if, and only if, one of the following conditions is true:

1. There are no n, n' such t and t' are the convolution of well-formed trees $\alpha \in T_\Sigma^{\#,n}, t_1, \ldots, t_m \in T_\Gamma^{\#,n}$ and $\alpha' \in T_\Sigma^{\#,n'}, t_1', \ldots, t_m' \in T_\Gamma^{\#,n'}$, respectively.
2. There are n, n' such t and t' are the convolution of well-formed trees $\alpha \in T_\Sigma^{\#,n}, t_1, \ldots, t_m \in T_\Gamma^{\#,n}$ and $\alpha' \in T_\Sigma^{\#,n'}, t_1', \ldots, t_m' \in T_\Gamma^{\#,n'}$, respectively. That is we can write $t = \langle t_n^\#[c_1/\alpha_1, \ldots, c_n/\alpha_n], t_n^\# c_1/t_{1,1}, \ldots, c_n/t_{1,n}], \ldots\ldots, t_n^\#[c_1/t_{m,1}, \ldots, c_n/t_{m,n}]\rangle$ and also $t' = \langle t_{n'}^\#[c_1/\alpha_1', \ldots, c_{n'}/\alpha_{n'}'], t_{n'}^\#[c_1/t_{1,1}', \ldots, c_{n'}/t_{1,n'}'], \ldots\ldots, t_{n'}^\#[c_1/t_{m,1}', \ldots, c_{n'}/t_{m,n'}']\rangle$. Then $t \sim_m^r t'$ if for all E_m^r equivalence classes κ: $|\{i \mid 1 \leq i \leq n, [\langle \alpha_i, t_{1,i}, \ldots, t_{m,i}\rangle]_{E_m^r} = \kappa\}| =_{f(r+m)^r} |\{i \mid 1 \leq i \leq n,' [\langle \alpha_i', t_{1,i}', \ldots, t_{m,i}\rangle']_{E_m^r} = \kappa\}|$.

One easily checks that $\sim_m^r$ is an equivalence relation with index bounded by

$$(f(r+m)^{r+m}+1)^{f(r+m)} + 1 \in 2^{\mathcal{O}((r+m)f(r+m)\log f(r+m))}$$

for all $r, m \in \mathbb{N}$. What is left is to verify is that $(\sim_m^r)_{r,m\in\mathbb{N}}$ is indeed an EF-congruence of $\mathfrak{c}^\times$. Therefore we check that the collection $(\sim_m^r)_{r,m\in\mathbb{N}}$ has the Properties 1 - 4 described in Definition 4.1. This is done in the lemmata below.

▶ **Lemma 4.7.** *The relation $\sim_m^r$ separates the trees that are the convolution of a tuple $(\alpha, t_1, \ldots, t_n)$ such that $(t_1, \ldots, t_m)$ represents a tuple of elements in $\mathcal{S}(\mathfrak{c}^\times[\alpha])$ from those trees that are not the convolution of such a tuple.*

Proof. Suppose $t = \langle \alpha, t_1, \ldots, t_m\rangle$ is a convolution of a tuple with $\alpha \in P^{\mathfrak{c}^\times}$ and $(t_1, \ldots, t_m) \in \mathcal{S}(\mathfrak{c}^\times[\alpha])$ and suppose t' is not the convolution of such a tuple. If t' is not a convolution, then none of the two conditions holds for t and t' and they are not equivalent. Otherwise there are n,

$n' \geq 1$ with $t = \langle t_n^{\#}[c_1/\alpha_1, \ldots, c_n/\alpha_n], t_n^{\#}[c_1/t_{1,1}, \ldots, c_n/t_{1,n}], \ldots, t_n^{\#}[c_1/t_{m,1}, \ldots, c_n/t_{m,n}]\rangle$ and $t' = \langle t_{n'}^{\#}[c_1/\alpha'_1, \ldots, c_{n'}/\alpha'_{n'}], t_{n'}^{\#}[c_1/t'_{1,1}, \ldots, c_{n'}/t'_{1,n'}], \ldots, t_{n'}^{\#}[c_1/t'_{m,1}, \ldots, c_{n'}/t'_{m,n'}]\rangle$.

From our assumption about t and t' we know that $\alpha_i \in P^{\mathfrak{c}}$ and $t_{1,i}, \ldots t_{m,i} \in \mathcal{S}(\mathfrak{c}[\alpha_i])$ for all $1 \leq i \leq n$ and there is a $1 \leq j \leq n'$ with $\alpha'_j \notin P^{\mathfrak{c}}$ or $\alpha'_j \in P^{\mathfrak{c}}$ but $t'_{\ell,j} \notin \mathcal{S}(\mathfrak{c}[\alpha_i])$ for some $1 \leq \ell \leq m$.

But then $\langle \alpha_i, t_{1,i}, \ldots, t_{m,i}\rangle \not\mathrel{E_m^r} \langle \alpha'_j, t'_{1,j}, \ldots, t'_{m,j}\rangle$, since the relation E_m^r fulfils Property 1 of Definition 4.1. Hence t and t' do not fulfil condition 2 and therefore $t \not\sim_m^r t'$. ◀

▶ **Lemma 4.8.** *If $t_1, \ldots, t_m \in \mathcal{S}(\mathfrak{c}[\alpha])$, $t'_1, \ldots, t'_m \in \mathcal{S}(\mathfrak{c}[\beta])$, and $\langle \alpha, \bar{t}\rangle \sim_m^0 \langle \beta, \bar{t}'\rangle$ then $(t_1, \ldots, t_m)$ and $(t'_1, \ldots, t'_m)$ satisfy the same atomic formulas in $\mathcal{S}(\mathfrak{c}[\alpha])$ and $\mathcal{S}(\mathfrak{c}[\beta])$, respectively.*

Proof. Suppose

- $\alpha = t_n^{\#}[c_1/\alpha_1, \ldots, c_n/\alpha_n], \beta = t_k^{\#}[c_1/\beta_1, \ldots, c_k/\beta_k] \in P^{\mathfrak{c}^\times}$,
- $t_i = t_n^{\#}[c_1/t_{i,1}, \ldots, c_n/t_{i,n}] \in \mathcal{S}(\mathfrak{c}^\times[\alpha])$ for $i \in \{1, \ldots, m\}$, and
- $t'_i = t_k^{\#}[c_1/t'_{i,1}, \ldots, c_k/t'_{i,k}] \in \mathcal{S}(\mathfrak{c}^\times[\beta])$ for $i \in \{1, \ldots, m\}$.

We show that if $(t_1, \ldots, t_m)$ and $(t_1, \ldots, t'_m)$ do not fulfil the same atomic propositions in $\mathcal{S}(\mathfrak{c}^\times[\alpha])$ and $\mathcal{S}(\mathfrak{c}^\times[\beta])$, respectively, then they are not $\sim_m^0$-equivalent. Consider an arbitrary atomic formula $Rx_{i_1} \ldots x_{i_r}$ and suppose $\mathcal{S}(\mathfrak{c}^\times[\alpha]) \models Rt_{i_1} \ldots, t_{i_r}$ and $\mathcal{S}(\mathfrak{c}^\times[\alpha']) \not\models Rt'_{i_1} \ldots, t'_{i_r}$. Then by definition $\mathcal{S}(\mathfrak{c}[\alpha_j]) \models Rt_{j,i_1} \ldots t_{j,i_r}$ for all $1 \leq j \leq n$ but $\mathcal{S}(\mathfrak{c}[\beta_\ell]) \not\models Rt'_{\ell,i_1} \ldots t'_{\ell,i_r}$ for some $1 \leq \ell \leq k$. Consequently $\langle \alpha_j, t_{j,1}, \ldots, t_{j,m}\rangle \not\mathrel{E_m^0} \langle \beta_\ell, t_{\ell,1}, \ldots, t_{\ell,m}\rangle$ for all $1 \leq j \leq n$ and therefore $\langle \alpha, t_1, \ldots, t_m\rangle \not\sim_m^0 \langle \beta, t'_1, \ldots, t'_m\rangle$. ◀

▶ **Lemma 4.9.** *If $s \sim_m^{r+1} s'$ then for all $t \in T_{(\Gamma \cup \{\#\})}$ there exists a $t' \in T_{(\Gamma \cup \{\#\})}$ such that $\langle s, t\rangle \sim_{m+1}^r \langle s', t'\rangle$.*

Proof. Let s, s' be two trees from $T_{(\Sigma \cup \{\#\}) \times (\Gamma \cup \{\#\})^m}$ such that $s \sim_m^{r+1} s'$ and $t \in T_{(\Gamma \cup \{\#\})}$. Then $s = \langle t_n^{\#}[c_1/\alpha_1, \cdots, c_n/\alpha_n], t_n^{\#}[c_1/t_{1,1}, \cdots, c_n/t_{1,n}], \ldots, t_n^{\#}[c_1/t_{m,1}, \cdots, c_n/t_{m,n}]\rangle$ and $s' = \langle t_k^{\#}[c_1/\alpha'_1, \cdots, c_k/\alpha'_k], t_k^{\#}[c_1/t'_{1,1}, \cdots, c_k/t'_{1,k}], \ldots, t_k^{\#}[c_1/t'_{m,1}, \cdots, c_k/t'_{m,k}]\rangle$ for some $n, k \geq 1$ and trees $\alpha_i, \alpha'_j \in T_\Sigma$ and $t_{i,j}, t'_{s,t} \in T_\Gamma$. Let t_{m+1} be an arbitrary tree from $T_{(\Gamma \cup \{\#\})}$. If $t_{m+1} \notin T_\Gamma^{\#,n}$ take some tree t'_{m+1} that is not in $T_\Gamma^{\#,k}$. Then $\langle s, t_{m+1}\rangle \sim_m^r \langle s', t'_{m+1}\rangle$ because of Condition 1. Otherwise $t_{m+1} = t_n^{\#}[c_1/t_{m+1,1}, \ldots c_n/t_{m+1,n}]$. For every E_m^r equivalence class κ let $\kappa(s) = \{i \in \{1, \ldots, n\} \mid [\langle \alpha_i, s_{i,1}, \ldots, s_{i,m}\rangle]_{E_m^r} = \kappa\}$.

Let $X_1^\kappa, \ldots, X_{\ell_\kappa}^\kappa$ be the partition of $\kappa(s)$ with respect to the E_{m+1}^r equivalence classes of $\{\langle \alpha, t_{1,i}, \ldots, t_{m+1,i}\rangle \mid i \in \kappa(t)\}$. Because $s \sim_m^{r+1} s'$ it is ensured that $|\kappa(s)| =_{f(m+r+1)^{r+1}} |\kappa(s')|$ and therefore we can find a partition $Y_1^\kappa, \ldots, Y_{\ell_\kappa}^\kappa$ of $\kappa(s')$ with $|X_i^\kappa| =_{f(r+m+1)^r} |Y_i^\kappa|$ (if $|\kappa(s)| < f(m+r+1)^{r+1}$ partition $\kappa(s')$ according to some bijection between $\kappa(s)$ and $\kappa(s')$. Otherwise, because $\ell_\kappa < f(m+r+1)$ there is at least one X_i^κ with $|X_i^\kappa| \geq f(m+r+1)$ which also ensures that we can find such a partition).

By construction, $\langle \alpha, t_{1,i}, \ldots, t_{m,i}\rangle E_m^{r+1} \langle \alpha', t'_{1,j}, \ldots, t'_{m,j}\rangle$ whenever $i \in X_k^\kappa$ and $j \in Y_k^\kappa$. Thus $\langle \alpha, t_{1,i}, \ldots, t_{m+1,i}\rangle E_{m+1}^r \langle \alpha', t'_{1,j}, \ldots, t'_{m+1,j}\rangle$ for some appropriate $t'_{m+1,j}$. Now choose $t'_{m+1} = t_k^{\#}[c_1/t'_{m+1,1}, \ldots, c_k/t'_{m+1,k}]$. By construction $\langle s, t_{m+1}\rangle \sim_{m+1}^r \langle s', t'_{m+1}\rangle$ due to Condition 2. ◀

In order to show that Property 4 is fulfilled, it is convenient to define a special kind of convolution for contexts. For $i \in \{1, \ldots, n\}$ let c_i be an Γ_i-context such such that $\mathrm{dom}_{c_1} = \cdots = \mathrm{dom}_{c_n} =: \mathrm{dom}$ and $c_1^{-1}(x) = \cdots = c_n^{-1}(x) =: w_x$. Then $\langle c_1, \ldots, c_n\rangle_c$ is the $(\Gamma_1 \times \cdots \times \Gamma_n)$-context with $\mathrm{dom}(\langle c_1, \ldots, c_n\rangle_c) = \mathrm{dom}$ and

$$\langle c_1, \ldots, c_n\rangle_c(w) = \begin{cases} (\gamma_1, \ldots, \gamma_n) & \text{if } w \neq w_x, \\ x & \text{otherwise.} \end{cases}$$

▶ **Lemma 4.10.** *The relations $(\sim_m^r)_{r,m\in\mathbb{N}}$ respect contexts.*

Proof. Suppose $s \sim_m^r s'$ and let c be a $((\Sigma \cup \{\#\}) \times (\Gamma \cup \{\#\})^m)$-context. We can assume that s and s' are equivalent due to Condition 2 and that

$$c = \langle t_n^{\#}[c_1/\alpha_1, \dots, c_n/\alpha_n], t_n^{\#}[c_1/t_{1,1}, \dots, c_n/t_{1,n}], \dots, t_n^{\#}[c_1/t_{m,1}, \dots, c_n/t_{m,n}]\rangle_c$$

for some $n \geq 1$ and $\langle \alpha_i, t_{1,i}, \dots t_{m,i}\rangle_c$ is a $(\Sigma \times \Gamma^m)$-context for exactly one $1 \leq i \leq n$ (because in any other case $c \circ t$ and $c \circ t'$ are equivalent by Condition 1). Fix this i and let $c' := \langle \alpha_i, t_{1,i}, \dots t_{m,i}\rangle$. There two cases that we need to consider. First if s, s' are elements of $T_\Sigma^{\#,1} \otimes (T_\Gamma^{\#,1})^{\otimes m}$ $(= T_\Sigma \otimes (T_\Gamma)^{\otimes m})$. Then the requirement of Condition 2 reduces to $sE_m^r s'$. But then $c' \circ tE_m^r c' \circ s'$ and hence $c \circ s \sim_m^r c \circ s'$. Otherwise we can even assume that

$$c = \langle t_n^{\#}[c_1/\alpha_1, \dots, c_n/x], t_n^{\#}[c_1/t_{1,1}, \dots, c_n/x], \dots, t_n^{\#}[c_1/t_{m,1}, \dots, c_n/x]\rangle_c$$

(again otherwise we would get equivalence by Condition 1). But then

$$\begin{aligned} c \circ s = \langle & t_{n+k-1}^{\#}[c_1/\alpha_1, \dots, c_{n-1}/\alpha_{n-1}, c_n/\beta_1, \dots, c_{n+k-1}/\beta_k], \\ & t_{n+k-1}^{\#}[c_1/t_{1,1}, \dots, c_{n-1}/t_{1,n-1}, c_n/s_{1,1}, \dots, c_{n+k-1}/s_{1,k}], \\ & \vdots \\ & t_{n+k-1}^{\#}[c_1/t_{m,1}, \dots, c_{n-1}/t_{m,n-1}, c_n/s_{m,1}, \dots, c_{n+k-1}/s_{m,k}]\rangle \end{aligned}$$

and

$$\begin{aligned} c \circ s' = \langle & t_{n+k-1}^{\#}[c_1/\alpha_1, \dots, c_{n-1}/\alpha_{n-1}, c_n/\beta'_1, \dots, c_{n+k-1}/\beta'_{k'}], \\ & t_{n+k-1}^{\#}[c_1/t_{1,1}, \dots, c_{n-1}/t_{1,n-1}, c_n/s'_{1,1}, \dots, c_{n+k-1}/s'_{1,k'}], \\ & \vdots \\ & t_{n+k-1}^{\#}[c_1/t_{m,1}, \dots, c_{n-1}/t_{m,n-1}, c_n/s'_{m,1}, \dots, c_{n+k-1}/s'_{m,k'}]\rangle \end{aligned}$$

Using that s and s' are equivalent by Condition 2, it is easy to see that also $c \circ t$ and $c \circ t'$ are equivalent. ◀

The preceding lemmata show that $(\sim_m^r)_{r,m\in\mathbb{N}}$ is an EF-congruence for $\mathfrak{c}^\times$, which completes the proof of Theorem 4.6. ◀

Another important class of operations under which uniform tree-automatic presentations are closed are parametrised first-order interpretations. Also in this case the complexity of the EF-congruence grows rather tamely under these operations.

▶ **Lemma 4.11.** *Let $\mathfrak{c}$ be a uniformly tree-automatic presentation of a class $\mathcal{C}$ of τ-structures and $\mathcal{I}$ be a parametrised τ-to-σ-interpretation of width ℓ that interprets for every $\mathfrak{A} \in \mathcal{C}$ a structure $\mathcal{I}(\mathfrak{A})$. Further let c be the maximal quantifier rank of any of the formulas in $\mathcal{I}$. If there is an $f(r+m)$ bounded EF-congruence for $\mathfrak{c}$ then there is a uniform tree-automatic presentation $\mathcal{I}^{\mathfrak{c}}$ of the class $\mathcal{I}^{\mathcal{C}} = \{\mathcal{I}^{\mathfrak{A}}(a) \mid \mathfrak{A} \in \mathcal{C}, a \in A\}$ with $g(r+m) := f((\ell+c)(r+m)+c)$ bounded EF-congruence.*

5 FPT Model Checking With Elementary Parameter Dependence

The runtime analysis from Section 3 not only enables us to show that first-order model checking is fixed parameter tractable on several classes of finite structures, but also gives us elementary bounds on the parameter dependence. In the following we write $\exp_k(x)$ for the k-fold tower of twos function applied to x, that is $\exp_0(x) = x$ and $\exp_{k+1}(x) = 2^{\exp_k(x)}$.

▶ **Theorem 5.1.** *Let $\mathfrak{c}$ be a uniformly tree-automatic presentation such that Algorithm 2 computes in time $T(|\varphi|)$ from $\mathfrak{c}$ the corresponding automaton $\mathcal{A}_\varphi$. Suppose for a class of finite structures $\mathcal{C}$ there is a function $f : \mathrm{code}(\mathcal{C}) \to \Gamma^*$ that computes in time $F(|w|)$ for every $w \in \mathrm{code}(\mathfrak{A})$ with $\mathfrak{A} \in \mathcal{C}$ a tree α with $\mathfrak{A} \cong \mathcal{S}(\mathfrak{c}[\alpha])$. Then* FO *model checking on $\mathcal{C}$ is decidable in time $\mathcal{O}(T(|\varphi|) \cdot |f(w)| + F(|w|))$.*

Proof. The runtime is achieved by the straight forward method of checking whether $\mathcal{A}_\varphi$ accepts $f(w)$. ◀

5.1 Boolean Algebras

Our simplest application of Theorem 4.6 and Theorem 5.1 is for the class of all finite Boolean algebras. It is well known that every finite Boolean algebra is isomorphic to a finite direct power of the two element Boolean algebra. Especially, every finite Boolean algebra contains exactly 2^n elements for some $n \geq 1$ and every finite Boolean algebra is uniquely determined by the number of elements. Because of this simple structure it is natural to consider succinct encodings of Boolean algebras as inputs. In the following we will assume that a Boolean algebra is given by the number of atoms, encoded in unary. In other words, a finite Boolean algebra $\mathfrak{B} = (B, \cap, \cup, \bar{}, \mathbf{0}, \mathbf{1})$ is encoded by the string $1^{\log|B|}$.

▶ **Theorem 5.2.** *First-order model checking is fixed parameter tractable on the class of all finite Boolean algebras. Given a Boolean algebra $\mathfrak{B}$ and an* FO *sentence φ one can decide in time $\exp_2(\mathrm{poly}(|\varphi|)) \log|\mathfrak{B}|$ whether $\mathfrak{B} \models \varphi$.*

Proof. The class that contains just the Boolean algebra $\mathfrak{B}_2 = (\{\mathbf{0}, \mathbf{1}\}, \cap, \cup, \bar{}, \mathbf{0}, \mathbf{1})$ has the trivial automatic presentation $\mathfrak{c}$ over the advice alphabet $\Sigma = \{a\}$ and the alphabet $\Gamma = \{0, 1\}$. The advice a (the tree of height 0 where the root is labeled with a) represents $\mathfrak{B}_2$ and the elements $\mathbf{0}$ and $\mathbf{1}$ are represented by 0 and 1, respectively. One checks that the relations $(E^r_m)_{r,m\in\mathbb{N}}$ where E^r_m is simply the identity relation on $T_{\hat{\Sigma}_m}$ are an EF-congruence with respect to $\mathfrak{c}$ and the index of E^r_m is bounded by $f(r+m) = 2^{r+m} + 2$ for all $r, m \in \mathbb{N}$.

As mentioned before, every finite Boolean algebra is a finite direct product of $\mathfrak{B}_2$ and hence $\mathfrak{c}^\times$ is a uniform presentation of the class of all finite Boolean algebras. According to Theorem 4.6, $\mathfrak{c}^\times$ has an EF-congruence bounded by $f'(r+m) \in 2^{\mathcal{O}((r+m+1)\left(2^{r+m}+2\right)\log(2^{r+m}+2))} \subseteq 2^{2^{\mathrm{poly}(|\varphi|)}}$. Using Theorem 5.1, we conclude that for a sentence φ of quantifier-rank r Algorithm 1 constructs the corresponding automaton $\mathcal{A}_\varphi$ in time $\mathcal{O}\left(|\varphi|\left(|\mathfrak{c}^\times|^{m+r} \cdot 2^{2^{\mathrm{poly}(|\varphi|)}}\right)^c\right) \subseteq 2^{2^{\mathrm{poly}(|\varphi|)}}$ (because $|\mathfrak{c}^\times|$ is constant). Note that the Boolean algebra with n atoms is represented by the tree $t^\#_n[c_1/a, \ldots, c_n/a]$ in $\mathfrak{c}^\times$. We can therefore transform the encoding of the Boolean algebra into the tree-representation in linear time. Finally the claim follows from Theorem 5.1. ◀

With respect to the height of the tower of twos in the parameter dependence this result is probably optimal, as stated by the following theorem.

▶ **Theorem 5.3.** *Unless $\bigcup_{c\in\mathbb{N}} \mathrm{STA}(*, 2^{cn}, n) = \mathrm{EXP}$ there is no algorithm that solves the model checking problem for finite Boolean algebras in time $2^{\mathrm{poly}(|\varphi|)} \cdot \log|\mathfrak{B}|$.*

Proof. It is known that the theory of all finite Boolean algebras is complete for the complexity class $\bigcup_{c\in\mathbb{N}} \mathrm{STA}(*, 2^{cn}, n)$. Further, using Lemma 4.2 and the computations of Theorem 5.2, we see that there is a constant c such that if $\mathfrak{B}$ and $\mathfrak{B}'$ are two Boolean algebras with at least 2^{r^c} many atoms then $\mathfrak{B} \equiv_r \mathfrak{B}'$. To check that a sentence φ of quantifier rank r belongs to the theory of finite Boolean algebras it is sufficient to check whether every finite Boolean algebra with at most 2^{r^c} many atoms models φ. If we could perform model-checking in time $\mathcal{O}(2^{\mathrm{poly}\,|\varphi|} \cdot \log|\mathfrak{B}|)$ we could hence solve the theory of finite Boolean algebras in time $\mathcal{O}\left(2^{\mathrm{poly}(|\varphi|)} \cdot \sum_{i=1}^{2^{r^c}} i\right) \subseteq 2^{\mathrm{poly}(|\varphi|)}$, which implies $\bigcup_{c\in\mathbb{N}} \mathrm{STA}(*, 2^{cn}, n) = \mathrm{EXP}$. ◀

▶ Remark. Needless to say than an analogue of Theorem 5.2 also holds if the Boolean algebra is encoded traditionally by the multiplication tables of the operators. Obviously one can compute the succinct encoding from the traditional encoding efficiently by simply counting the number of atoms.

However, one could also argue that our encoding for the Boolean algebras is not optimal. Indeed a finite Boolean algebra $\mathfrak{B}$ can be encoded by a word of length $\lceil \log\log|\mathfrak{B}|\rceil$ when we encode the number of atoms by its binary expansion. In this case our algorithm would not have a polynomial runtime in the size of the encoding of the structure because the advice would be of exponential size. However we could slot in a kernelisation procedure ahead. As we already explained in the proof of Theorem 5.3, there is a fixed polynomial p such that all finite Boolean algebras with at least $2^{p(k)}$ atoms are indistinguishable by a first-order Formula of quantifier rank at most k. In turn we can compute for a given finite Boolean Algebra $\mathfrak{B}$ and a natural number k an advice α of size $\mathcal{O}\left(2^{2^{p(k)}}\right)$ such that $\mathcal{S}(\mathfrak{c}[\alpha]) \equiv_k \mathfrak{B}$ (where $\mathfrak{c}$ is the presentation of the finite Boolean algebras constructed in Theorem 5.2). Because we are more interested in the application of automata based presentations than on encoding issues we will not work out the details here.

5.2 Finite Groups

Probably a bit more interesting is the class of all finite groups. In [10], Grohe posed the question on which classes of finite groups first-order model checking is fixed parameter tractable. In order to tackle this question we propose a structural parameter on finite groups. The Remak-Krull-Schmidt Theorem [16] states that a factorization of $\mathfrak{G} = \mathfrak{G}_1 \otimes \mathfrak{G}_2 \otimes \cdots \otimes \mathfrak{G}_n$ into indecomposable subgroups $\mathfrak{G}_i$ is unique up to permutation and isomorphism of the occurring subgroups for any finite group $\mathfrak{G}$. Therefore the size of the largest non-abelian subgroup in such a factorisation is uniquely determined. This leads to the following parameter.

▶ **Definition 5.4.** Let $\mathfrak{G}$ be a finite group. The *non-abelian decomposition width* of $\mathfrak{G}$ is $\mathrm{dw}(\mathfrak{G}) = \max(\{|\mathfrak{G}'| \mid \mathfrak{G}' \text{ is non-abelian, indecomposable, and } \mathfrak{G} \cong \mathfrak{G}' \oplus \mathfrak{G}''\})$ the size of a maximal non-abelian indecomposable factor of $\mathfrak{G}$.

Note that the finite abelian groups are exactly the groups with non-abelian decomposition width one. As for the case of Boolean algebras, finite abelian groups have a quite simple structure. By the classification of finitely generated abelian groups every finite abelian group $\mathfrak{G}$ is isomorphic to a finite sum of finite cyclic groups. That is $\mathfrak{G} \cong \mathbb{Z}_{n_1} \oplus \cdots \oplus \mathbb{Z}_{n_k}$ for some $k \geq 1$ and $n_1, \ldots, n_k \geq 1$. Hence, a finite abelian group can be encoded by a sequence of natural numbers $(n_1, \ldots, n_k)$. Bova and Martin have independently shown in [2] that first-order model-checking is FPT on the class of all finite abelian groups. Their algorithm uses a quantifier elimination procedure. However, their analysis of the algorithm only yields a non-elementary parameter dependence. We will show that the automata based approach yields an algorithm with elementary parameter dependence.

Algorithm 2 Decomposing a Finite Abelian Group into Cyclic Factors.

Input: Finite abelian group $\mathfrak{G}$
Output: String $\text{bin}(n_1)\#\cdots\#\text{bin}(n_k)$ such that $\mathfrak{G} \cong \mathbb{Z}_{n_1} \oplus \cdots \oplus \mathbb{Z}_{n_k}$

```
procedure DECOMPOSE(𝔊)
    Compute g with |g| maximal in 𝔊
    if ⟨g⟩ = 𝔊 then
        return bin(|g|)
    else
        w ← DECOMPOSE(𝔊/⟨g⟩)
        return bin(|g|)#w
    end if
end procedure
```

▶ **Theorem 5.5.** FO-*model-checking is* FPT *on the class of all finite abelian groups. More precisely one can decide given a finite abelian group* $\mathfrak{G}$ *and a formula* $\varphi \in \text{FO}$ *in time* $\mathcal{O}\left(\exp_4(\text{poly}(|\varphi|)) \cdot \log|\mathfrak{G}|\right)$ *whether* $\mathfrak{G} \models \varphi$.

Proof. Durand-Gasselin and Habermehl gave in an automatic presentation $\mathfrak{d}$ of Presburger arithmetic and proved that there is a $f(m+r) = \exp_3(c(m+r))$ bounded EF-congruence with respect to $\mathfrak{d}$ for some $c \in \mathbb{N}$ [6, Lemma 15].

We construct a uniform presentation of all finite cyclic groups from $\mathfrak{d}$ by a parametrised first-order interpretation $\mathcal{I} = (\delta(n,x), \varphi_\circ(n,x,y,z))$ in Presburger Arithmetic. It is a well known fact that such an interpretation exists. Then $\mathcal{I}^{(\mathbb{N},+)}(n) \cong \mathbb{Z}_n$ for all $n \in \mathbb{N}$ and therefore $\mathcal{I}^{\mathfrak{d}}$ is a uniform presentation of the class of all finite cyclic groups. By Lemma 4.11 there is a constant c' such that $\mathcal{I}^{\mathfrak{d}}$ has a $g(r+m) = \exp_3(c'(r+m))$ bounded EF-congruence. Further $(\mathcal{I}^{\mathfrak{d}})^\times$ is a uniform presentation of the class of all finite abelian groups and Theorem 4.6 tells us that it has a $(g(r+m)^r)^{g(r+m)} \in \exp_4(\text{poly}(|\varphi|))$ bounded EF-congruence. Note that in $(\mathcal{I}^{\mathfrak{d}})^\times$ a group $\mathfrak{G} \cong \mathbb{Z}_{n_1} \oplus \cdots \oplus \mathbb{Z}_{n_k}$ is represented by the tree $t_k^{\#}[c_1/\text{bin}^R(n_1), \ldots, c_k/\text{bin}^R(n_k)]$ (The presentation in [6] uses binary encoding). Of course this tree can trivially be computed in linear time from the encoding $(n_1, \ldots, n_k)$ of G. By applying Theorem 5.1 we conclude that our algorithm solves the model-checking problem for finite abelian groups in time $\mathcal{O}\left(\exp_4(\text{poly}(|\varphi|)) \cdot \log|G|\right)$. ◀

▶ Remark. Although the encoding of an abelian group by the orders of its cyclic factors makes it trivial to compute the tree-presentation because it makes the relevant structural properties of the group explicit, it is still true that an analog of Theorem 5.5 holds if the group is encoded by its multiplication table. Indeed Algorithm 2 provides a simple procedure to compute the cyclic factors of the group in linear time. To see this, note that if g is an element of maximal order in a finite abelian group $\mathfrak{G}$ then $\mathfrak{G} \cong \langle g \rangle \oplus \mathfrak{G}/\langle g \rangle$. The Algorithm 2 therefore computes a representant of a decomposition of $\mathfrak{G}$ into cyclic factors. The computation of an element with maximal order can be done in time $\mathcal{O}(|\mathfrak{G}|^2)$ by computing the order of every element. The group $\mathfrak{G}/\langle g \rangle$ can also be computed in time $\mathcal{O}(|\mathfrak{G}|^2)$ by computing the multiplication table on the cosets of $\langle g \rangle$. Finally the procedure DECOMPOSE($\mathfrak{G}$) is called at most $\log_2(|\mathfrak{G}|)$ times because $|\mathfrak{G}/\langle g \rangle| = |\mathfrak{G}|/|g|$. Together this gives a running time of $\mathcal{O}(|\mathfrak{G}|^2 \cdot \log(|\mathfrak{G}|))$, which is linear in the size of the multiplication table.

Finally, we turn our attention to encoding issues. As it was the case for Boolean algebras, there is an encoding of finite abelian groups, which in some cases allows for a considerably more succinct presentation. More precisely an abelian group $\mathfrak{G} \cong (\mathbb{Z}_{n_1})^{k_1} \times \cdots \times (\mathbb{Z}_{n_\ell})^{k_\ell}$

can be encoded by the tuple of pairs $((n_1, k_1), \ldots, (n_\ell, k_\ell))$. Again, using this encoding we would not directly obtain an FPT-algorithm from our method. However, using the same argument as for the Boolean algebras, for some fixed polynomial p we can truncate the second components of each pair to $\exp_3(p(r))$ in a preprocessing step, where r is the quantifier rank of the formula under consideration. Again we will leave the details of this approach to the reader.

We extend our ideas from abelian groups to groups of bounded non-abelian decomposition width.

▶ **Theorem 5.6.** *First-order model checking is* FPT *on the class of all finite groups with bounded non-abelian decomposition width. More precisely there exists a constant c such that we can decide in time $\mathcal{O}(\exp_4(\text{poly}(|\varphi|)) \cdot \log|\mathfrak{G}| + |\mathfrak{G}|^c)$ whether $\mathfrak{G} \models \varphi$.*

5.3 Graphs of bounded Tree-Depth and MSO Model Checking

Algorithmic meta-theorems for MSO are particularly interesting because MSO is capable of defining many NP-complete problems such as 3-colourability. The most famous result of this kind is probably the theorem of Courcelle that every MSO-definable query can be decided in linear time on the class of all graphs with treewidth at most c for any given constant $c \in \mathbb{N}$ [5]. Because trees have treewidth one, it is immediately clear that the parameter dependence in Courcelle's Theorem must be non-elementary. Tree-depth is another parameter on graphs that has recently drawn quite some attention. Tree-depth is a more restrictive parameter than treewidth. Indeed, every class of graphs of bounded tree-depth has also bounded treewidth but there are classes of graphs of bounded treewidth that have unbounded tree-depth. It was shown by Gajarský and Hliněný that, in terms of the parameter dependence, MSO-model-checking can be performed significantly faster on graphs of bounded tree-depth [13]. Their algorithm relies on kernelisation to perform fast MSO-model-checking on trees of bounded depth. However, transferring their arguments into our framework reveals that no specialised algorithm is needed to achieve this runtime.

▶ **Definition 5.7.** The **tree-depth** of a graph $G = (V, E)$ is recursively defined as

$$\text{td}(G) := \begin{cases} 1, & \text{if } |V| = 1 \\ \min\{\text{td}(G \restriction V \setminus \{v\}) \mid v \in V\} + 1 & \text{if } G \text{ is connected and } |V| > 1 \\ \max_{1 \le i \le n} \text{td}(G_i) & G \text{ has components } G_1, \ldots, G_n \end{cases}$$

An equivalent characterisation is the minimal height of a rooted forest such that G is isomorphic to a subgraph of the symmetric closure of the ancestor-descendant graph of that forest.

Again a straight forward encoding yields for every $h > 0$ a uniformly automatic presentation of the class of all graphs of tree-depth at most h. Translating the ideas of [13] into our framework shows that our generic algorithm performs just as good as the best known specialized algorithms.

▶ **Theorem 5.8.** *The* MSO *model checking problem for graphs of tree-depth at most h is fixed parameter tractable. Given an* MSO *sentence φ and a graph $\mathfrak{G}$ of tree-depth at most h one can decide in time $\mathcal{O}\left(\exp_{(h+2)}(\text{poly}(|\varphi|)) \cdot \text{poly}(|\mathfrak{G}|)\right)$ whether $\mathfrak{G} \models \varphi$.*

References

1. Faried Abu Zaid, Erich Grädel, and Frederic Reinhardt. Advice Automatic Structures and Uniformly Automatic Classes. In *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*, 2017. URL: http://www.logic.rwth-aachen.de/pub/abuzaid/AbuGraRei17.pdf.
2. Simone Bova and Barnaby Martin. First-Order Queries on Finite Abelian Groups. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic (CSL 2015)*, volume 41 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 41–59, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CSL.2015.41.
3. Thomas Colcombet and Christof Löding. Transforming structures by set interpretations. *CoRR*, abs/cs/0703039, 2007. arXiv:cs/0703039.
4. B. Courcelle, J. A. Makowsky, and U. Rotics. Linear Time Solvable Optimization Problems on Graphs of Bounded Clique-Width. *Theory Comput. Systems*, 33(2):125–150, April 2000. doi:10.1007/s002249910009.
5. Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, March 1990. doi:10.1016/0890-5401(90)90043-H.
6. Antoine Durand-Gasselin and Peter Habermehl. Ehrenfeucht-Fraïssé goes elementarily automatic for structures of bounded degree. In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France*, volume 14 of *LIPIcs*, pages 242–253. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012. doi:10.4230/LIPIcs.STACS.2012.242.
7. Andrzej Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fundamenta Mathematicae*, 49(2):129–141, 1961. URL: https://eudml.org/doc/213582.
8. Jochen Eisinger. Upper Bounds on the Automata Size for Integer and Mixed Real and Integer Linear Arithmetic (Extended Abstract). In Michael Kaminski and Simone Martini, editors, *Computer Science Logic*, number 5213 in Lecture Notes in Computer Science, pages 431–445. Springer Berlin Heidelberg, September 2008. DOI: 10.1007/978-3-540-87531-4_31. URL: http://link.springer.com/chapter/10.1007/978-3-540-87531-4_31.
9. Jeanne Ferrante and Charles W. Rackoff. *The Computational Complexity of Logical Theories*, volume 718 of *Lecture Notes in Mathematics*. Springer Berlin Heidelberg, 1979. URL: http://link.springer.com/10.1007/BFb0062837.
10. JÖRG Flum, ERICH Grädel, and THOMAS Wolke, editors. *Logic and Automata: History and Perspectives*. Amsterdam University Press, 2008. URL: http://www.jstor.org/stable/j.ctt46mv83.
11. Roland Fraïssé. Sur l'extension aux relations de quelques propriétés des ordres. *Annales scientifiques de l'École Normale Supérieure*, 71(4):363–388, 1954. URL: https://eudml.org/doc/81696.
12. M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. In *17th Annual IEEE Symposium on Logic in Computer Science, 2002. Proceedings*, pages 215–224, 2002. doi:10.1109/LICS.2002.1029830.
13. Jakub Gajarsky and Petr Hlineny. Faster Deciding MSO Properties of Trees of Fixed Height, and Some Consequences. In Deepak D'Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012)*, volume 18 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 112–123, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSTTCS.2012.112.

14 Martin Grohe. Logic, Graphs, and Algorithms. *Electronic Colloquium on Computational Complexity (ECCC)*, 14(091), 2007. URL: http://eccc.hpi-web.de/eccc-reports/2007/TR07-091/index.html.

15 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding First-order Properties of Nowhere Dense Graphs. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 89–98, New York, NY, USA, 2014. ACM. doi:10.1145/2591796.2591851.

16 Thomas W. Hungerford. *Algebra*, volume 73 of *Graduate Texts in Mathematics*. Springer New York, New York, NY, 1980. URL: http://link.springer.com/10.1007/978-1-4612-6101-8.

17 Neeraj Kayal and Timur Nezhmetdinov. Factoring Groups Efficiently. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 5555 of *Lecture Notes in Computer Science*. Springer Verlag, 2009. URL: http://research.microsoft.com/apps/pubs/default.aspx?id=102435.

18 Felix Klaedtke. Ehrenfeucht–Fraïssé goes automatic for real addition. *Information and Computation*, 208(11):1283–1295, November 2010. doi:10.1016/j.ic.2010.07.003.

19 Dietrich Kuske and Markus Lohrey. Automatic structures of bounded degree revisited. *J. Symbolic Logic*, 76(4):1352–1380, December 2011. doi:10.2178/jsl/1318338854.

20 Alexander Langer, Felix Reidl, Peter Rossmanith, and Somnath Sikdar. Evaluation of an MSO-Solver. In David A. Bader and Petra Mutzel, editors, *ALENEX*, pages 55–63. SIAM / Omnipress, 2012. URL: http://dblp.uni-trier.de/db/conf/alenex/alenex2012.html#LangerRRS12.

21 Derek C. Oppen. A $2^{2^{2^{pn}}}$ upper bound on the complexity of Presburger Arithmetic. *Journal of Computer and System Sciences*, 16(3):323–332, June 1978. doi:10.1016/0022-0000(78)90021-1.

22 Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. A Faster Parameterized Algorithm for Treedepth. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming*, number 8572 in Lecture Notes in Computer Science, pages 931–942. Springer Berlin Heidelberg, July 2014. DOI: 10.1007/978-3-662-43948-7_77. URL: http://link.springer.com/chapter/10.1007/978-3-662-43948-7_77.

23 Detlef Seese. Linear Time Computable Problems and First-Order Descriptions. *Mathematical Structures in Computer Science*, 6(6):505–526, 1996.

A Proofs Omitted from Section 4

In order to analyse Algorithm 1 with respect to the given presentation, we observe the following runtime bounds for the subroutines. We omit the proofs here as they are easily obtainable by a straight forward analysis of the respective routines.

▶ **Lemma A.1.** *The procedure* Intersect$(\mathcal{A}_1, \mathcal{A}_2)$ *computes a tree-automaton* $\mathcal{A}$ *with* s *states and* $L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$ *in time* $\mathcal{O}(|\Gamma| \cdot s^2)$*, where* s *is the number of states reachable from the initial state in the product automaton* $\mathcal{A}_1 \times \mathcal{A}_2$*.*

▶ **Lemma A.2.** *The procedure* Determinize$(\mathcal{A})$ *computes a deterministic tree-automaton* $\mathcal{A}'$ *with* s *states and* $L(\mathcal{A}') = L(\mathcal{A})$ *in time* $\mathcal{O}(|\Gamma| \cdot s^2)$*, where* s *is the number of states reachable from the initial state in the power set automaton of* $\mathcal{A}$*.*

▶ **Lemma A.3.** *Let* Γ *be a ranked alphabet,* $\sim$ *an equivalence relation on* T_Γ*, and* $\mathcal{A}_1 = (Q_1, \Gamma, \delta_1, q_{01}, F_1)$*,* $\mathcal{A}_2 = (Q_1, \Gamma, \delta_1, q_{01}, F_1)$ *tree-automata. Suppose* $t \sim t'$ *implies* $\delta_i^*(t) = \delta_i^*(t')$ *for all* $i \in \{1, 2\}$ *and for all* $t, t' \in T_\Gamma$*. Then the number of reachable states from the initial state in* $\mathcal{A}_1 \times \mathcal{A}_2$ *is bounded by the index of* $\sim$*.*

▶ **Theorem 4.3.** *Let $\mathfrak{c} = (\mathcal{A}, (\mathcal{A}_R)_{R\in\tau})$ be a uniformly tree-automatic presentation of a class of τ-structures. Suppose there is an $f(r+m)$ bounded EF-congruence $(E^r_m)_{r,m\in\mathbb{N}}$ for $\mathfrak{c}$. Then for every $\varphi(x_1,\ldots,x_m) \in \mathrm{FO}$ of quantifier rank r Algorithm 2 computes the automaton $\mathcal{A}_\varphi$ in time $\mathcal{O}(|\varphi|(|\mathfrak{c}|^{m+r}\cdot f(m+r))^c)$ for some constant c.*

Proof. We prove the claim by induction over the structure of φ. Actually we prove an extended claim, namely that the procedure computes the automaton $\mathcal{A}_\varphi$ in the given time and $\mathcal{A}_\varphi$ has the property $\delta^*_{\mathcal{A}_\varphi}(t) = \delta^*_{\mathcal{A}_\varphi}(t')$ for all $t, t' \in T_{\hat{\Sigma}_m}$ with $tE^r_m t'$.

Case $\varphi = R(x_{i_1},\ldots,x_{i_k})$: Obviously $|\mathcal{A}_\varphi| \leq |\mathfrak{c}|^m$ and therefore there is a fixed polynomial p such that $\mathcal{A}_\varphi$ is constructed in time $p(|\mathfrak{c}|^m)$. Further, by construction, the automata $\mathcal{A}'_R$ and $\mathcal{A}_D$, from which $\mathcal{A}_\varphi$ is build up, are minimal. Let s, s' be two trees from $T_{\hat{\Sigma}_m}$ with $sE^0_m s'$. Then by Property 4 also $(c\circ s)E^0_m(c\circ s')$ for all $\hat{\Sigma}_m$-contexts. If $c\circ s$ is not a convolution of a tuple $(\alpha,\bar{t})$ with $\bar{t}\in\mathcal{S}(\mathfrak{c}[\alpha])$ then because of the first property of E^0_m the same holds for $c\circ s'$. Hence $c\circ s \notin L(\mathcal{A}_\varphi)$ and $c\circ s \notin L(\mathcal{A}_\varphi)$. Otherwise $c\circ s = \langle\alpha,\bar{t}\rangle$ and $c\circ s' = \langle\beta,\bar{t}'\rangle$ and Property 2 yields $\langle\alpha,\bar{t}\rangle \in L(\mathcal{A}_\varphi) \Leftrightarrow \langle\beta,\bar{t}'\rangle \in L(\mathcal{A}_\varphi)$. We obtain from Myhill-Nerode Theorem for tree-languages that $\delta^*_{\mathcal{A}_\varphi}(s) = \delta^*_{\mathcal{A}_\varphi}(s')$.

Case $\varphi = \psi(x_1,\ldots,x_m)\wedge\gamma(x_1,\ldots,x_m)$: Let $\mathcal{A}_\psi$ and $\mathcal{A}_\gamma$ be the automata constructed by Compose in the recursion step. By the induction hypothesis, we know that all pairs of tuples t, t' that are related by E^r_m the computation of $\mathcal{A}_\psi$ and $\mathcal{A}_\gamma$ reach the same state. Lemma A.3 tells us that the number of reachable states in $\mathcal{A}_\varphi\times\mathcal{A}_\psi$ is bounded by $f(m+r)$. The automata $\mathcal{A}_\psi$ and $\mathcal{A}_\gamma$ are computed in at most $d|\psi|(|\mathfrak{c}|^{m+r}\cdot f(m+r))^c + d|\gamma|(|\mathfrak{c}|^{m+r}\cdot f(m+r))^c$ many steps and, according to Lemma A.1, the computation of $\mathcal{A}_\varphi$ takes at most $d'|\hat{\Sigma}_m|f(m+r)^2$. But $|\mathfrak{c}|^{m+r}$ is an upper bound for $|\hat{\Sigma}_m|$. Hence the overall runtime is bounded by $d(|\psi|+|\gamma|+1)(|\mathfrak{c}|^{m+r}\cdot f(m+r))^c = d|\varphi|(|\mathfrak{c}|^{m+r}\cdot f(m+r))^c$. The property $tE^r_m t' \Rightarrow \delta^*_{\mathcal{A}_\varphi}(t) = \delta^*_{\mathcal{A}_\varphi}(t')$ follows directly from the induction hypothesis and the fact that $\delta^*(t) = (\delta^*_{\mathcal{A}_\psi}(t), \delta^*_{\mathcal{A}_\gamma}(t))$.

Case $\varphi = \neg\psi(x_1,\ldots,x_m)$: By the induction hypothesis the automaton $\mathcal{A}_\psi$ is constructed in time $d|\psi|(|\mathfrak{c}|^{m+r}\cdot f(m+r))^c$. The automaton $\mathcal{A}_D$ is the minimal automaton that recognises exactly the words of the form $\langle\alpha,t_1,\ldots,t_m\rangle$, where $\alpha\in P^{\mathfrak{c}}$ and $t_1,\ldots,t_m$ are elements of $\mathcal{S}(\mathfrak{c}[\alpha])$. Using the properties 1 and 4 of Definition 4.1, we see that for all $t, t' \in T_{\hat{\Sigma}_m}$ with $tE^r_m t'$ and all $\hat{\Sigma}_m$-contexts c it is the case that $c\circ t \in L(\mathcal{A}_D) \Leftrightarrow c\circ t' \in L(\mathcal{A}_D)$. Therefore we can once again apply the lemmata A.3 and A.1 to establish that also $\mathcal{A}_\varphi$ is constructed in the right amount of time and has the proclaimed property (recall that $\mathcal{A}_\varphi$ is the product automaton of $\overline{\mathcal{A}_\psi}$ and $\mathcal{A}_D$).

Case $\varphi = \exists x_{m+1}\psi(x_1,\ldots,x_m,x_{m+1})$: Let $\mathcal{A}_\psi$ be the automaton that is constructed in the recursion step. Then $\mathcal{A}_\varphi$ is essentially the reachable part of the power-set automaton of the projection automaton derived from $\mathcal{A}_\psi$ under the projection $(\sigma,\gamma_1,\ldots,\gamma_{m+1}) \mapsto (\sigma,\gamma_1,\ldots,\gamma_m)$. Now suppose $sE^{r+1}_m s'$ for some $s, s' \in T_{\hat{\Sigma}_m}$. Then $q \in \delta^*_{\mathcal{A}_\varphi}(s)$ if and only if there is a $t\in T_\Gamma$ such that $\delta^*_{\mathcal{A}_\psi}(\langle s,t\rangle) = q$. But then, by Property 3 of Definition 4.1, there is also a $t'\in T_\Gamma$ with $\langle s,t\rangle E^r_m\langle s',t'\rangle$. By the induction hypothesis $\delta^*_{\mathcal{A}_\psi}(s',t') = q$ and thus $q\in\delta^*_{\mathcal{A}_\varphi}(s')$. This shows that $sE^r_m s'$ implies $\delta^*_{\mathcal{A}_\varphi}(s) = \delta^*_{\mathcal{A}_\varphi}(s')$. Consequently the number of reachable states in the aforementioned power set automaton is bounded by $f(m+r)$. We can now apply the induction hypothesis and Lemma A.2 to conclude that the algorithm takes at most $d|\varphi|(\mathfrak{c}^{m+r}f(m+r))^c$ many steps to compute $\mathcal{A}_\varphi$. ◀

B Proofs Omitted from Section 5

▶ **Theorem 5.6.** *First-order model checking is* FPT *on the class of all finite groups with bounded non-abelian decomposition width. More precisely there exists a constant c such that we can decide in time $\mathcal{O}(\exp_4(\mathrm{poly}(|\varphi|)) \cdot \log|\mathfrak{G}| + |\mathfrak{G}|^c)$ whether $\mathfrak{G} \models \varphi$.*

Proof. First we build a trivial presentation $\mathfrak{d}$ for the groups of order at most d. Let $\mathfrak{G}_1, \dots, \mathfrak{G}_n$ be an enumeration of the non-abelian groups of size at most d (up to isomorphism). The advice alphabet is $\Sigma = \{g_1, \dots, g_n\}$. The input alphabet Γ is extended by new letters $a_1, \dots, a_d$. For every $1 \leq i \leq n$ we choose a bijection $\pi_i : \{a_1, \dots, a_{|\mathfrak{G}_i|}\} \to \mathfrak{G}_i$ and construct the automata that recognise the languages $\{\langle g_i, a_j\rangle \mid 1 \leq i \leq n, j \leq |\mathfrak{G}_i|\}$ and $\{\langle g_i, a_x, a_y, a_z\rangle \mid 1 \leq i \leq n, 1 \leq x, y, z \leq |\mathfrak{G}_i|, \pi_i(a_x) \circ_{\mathfrak{G}_i} \pi_i(y) = \pi_i(a_z)\}$. Note that the trivial EF-congruence for $\mathfrak{d}$ is $g(m+r) = G(d)d^{r+m}$ bounded, where $G(d)$ is the number of groups of size at most d.

Let $\mathfrak{c}$ be the uniform presentation of the cyclic groups as described previously. We build automata that recognize the alphabet-disjoint union of the languages in $\mathfrak{d}$ and corresponding languages from $\mathfrak{c}$ and obtain a presentation $\mathfrak{e}$ of all cyclic groups and groups of order at most d. It is not hard to see that this presentation is also $\exp_3(\mathrm{poly}(|\varphi|))$ bounded. Basically the union of the EF-congruences for $\mathfrak{d}$ and $\mathfrak{c}$ (where the "is not a tuple of the presentation" equivalence class of $\mathfrak{d}$ is merged with the "is not a convolution" equivalence class of $\mathfrak{c}$) is an EF-congruence for $\mathfrak{e}$. Then $\mathfrak{e}^\times$ is a presentation of the class of all finite groups with bounded abelian decomposition width at most d. By Theorem 4.6, $\mathfrak{e}^\times$ is $\exp_4(\mathrm{poly}(|\varphi|))$ bounded.

A decomposition of $\mathfrak{G} = \mathfrak{G}_1 \oplus \cdots \oplus \mathfrak{G}_k \oplus \mathbb{Z}_{n_1} \oplus \cdots \oplus \mathbb{Z}_{n_\ell}$ with non-abelian indecomposable factors $\mathfrak{G}_1, \dots, \mathfrak{G}_k$ can be computed in polynomial time [17]. From the decomposition we can compute in linear time an advice that represents $\mathfrak{G}$. Note that such an advice has logarithmic size in $|\mathfrak{G}|$. Applying Theorem 5.1 completes the proof. ◀

In order to handle MSO model-checking on graphs of bounded tree-depth we need to enrich the structure by the powerset of the universe in order to simulate quantification over sets.

▶ **Definition B.1** ([3])**.** Let $\mathfrak{A} = (A, R_1, \dots, R_n)$ be a τ-structure. The **power set structure** $\mathcal{P}(\mathfrak{A})$ is the $(\tau \uplus \{\subseteq\})$-structure $(\mathcal{P}(A), R_1^{\mathcal{P}(\mathfrak{A})}, \dots, R_n^{\mathcal{P}(\mathfrak{A})}, \subseteq)$, where $(\mathcal{P}(A), \subseteq)$ is the powerset lattice on A and $R_i^{\mathcal{P}(\mathfrak{A})} = \{(\{a_1\}, \dots, \{a_{r_i}\}) \in \mathcal{P}(A)^{r_i} \mid (a_i, \dots, a_{r_i}) \in R_i\}$ for all $i \in \{1, \dots, n\}$.

Clearly the MSO-theory of $\mathfrak{A}$ is reducible to the FO-theory of $\mathcal{P}(\mathfrak{A})$ and vice versa. In the following we also need to make a distinction between trees that serve as an input to a tree automaton and an unordered rooted tree in the graph theoretic sense. A **finite unordered labeled tree-structure** $\mathfrak{T}$ is a tuple $(V, E, P_1, \dots, P_n, r)$ where

- V is a finite set of nodes,
- $E \subseteq \binom{V}{2}$ such that (V, E) is connected and acyclic,
- $P_i \subseteq V$ for all $1 \leq i \leq n$, and
- $r \in V$ is the root of the tree.

There are standard techniques to encode a finite unordered tree-structures of unbounded degree by trees of bounded degree.

▶ **Definition B.2.** For a finite unordered tree-structure $\mathfrak{T} = (V, E, P_1, \dots, P_n, r)$ the set of **tilts** of $\mathfrak{T}$, $\mathrm{tilt}(\mathfrak{T}) \subseteq T_{\mathcal{P}(\{1,\dots,n\})}$, is inductively defined by the following rules.

- if $\mathfrak{T} = (\{v\}, P_1, \dots, P_n, v)$ then $\mathrm{tilt}(\mathfrak{T}) = \{t\}$, where $\mathrm{dom}(t) = \{\epsilon\}$ and $t(\epsilon) = \{i \mid v \in P_i\}$

- if $\mathfrak{T} = (V, P_1, \dots, P_n, r)$ is of depth $h > 1$ then $t \in \text{tilt}(\mathfrak{T})$ if, and only if, there is an enumeration $\mathfrak{T}_0, \dots, \mathfrak{T}_k$ of the subtrees induced by the children of the root r of $\mathfrak{T}$ such that there are trees $t_0, \dots, t_k$ with
  - t_i is a tilt of $\mathfrak{T}_i$,
  - $\text{dom}(t) = \bigcup_{0 \le i \le k} \{1^i 0\}\text{dom}(t_i)$,
  - $t(w) = \begin{cases} \{i \mid w \in P_i\} & w = \epsilon \\ X & w \in \{1\}^i, 1 \le i \le k \\ t_i(w') & w = 1^i 0 w', 0 \le i \le k \end{cases}$

Note that if t is a tilt of a tree-structure $\mathfrak{T}$ and $v \in \text{dom}(t)$ with $t(v) \neq X$ then v corresponds to a node of depth $|v|_0 + 1$ in $\mathfrak{T}$.

▶ **Lemma B.3.** *Let $h \in \mathbb{N}$ be some fixed number. Then the class $\mathcal{C}_h$ of all power set structures of graphs of tree-depth at most h is uniformly tree-automatic.*

Proof. The advice set consists of all tilts of tree-structures $(V, E, P_1, \dots, P_{h-1})$ of depth at most $h+1$ such that every node of depth ℓ appears only in sets P_i with $i + 1 < \ell$. This is obviously a regular set. Such a tree α presents (the isomorphism type of) the graph $\mathfrak{G} = (V, E)$ with $V = \text{dom}(\alpha) \cap (\{0,1\}^*\{0\})$ and $E = \{\{v, w\} \mid v \preceq w \text{ and } |v|_0 \in \alpha(w)\}$. If α is a tilt of an optimal decomposition of $\mathfrak{G}$ then the subtrees induced by the nodes in $\text{dom}_\alpha \cap \{1\}^*\{0\}$ correspond to the connected components of $\mathfrak{G}$. Building a uniformly tree-automatic presentation $\mathfrak{c} = (\mathcal{A}, \mathcal{A}_E, \mathcal{A}_\subseteq)$ is then straight forward. The automaton $\mathcal{A}$ is chosen such that $L(\mathcal{A}[\alpha]) = \{t \in T_{\{0,1,X\}} \mid \text{dom}(t) = \text{dom}(\alpha) \wedge \forall w \in \text{dom}(\alpha) : \alpha(w) = X \to t(w) = X\}$. A tree $t \in L(\mathcal{A}[\alpha])$ represents the set $\{v \in \text{dom}(\alpha) \mid t(v) = 1\}$. Then the relation $\subseteq$ is trivially regular and the relation E can also be recognised with the advice α, because the prefix relation is regular on the domain of a tree and $|w|_0 \le h$ for every $w \in \text{dom}(t)$ and every $t \in L(\mathcal{A}[\alpha])$, so an automaton can check whether w is the first ancestor with $|w|_0 = i$ of a node v with $i \in t(v)$. ◀

For a tree $t \in T_\Sigma$, $w \in \text{dom}_t$, and $a \in \Sigma$ we write $t[w \to a]$ for the tree that is obtained by replacing the label of the node w by a.

▶ **Theorem 5.8.** *The* MSO *model checking problem for graphs of tree-depth at most h is fixed parameter tractable. Given an* MSO *sentence φ and a graph $\mathfrak{G}$ of tree-depth at most h one can decide in time $\mathcal{O}\left(\exp_{(h+2)}(\text{poly}(|\varphi|)) \cdot \text{poly}(|\mathfrak{G}|)\right)$ whether $\mathfrak{G} \models \varphi$.*

Proof. We define the EF-congruence on the basis of equivalence relations $(\sim^h_{r,k})_{r,k \in \mathbb{N}}$ on $(P_1, \dots, P_k)$-labeled tree-structures of depth h:

- For tree-structures $\mathfrak{S}, \mathfrak{T}$ of depth 1 we define $\mathfrak{S} \sim^1_{r,k} \mathfrak{T} :\Leftrightarrow \mathfrak{S} \cong \mathfrak{T}$.
- Let $\mathfrak{S}, \mathfrak{T}$ be trees of depth $h+1$ and let $\mathfrak{S}^i_1, \dots, \mathfrak{S}^i_{n_i}$ be the subtrees of depth i rooted in a child node of the root in $\mathfrak{S}$ for all $i \le h$ and let $\mathfrak{T}^i_1, \dots, \mathfrak{T}^i_{n'_i}$ be the corresponding trees with respect to $\mathfrak{T}$. Then $\mathfrak{S}$ and $\mathfrak{T}$ are $\sim^{h+1}_{r,k}$-equivalent if, and only if, the roots of $\mathfrak{T}$ and $\mathfrak{S}$ share the same labels and for all $i < h$ and all $\sim^i_{r,k}$-equivalence classes κ

$$|\{j \in \mathbb{N} \mid j \le n_i, \mathfrak{S}^i_j \in \kappa\}| =_{\text{index}(\sim^i_{r,k})^r + 1} |\{j \in \mathbb{N} \mid j \le n'_i, \mathfrak{T}^i_j \in \kappa\}|$$

The proof of [13, Theorem 3.1] can be easily adapted to show that no FO-formula with r quantifiers can distinguish between two power set structures of two $\sim^h_{r,(r+k)}$-equivalent $(P_1, \dots, P_k)$-labeled tree-structures of depth h.

Moreover, a straightforward induction shows that whenever two such tree-structures $\mathfrak{S}, \mathfrak{T}$ of depth h are $\sim^h_{0,k}$-equivalent then the following two observations hold for every path $v_0 v_1 \dots v_n$ in $\mathfrak{S}$ starting from the root:

1. There is a path $w_0 w_1 \dots w_n$ in $\mathfrak{T}$ starting from the root of $\mathfrak{T}$ such that for all $0 \leq i \leq n$ the nodes v_i and w_i share the same labels, that is $v_i \in P_j^{\mathfrak{S}} \Leftrightarrow w_i \in P_j^{\mathfrak{T}}$ for all $1 \leq j \leq k$.
2. If for some subsets $I \subseteq \{1, \dots, n\}$, $J \subseteq \{1, \dots, k\}$ the nodes v_i with $i \in I$ are unique in the sense that for every path $v_0' v_1' \dots v_n'$ with $v_i \in P_j^{\mathfrak{S}} \Leftrightarrow v_i' \in P_j^{\mathfrak{S}}$ for all $i \in I$ and $j \in J$ implies $v_i = v_i'$ for all $1 \leq i \leq n$ then there is also a unique path $w_0 w_1 \dots w_n$ $\mathfrak{T}$ with $w_i \in P_j^{\mathfrak{T}} \Leftrightarrow v_i \in P_j^{\mathfrak{S}}$ for all $i \in I, j \in J$.

Let $\sim_{r,k}^{\leq h} := \bigcup_{1 \leq i \leq h} \sim_{r,k}^{i}$. We define an EF-congruence for the presentation in Lemma B.3 from $\sim_{r,k}^{\leq h}$. Let h be fixed.

In a first step, we partition the set of all $(\mathcal{P}(\{1, \dots, h+m-1\}) \uplus \{X\})$-labeled trees into $2h+1$ classes $T_1^m, \dots, T_h^m, Q_1^m, \dots, Q_h^m, F$.

- A tree t is in T_i^m if, and only if, t is a tilt of a tree of depth i.
- A tree t is in Q_i^m if, and only if, t is not a tilt of a tree of depth i but $t[\epsilon \to \emptyset]$ is a tilt of a tree of depth i (this is exactly the case if $t = t'[\epsilon \to X]$ for some tilt t' of a tree of depth i).
- All other trees are in F.

The EF-congruence is then defined by

$$\begin{aligned} tE_m^r t' :\Leftrightarrow & \exists 0 \leq i \leq h : (t \in T_i^m \wedge t' \in T_i^m \wedge \\ & \exists \mathfrak{S}, \mathfrak{S}' : t \in \mathrm{tilt}(\mathfrak{S}) \wedge t' \in \mathrm{tilt}(\mathfrak{S}') \wedge \mathfrak{S} \sim_{r,(r+m+k)}^{i} \mathfrak{S}') \\ & \vee \exists 0 \leq i \leq h : (t \in Q_i^m \wedge t' \in Q_i^m \wedge \\ & \exists \mathfrak{S}, \mathfrak{S}' : t[\epsilon \to \emptyset] \in \mathrm{tilt}(\mathfrak{S}) \wedge t'[\epsilon \to \emptyset] \in \mathrm{tilt}(\mathfrak{S}'') \wedge \mathfrak{S} \sim_{r,(r+m+k)}^{i} \mathfrak{S}') \\ & \vee t, t' \in F \end{aligned}$$

For Property 1 let us consider under which circumstances a tree t does not present a graph of tree-depth at most h. First of all t might not be a tilt of a tree-structure of depth at most $h+1$. In this case $t \in F$ or $t \in Q_i$ for some $i \leq h+1$. In this case E_m^r seperates t from all trees that represent a graph from $\mathcal{C}_h$. Otherwise t might be the tilt of a tree-structure $\mathfrak{T}$ of depth at most $h+1$ but there is a note $v \in \mathfrak{T}$ of depth i with $v \in P_i^{\mathfrak{T}}$ and $i+1 \geq j$. But then by Observation 1 every E_m^r-equivalent tree-structure contains also a node of depth j which is contained in P_i and therefore does also not present a graph from $\mathcal{C}_h$.

We use Observation 2 to show that Property 2 is fulfilled. Let s and t be $(\mathcal{P}(\{1, \dots, h+m-1\}) \cup \{X\})$-labeled trees that present Structures in $\mathfrak{c}$ with $sE_m^r t$. Let $\mathfrak{S}, \mathfrak{T}$ be the tree-structures with $s \in \mathrm{tilt}(\mathfrak{S})$ and $t \in \mathrm{tilt}(\mathfrak{T})$, let $(\mathfrak{G}_s, V_1, \dots, V_m)$ be the tuple presented by s, and $(\mathfrak{G}_t, W_1, \dots, W_m)$ be the tuple presented by t. If $\mathfrak{G}_s \models E(V_i, V_j)$ for some $i, j \leq m$ then V_i and V_j are singletons and therefore there are unique nodes v_i, v_j with $v_i \in P_{h+i-1}^{\mathfrak{S}}$ and $v_j \in P_{h+j-1}^{\mathfrak{S}}$. Further v_i and v_j are ordered by the ancestor-relationship. Without loss generality assume that v_i is an ancestor of v_j and let d be the depth of v_i in $\mathfrak{S}$. Then $v_j \in P_{d-1}^{\mathfrak{S}}$. By Observation 2 there must be unique nodes w_i, w_j with $w_i \in P_{h+i-1}^{\mathfrak{T}}$ and $w_j \in P_{h+j-1}^{\mathfrak{T}}$. Further w_i has depth d, is an ancestor of w_j, and $w_j \in P_{d-1}^{\mathfrak{T}}$. Hence $\mathfrak{G}_t \models E(W_i, W_j)$. If $\mathfrak{G}_s \not\models V_i \subseteq V_j$ then there is node $v \in \mathrm{dom}_s$ such that $i \in s(v)$ but $j \notin s(v)$. Using similar arguments as in the previous case we can follow that there is also a $w \in \mathrm{dom}_t$ with $i \in s(v)$ and $j \notin s(v)$. Hence $\mathfrak{G}_t \not\models W_i \subseteq W_j$. The case of $\mathfrak{G}_s \not\models V_i = V_j$ is analogous.

In order to establish Property 3 suppose $sE_m^{r+1} t$. Let s' be any tree that can be derived from s by adding the label $(h+m)$ to some nodes $w \in \mathrm{dom}(s) \cap \{0,1\}^*\{0\}$. We distinguish three cases.

Case $s, t \in F$: then $t' \in F$ and we can extend the labeling of t in an arbitrary way to obtain an E^r_{m+1}-equivalent t'.

Case $s, t \in T^m_i$ for some $1 \leq i \leq h$: then there is a $(P_1, \ldots, P_{h+m-1})$-labeled tree-structures $\mathfrak{S}, \mathfrak{T}$ of depth i with $s \in \text{tilt}(\mathfrak{S})$ and $t \in \text{tilt}(\mathfrak{T})$. Further there is a set $X_{\mathfrak{S}} \subseteq \mathfrak{S}$ such that s' is a tilt of $(\mathfrak{S}, X_{\mathfrak{S}})$. Because $\mathfrak{S} \sim^h_{(r+1),((r+1)+h+m)} \mathfrak{T}$ there must be a set $X_{\mathfrak{T}} \subseteq \mathfrak{T}$ such that $(\mathfrak{S}, X_{\mathfrak{S}}) \sim^h_{r,(r+h+(m+1))} (\mathfrak{T}, X_{\mathfrak{T}})$. Finally choose the extension t' of the labeling of t such that $t' \in \text{tilt}((\mathfrak{T}, X_{\mathfrak{T}}))$. Then $t' E^r_{m+1} s'$.

Case $s, t \in Q^m_i$ for some $1 \leq i \leq h$: the case follows analogously to the previous one by considering $s[\epsilon \to \emptyset]$ and $t[\epsilon \to \emptyset]$.

At last, we see that Property 4 holds. Indeed, if $t \in F$ then $(c \circ t) \in F$ for every context c. For the case $s, t \in T^m_i$ for some $1 \leq i \leq h$ one can distinguish two cases based on the structure of the context c.

Case $c^{-1}(x) \in \{0,1\}^*\{1\} \cup (\{1\}^*\{0\})^{>(h-i)}$: then $(c \circ s)$ and $(c \circ t)$ do not present trees of depth at most h and hence $s, t \in F$.

Case $c^{-1}(x) \in (\{1\}^*\{0\})^{\leq(h-i)}$: there are three subcases that might occur.

- It might be that $(c \circ t) \in F$ and $(c \circ s) \in F$ (because c is a "template" of a tree of depth larger than h or c contains an inconsistent labeling). in this case equivalence is guaranteed by definition.
- It is also possible that $(c \circ t) \in T^m_j$ and $(c \circ s) \in T^m_j$ for some $i \leq j \leq h$. Then let $\mathfrak{S}, \mathfrak{T}$ be trees of depth j such that $(c \circ t) \in \text{tilt}(\mathfrak{S})$ and $(c \circ s) \in \text{tilt}(\mathfrak{T})$. By induction over $j - i$ one shows that $\mathfrak{S} \sim^j_{r,r+h+m} \mathfrak{T}$. For $j - i = 0$ this is the case by definition. For $j - i = k + 1$ let $\mathfrak{S}_1, \ldots, \mathfrak{S}_\ell$ and $\mathfrak{T}_1, \ldots, \mathfrak{T}_\ell$ be the subtrees of $\mathfrak{S}$ and $\mathfrak{T}$ that are rooted in the children of the roots $\mathfrak{S}$ and $\mathfrak{T}$, respectively. Without loss of generality assume that $\mathfrak{S}_1$ and $\mathfrak{T}_1$ are the subtrees which resulted from adding s and t into the context c. Then by the induction hypothesis $\mathfrak{S}_1 \sim^h_{r,r+h+m} \mathfrak{T}_1$ and also $\mathfrak{S}_n \cong \mathfrak{T}_n$ for all $1 < n \leq \ell$. But then for all $n < j$ and all $\sim^n_{r,r+h+m}$-equivalence classes τ the number of τ-children of the root in $\mathfrak{S}$ is equal to the number in $\mathfrak{T}$, hence $\mathfrak{S} \sim^j_{r,r+h+m} \mathfrak{T}$ and therefore $(c \circ s) E^r_m (c \circ t)$.
- The last case that might happen is $(c \circ t) \in Q^m_j$ and $(c \circ s) \in Q^m_j$ for some $i \leq j \leq h$. In this case we might again argue analogously to the previous cases by considering $(c \circ t)[\epsilon \to \emptyset]$ and $(c \circ s)[\epsilon \to \emptyset]$.

Next, let us estimate the index of E^r_m. By the definition of E^r_m, $\text{index}(E^r_m) \leq 1 + 2\sum_{i=0}^{h+1} \text{index}(\sim^i_{r,r+m+h+1})$. An inductive analysis of $\text{index}(\sim^i_{r,r+m+h+1})$ (see [13, Lemma 3.1 c)]) shows $\text{index}(\sim^i_{r,r+m+h+1}) \in \exp_{(i+1)}(\text{poly}(r + m + h + 1))$. Applying this to the above estimation yields $\text{index}(E^r_m) \in \exp_{(h+2)}(\text{poly}(r + m))$.

In order to fulfil the prerequisites of Theorem 5.1 we can apply textbook methods to compute the decomposition of a graph of fixed tree-depth (see for instance [22]). From the decomposition the construction of an advice for the presentation in Lemma B.3 can be performed efficiently. ◀

Towards a General Direct Product Testing Theorem

Elazar Goldenberg
The Academic College of Tel Aviv-Yaffo, Israel
elazargo@mta.ac.il

Karthik C. S.[1]
Weizmann Institute of Science, Israel
karthik.srikanta@weizmann.ac.il

Abstract

The Direct Product encoding of a string $a \in \{0,1\}^n$ on an underlying domain $V \subseteq \binom{[n]}{k}$, is a function $\mathsf{DP}_V(a)$ which gets as input a set $S \in V$ and outputs a restricted to S. In the Direct Product Testing Problem, we are given a function $F : V \to \{0,1\}^k$, and our goal is to test whether F is close to a direct product encoding, i.e., whether there exists some $a \in \{0,1\}^n$ such that on most sets S, we have $F(S) = \mathsf{DP}_V(a)(S)$. A natural test is as follows: select a pair $(S, S') \in V$ according to some underlying distribution over $V \times V$, query F on this pair, and check for consistency on their intersection. Note that the above distribution may be viewed as a weighted graph over the vertex set V and is referred to as a test graph.

The testability of direct products was studied over various domains and test graphs: Dinur and Steurer (CCC '14) analyzed it when V equals the k-th slice of the Boolean hypercube and the test graph is a member of the Johnson graph family. Dinur and Kaufman (FOCS '17) analyzed it for the case where V is the set of faces of a Ramanujan complex, where in this case $V = O_k(n)$. In this paper, we study the testability of direct products in a general setting, addressing the question: what properties of the domain and the test graph allow one to prove a direct product testing theorem?

Towards this goal we introduce the notion of coordinate expansion of a test graph. Roughly speaking a test graph is a coordinate expander if it has global and local expansion, and has certain nice intersection properties on sampling. We show that whenever the test graph has coordinate expansion then it admits a direct product testing theorem. Additionally, for every k and n we provide a direct product domain $V \subseteq \binom{n}{k}$ of size n, called the Sliding Window domain for which we prove direct product testability.

2012 ACM Subject Classification Theory of computation → Probabilistic computation

Keywords and phrases Property Testing, Direct Product, PCP, Johnson graph, Ramanujan Complex, Derandomization

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.11

Acknowledgements We are truly grateful to Irit Dinur for her constant support throughout this project and for her many illuminating and helpful discussions and comments. We also thank the anonymous reviewers for their detailed and useful feedback.

[1] This work was supported by Irit Dinur's ERC-CoG grant 772839.

38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018).
Editors: Sumit Ganguly and Paritosh Pandya; Article No. 11; pp. 11:1–11:17

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

1 Introduction

The direct product encoding of a function is a way to aggregate multiple values of the input function using a single query. Justifying the vague intuition that it is much harder to compute multiple values of a function rather then a single value of it, the direct product encoding has been successfully used in several contexts of hardness amplification. The hardness can either measure the fraction of inputs on which every reasonable-time algorithm fails to compute the input function, or the fraction of unsatisfied assignments of a given CNF-formula or the communication complexity of the function.

In most of the PCP constructions an assignment to the given input is broken into many tiny pieces. Each small piece is encoded individually and then one should be able to test whether these tiny pieces could be stitched together into a global assignment. This testability task is referred to as an agreement test, and instantiations of it include low degree tests such as the plane vs. plane [12], the line vs. line test [1] and the cube vs. cube test [2], and the direct product test used in [8].

More concretely, we associate the direct product encoding of strings of size n, with some underlying domain[2] V which is a collection of subsets of $[n]$ of cardinality k. Given a string $a \in \{0,1\}^n$ its direct product encoding on the domain V, denoted by $\mathsf{DP}_V(a)$, is defined as follows: For every set $S \in V$ we define $\mathsf{DP}_V(a)(S) = a|_S$ (where $a|_S$ is the restriction of a to the coordinates in S). In this paper we study the testability of this encoding, namely: Given $F : V \to \{0,1\}^k$ we want to decide whether F agrees with some $\mathsf{DP}_V(a)$ on most sets S while querying F only on a few locations, specifically two. In other words, we focus on two-query tests in the paper where we pick a pair of subsets (both in the domain) according to some fixed distribution and then check if the two subsets agree on their intersection. We say that a domain V admits a direct product testing theorem if there exists a two-query test $\mathcal{T}$ satisfying the following: For every $\varepsilon \geq 0$ and $F : V \to \{0,1\}^k$ if $\mathcal{T}$ accepts F with probability $1-\varepsilon$, then we have $F(S) = \mathsf{DP}_V(a)(S)$ for some $a \in \{0,1\}^n$ on $1 - O(\varepsilon)$-fraction of the sets S in V, where the constant behind the O notation is independent of $|V|$ and k.

This question was studied under various domains. Dinur and Steurer [9] analyzed a two-query test under the domain $V = \binom{[n]}{k}$. Recently, Dinur and Kaufman [6] studied this question in a much shrunken domain, which is obtained by considering the set of the faces of a high dimensional expander. However, both of these proofs are tailored to the structure of their own domain and cannot be (trivially) generalized to other domains. It is natural to ask whether a more generalized argument can be applied covering both of these domains, and on which domains it may be applied. The main question we are investigating is as follows:

Which domains admit a two-query direct product testing theorem?

Let us elaborate more about the previous proofs. The proofs given by [9] and [6] first analyze the testability in the high error regime, i.e. when the acceptance probability is slightly bounded away from 0. They show that any function that passes the test with non-negligible probability ε must agree with some legal codeword $\mathsf{DP}_V(a)$ on $\Omega(\varepsilon)$ fraction of sets. Then they analyze the test in the low error regime, i.e. when the acceptance probability of the test is close to 1. Finally they stitch local tiny agreements into a single codeword and show that the agreement is almost everywhere.

[2] For the ease of presentation, we only consider domains which are a subset of $\binom{[n]}{k}$ in this section. However, in the rest of the paper we consider V which is a collection of subsets of $[n]$, and all our results are proved for this more general case.

We would like to establish a direct product testing theorem using a more straightforward approach: we decode a string from the input function F using the majority operator and then show that if the test passes with high probability then F is close to the direct product encoding of the decoded string. More precisely, given the input function F, we define a string $a \in \{0,1\}^n$ as follows: for every coordinate $i \in [n]$ we set a_i to be the majority value of $F(S)_i$, where the majority is taken over the sets that contain i. Next we show that if F passes the test with probability $1-\varepsilon$ then F must be $1-O(\varepsilon)$-close to $\mathsf{DP}_V(a)$. We remark that Dinur and Reingold [8] indeed followed this proof strategy, however, their proof admits only a relaxed notion of closeness between the input function and the direct product encoding of the decoded string (namely, that on most sets S, $F(S)$ and $\mathsf{DP}_V(a)(S)$ agree *only* on most of the coordinates in S).

Observe that any two-query test on a domain V gives rise to a weighted graph whose vertex set is V and the weight we assign for each pair (S, S') is the probability of this pair being picked by the test[3]. We refer to this graph as the test graph. We say that a test graph yields a tester for the domain V, if for every $\varepsilon \geq 0$ and every function $F : V \to \{0,1\}^k$ the following holds: if the test accepts F with probability $1-\varepsilon$, then F must be $1-O(\varepsilon)$-close to some $\mathsf{DP}_V(a)$. Here the test corresponds to picking an edge (S, S') at random (according to the distribution of weights on the edges) and accepting if and only if $F(S)|_{S\cap S'} = F(S')|_{S\cap S'}$.

Another proof insight that we desire is the explicit use of the properties of the underlying test graph. For example, one property that the test graph must satisfy to be a tester is that for most edges (S, S') the intersection between S and S' is linear in k. Assume not, then we consider the following construction of F: We start from $F = \mathsf{DP}_V(a)$ for some $a \in \{0,1\}^n$ and then for each $S \in V$ we reset the value of $F(S)_i$ for some random $i \in S$. Then for most sets (S, S') with small intersection the test accepts but F is far from any direct product codeword. Another property that the test graph must have is some notion of expansion. Summing up, our more refined question is as follows:

> *What properties of the test graph yields a tester for its underlying domain?*

1.1 Our Results

Our conceptual contributions in this paper are two-fold. First, we introduce a notion called *coordinate expansion* which captures the properties of direct product testable domains. Second, we introduce the sliding window domain which is of size exactly *equal* to the universe and is direct product testable. Our main technical contribution is showing that domains having coordinate expansion with certain parameters admit a direct product theorem.

1.1.1 A General Direct Product Theorem

We introduce below the notion of coordinate expansion. Informally, a coordinate expander has both global and local expansion properties, and has good intersection properties.

▶ **Definition 1** ((λ,ρ)-Coordinate Expander)**.** Let $G = (V, E)$ be a test graph, where $V \subseteq \binom{[n]}{k}$. For $i \in [n]$ let $V_i = \{S \in V | i \in S\}$ and G_i be the subgraph of G induced by the vertices in V_i. The graph G is called (λ,ρ)-coordinate expander if:

[3] In this paper we analyze test graphs which are undirected.

1. $\lambda(G) < \lambda$ (where $\lambda(G) = \max\{|\lambda_2(A_G)|, |\lambda_{|V|}(A_G)|\}$ and A_G is the normalized adjacency matrix of G).
2. For every $i \in [n]$ we have that $\lambda(G_i) < \lambda$ and for each $S \in V_i$ the probability that a uniformly random neighbor S' of S is in V_i is at least ρ.
3. For every subset S and $T \subseteq S$, satisfying $|T| \geq 2/\rho$, the probability that for a uniformly random neighbor S' of S we have $|S' \cap T| \leq \rho |T| /2$ is upper bounded bounded by λ .

Notice that condition 1 implies that the test graph must be a good expander (in the traditional sense). Moreover, condition 2, implies that on certain local subsets (i.e., subsets containing a common coordinate) of vertices, the induced subgraph must be expanding as well. Finally, condition 3 implies that the neighbors of every subset S samples well every subset T of S.

Observe that condition 2 is necessary for the test graph to be a direct product tester. To see this, consider a test graph that does not satisfy this property, namely, there exists a coordinate $i \in [n]$ for which: there exits a set $B_i \subset V_i$ such that $\Pr_{S' \in V_i}[S' \notin B_i | S \in B_i] = o(1)$. Then, we show that the test graph does not yield a tester. Indeed, consider the following construction of F: we first choose $F = \mathsf{DP}_V(a)$ for some $a \in \{0,1\}^n$. Then for every $S \in B_i$ we change the value of $F(S)_i$ to $1 - a_i$. Clearly, the distance of F from a direct product encoding equals $\delta := |B_i| / |V|$. However, the rejection probability equals:

$$2 \cdot \Pr_{S' \sim S}[S \in B_i \text{ and } S' \in V_i \setminus B_i] \leq 2 \cdot \Pr[S \in B_i] \cdot \Pr[S' \in V_i] \cdot \Pr_{S' \in V_i}[S' \notin B_i | S \in B_i] = o(1) \cdot \delta.$$

Then, we show our main technical result that coordinate expansion implies direct product testing (for a certain range of parameters).

▶ **Theorem 2.** *Let $\rho \geq 1/2$ and $\lambda \leq 1/33$. Let $G = (V, E)$ be a test graph, $V \subseteq \binom{[n]}{k}$, let $\varepsilon \geq 0$, and $F : V \to \{0,1\}^k$. Let G be a (λ, ρ)-coordinate expander. If F passes the test implied by the test graph G with probability $1 - \varepsilon$ then F is $1 - O(\varepsilon)$-close to $\mathsf{DP}_V(a)$ for some $a \in \{0,1\}^n$.*

The overview of the above proof is given in Section 1.2. Also, as an application of the above theorem, we show[4] a direct product theorem for the test graph isomorphic to the Johnson graph $J(n,k)$ when k is close to $n/2$, where $J(n,k)$ is a graph whose vertex set is the set of all subsets of $[n]$ of cardinality k, and two subsets have an edge if their intersection is equal to $k/2$. This should be compared to [9], where they show the direct product for the Johnson graph for all the layers up to $n/2$ (i.e., for all $J(n,k)$ where $k \leq n/2$).

The main open problem stemming from our work is to improve the parameters in Theorem 2. In particular, does the following hold?

▶ **Open Problem 3.** *Does $(1/2, 1/2)$-coordinate expansion imply a direct product theorem?*

A positive resolution of the above open question would imply direct product testability on the test graph isomorphic to the Johnson graph for every layer of the Boolean hypercube (completely recovering the results in [9]). It even implies a direct product testability on a new domain: where the subsets are stemming from d-dimensional subspaces of $\mathbb{F}_2^m$ and two subsets are connected by an edge if they intersect on a $(d-1)$-dimensional subspace (this is referred to as the Grassmann graph). Finally, we would like to recall that Theorem 2

[4] The claim as written here is slightly inaccurate. Please refer to Appendix B for a precise statement.

states that $(1/33, 1/2)$-coordinate expansion implies a direct product theorem, i.e., in order to positively resolve Open Problem 3, we might need to improve the analysis in the proof of Theorem 2 to accommodate test graphs with weaker expansion properties.

In fact, if we can resolve Open Problem 3 in a slightly stronger way i.e., if we show that for some small enough constant $\gamma > 0$, we have $(1/2 + \gamma, 1/2)$-coordinate expansion implies a direct product theorem then we recover the testability result of [6] on Ramanujan complexes. Summarizing, we view the study of coordinate expansion as providing a unified framework to prove direct product theorems. Also, it might be useful in the future to establish direct product testability for new domains (in a black-box manner).

1.1.2 Sliding Window Domain

In this subsubsection, we define a new direct product testable domain which we call the sliding window domain, and also discuss about the necessary and sufficient structure that a domain (and test graph) should have, in order to admit direct product testing.

For every n, k, the sliding window domain $\mathcal{A} \subseteq \binom{[n]}{k}$ is the collection of all contiguous k-sized subsets (windows) of $[n]$, i.e., $\mathcal{A} = \{\{i, \ldots, i+k-1\} \mid i \in [n]\}$, where the addition is done modulo n. Two vertices (i.e., subsets in $\mathcal{A}$) have an edge in the test graph, if their intersection is non-empty. Notice that $|\mathcal{A}| = n$ and yet we show that it admits a direct product theorem (see Theorem 9 for a simple proof).

Let us put the above result in context with the recent breakthrough of Dinur and Kaufman [6]. In [6], the authors obtain a direct product testable domain (subset of $\binom{[n]}{k}$) of size $O(2^{k^2} n)$. The domain arises from the highly non-trivial object called Ramanujan complex. Such a domain is studied because apart from admitting a direct product theorem over a domain of size linear in the universe (i.e., n), it also has other desirable properties such as *distance amplification* which are needed for applications in gap and hardness amplification. Thus, our direct product testing result (Theorem 9) provides a conceptual clarification that if one is only interested in direct product testing as a property testing question, then there is a very simple domain of size n, namely the sliding window domain, which is testable.

Roughly speaking, a domain (subset of $\binom{[n]}{k}$) has distance amplification if for every two strings of relative distance δ, the relative distance between their direct product encoding is $\Omega(k\delta)$. This seems to be a crucial property for PCP applications of direct product testing. Thus, the construction of the sliding window domain provides a conceptual clarification as to why we need high dimensional expanders: we can obtain direct product testing from simple constructions like the sliding window domain and we can obtain distance amplification from known constructions of vertex expanders (see Appendix D); but to obtain both simultaneously, [6] needed high dimensional expanders. We leave it as an open question whether there exists a simple construction admitting both direct product testability and distance amplification.

▶ **Open Problem 4.** *Is there a (relatively) simple domain of linear size in the universe (i.e., n) for which we have both direct product testing and distance amplification?*

Lack of Global Expansion. We would like to now briefly discuss about the minimal structure of the domain (and the test graph) sufficient to prove a direct product theorem. This is highlighted by the sliding window domain, an in particular by the proof of its testability (Lemma 10 to be precise). Notice that $G_{\mathcal{A}}$ has very bad edge-expansion/vertex-expansion but is a very good local expander, i.e., the induced subgraph containing any particular coordinate

has good expansion (in fact is a clique). Lemma 10 guarantees that in such situations[5] the domain admits direct product testing if for every vertex in the test graph, and every element in that vertex, the probability of retaining that element when moving to a uniformly random neighbor is bounded from below by a positive constant. The probability of retaining a coordinate when moving to a random neighbor is 1/2 in $\mathcal{A}$, and thus $\mathcal{A}$ admits a direct product theorem. Therefore, $\mathcal{A}$ demonstrates that direct product testing does not require the test graph to be an expander (like the Johnson/Ramanujan graph) but only needs to have certain local expansion properties. Finally, recall that we had earlier argued that local expansion is necessary (to justify the need for condition 2 in Definition 1) for direct product testing.

Finally, it seems that conditions 1 and 3 in coordinate expansion are not (necessarily) needed for direct product testing, but are merely artifacts of our proof (Theorem 2). However, these conditions might imply distance amplification[6] and are typically guaranteed in structured domains of interest (namely, Johnson, Grassmannian, and Ramanujan).

1.2 Technical Contribution: Proof Overview of Theorem 2

For the sake of convenience, through out this subsection, we fix $V = \binom{[n]}{k}$ and the test would pick pairs (S, S') that intersect on $k/2$ elements and checks for agreement. As suggested above there is a natural way to decode any function $F : V \to \{0,1\}^k$ using the majority operator: define a string $a \in \{0,1\}^n$ by setting a_i to be the majority value of $F(S)_i$ for all $S \ni i$. We define $B = \{S | F(S) \neq \mathsf{DP}_V(a)(S)\}$, i.e., B is the subset of the domain that disagrees with the direct product encoding of the decoded string. Also for $S \in B$ we call $i \in S$ conflicting if $F(S)_i \neq a_i$. Our goal is to show that the test rejects with probability $\Omega(|B| / |V|)$ as $|B|/|V|$ is the relative distance between F and $\mathsf{DP}_V(a)$.

Indeed fix $S \in B$, then it must contain at least one conflicting coordinate, say i. Observe that with probability 1/2 we also have that $i \in S'$. Now if S' were a random set containing i, then since at least half of the elements that contain i agree with the majority value, the test rejects with probability 1/2. And the overall rejection probability of the test would be at least $\frac{|B|}{4|V|}$ and we are done.

However, S' is not a random set that contains i, it intersects with S on further $k/2 - 1$ coordinates. Therefore, it may well be that among the neighbors of S that contain i we do not see the majority value so often. A natural way to overcome this is by aggregating all Ss' that contain i and disagree with the majority value on i. We could try to show that if we start from some member of this set then with constant probability we reach S' that contains i and resides outside of this aggregated set (by using the local expansion property). But this leads into another problem: using this argument sets S that contain many conflicting coordinates are counted many times, whereas sets that contain few conflicting coordinates are counted much less.

Our analysis proceeds by studying the variance of the number of conflicting coordinates in the following manner. We first sort the set B based on the number of their conflicting coordinates. Let B_L (resp. B_H) be the first (resp. last) third of the elements in B according the sorting. We first show that if the number of conflicting coordinates of each member in B_L is much smaller than it is in B_H, then the test rejects with probability $\Omega(\frac{|B|}{|V|})$. To show this, we prove that whenever the test picks $S \in B_H$ then with constant probability S' is in

[5] Lemma 10 can be generalized to accommodate test graphs which are locally subgraphs that strongly satisfy the expander mixing lemma.

[6] This would be an interesting question to resolve in either direction.

$B_L \cup \{V \setminus B\}$ (by using the global expansion property). Moreover, there is a large subset Γ of conflicting coordinates in S which are also in S' (follows from condition 3 in Definition 1). However, S' has few conflicting coordinates in total (by our choice of S'), and thus, there must be a coordinate in Γ that agrees with the majority value on S' but disagrees on it on S and hence the test rejects the edge (S, S').

On the other hand, if the number of conflicting coordinates does not vary a lot among these sets, then we analyze the test by selecting (at random) a single conflicting coordinate in S and focusing on the rejection probability based only on the value of the selected coordinate.

1.3 Related Work

The question of testing the direct product was studied extensively when the underlying domain $V = \binom{[n]}{k}$ [10, 8, 5, 9, 11]. In this setting, Goldreich and Safra [10] proposed a constant query test. Dinur and Reingold [8] suggested the two-query test mentioned above and analyzed it in the high acceptance regime but with a relaxed distance measure.

The state of the art in this context is the result of Dinur and Steurer[7] [9] dealing with the domain $V = \binom{[n]}{k}$ where k varies between 2 and $n/2$. They analyze the aforementioned two-query test with $k/2$-intersection size. They analyze it in the high acceptance regime and show that $\binom{[n]}{k}$ indeed admits a direct product testing theorem. The proof is quite involved and in particular analyzes first the low acceptance regime. Recently, in a breakthrough paper, Dinur and Kaufman [6] analyzed the two-query test when the underlying domain is obtained from the set of faces of a Ramanujan complex. Their approach crucially relies on the result of [9].

We remark that the direct product testability question was further analyzed in the low acceptance regime under the domain $\binom{[n]}{k}$, see [5, 11, 7] and also under the domain where the universe is $\mathbb{F}_2^m$, and the domain is the set of all subspaces of $\mathbb{F}_2^m$ [11].

1.4 Organization of the Paper

Section 2 lists the notations and technical tools that we use in the paper. In Section 3 we formalize the notion of direct products and their testing. In Section 4 we prove our main technical result, namely, that whenever the underlying test graph is a (λ, ρ)-coordinate expander it admits a direct product testing theorem. Finally, in Section 5 we introduce the sliding window domain for which we show a direct product theorem.

2 Preliminaries

In this section, we list the notations and technical tools used in this paper.

Notations. We use the following notations throughout the paper. We denote the set $\{1, \ldots, n\}$ by $[n]$. For any $n, k \in \mathbb{N}$, with $k \leq n$, we denote by $\binom{[n]}{k}$, all subsets of $[n]$ of cardinality k. For any set S, we denote by $\mathcal{P}(S)$ the power set of S, i.e., the set of all subsets of S. For any graph $G(V, E)$ and any two subsets $S, T \subseteq V$, we denote by $E(S, T)$ the set of all edges between S and T. For any $x, y \in \{0, 1\}^n$, we denote by $\Delta(x, y)$ the relative Hamming distance between x and y given by the fraction of coordinates in which x and y differ.

[7] The result in [9] is stated in the language of tuples, i.e., the domain is a subset of $[n]^k$, but their result also holds when the domain is a collection of k-sized subsets of $[n]$. See [4] for more details.

Johnson Graph Family. For every $n, k, t \in \mathbb{N}$ such that $t \leq k \leq n$, $J(n,k,t)$ is a graph which is a member of the Johnson graph family, whose vertex set is $\binom{[n]}{k}$, and whose edge set is $\{(S,S') \mid S,S' \in \binom{[n]}{k}, |S \cap S'| = t\}$.

Expander Mixing Lemma. The following is a standard claim concerning the expansion of two sets in expander graphs. For completeness we include a proof in Section A:

▶ **Claim 5.** *Let $G = (V,E)$ be a d-regular graph and A be its adjacency matrix. Let λ be its second largest eigenvalue in absolute value. Let $S, T \subseteq V$ satisfying: $|S| \leq |V|/2$ then:*

$$\Pr_{(u,v)}[v \in T | u \in S] \leq \frac{|T|}{|V|} + \frac{\lambda}{d}\sqrt{\frac{|T|}{|S|}},$$

where the probability is given by first picking u uniformly at random from S, and then picking v according to A. Furthermore, let μ be a distribution on S satisfying that for every two elements $b, b' \in S$: $\mu(b) \leq c\mu(b')$, then:

$$\Pr_{(u,v)}[v \in T \mid u \sim \mu] \leq \frac{|T|}{|V|} + \frac{\lambda}{d} \cdot \sqrt{\frac{c\,|T|}{|S|}}$$

3 Direct Product Testing: The Setting

In this section, we formalize the notion of direct products and their testing. Specifically, we formalize the notion of direct product testing through test graphs, which is slightly non-standard but it helps in introducing the notion of coordinate expansion in a later section succinctly.

For every subset S of $[n]$, let $\mathcal{F}_S$ be the class of all functions whose domain is S and range is $\{0,1\}$. Let $V \subseteq \mathcal{P}([n])$ be the domain of the direct product. Let $\mathcal{F}_V$ be the class of all functions whose domain is V and maps every subset S in V to a function in $\mathcal{F}_S$. The direct product encoding is a function $\mathsf{DP}_V : \{0,1\}^n \to \mathcal{F}_V$ defined as follows: for every string $a \in \{0,1\}^n$, and every subset $S \in V$, let $\mathsf{DP}_V(a)_S$ be defined as the projection function which maps S to a_S, the string a restricted to only the coordinates in S.

▶ **Definition 6.** For two functions $F, G \in \mathcal{F}_V$ we define their relative distance as:

$$\Delta(F,G) = \frac{|\{S \in V | F(S) \neq G(S)\}|}{|V|}.$$

For a function F and a set of functions $\tilde{G}$ we define the distance between F and $\tilde{G}$ as the minimal distance between F and some function $G \in \tilde{G}$. If $\Delta(F,\tilde{G}) \leq \delta$, we say that F is $1-\delta$-close to $\tilde{G}$, otherwise, it is δ-far from $\tilde{G}$.

For every function $F \in \mathcal{F}_V$, we define $\mathsf{dec}(F)$ as follows: Given F construct $a^F \in \{0,1\}^n$ in the following way,

$$a_i^F = \operatorname*{\mathsf{maj}}_{\substack{S \in V \\ S \ni i}}(F(S)_i).$$

Then, we define $\mathsf{dec}(F) := \mathsf{DP}_V(a^F)$.

Let G_V be a graph whose vertex set is V. Then we interpret G_V as a test graph on functions defined on $\mathcal{F}_V$ in the following sense:

Test $\mathcal{T}(G_V)$:
Input: A function $F \in \mathcal{F}_V$.
Procedure: Pick an edge (S, S') in G_V uniformly at random.
Output: Accept if and only if $F(S)|_{S\cap S'} = F(S')|_{S\cap S'}$.

It is important to note that we allow self loops and multiple edges between a pair of vertices. Also, we can generalize the above direct product testing setting to the case when V is a multiset of $\mathcal{P}([n])$, and the results in this paper still hold. However, we choose not to handle this more general setting for the sake of clarity of presentation. The above remark also applies to the case of studying test graphs which are not regular in degree, that are not considered in this paper. Finally, throughout the paper, we drop the subscript V in G_V, if V is clear from the context.

4 Direct Product Testing: Coordinate Expansion

In this section we prove our main technical result, namely, that whenever the underlying test graph is a (λ, ρ)-Coordinate Expander (defined next) it admits a direct product testing theorem.

▶ **Definition 7** ((λ, ρ)-Coordinate Expander)**.** Let $n \in \mathbb{N}$ and let $G = (V, E)$ be a test graph, where $V \subseteq \mathcal{P}([n])$. For $i \in [n]$ let $V_i = \{S \in V | i \in S\}$ and G_i be the subgraph of G induced by the vertices in V_i. Let $\lambda(G) = \max\{|\lambda_2(A_G)|, |\lambda_{|V|}(A_G)|\}$, where A_G is the normalized adjacency matrix of G. The graph G is called (λ, ρ)-coordinate expander if:

1. $\lambda(G) < \lambda$ and for every $i \in [n]$ we have $\lambda(G_i) < \lambda$.
2. For every $i \in [n]$ and for each $S \in V_i$ we have $\Pr_{S'\sim S}[S' \in V_i] \geq \rho$.
3. For every subset S and $T \subseteq S$, satisfying $|T| \geq 2/\rho$, we have $\Pr_{S'\sim S}[|S' \cap T| \leq \rho|T|/2] \leq \lambda$.

Informally, a domain is a coordinate expander if the test graph is an expander and every induced subgraph of the test graph containing a fixed coordinate is also an expander[8], and it has good correlation/intersection properties – i.e., for any subset S and coordinate $i \in S$, an uniformly random neighbor of S contains i with constant probability (say $\rho > 0$), and for every S in the domain, and any subset T of S, the number of elements of T that we see in a random neighbor of S is close to the expected number, which is $\rho \cdot |T|$. Below, we see that coordinate expansion of the test graph implies a direct product theorem for the underlying domain.

▶ **Theorem 8.** *Let $n \in \mathbb{N}$, and let $\rho \geq 1/2$ and $\lambda \leq 1/33$ be some constants. Let $G = (V, E)$ be a graph, $V \subseteq \mathcal{P}([n])$, let $\varepsilon \geq 0$, and $F \in \mathcal{F}^V$. Let G be a (λ, ρ)-coordinate expander. If F passes $\mathcal{T}(G)$ with probability $1 - \varepsilon$ then F is $1 - O(\varepsilon)$-close to $\mathsf{dec}(F)$.*

Proof. Let $F^* := \mathsf{dec}(F) = \mathsf{DP}_V(a^F)$. We define $B, C \subseteq V$ as follows:

$$B = \{S \mid F(S) \neq F^*(S)\} \text{ and } C = V \setminus B.$$

Let $\beta = |B| / |V|$. Given a subset $S \in V$ we say that a coordinate i is conflicting if the value of $F(S)$ at i does not equal a_i^F. For a set S denote by $B(S)$ the set of conflicting coordinates in S. We show that $\mathcal{T}(G)$ rejects with probability at least $\Omega(\beta)$.

[8] Actually, the property of an expander that we need is that for any two sets of vertices S, T in the graph, the number of edges between S and T is roughly equal to $\alpha|S||T|$, where α is the density of the edge set of the graph.

Let us sort in ascending order the elements of B based on the number of coordinates on which they disagree with F^*. For a parameter $0 \leq p \leq 1$ we define the set $B_{\geq p}$ as the set of last $(1-p)\,|B|$ elements of B (and similarly the set $B_{\leq p}$ is the set of the first $p\,|B|$ elements of B). We denote by m_p the number of conflicting coordinates of the $p\,|B|$-th element of B.

Let $c = 3/40$. We consider two cases based on $m_c, m_{1/2}$ and m_{1-c}.

Case 1: $m_{1-c} > \frac{2}{\rho} m_{1/2}$ or $m_{1/2} > \frac{2}{\rho} m_c$

For both the possibilities we have similar arguments, which is why they are clubbed under one case, but will be handled separately for ease of presentation.

Case 1A: $m_{1-c} > \frac{2}{\rho} m_{1/2}$

The probability that an uniformly random $S \in V$ is in $B_{\geq 1-c}$ is $c\beta$. Now by Claim 5, we get that

$$\Pr[S' \in B_{>1/2} | S \in B_{\geq 1-c}] < \beta/2 + \lambda\sqrt{\frac{1}{2c}},$$

so with probability at least $1 - \beta/2 - \lambda\sqrt{\frac{1}{2c}}$ if $S \in B_{\geq 1-c}$ then $S' \in B_{\leq 1/2} \cup C$.

Now, by the third property of (λ, ρ)-coordinate expander, the probability that $|S' \cap B(S)| \leq \frac{\rho}{2}|B(S)|$ is at most λ. Notice that the probability that $|S' \cap B(S)| \leq m_{1/2}$ is at least the probability that $|S' \cap B(S)| \leq \frac{\rho}{2}|B(S)|$ (because $m_{1/2} < \frac{\rho}{2} m_{1-c} \leq \frac{\rho}{2}|B(S)|$). Hence we have that the probability that $|S' \cap B(S)| \leq m_{1/2}$ is at most λ.

Overall, using union bound, conditioned on $S \in B_{\geq 1-c}$, the probability that $S' \in B_{\leq 1/2} \cup C$ and $|S' \cap B(S)| > m_{1/2}$ is at least $1 - \beta/2 - \lambda\sqrt{\frac{1}{2c}} - \lambda$. But in such a case since $S' \in B_{\leq 1/2} \cup C$ we get $|B(S')| \leq m_{1/2}$, so there exists at least one coordinate $i \in S \cap S'$ on which $F(S')_i = a_i^F$ but $F(S)_i \neq a_i^F$, so the test rejects. In total $\mathcal{T}$ rejects with probability at least $c\beta\left(1 - \beta/2 - \lambda\sqrt{\frac{1}{2c}} - \lambda\right) \geq c\beta\left(1/2 - \lambda\left(\sqrt{\frac{1}{2c}} + 1\right)\right)$ (where we used a trivial bound that $\beta \leq 1$). Notice that $1/2 - \lambda\left(\sqrt{\frac{1}{2c}} + 1\right) > 0$ holds for $c = 3/40$ whenever $\lambda \leq 0.13$.

Case 1B: $m_{1/2} \geq \frac{2}{\rho} m_c$

In this case we would like to mimic the proof strategy of the previous case. That is we would like to show that with non-zero constant probability a random neighbor in $B_{\geq 1/2}$ is in $B_{\leq c} \cup C$. By an application of Claim 5, we get:

$$\Pr[S' \in B_{>c} | S \in B_{\geq 1/2}] < (1-c)\beta + \lambda\sqrt{2-2c},$$

so with probability at least $1 - (1-c)\beta - \lambda\sqrt{2-2c}$ if $S \in B_{\geq 1/2}$ then $S' \in B_{\leq c} \cup C$.

Now, by the third property of (λ, ρ)-coordinate expander, the $\Pr[|S' \cap B(S)| \leq \frac{\rho}{2}|B(S)|]$ is at most λ. Notice that $m_c \leq \frac{\rho}{2} m_{1/2} \leq \frac{\rho}{2} \cdot |B(S)|$ and thus $\Pr[|S' \cap B(S)| \leq \frac{\rho}{2}|B(S)|] \geq \Pr[|S' \cap B(S)| \leq m_c]$. Therefore we have $\Pr[|S' \cap B(S)| \leq m_c] \leq \lambda$.

Overall, using union bound, conditioned on $S \in B_{\geq 1/2}$, the probability that $S' \in B_{\leq c} \cup C$ and $|S' \cap B(S)| > m_c$ is at least $1 - (1-c)\beta - \lambda\sqrt{2-2c} - \lambda$. But in such a case since $S' \in B_{\leq c} \cup C$ we get $|B(S')| \leq m_c$, so there exists at least one coordinate $i \in S \cap S'$ on which $F(S')_i = a_i^F$ but $F(S)_i \neq a_i^F$, so the test rejects. In total $\mathcal{T}$ rejects with probability at least $\frac{\beta}{2}\left(1 - (1-c)\beta - \lambda\sqrt{2-2c} - \lambda\right) \geq \frac{\beta}{2}\left(c - \lambda\left(\sqrt{2-2c} + 1\right)\right)$ (where we used a trivial bound that $\beta \leq 1$). Notice that $\left(c - \lambda\left(\sqrt{2-2c} + 1\right)\right) > 0$ holds for $c = 3/40$ whenever $\lambda \leq 0.03177$.

Case 2: $m_{1-c} \leq \frac{4}{\rho^2} m_c$

Define $B_{(c,1-c)}$ as the set $B \setminus (B_{\leq c} \cup B_{\geq 1-c})$. Observe that in $B_{(c,1-c)}$ the number of conflicting coordinates is between m_c and $4m_c/\rho^2$. Now we would like to consider a different test $\mathcal{T}'(G)$ that selects S, S' according to G. If $S \notin B_{(c,1-c)}$ then $\mathcal{T}'$ accepts. Otherwise, it picks uniformly at random $i_0 \in B(S)$ and checks for consistency *only* on i_0, namely: It rejects iff $i_0 \in S'$ and $F(S)_{i_0} \neq F(S')_{i_0}$. Clearly the rejection probability of $\mathcal{T}'(G)$ is at most the rejection probability of $\mathcal{T}(G)$. We conclude the proof by showing that $\mathcal{T}'(G)$ rejects F with probability $\Omega(\beta)$.

With probability $(1-2c)\beta$ the test $\mathcal{T}'$ selects $S \in B_{(c,1-c)}$ and we would like to analyze the rejection probability conditioned on that. For this sake we bound the probability of the following events:

- E_1 is the event where $S' \in B_{\leq c} \cup B_{\geq 1-c}$.
- E_2 is the event where $i_0 \in S'$ and $S' \notin \tilde{B}_{i_0}$ where $\tilde{B}_i = \{S \in B_{(c,1-c)} | F(S)_i \neq a_i^F\}$.

If the event E_2 occurs but E_1 does not, then it must be the case that $F(S')_{i_0} = a_{i_0}^F$. Hence $\mathcal{T}'$ rejects. As a consequence $\Pr[\mathcal{T}' \text{ rejects}] \geq (1-2c)\beta(\Pr[E_2 | S \in B_{(c,1-c)}] - Pr[E_1 | S \in B_{(c,1-c)}])$. Thus it suffices to show that $(\Pr[E_2 | S \in B_{(c,1-c)}] - Pr[E_1 | S \in B_{(c,1-c)}])$ is a positive constant bounded away from 0.

To bound the probability for the event E_1 we use Claim 5: The probability of E_1 conditioned on $S \in B_{(c,1-c)}$ is at most $2c\beta + \lambda\sqrt{\frac{2c}{1-2c}}$.

Since the graph G is a (λ, ρ)-coordinate expander then for each $i \in S$, we have that $\Pr[i \in S'] \geq \rho$, in particular this is true for i_0, hence: $\Pr[i_0 \in S'] \geq \rho$.

Now we divide the event E_2 into disjoint events depending on the value of i_0 and bound the rejection probability of $\mathcal{T}'$ conditioned on specific value of i_0. Fix $i \in [n]$ and assume that $\mathcal{T}'$ selects $S, S' \in V_i$ and sets $i_0 = i$ (so $S \in \tilde{B}_i$). We denote by β_i the fraction $\frac{|\tilde{B}_i|}{|V_i|}$. Observe that $\beta_i \leq 1/2$, since otherwise the majority value would become the value of $F(S)_i$, but we have $S \in \tilde{B}_i$.

Note, that under the assumption that $\mathcal{T}'$ selects $i_0 = i$ and $S \in \tilde{B}_i$, sets S with few conflicting coordinates are more likely to be chosen than those who have many of them. However, since by our assumption the number of conflicting coordinates is between m^* and $\frac{4}{\rho^2}m^*$, then sets with m^* conflicting coordinates are only $4/\rho^2$-times more probable than those having $\frac{4}{\rho^2}m^*$-conflicting coordinates. Denote by μ the distribution of picking $S \in \tilde{B}_i$ assuming that $\mathcal{T}'$ selects $i_0 = i$. By an application of Claim 5 we get:

$$\Pr_{S \sim \mu, S'}[S' \in \tilde{B}_i] \leq \frac{|\tilde{B}_i|}{|V_i|} + \lambda\sqrt{\frac{4}{\rho^2}} \leq \frac{1}{2} + 2\lambda/\rho$$

So we get that,

$$\Pr[E_2 | S \in B_{(c,1-c)}] = \left(1 - \Pr_{S \sim \mu, S'}[S' \in \tilde{B}_i \mid i_0 \in S']\right) \cdot \Pr[i_0 \in S'] \geq \frac{\rho}{2} - 2\lambda.$$

Summing up, we get:

$$\begin{aligned}
\Pr[\mathcal{T}' \text{ rejects} | S \in B_{(c,1-c)}] &\geq \Pr[E_2 | S \in B_{(c,1-c)}] - \Pr[E_1 | S \in B_{(c,1-c)}] \\
&\geq \frac{\rho}{2} - 2\lambda - \left(2c + \lambda\sqrt{\frac{2c}{1-2c}}\right) \\
&\geq \frac{1}{4} - 2c - \lambda\left(2 + \sqrt{\frac{2c}{1-2c}}\right),
\end{aligned}$$

a constant bounded away from 0 for $c = 3/40$ whenever $\lambda < 0.04$. ◀

In Appendix B, we consider the test graph $J(n, k, k/2)$ and show a direct product theorem when k is close to $n/2$.

5 Sliding Window Domain

In this section, we introduce the sliding window domain for which we show a direct product theorem.

Construction. Let $k, n \in \mathbb{N}$ such that $k \leq n$. Let $\mathcal{A}$ be a collection of n subsets of $[n]$ of Hamming weight k.

$$\mathcal{A} = \{\{i, \ldots, i+k-1\} \mid i \in [n]\},$$

where the addition is done[9] modulo n.

Testability. The domain of our direct product test is $\mathcal{A}$. The corresponding test is as follows:

Test $\mathcal{T}$:
Input: A function $F : \mathcal{A} \to \{0,1\}^k$.
Procedure: Pick uniformly at random $S \in \mathcal{A}$. Then pick uniformly at random $S' \in \mathcal{A}$ such that $S \cap S' \neq \emptyset$.
Output: Accept if and only if $F(S)|_{S\cap S'} = F(S')|_{S\cap S'}$.

The test graph $G_{\mathcal{A}}$ of the above is given by the vertex set $\mathcal{A}$ and the edge set $\{(S, S') \mid S \cap S' \neq \emptyset\}$. The correctness of the above test is shown below. We would like to emphasize that $|\mathcal{A}| = n$ and yet admits a direct product theorem.

▶ **Theorem 9.** *Let $\varepsilon \geq 0$ and $F \in \mathcal{F}_{\mathcal{A}}$. If F passes $\mathcal{T}(G_{\mathcal{A}})$ with probability $1-\varepsilon$ then F is $(1-4\varepsilon)$-close to $\mathsf{dec}(F)$.*

Proof. We will in fact prove a more general direct product testing result.

▶ **Lemma 10.** *Let $n \in \mathbb{N}$ and $G = (V, E)$ be a d-regular graph where $V \subseteq \mathcal{P}([n])$, let $\varepsilon \geq 0$, and $F \in \mathcal{F}_V$. For every $i \in [n]$, let the induced subgraph of V_i in G be a clique (with self loops). Additionally, let $c > 0$ be a constant such that for every $S \in V$ and every $i \in S$, the probability that a uniformly random neighbor S' of S in G contains i is at least c. If F passes $\mathcal{T}(G)$ with probability $1-\varepsilon$ then F is $(1-\frac{2\varepsilon}{c})$-close to $\mathsf{dec}(F)$.*

Now we show that the above lemma gives the proof of the theorem. Let $\mathcal{A}_i = \{S \in \mathcal{A} \mid i \in S\}$. Note that for every $i \in [n]$, the induced subgraph of $\mathcal{A}_i$ in G is a clique (with self loops) because any two subsets in $\mathcal{A}_i$ have i in their intersection and thus have non-empty intersection. Also for every $S \in \mathcal{A}$ and every $i \in S$, the probability that a uniformly random neighbor S' of S in G contains i is at least $1/2$. Thus, from Lemma 10 the theorem follows. ◀

We complete the proof of the above theorem by showing Lemma 10 below.

[9] Strictly speaking, the addition is done modulo n and then the resulting number is incremented by one.

Proof of Lemma 10. Let $F^* := \mathsf{dec}(F) = \mathsf{DP}_V(a^F)$. Let $B \subseteq V$ be defined as follows:

$$B = \{S \mid F(S) \neq F^*(S)\}.$$

Let $C_i \subseteq V_i$ be defined as follows:

$$C_i = \{S \in V_i \mid F(S)_i = a_i^F\}.$$

By definition of a_i^F, it is clear that $|C_i| \geq |V_i|/2$.

Since F passes $\mathcal{T}(G)$ with probability $1 - \varepsilon$ this implies that the number of edges that fail $\mathcal{T}(G)$ is at most $\varepsilon \cdot \frac{|V|d}{2}$.

Fix $S \in B$. Fix $i \in [n]$ (arbitrarily) such that $F(S)_i \neq F^*(S)_i$. Now observe that whenever $S' \in C_i$, the test $T(G)$ rejects the edge (S, S') in G because $F(S)_i \neq a_i^F = F(S')_i$. This implies that there are at least $|C_i| \geq |V_i|/2 \geq {}^{cd}/_2$ many edges incident on S that fail the test $T(G)$. Therefore, there are in total at least $|B| \cdot {}^{cd}/_4$ edges that fail the test. Recall that the total number of rejected edges is at most $\varepsilon \cdot \frac{|V|d}{2}$. Thus we have that $|B|/|V| \leq \frac{2\varepsilon}{c}$. The proof is concluded by noting that the distance between F and F^* is exactly $|B|/|V|$. ◀

Note that Lemma 10 holds even when the induced subgraph of V_i in G is a clique without self loops. In Appendix C, we provide a couple of direct product theorems on domains that are known in literature as an immediate consequence of this lemma.

Lack of Global Expansion. Notice that $G_{\mathcal{A}}$ has very bad edge-expansion/vertex-expansion but is a very good local expander, i.e., the induced subgraph containing any particular coordinate has good expansion (in fact is a clique). Lemma 10 guarantees that and thus $\mathcal{A}$ admits a direct product theorem. Therefore, $\mathcal{A}$ demonstrates that direct product testing does not require the test graph to be an expander (like the Johnson/Ramanujan graph) but only to have certain local expansion properties.

Sub-linear Size Domains. We remark here that we could consider subsets $\tilde{\mathcal{A}}$ of $\mathcal{A}$ of size smaller than n which *still* admit a direct product theorem. For example consider $\tilde{\mathcal{A}}$ as follows:

$$\tilde{\mathcal{A}} = \{\{{}^{ik}/_2, \ldots, {}^{ik}/_2 + k - 1\} \mid i \in [2n/k]\},$$

and the test graph $G_{\tilde{\mathcal{A}}}$ is given by the vertex set $\tilde{\mathcal{A}}$ and the edge set $\{(S, S') \mid S \cap S' \neq \emptyset\}$. It is easy to see that $\tilde{\mathcal{A}}$ admits a direct product theorem by applying Lemma 10. Again, we emphasize that $|\tilde{\mathcal{A}}| = 2n/k$ and yet admits a direct product theorem.

Comparison with Dinur and Kaufman. One might wonder that if direct product testing results can be established on linear sized direct product domains using simple constructions such as the sliding window domain then, why did [6] work so hard and use extremely heavy objects such as high dimensional expanders to obtain linear sized direct product domains. This is because for applications to gap and hardness amplification, it is desirable that a direct product domain also has distance amplification (defined below) and high dimensional expanders have distance amplification whereas the sliding window domain does not.

▶ **Definition 11** (Distance Amplification, [6])**.** A direct product domain $V \subseteq \binom{[n]}{k}$ is said to have distance amplification if for every $x, y \in \{0,1\}^n$ such that $\delta := \Delta(x, y) < {}^1/_k$, we have that $\Delta(\mathsf{DP}_V(x), \mathsf{DP}_V(y)) = \Omega(k\delta)$.

Thus, the construction of the sliding window domain provides a conceptual clarification as to why we need high dimensional expanders: we can obtain direct product testing from simple constructions like the sliding window domain and we can obtain distance amplification from known constructions of vertex expanders (see Appendix D); but to obtain both simultaneously, [6] needed high dimensional expanders.

References

1 Sanjeev Arora and Madhu Sudan. Improved low-degree testing and its applications. In *IN 29TH STOC*, pages 485–495, 1997.

2 Amey Bhangale, Irit Dinur, and Inbal Livni Navon. Cube vs. Cube Low Degree Test. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, pages 40:1–40:31, 2017. doi:10.4230/LIPIcs.ITCS.2017.40.

3 Andries E. Brouwer, Sebastian M. Cioabă, Ferdinand Ihringer, and Matt McGinnis. The smallest eigenvalues of Hamming graphs, Johnson graphs and other distance-regular graphs with classical parameters. *Journal of Combinatorial Theory, Series B*, 2018. doi:10.1016/j.jctb.2018.04.005.

4 Roee David, Irit Dinur, Elazar Goldenberg, Guy Kindler, and Igor Shinkar. Direct Sum Testing. *SIAM J. Comput.*, 46(4):1336–1369, 2017. doi:10.1137/16M1061655.

5 Irit Dinur and Elazar Goldenberg. Locally Testing Direct Product in the Low Error Range. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 613–622, 2008. doi:10.1109/FOCS.2008.26.

6 Irit Dinur and Tali Kaufman. High Dimensional Expanders Imply Agreement Expanders. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 974–985, 2017. doi:10.1109/FOCS.2017.94.

7 Irit Dinur and Inbal Livni Navon. Exponentially Small Soundness for the Direct Product Z-Test. In *32nd Computational Complexity Conference, CCC 2017, July 6-9, 2017, Riga, Latvia*, pages 29:1–29:50, 2017. doi:10.4230/LIPIcs.CCC.2017.29.

8 Irit Dinur and Omer Reingold. Assignment Testers: Towards a Combinatorial Proof of the PCP Theorem. *SIAM J. Comput.*, 36(4):975–1024, December 2006. doi:10.1137/S0097539705446962.

9 Irit Dinur and David Steurer. Direct Product Testing. In *IEEE 29th Conference on Computational Complexity, CCC 2014, Vancouver, BC, Canada, June 11-13, 2014*, pages 188–196, 2014. doi:10.1109/CCC.2014.27.

10 Oded Goldreich and Shmuel Safra. A Combinatorial Consistency Lemma with Application to Proving the PCP Theorem. *SIAM J. Comput.*, 29(4):1132–1154, 2000. doi:10.1137/S0097539797315744.

11 Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. New Direct-Product Testers and 2-Query PCPs. *SIAM J. Comput.*, 41(6):1722–1768, 2012. doi:10.1137/09077299X.

12 Ran Raz and Shmuel Safra. A Sub-constant Error-probability Low-degree Test, and a Sub-constant Error-probability PCP Characterization of NP. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 475–484, New York, NY, USA, 1997. ACM. doi:10.1145/258533.258641.

A Missing Proofs

Proof of Claim 5. We prove only the furthermore part. The first part follows by plugging $c = 1$. For a set $S \subseteq V$ we denote by $\mathbf{1}_S$ the characteristic vector of S. Let $p_\mu \in \mathbb{R}^n$ be the

distribution vector that describes μ. First observe that:

$$\Pr_{(u,v)}[v \in T \mid u \sim \mu] = \frac{1}{d} \cdot (p_\mu)^t \cdot A \cdot \mathbf{1}_T,$$

where the probability is taken over u that is drawn according to μ and v is a uniformly random neighbor of u. Note that $\mathbf{1}_V$ is an eigenvector of A corresponding to the largest eigenvalue (in absolute value) of d. We decompose the vectors: $p_\mu, \mathbf{1}_T$ as follows:

$$p_\mu = \frac{1}{|V|}\mathbf{1}_V + \vec{p} \text{ and } \mathbf{1}_T = \gamma \mathbf{1}_V + \vec{t}.$$

Note that $\gamma = \frac{|T|}{|V|}$ and $\vec{p}, \vec{t}$ are both orthogonal to $\mathbf{1}_V$ and let $\beta = \frac{|S|}{|V|}$. In these notations:

$$\begin{aligned} \frac{1}{d} \cdot (p_\mu)^t \cdot A \cdot \mathbf{1}_T &= \frac{1}{d}\left(\frac{1}{|V|}\mathbf{1}_V + \vec{p}\right)^t \cdot A \cdot (\gamma \mathbf{1}_V + \vec{t}) \\ &= \gamma + \langle \vec{p}A, \vec{t} \rangle \\ &\leq \gamma + \frac{\lambda}{d} \left\|\vec{p}\right\| \cdot \left\|\vec{t}\right\|, \end{aligned}$$

where in the last step we used the Cauchy-Schwarz inequality and the fact that $\|\vec{p}A\| \leq \lambda \|\vec{p}\|$. Now since the value of each coordinate of p_μ is upper bounded by $\frac{c}{|S|}$ we get: $\left\|\vec{p}\right\|^2 = \|p_\mu\|^2 - \frac{1}{|V|^2}\|\mathbf{1}_V\|^2 \leq \frac{c}{|S|} - \frac{1}{|V|}$, and $\left\|\vec{t}\right\|^2 = (\gamma(1-\gamma))|V|$. So:

$$\begin{aligned} \Pr_{(u,v)}[v \in T \mid u \sim \mu] &= \frac{1}{d} \cdot (p_\mu)^t \cdot A \cdot \mathbf{1}_T \\ &\leq \gamma + \frac{\lambda}{d} \cdot \sqrt{\left(\frac{c}{|S|} - \frac{1}{|V|}\right)\gamma(1-\gamma)|V|} \\ &\leq \gamma + \frac{\lambda}{d} \cdot \sqrt{\frac{c\gamma|V|}{|S|}} \\ &= \frac{|T|}{|V|} + \frac{\lambda}{d} \cdot \sqrt{\frac{c|T|}{|S|}} \end{aligned}$$

◀

B Application of Theorem 8: $\Omega(n)$-slice of the Hypercube

In this section, we consider the test graph $J(n, k, k(0.5+\varepsilon))$, where ε is some small constant. The domain of the direct product encoding is $\binom{[n]}{k}$. The pair (S, S') is connected by an edge if and only if: $|S \cap S'| = k/2(0.5+\varepsilon)$. We show that:

▶ **Claim 12.** *Let $\varepsilon = 1/64$. Let $n \in \mathbb{N}$, let $1/2 - \varepsilon \leq c \leq 1/2$ be a constant and let $k = c \cdot n$, then the graph $J(n, k, k \cdot (1/2+\varepsilon))$ is $(1/33, 1/2)$-coordinate expander for large enough n.*

Proof.

1. The proof of the second largest eigenvalue in absolute value was recently confirmed in [3] and we use it below. Note that $\lambda_0 = \binom{k}{k/2+\varepsilon k}\binom{n-k}{k/2-\varepsilon k}$, which is the degree of the graph. Theorem 3.10 in [3] states that λ_1 below is the second largest eigenvalue in absolute value when

$$\lambda_1 = -\binom{1}{0}\binom{k}{k/2+\varepsilon k}\binom{n-k-1}{k/2+\varepsilon k-1} + \binom{1}{1}\binom{k-1}{k/2+\varepsilon k}\binom{n-k}{k/2+\varepsilon k} \leq \lambda_0/33$$

2. Fix $i \in [n]$. Then the graph G_i is isomorphic to $J(n-1, k-1, k/2-1+\varepsilon k)$. Therefore by the first item $\lambda(G_i) < \lambda$. Clearly, for every value of $i \in [n]$ and for each $S \in V_i$ the probability that $i \in S'$ equals $1/2 + \varepsilon$.
3. We verify here the proof for $t > 16$. The case when $t = 4, 8, 12, 16$ can be routinely calculated and verified.

$$\begin{aligned}
\Pr[|T \cap S'| \leq t/4] &\leq \frac{t}{4} \cdot \frac{\binom{t}{t/4} \cdot \binom{k-t}{k/2-t/4}}{\binom{k}{k/2}} \\
&\leq \frac{t}{4} \cdot \frac{\binom{t}{t/4} \cdot \binom{k-t}{k/2-t/2}}{\binom{k}{k/2}} \\
&\leq \frac{t}{4} \cdot (1.01) 2^{H(1/4)t} (1.01) \frac{2^{k-t}}{\sqrt{k-t}} \cdot \frac{2\sqrt{k/2}}{2^k} \\
&\leq \frac{t}{2} \cdot (1.02) \sqrt{\frac{k}{k-t}} \cdot 2^{-.43t} \\
&< 1/33
\end{aligned}$$

Where in third line we used Stirling's approximation that for all $n \geq 16$ to derive: $\frac{2^n}{2\sqrt{n/2}} \leq \binom{n}{n/2} \leq (1.01)\frac{2^n}{\sqrt{\pi n/2}}$ and $\binom{n}{\epsilon n} \leq (1.01)2^{H(\epsilon)n}$. ◀

As a corollary we get that we test the direct product encoding when the domain V equals $\binom{[n]}{k}$ for values of k which are close to $n/2$. Recall that [9] established this result for all $k \leq n/2$.

C Simple Applications of Lemma 10

In this subsection, we consider two direct product domains, namely $\binom{[n]}{n/2}$ and $\binom{[n]}{2}$ and prove a direct product theorem for these domains when the test graph is a clique and a member of Johnson graph family respectively.

C.1 $n/2$ slice of the Hamming cube

A natural two-query test on the $n/2$ slice of the Hamming cube is as follows:

Test $\mathcal{T}$:
Input: A function $F : \binom{[n]}{n/2} \to \{0,1\}^{n/2}$.
Procedure: Pick uniformly and independently at random $S, S' \in \binom{[n]}{n/2}$.
Output: Accept if and only if $F(S)|_{S \cap S'} = F(S')|_{S \cap S'}$.

We now interpret the above test in the language established in Section 3. In the above test, the domain V of the direct product is $\binom{[n]}{n/2}$ and the test graph G is a clique with self loops. Therefore, for every $i \in [n]$, the induced subgraph of V_i in G is a clique (with self loops). And, for every $S \in V$ and every $i \in S$, the probability that a uniformly random neighbor S' of S in G contains i is $1/2$. Thus, from Lemma 10 we have that for any $F \in \mathcal{F}_V$, if F passes $\mathcal{T}(G)$ with probability $1 - \varepsilon$ then F is $(1 - 4\varepsilon)$-close to $\mathsf{dec}(F)$.

C.2 $J(n, 2, 1)$ of the Johnson Graph Family

For the domain $\binom{[n]}{2}$, we note that if we pick two elements from $\binom{[n]}{2}$ uniformly and independently at random then they have empty intersection with probability almost 1. Therefore,

the same test as for the $n/2$ slice of the Hamming cube does not work here. Nonetheless, there is still a natural two-query test for the domain $\binom{[n]}{2}$ described as follows:

Test $\mathcal{T}$:
Input: A function $F : \binom{[n]}{2} \to \{0,1\}^2$.
Procedure: Pick uniformly at random $S \in \binom{[n]}{2}$. Then pick uniformly at random $S' \in \binom{[n]}{2}$ such that $|S \cap S'| = 1$.
Output: Accept if and only if $F(S)|_{S \cap S'} = F(S')|_{S \cap S'}$.

We now interpret the above test in the language established in Section 3. In the above test, the domain V of the direct product is $\binom{[n]}{2}$ and the test graph G is $J(n, 2, 1)$. Note that for every $i \in [n]$, the induced subgraph of V_i in G is a clique (without self loops) because any two distinct subsets in V_i have i in their intersection and thus have intersection size equal to 1. Also for every $S \in V$ and every $i \in S$, the probability that a uniformly random neighbor S' of S in G contains i is $1/2$. Thus, from Lemma 10 we have that for any $F \in \mathcal{F}_V$, if F passes $\mathcal{T}(G)$ with probability $1 - \varepsilon$ then F is $(1 - 4\varepsilon)$-close to $\mathsf{dec}(F)$.

D Linear Sized Domains having Distance Amplification

In this section, we show how to construct a collection of sets which have distance amplification. To do so we rely on the existence of vertex expanders.

▶ **Definition 13** (Vertex Expansion). Let $G(V, E)$ be a d-regular graph. For every subset $S \subseteq V$ let $\partial(S) = \{u \in V \setminus S \mid \exists v \in S \text{ such that } (u, v) \in E\}$. The vertex isoperimetric constant $h(G)$ is defined as follows:

$$h(G) = \min_{0 \leq |S| \leq |V|/d} \frac{|\partial(S)|}{|S| \cdot d}.$$

We say that G is a vertex expander if $h(G)$ is a constant bounded away from 0.

▶ **Theorem 14** (Folklore). *For all $d > 2$, a random d-regular graph is a vertex expander with high probability.*

Given a d-regular graph $G(V, E)$ (where $n := |V|$) which is a vertex expander with vertex isoperimetric constant $\gamma > 0$, we show how to construct $\mathcal{A}_G \subseteq \binom{[n]}{d}$ of cardinality n such that $\mathcal{A}_G$ has distance amplification. We identify the vertices in V with $[n]$ and construct $\mathcal{A}_G$ as follows:

$$\mathcal{A}_G = \{\partial(\{v\}) \mid v \in V\}.$$

▶ **Claim 15.** *$\mathcal{A}_G$ has distance amplification.*

Proof. Fix distinct $x, y \in \{0,1\}^n$. Let $\delta := \Delta(x, y) \leq 1/d$. Let $R \subseteq [n]$ be the set of coordinates on which x and y differ. Clearly, $|R| \leq n/d$. The number of subsets in $\mathcal{A}_G$ that contain an element in R is at least $\gamma d|R|$. Therefore we have $\Delta(\mathsf{DP}_{\mathcal{A}_G}(x), \mathsf{DP}_{\mathcal{A}_G}(y)) \geq \gamma\delta d$. ◀

Space Complexity of Two Adaptive Bitprobe Schemes Storing Three Elements

Deepanjan Kesh
Indian Institute of Technology Guwahati, Guwahati, Assam 781039, India
deepkesh@iitg.ac.in

Abstract

We consider the following set membership problem in the bitprobe model – that of storing subsets of size at most three from a universe of size m, and answering membership queries using two adaptive bitprobes. Baig and Kesh [2] proposed a scheme for the problem which takes $\mathcal{O}(m^{2/3})$ space. In this paper, we present a proof which shows that any scheme for the problem requires $\Omega(m^{2/3})$ amount of space. These two results together settle the space complexity issue for this particular problem.

2012 ACM Subject Classification Information systems → Data structures

Keywords and phrases Data structures, set membership problem, bitprobe model, lower bound

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.12

Acknowledgements I want to thank Mirza Galib Anwarul Husain Baig for certain corrections and useful suggestions.

1 Introduction

Given a universe $\mathcal{U}$ containing m elements, consider the problem of storing an arbitrary subset $\mathcal{S}$ of size at most n. Once we have stored some such subset, we are required to answer membership queries of the form "Is x in $\mathcal{S}$?" The solutions to these problems are referred to as *schemes*. The resources that we consider to evaluate schemes for the problem are the space required by the data structure, denoted by s, and the number of bits of the data structure accessed to answer the membership queries, denoted by t. This particular class of static membership problems is referred to in the literature as the *bitprobe model*.

1.1 The Bitprobe Model

Schemes for the bitprobe model are further classified based on how the decision is made to probe a particular bit of the data structure for some query. If for a given query, the location of a bitprobe is independent of the result obtained from the previous bitprobes, then such a scheme is called a *non-adaptive scheme*. On the other hand, if the location of the current bitprobe depends on the results obtained from the previous bitprobes, then such a scheme is called an *adaptive scheme*.

Given a universe $\mathcal{U}$ and the size of the subset to be stored, say n, the design of any scheme has two components – the storage scheme and the query scheme. Given an arbitrary subset $\mathcal{S}$ of size at most n, the storage scheme sets the bits of the data structure such that the membership queries can be answered correctly. The query scheme handles arbitrary queries of the form "Is x in $\mathcal{S}$?"

Radhakrishnan *et al.* [7] introduced the following notation to represent the various schemes in the model – a scheme that takes s amount of space and requires t bitprobes to answer membership queries correctly is denoted as $(n, m, s, t)_A$ or $(n, m, s, t)_N$ depending on whether

the scheme is adaptive or non-adaptive, respectively. Sometimes, the notations $s_A(n, m, t)$ and $s_N(n, m, t)$ are used to denoted the space requirement for the respective schemes.

1.2 The Problem Statement

The space complexity for two adaptive bitprobe schemes storing subsets of size one ($n = 1, t = 2$) is well understood – the lower bound is $\Omega(m^{1/2})$, and there is an explicit scheme that matches this lower bound [1, 5]. For subsets of size two ($n = 2, t = 2$), there is a scheme due to Radhakrishnan *et al.* [6] that takes $\mathcal{O}(m^{2/3})$ amount of space. They further conjectured that this is also the minimum space required for the problem. Though progress has been made towards proving the lower bound [6, 7], the problem still remains open.

In this paper, we consider the problem of storing subsets of size at most three, and answering membership queries using two adaptive bitprobes, i.e. $n = 3$ and $t = 2$. Particularly, we look into the lower bound on space for the class schemes solving the problem.

Garg and Radhakrishnan [4] has proposed a general upper and lower bound for all adaptive schemes using two bitprobes, which are as follows.

$$\begin{aligned} s_A(n, m, 2) &= \mathcal{O}(m^{1-\frac{1}{4n+1}}). \\ s_A(n, m, 2) &= \Omega(m^{1-\frac{1}{\lfloor n/4 \rfloor}}) \end{aligned}$$

When applied to the particular case of storing three elements, the upper and lower bounds come out to be $\mathcal{O}(m^{12/13})$ and $\Omega(1)$, respectively. Garg [3] improved the general upper bound in his thesis to the following.

$$s_A(n, m, 2) = \mathcal{O}(m^{1-\frac{1}{4n-1}})$$

This improves the bound for the three element case to $\mathcal{O}(m^{11/12})$. Further improvement of the upper bound was made by Baig and Kesh [2] when they came up with a scheme that takes $\mathcal{O}(m^{2/3})$ space.

A much better lower bound was proposed by Radhakrishnan *et al.* [7] when they proved that for storing subsets of size at most two ($n = 2$), the space required is $\Omega(m^{4/7})$. As a corollary, their result puts a lower bound for the scenario when $n = 3$.

In this paper, we make the following claim – an adaptive scheme storing subsets of size at most three from a universe of size m and answering membership queries using two bitprobes requires $\Omega(m^{2/3})$ amount of space, i.e.

$$s_A(3, m, 2) = \Omega(m^{2/3}) \text{ (Theorem 19).}$$

This claim, along with the scheme due to Baig and Kesh [2] resolves the space complexity question for $n = 3$ and $t = 2$.

2 Two Bitprobe Schemes

In this section, we discuss the components of an adaptive two-bitprobe scheme, restate a few notations from the literature, and introduce some new ones used in the proof of our claim.

2.1 The Decision Tree

The data structure for two-bitprobe adaptive schemes consists of three tables, namely $\mathcal{A}, \mathcal{B}$, and $\mathcal{C}$. Every element in our universe has a location reserved in each of the three tables, a location which stores a single bit. For an element x, we use the notations $\mathcal{A}(x), \mathcal{B}(x)$, and $\mathcal{C}(x)$ to denote its location in the three tables. We abuse this notation a bit, and use these notations to also denote the bits stored in those locations.

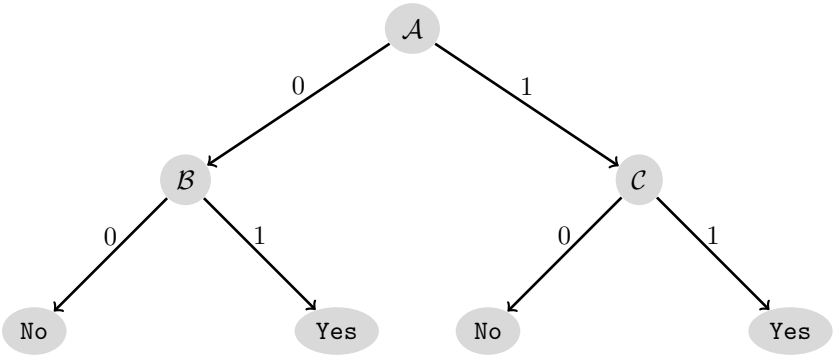


Figure 1 The decision tree of an element.

Given a subset $\mathcal{S}$ of the universe $\mathcal{U}$, the storage scheme sets the bits of these tables in such a way that the query scheme answers membership questions correctly. The arrangement and the purpose of these three tables will become more apparent from the query scheme, discussed below.

The design of the query scheme is as follows. Upon the query "Is x in $\mathcal{S}$?", the first bitprobe in made in table $\mathcal{A}$ at the location $\mathcal{A}(x)$. Depending on whether the bit stored in the location is 0 or 1, the second bitprobe is made in table $\mathcal{B}$ or $\mathcal{C}$, respectively. Finally, if the second bitprobe returned 1, we declare that the element x is a member of $\mathcal{S}$, else if 0 is returned, we declare that x is not a member of $\mathcal{S}$.

The description of the query scheme can be represented in the form of tree, shown in Figure 1, and is known as the *decision tree*.

2.2 Blocks

We borrow the terminology introduced in Radhakrishnan *et al.* [6] and define the notion of *blocks*.

▶ **Definition 1.** The set of all elements of the universe $\mathcal{U}$ that query the same location in table $\mathcal{A}$ is said to form a *block*.

It follows that if elements u and v belong to the same block, then $\mathcal{A}(u) = \mathcal{A}(v)$. Consequently, we have as many blocks as there are bits in table $\mathcal{A}$. Blocks are significant for the following reason – all the elements of a block will either query table $\mathcal{B}$ or $\mathcal{C}$, depending on whether the bit corresponding to the block stores a 0 or a 1, respectively.

Given a block, each of its elements will be numbered uniquely starting from 1. We will call this number corresponding to an element within a block as the *index* of the element.

The notion of blocks together with the notion of indices gives us a unique way of identifying the elements of the universe $\mathcal{U}$ – the block number in table $\mathcal{A}$, and the index within that block. Henceforth, we are going to use the following notation to label any element. If an element belongs to block a, and its index within the block is i, then we are going to address that element as a_i.

As a block is essentially a set, we will use the notation $|a|$ to denote the number of elements block a contains. Table $\mathcal{A}$ being a collection of blocks, we use the notation $|\mathcal{A}|$ to denote its size.

2.3 Sets

In tables $\mathcal{B}$ and $\mathcal{C}$, for the sake of convenience, which will become apparent as the proof progresses, we use the term **sets** instead of blocks for elements querying the same location.

▶ **Definition 2.** Elements that query the same location in table $\mathcal{B}$ are said to belong the same *set*. The same terminology is used for elements that query the same location in table $\mathcal{C}$.

So, there are as many sets in tables $\mathcal{B}$ and $\mathcal{C}$ as there are bits. Similar to table $\mathcal{A}$, we will use the notation $|\mathcal{B}|$ and $|\mathcal{C}|$ to denote the sizes of the respective tables.

We now define two of the key notions employed in the proof of the lower bound, that of the *mass* of a set and the *universe* of a set.

▶ **Definition 3.** The *mass* of a set is the total number of elements in all of those blocks of table $\mathcal{A}$ which has one or more elements in the set. For a set W, its mass is denoted by m_W.

▶ **Definition 4.** Given the set W, we construct a new set corresponding to W using the following steps.

Step 1. Collect all the elements in all of those blocks of table $\mathcal{A}$ which has a member in set W.

Step 2. From the resulting set, remove the elements of set W.

This set will be denoted as U_W, the *universe* of W. The size of this set is

$$|U_W| = m_W - |W|. \tag{1}$$

To take an example, suppose the set $W = \{\ a_1, e_1, f_2, h_3\ \}$. Let the members of the relevant blocks a, e, f, and h be

$$a = \{a_1, a_2, a_3\};$$
$$e = \{e_1, e_2\};$$
$$f = \{f_1, f_2, f_3, f_4\}; \textit{and}$$
$$h = \{h_1, h_2, h_3, h_4, h_5\},$$

then,

$$m_W = |a| + |e| + |f| + |h|$$
$$U_W = \{a_2, a_3, e_2, f_1, f_3, f_4, h_1, h_2, h_4, h_5\}.$$

3 Clean and Dirty Sets

In this section, we define and discuss two categories of sets, namely *clean* sets and *dirty* sets.

▶ **Definition 5.** A set is said to be *dirty* if the set contains more than one element from some block of table $\mathcal{A}$. On the other hand, if all of the elements of a set are from distinct blocks of table $\mathcal{A}$, then that set is said to be *clean*.

For example, the set $\{\ a_1, a_2, b_3, c_4\ \}$ is a dirty set as it contains two elements from block a. On the other hand, the set $\{\ e_1, f_1, g_3\ \}$ is a clean set. Note that same indices, as in the later set, are allowed, but same block numbers are not.

We now make an important observation about the relationship between blocks and dirty sets.

▶ **Lemma 6.** *If a set in any of the tables is dirty due to the elements of a block of table $\mathcal{A}$, then all of the elements of that block must belong to distinct sets in the other table.*

In other words, if some block of table $\mathcal{A}$ makes some set dirty in table $\mathcal{B}$, then it cannot make any set dirty in table $\mathcal{C}$, and vice versa. This is similar to what has been considered by Radhakrishnan *et al.* [6] as part of item 4 in Section 4. We reprove it in our teminology.

Proof. Without loss of generality, let the elements a_1 and a_2 (the first and the second elements of block a) belong to the same set W in table $\mathcal{B}$. So, the set W is dirty due to block a. We will prove that the elements of this block will necessarily belong to distinct sets in table $\mathcal{C}$.

Let us construct the subset $\mathcal{S}$ so as to contain the element a_1 but not the element a_2. In this case, $\mathcal{A}(a)$ cannot be 0. If $\mathcal{A}(a)$ is indeed 0, then upon query for the element a_1, we would get a 0 from table $\mathcal{A}$, and the second query for a_1 must be in table $\mathcal{B}$. As a_1 belongs to the set W of table $\mathcal{B}$, and as a_1 belongs to the subset $\mathcal{S}$, the bit corresponding to the set W must be set to 1.

Under this assignment, we look into the queries for the element a_2. As $\mathcal{A}(a) = 0$, the second query for a_2 will be in table $\mathcal{B}$. As we have assumed that a_1 and a_2 belong to the set W in $\mathcal{B}$, the second query for a_2 will be to the bit corresponding to the set W. As the bit stored is 1, we will deduce that a_2 belongs to $\mathcal{S}$, which would be incorrect.

So, $\mathcal{A}(a)$ cannot store 0, and hence it must store 1. So, the second query for all the elements of block a will be made in table $\mathcal{C}$.

In table $\mathcal{C}$, if two elements of block a are again together in some set, we can put one of the elements in $\mathcal{S}$ but not the other, and we will reach a contradiction similar to the one above. Note that the subset $\mathcal{S}$ is allowed to contain at most three elements, and to arrive at the contradiction we need to put at most two elements in $\mathcal{S}$.

We can thus conclude that the elements of block a must belong to distinct sets in table $\mathcal{C}$. ◀

The next lemma shows the relationship between multiple blocks that create dirty sets in table $\mathcal{B}$

▶ **Lemma 7.** *Consider all of those blocks of table $\mathcal{A}$ that make one or more sets dirty in table $\mathcal{B}$. All of the elements from all of those blocks must necessarily be in distinct sets of table $\mathcal{C}$.*

Proof. Without loss of generality, let the elements a_1 and a_2 of block a make some set dirty in table $\mathcal{B}$. Putting one of them in subset $\mathcal{S}$ but not the other, and reasoning along the lines of the proof of Lemma 6, we will have $\mathcal{A}(a) = 1$. Similarly, if we have the elements b_1 and b_2 of block b making some set dirty in table $\mathcal{B}$, we can put one of them in $\mathcal{S}$ and not the other, and ensure that $\mathcal{A}(b) = 1$.

Now, suppose that the elements a_i and b_j belong to a set X in table $\mathcal{C}$. In this scenario, we will add a_i to subset $\mathcal{S}$ as its third member. As the second query for a_i will be in table $\mathcal{C}$, we will have to set the bit corresponding to the set X to 1.

With this assignment, we look in the query "Is b_j in $\mathcal{S}$?" As $\mathcal{A}(b)$ is 1, the second query of b_j will be in table $\mathcal{C}$. As it belongs to set X, we will get a 1 for the second query and incorrectly deduce that b_j is a member of $\mathcal{S}$.

This tells us that all the elements of blocks a and b must belong to distinct sets of table $\mathcal{C}$. It is to be noted that it has been implicitly assumed that the elements a_1, a_2, b_1, and b_2 are distinct from a_i and b_j, which need not necessarily be true. In such a case too it can be argued, as above, that the elements of the two blocks cannot share a set in table $\mathcal{C}$. ◀

The aforementioned restrictions on the blocks creating dirty sets help us to estimate the total number elements in all of those blocks of table $\mathcal{A}$ which are responsible for creating dirty sets in table $\mathcal{B}$.

Consider those blocks of table $\mathcal{A}$ that create dirty sets in table $\mathcal{B}$. Let the total number of elements in all of those blocks combined be $N_{\mathcal{B}}$. Lemma 7 tells us that of those elements must belong to distinct sets in table $\mathcal{C}$. This observation immediately puts the following bound on $N_{\mathcal{B}}$ –

$$N_{\mathcal{B}} \leq |\mathcal{C}|.$$

We can do the same exercise for table $\mathcal{C}$, and count the total number of elements in all of the blocks responsible for creating dirty sets in table $\mathcal{C}$. If that number is $N_{\mathcal{C}}$, then we will arrive at the relation

$$N_{\mathcal{C}} \leq |\mathcal{B}|.$$

In our data structure, let us remove all of those $N_{\mathcal{B}}$ elements from their respective sets and put in singleton sets in table $\mathcal{B}$, and we do the same for the $N_{\mathcal{C}}$ elements in table $\mathcal{C}$. This will make all of the sets in the tables $\mathcal{B}$ and $\mathcal{C}$ clean. Of course, this comes with an additional cost to the size our data structure, and the its new size will be

$$|\mathcal{A}| + 2|\mathcal{B}| + 2|\mathcal{C}|.$$

If the sizes of all of the tables in our initial data structure be s each, resulting in the total size to be $3s$ to begin with, after the adjustment mentioned above we will have a data structure whose size is at most $5s$, an increase by a constant factor, and no asymptotic penalty.

We can further introduce s empty blocks in table $\mathcal{A}$ and make the sizes of the three tables uniform. With these observations, we can make the following claim.

▶ **Theorem 8.** *Given a $(3, m, s, 2)_A$-scheme, we can have an equivalent $(3, m, 2 \times s, 2)_A$ adaptive scheme where the data structure has only clean sets.*

Henceforth, we will talk exclusively about schemes with clean sets only, and whose table sizes are all equal, and prove the lower bound for this class of schemes. Theorem 8 guarantess that the lower bound claim will also hold for the general class of schemes, with or without dirty sets.

4 Mass of a set

The following relationship holds between the total mass of all the sets of tables $\mathcal{B}$ and $\mathcal{C}$, and the sizes of the blocks of table $\mathcal{A}$.

▶ **Lemma 9.** *For the sets of tables $\mathcal{B}$ and $\mathcal{C}$, and the blocks of table $\mathcal{A}$, the following equality is true.*

$$\sum_{W \in \mathcal{B}} m_W = \sum_{X \in \mathcal{C}} m_X = \sum_{a \in \mathcal{A}} |a|^2 \tag{2}$$

Proof. Consider a block a of table $\mathcal{A}$. As we are dealing with schemes containing clean sets only, the elements of the block a will be distributed in exactly $|a|$ sets of table $\mathcal{B}$. This implies that block a will contribute to the masses of $|a|$ sets of table $\mathcal{B}$. In other words, the term $|a|$ will occur as summand in the masses of $|a|$ sets of table $\mathcal{B}$.

So, in the total mass of all the sets of table $\mathcal{B}$, $|a|$ will occur as a summand exactly $|a|$ times. In other words, the contribution of block a to the total mass of table $\mathcal{B}$ is $|a|^2$, and the equality follows.

We can similarly argue about table $\mathcal{C}$. ◀

▶ **Lemma 10.** *The following inequality holds between the masses of the sets of tables $\mathcal{B}$ and $\mathcal{C}$, and the size of table $\mathcal{A}$ –*

$$\sum_{W \in \mathcal{B}} m_W = \sum_{X \in \mathcal{C}} m_X \geq \frac{m^2}{|\mathcal{A}|}. \tag{3}$$

Proof. Consider the sum from Equation 2

$$\sum_{a \in \mathcal{A}} |a|^2.$$

Using the arithmetic mean geometric mean inequality, we can show that the sum is minimized when all the summands are equal, i.e.

$$\sum_{a \in \mathcal{A}} |a|^2 \geq |\mathcal{A}| \times \left(\frac{\sum_{a \in \mathcal{A}} |a|}{|\mathcal{A}|} \right)^2.$$

By using the fact that $\sum_{a \in \mathcal{A}} |a| = m$, we get the desired R.H.S. ◀

It is interesting to note that the total mass of either of the tables $\mathcal{B}$ and $\mathcal{C}$ is minimized when all of the blocks of table $\mathcal{A}$ are of equal size.

5 Bad Elements

In this section, we give a characterisation of certain elements of our universe $\mathcal{U}$ as being *bad* for some particular sets of table $\mathcal{C}$.

▶ **Definition 11.** Suppose an element a_i from block a of table $\mathcal{A}$ belongs to a set W of table $\mathcal{B}$, and to a set X in table $\mathcal{C}$. Such an element is said to be a *bad element* for the set X if the following holds:

1. a_i shares the set W with two other elements b_j and c_k, from blocks b and c, respectively.
2. There exists elements b_l and c_n, different from the elements b_j and c_k, such that they share a set in table $\mathcal{C}$.

Figure 2 describes pictorially the notion of a bad element.

As in the above instance, let us suppose that the element a_i is a bad element for the set X of table $\mathcal{C}$. We discuss below why, given such an arrangement of elements, is a_i referred to as bad for the set X.

5.1 Property of a bad element

Consider the following subset $\mathcal{S} = \{ a_i, b_l \}$. We show that if we want to store this subset, then $\mathcal{A}(a)$ must be set to 1, and we show this by contradiction.

If it is indeed the case that $\mathcal{A}(a) = 0$, then upon query for the element a_i, we will go to table $\mathcal{B}$ for the second query. As a_i belongs to the set W in table $\mathcal{B}$, and as it is also a member of subset $\mathcal{S}$, the bit corresponding to W must be set to 1.

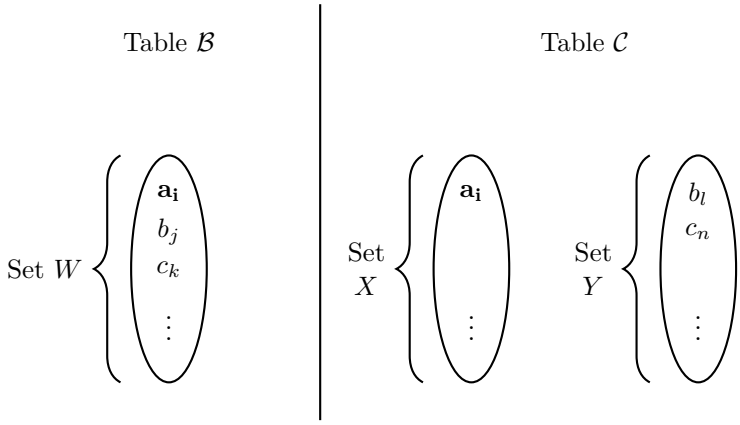


Figure 2 In this arrangement, a_i is a bad element for the set X. Note that $j \neq l$ and $k \neq n$. Further, the sets X and Y need not necessarily be distinct.

This would imply that $\mathcal{A}(b) = 1$. If it is not, and $\mathcal{A}(b)$ is set to 0, then the second query for the element b_j would in table $\mathcal{B}$. As b_j is a member of the set W in table $\mathcal{B}$, we would get a 1 against the second query and incorrectly assume b_j is a member of $\mathcal{S}$. So, we see that $\mathcal{A}(b)$ must be 1. We can similarly argue that $\mathcal{A}(c)$ must also be 1.

As shown in Figure 2, the elements b_l and c_n belong to the set Y in table $\mathcal{C}$. From the arrangement of elements above, we can further deduce that as b_l is a member of $\mathcal{S}$, and it is also a member of the set Y in table $\mathcal{C}$, the bit corresponding to the set Y must be set to 1. If we now consider the query "Is c_n in $\mathcal{S}$?", we would find out that we would incorrectly get that c_n is a member of $\mathcal{S}$.

This shows that if the subset $\mathcal{S}$ contains the elements a_i and b_l, then $\mathcal{A}(a)$ cannot be 0, and hence it must be set to 1. We summarise our findings in the following lemma.

▶ **Lemma 12.** *If the subset $\mathcal{S}$ contains the elements a_i and b_l, then $\mathcal{A}(a)$ must be set to 1.*

5.2 Universe of X

We would show that no two elements of $U_X \setminus a$, the universe of X minus the elements of block a, can share a set in table $\mathcal{B}$, and we arrive at this by contradiction.

Let us suppose, without loss of generality, that the elements x_1 and y_1 belong to set X, implying that the elements of the blocks x and y will be part of U_X. Let us further assume that the elements x_2 and y_2, which are members of U_X, share the set Z in table $\mathcal{B}$. The arrangement of the elements can be seen in Figure 3.

In this scenario we will show next that while trying to store the subset $\mathcal{S} = \{ a_i, b_l, x_2 \}$, the query for the element y_2 will give an incorrect answer.

As the subset $\mathcal{S}$ contains the elements a_i and b_l, Lemma 12 tells us that $\mathcal{A}(a)$ must be equal to 1. This means that the second query for a_i will be in table $\mathcal{C}$. a_i belongs to the set X in this table, and hence the bit corresponding to the X must be set to 1.

The element x_1 is not a member of $\mathcal{S}$, so the second query for this element cannot be in table $\mathcal{C}$. If it is, then it will query the bit corresponding to the set X and get a 1, implying x_1 *is* a member of $\mathcal{S}$. To ensure that the second query is in table $\mathcal{B}$, we ought to have $\mathcal{A}(x) = 0$. We can argue similarly about the block y, and conclude that $\mathcal{A}(y) = 0$.

With $\mathcal{A}(x) = 0$, the second query for the element x_2 must be in table $\mathcal{B}$. Considering the fact that x_2 is a member of the subset $\mathcal{S}$, the bit corresponding to the set Z in table $\mathcal{B}$ must be set to 1(Figure 3).

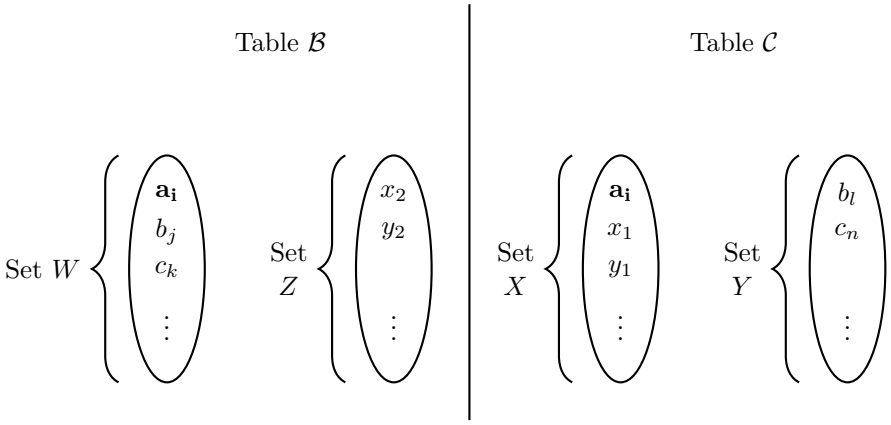


Figure 3 In this arrangement, a_i is a bad element. Note that $j \neq l$ and $k \neq n$.

Let us now consider the query "Is y_2 in $\mathcal{S}$?" As $\mathcal{A}(y) = 0$, the second query for y_2 will be in table $\mathcal{B}$. In this table, y_2 belongs to the set Z, and hence, the second query for this element will return a 1, implying incorrectly that y_2 is a member of $\mathcal{S}$.

We summarise our findings in the following lemma.

▶ **Lemma 13.** *Suppose that an element a_i is bad for a set X in table $\mathcal{C}$. Then, the elements of $U_X \setminus a$ must belong to distinct sets in table $\mathcal{B}$.*

5.3 Bounded Sets of Table $\mathcal{C}$

Lemma 13 tells us that the elements of $U_X \setminus a$ must belong to distinct sets in table $\mathcal{B}$. Hence, the size of U_X is bounded by the size of $\mathcal{B}$.

$$|U_X| - |a| \leq |\mathcal{B}|$$
$$\implies m_X \leq |\mathcal{B}| + |a| + |X| \text{ (from Equation 1)},$$

giving us the following corollary to the lemma above.

▶ **Corollary 14.** *If a set X of table $\mathcal{C}$ contains a bad element from a block a of table $\mathcal{A}$, then the mass of X must satisfy the following inequality.*

$$m_X \leq |\mathcal{B}| + |a| + |X|. \tag{4}$$

We now come to reason why elements with the property, as stated in Definition 11, are said to be bad for sets of table $\mathcal{C}$. A bad element in a set of table $\mathcal{C}$ puts an upper bound on the mass of that set. For small data structures, the sizes of the sets of the tables $\mathcal{B}$ and $\mathcal{C}$ must be large, so that the number of distinct sets is small. A bad element in a set, on the other hand, restricts the size of the set.

For easy reference, we characterise these sets as *bounded sets* because their mass has an upper bound.

▶ **Definition 15.** A set in table $\mathcal{C}$ which contains one or more bad elements is called a *bounded set.*

5.4 Large Sets of Table $\mathcal{B}$

▶ **Lemma 16.** *Consider a set W of table $\mathcal{B}$ whose mass satisfies the following inequality.*

$$m_W \geq 2 \times |\mathcal{C}| + |W| + 1.$$

Then, all of the elements of set W are bad elements.

Proof. If some set W of table $\mathcal{B}$ satisfies the above inequality, then the size of the universe of W, which is $m_W - |W|$, has more than twice the number of elements than there are sets in table $\mathcal{C}$. Hence, there is at least one set in table $\mathcal{C}$ which contains three elements or more of U_W.

Without loss of generality, let us assume that all of the elements of W have index 1, i.e. $W = \{ a_1, b_1, c_1, d_1, \dots \}$. Let us assume further that the set X of table $\mathcal{C}$ is the set containing at least three elements of U_W; let those elements be a_2, b_2, and c_2.

If we consider the element c_1, it satisfies the definition of a bad element - a_1 and b_1 along with c_1 belong to the set W in table $\mathcal{B}$, and the elements a_2 and b_2 belong to the set X in table $\mathcal{C}$. We can say the same for every element of W except for a_1 and b_1. So, we get $|W| - 2$ bad elements in the set W.

a_1 is also a bad element due to b_1, c_1, b_2, c_2, and similarly b_1. So, all of the elements of W are bad for one or the other set of table $\mathcal{C}$. ◀

We characterise these sets of table $\mathcal{B}$ as *large sets.*

▶ **Definition 17.** A set W in table $\mathcal{B}$ is called a *large set* if its mass satisfies the following inequality.

$$m_W \geq 2 \times |\mathcal{C}| + |W| + 1. \tag{5}$$

We next highlight an important relation between the masses of large sets of table $\mathcal{B}$ and the bounded sets of table $\mathcal{C}$.

▶ **Lemma 18.** *The total mass of the large sets of table $\mathcal{B}$ is less than or equal to the total mass of the bounded sets of table $\mathcal{C}$.*

Proof. Let a_i be an element that belongs to a large set W in table $\mathcal{B}$. Then, two things hold true – a_i is a bad element, and a_i contributes the amount $|a|$ to the mass of the large set W.

In table $\mathcal{C}$, if a_i belongs to set X, then two things hold true here as well – X is a bounded set, and a_i contributes the amount $|a|$ to the mass of X.

So, every contribution to the total mass of a large set will also have an equal amount of contribution to the total mass of bounded sets.

Additionally, there could be bad elements due to sets that are not large, or there would be elements in bounded sets that are not bad. Both of these will contribute to the mass of bounded sets, but not to that of large sets. Consequently, the inequality follows. ◀

6 The Lower Bound

Baig and Kesh [2] have shown that there exists an adaptive scheme that stores subsets of size at most three elements from a universe of size m, and answers memberships queries using two bitprobes. The space required by the scheme is $3 \times m^{2/3}$. In other words, we have a $(3, m, 3 \times m^{2/3}, 2)_A$-scheme.

In this section, we will show that a $(3, 2 \times m, 3 \times m^{2/3}, 2)_A$ scheme cannot exist. Thus, a $3 \times m^{2/3}$-sized datastructure is not sufficient if the universe size is doubled, giving us the desired contradiction.

Section 3 tells us that we will only have to look into schemes with data structures having the following properties – all of the sets are clean, and the sizes of the three tables are equal. So we have the following setting.

$$\begin{aligned} |U| &= 2m \\ n &= 3 \\ t &= 2 \\ |\mathcal{A}| = |\mathcal{B}| = |\mathcal{C}| &= m^{2/3} \end{aligned}$$

If the total number of elements is $2m$, then Lemma 10 tells us that the total mass of all the sets of table $\mathcal{B}$ or table $\mathcal{C}$ is at least

$$\frac{(2m)^2}{|\mathcal{A}|} = \frac{(2m)^2}{m^{2/3}} = 4m^{4/3}.$$

The mass of a set W which is not large is at most

$2 \times |C| + |W| = 2m^{2/3} + |W|$ (from Equation 5).

In the worst case, the total number of such sets could at most $m^{2/3}$ – the size of table $\mathcal{B}$ – and the total number of elements belonging to such sets could be $2m$. So, the total mass of all such non-large sets of table $\mathcal{B}$ is

$$m^{2/3} \times 2 \times m^{2/3} + 2m = 2 \times m^{4/3} + 2m.$$

This means that the total mass of the large sets of table $\mathcal{B}$ is at least

$$4m^{4/3} - 2 \times m^{4/3} - 2m = 2m^{4/3} - 2m \tag{6}$$

If table $\mathcal{C}$, the mass of a bounded set X is at most

$|\mathcal{B}| + |a| + |X| = m^{2/3} + |a| + |X|$ (Corollary 14).

Here, a is the block to which the bad element belongs. In the case that all of the sets of table $\mathcal{C}$ are bounded, then the total mass of all bounded sets is at most

$$m^{2/3} \times m^{2/3} + 2m + 2m = m^{4/3} + 4m. \tag{7}$$

Equations 6 and 7 tell us that as long as $m \geq 7^3$, the total mass of large sets of table $\mathcal{B}$ is strictly greater than the total mass of bounded sets of table $\mathcal{C}$, which is absurd as it contradicts Lemma 18.

This tells us that a $(3, 2 \times m, 3 \times m^{2/3}, 2)_A$-scheme cannot exist. We now arrive at the final result.

▶ **Theorem 19.** $s_A(3, m, 2) = \Omega(m^{2/3})$.

7 Conclusion

In this paper, we have provided a lower bound for two-bitprobe adaptive schemes storing subsets of size at most three (Theorem 19), which matches with the upper bound for the problem proposed by Baig and Kesh [2]. This, as alluded to earlier, settles the space complexity problem for this particular n and t.

The lower bound for the problem where $n = 2$ and $t = 2$, conjectured by Radhakrishnan *et al.* [6] to be $\Omega(m^{2/3})$, still remains open. We hope that the notions of the mass of a set (Definition 3) and the universe of a set (Definition 4) would help us better understand the data structure for this problem, and consequently resolve the conjecture.

References

1 Noga Alon and Uriel Feige. On the power of two, three and four probes. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 346–354, 2009.

2 Mirza Galib Anwarul Husain Baig and Deepanjan Kesh. Two New Schemes in the Bitprobe Model. In *WALCOM: Algorithms and Computation - 12th International Conference, WALCOM 2018, Dhaka, Bangladesh, March 3-5, 2018, Proceedings*, pages 68–79, 2018.

3 Mohit Garg. *The Bit-probe Complexity of Set Membership*. PhD thesis, School of Technology and Computer Science, Tata Institute of Fundamental Research, Homi Bhabha Road, Navy Nagar, Colaba, Mumbai 400005, India, 2016.

4 Mohit Garg and Jaikumar Radhakrishnan. Set membership with a few bit probes. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 776–784, 2015.

5 Moshe Lewenstein, J. Ian Munro, Patrick K. Nicholson, and Venkatesh Raman. Improved Explicit Data Structures in the Bitprobe Model. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, pages 630–641, 2014.

6 Jaikumar Radhakrishnan, Venkatesh Raman, and S. Srinivasa Rao. Explicit Deterministic Constructions for Membership in the Bitprobe Model. In *Algorithms - ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings*, pages 290–299, 2001.

7 Jaikumar Radhakrishnan, Smit Shah, and Saswata Shannigrahi. Data Structures for Storing Small Sets in the Bitprobe Model. In *Algorithms - ESA 2010, 18th Annual European Symposium, Liverpool, UK, September 6-8, 2010. Proceedings, Part II*, pages 159–170, 2010.

New Constructions with Quadratic Separation between Sensitivity and Block Sensitivity

Siddhesh Chaubal
University of Texas at Austin, USA
siddhesh@cs.utexas.edu

Anna Gál
University of Texas at Austin, USA
panni@cs.utexas.edu

Abstract

Nisan and Szegedy [14] conjectured that block sensitivity is at most polynomial in sensitivity for any Boolean function. There is a huge gap between the best known upper bound on block sensitivity in terms of sensitivity – which is exponential, and the best known separating examples – which give only a quadratic separation between block sensitivity and sensitivity.

In this paper we give various new constructions of families of Boolean functions that exhibit quadratic separation between sensitivity and block sensitivity. Our constructions have several novel aspects. For example, we give the first direct constructions of families of Boolean functions that have both 0-block sensitivity and 1-block sensitivity quadratically larger than sensitivity.

2012 ACM Subject Classification Theory of computation → Complexity classes

Keywords and phrases Sensitivity Conjecture, Boolean Functions, Complexity Measures

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.13

1 Introduction

The Sensitivity Conjecture posed by Nisan and Szegedy [14] is one of the most intriguing, yet elusive problems in computational complexity theory.

The *sensitivity* $s(f)$ of a Boolean function f is the maximum over all inputs x of the number of coordinate positions i such that changing the value of the i-th bit of x changes the value of the function. The *block sensitivity* $bs(f)$ of a Boolean function f is the maximum over all inputs x of the number of disjoint blocks of bits such that changing the value of all bits of x in any given block changes the value of the function. (See Section 2 for more formal definitions.) Sensitivity was introduced by Cook, Dwork and Reischuk [8] as a measure to prove lower bounds on the parallel complexity of Boolean functions in the CREW PRAM model. Nisan [13] defined the more general block sensitivity measure, and showed that the CREW PRAM complexity of any Boolean function f is characterized by its block sensitivity up to constant factors as $\Theta(\log bs(f))$. Nisan also showed that several other complexity measures, including certificate complexity and decision tree depth are polynomially related to block sensitivity. Nisan and Szegedy [14] showed that the degree of real polynomials representing a Boolean function f is also polynomially related to its block sensitivity. These relations extend to approximate representation by real polynomials and to randomized and quantum decision tree depth. Thus, a number of important complexity measures are polynomially related to block sensitivity. See [6, 11] for a survey.

However, it remains open to fully understand the relationship between sensitivity and block sensitivity. Of course for any Boolean function f, $s(f) \leq bs(f)$. Nisan and Szegedy [14] conjectured that block sensitivity is at most polynomial in sensitivity for any Boolean

38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018).
Editors: Sumit Ganguly and Paritosh Pandya; Article No. 13; pp. 13:1–13:16

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

function f. They even raised the possibility that $bs(f) = O(s(f)^2)$. This possibility is still not ruled out - the best separation so far remains quadratic. The current best upper bound on block sensitivity in terms of sensitivity by Ambainis et al. [2, 4] is exponential: $bs(f) \leq s(f)2^{s(f)-1}$. (More precisely, $bs(f) \leq \max\{2^{s(f)-1}(s(f) - \frac{1}{3}), s(f)\}$ [4].) This improves the earlier upper bounds of Kenyon and Kutin and Simon [12, 16].

The first example of a function with quadratic separation between its sensitivity and block sensitivity was given by Rubinstein [15] who constructed a function f with $bs(f) = \frac{1}{2}s(f)^2$. Other constructions with quadratic separation were given in [19, 7, 10, 5]. The largest separation so far is achieved by the construction of Ambainis and Sun [5] who gave a function f with $bs(f) = \frac{2}{3}s(f)^2 - \frac{1}{3}s(f)$.

Improving the constant $\frac{2}{3}$ in the separation would be interesting, since a function f with $bs(f) > cs(f)^2$ for a constant $c > 1$ would imply a construction with superquadratic separation by iterated composition of the function f [3].

In order to better understand the relationship between sensitivity and block sensitivity, the one-sided versions of the measures 0-sensitivity $s_0(f)$, 1-sensitivity $s_1(f)$, 0-block sensitivity $bs_0(f)$ and 1-block sensitivity $bs_1(f)$ have also been extensively studied. These measures are obtained by restricting attention to inputs $x \in f^{-1}(0)$ for defining 0-sensitivity and 0-block sensitivity and to inputs $x \in f^{-1}(1)$ for defining 1-sensitivity and 1-block sensitivity, respectively. (See Section 2 for formal definitions.) Then $s(f) = \max\{s_0(f), s_1(f)\}$ and $bs(f) = \max\{bs_0(f), bs_1(f)\}$.

Ambainis and Prusis [3] (improving the constant of a statement in [12]) proved that $bs_0(f) \leq \frac{2}{3}s_0(f)C_1(f)$ where $C_1(f)$ denotes the 1-certificate complexity of f. See Section 2 for the definition of certificate complexity. On the other hand, [13] proved that $C_1(f) \leq bs_1(f)s_0(f)$. The analogous statements also hold for upper bounding bs_1 and C_0, respectively. Combining these results implies that in order to obtain much stronger separation between sensitivity and block sensitivity it is necessary to construct functions f such that both $bs_0(f)$ and $bs_1(f)$ are significantly larger than $s(f)$.

Avishay Tal [18] pointed out to us, that one can get such examples by the following trick. Let g be any function with $bs(g) = \Omega(s(g)^2)$, then taking $f(x, y) = g(x) \vee \neg g(y)$ will give $\min\{bs_0(f), bs_1(f)\} = \Omega(s(f)^2)$. Notice however that in this example the function f will not give an asymptotically larger separation between its block sensitivity and sensitivity than what was achieved by the function g unless $bs_1(g) = \theta(bs_0(g))$. Thus, limitations on the separation that follow from properties of the function g will be inherited by the function f. By direct constructions, the largest simultaneous separation has been $\min\{bs_0(f), bs_1(f)\} = \Omega(s(f)^{\log_2 3})$ in [1]. On the other hand, all previous direct constructions with quadratic separation between $bs(f)$ and $s(f)$ had $\min\{bs_0(f), bs_1(f)\} = O(s(f))$.

1.1 Our Results

In this paper we give various new constructions of families of Boolean functions that exhibit quadratic separation between sensitivity and block sensitivity. Our constructions have several novel aspects.

We provide the first direct constructions of families of Boolean functions f with $\min\{bs_0(f), bs_1(f)\} = \Omega(s(f)^2)$. Our simultaneous quadratic separation of both 0-block sensitivity and 1-block sensitivity from sensitivity is based on a more refined study of the effects of function composition on these measures. We also present sufficient conditions for achieving such simultaneous separations and give several examples of functions satisfying these conditions.

All previous constructions - with the exception of Chakraborty's functions [7] - were of the form $f = OR_m \circ g_k$ that is $f : \{0,1\}^{mk} \to \{0,1\}$ was obtained by composing the m-bit OR function with an appropriately chosen inner function g on k bits. Chakraborty [7] did

not use function composition at all. As for the choice of the inner function, [10] defined the inner function g based on codewords of a Hamming code. All other constructions (including Chakraborty [7]) used the presence of certain patterns in the input x to set the function value $g(x)$ to 1.

We observe that other function compositions instead of OR-composition can also yield quadratic separations. We define new functions, that could be used as inner or outer functions, based on algebraic criterions related to multiplication in finite fields or polynomial multiplication.

We also give new examples of functions to match the current best constant of $\frac{2}{3}$ by Ambainis and Sun [5] among the known quadratic separations. We give a general condition for achieving quadratic separations with the $\frac{2}{3}$ constant for functions defined by families of certificates. The function by Ambainis and Sun [5] fits into this framework.

2 Preliminaries

Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. For $x \in \{0,1\}^n$ and $i \in [n]$ we denote by x^i the input obtained by flipping the i-th bit of x. More generally, for $S \subseteq [n]$ we denote by x^S the input obtained by flipping the bits of x in all coordinates in the subset S.

▶ **Definition 1** (Sensitivity)**.** The sensitivity $s(f,x)$ of a Boolean function f on input x is the number of coordinates $i \in [n]$ such that $f(x) \neq f(x^i)$. The 0-sensitivity and 1-sensitivity of f are defined as $s_0(f) = \max\{s(f,x) : f(x) = 0\}$ and $s_1(f) = \max\{s(f,x) : f(x) = 1\}$, respectively. The sensitivity of f is defined as $s(f) = \max\{s(f,x) : x \in \{0,1\}^n\} = \max\{s_0(f), s_1(f)\}$.

▶ **Definition 2** (Block Sensitivity)**.** The block sensitivity $bs(f,x)$ of a Boolean function f on input x is the maximum number of pairwise disjoint subsets $S_1, \ldots, S_k$ of $[n]$ such that for each $i \in [k]$ $f(x) \neq f(x^{S_i})$. The 0-block sensitivity and 1-block sensitivity of f are defined as $bs_0(f) = \max\{bs(f,x) : f(x) = 0\}$ and $bs_1(f) = \max\{bs(f,x) : f(x) = 1\}$, respectively. The block sensitivity of f is defined as $bs(f) = \max\{bs(f,x) : x \in \{0,1\}^n\} = \max\{bs_0(f), bs_1(f)\}$.

It is convenient to refer to coordinates $i \in [n]$ such that $f(x) \neq f(x^i)$ as *sensitive bits* for f on x. Similarly, a subset $S \subseteq [n]$ is called a *sensitive block* for f on x if $f(x) \neq f(x^S)$.

▶ **Definition 3** (Partial assignment)**.** Given an integer $n > 0$, a partial assignment α is a function $\alpha\colon [n] \to \{0,1,\star\}$. A partial assignment α corresponds naturally to a setting of n variables $(x_1, x_2, \ldots x_n)$ to $\{0,1,\star\}$ where x_i is set to $\alpha(i)$. The variables set to $\star$ are called unassigned or free, and we say that the variables set to 0 or 1 are fixed.
We say that $x \in \{0,1\}^n$ agrees with α if $x_i = \alpha(i)$ for all i such that $\alpha(i) \neq \star$.
The size of a partial assignment α is defined as the number of fixed variables of α.

▶ **Definition 4** (Certificate)**.** For a function $f\colon \{0,1\}^n \to \{0,1\}$ and input $x \in \{0,1\}^n$ a partial assignment α is a certificate of f on x if x agrees with α and any input y agreeing with α satisfies $f(y) = f(x)$.
The size of a certificate α is defined as the size of the partial assignment α.

▶ **Definition 5** (Certificate Complexity)**.** The certificate complexity $C(f,x)$ of a Boolean function f on input x is the size of the smallest certificate of f on x. The 0-certificate complexity and 1-certificate complexity of f are defined as $C_0(f) = \max\{C(f,x) : f(x) = 0\}$ and $C_1(f) = \max\{C(f,x) : f(x) = 1\}$, respectively. The certificate complexity of f is defined as $C(f) = \max\{C(f,x) : x \in \{0,1\}^n\} = \max\{C_0(f), C_1(f)\}$.

▶ **Definition 6** (Function defined by a set of partial assignments)**.** Let $C = \{c_1, c_2, \dots c_t\}$ be a set of partial assignments $c_1, c_2, \dots c_t \colon [n] \to \{0, 1, \star\}$.
Then C naturally defines a function $g_C \colon \{0,1\}^n \to \{0,1\}$ as:
$g_C(x) = 1$ iff x agrees with some partial assignment $c_i \in C$.

▶ **Definition 7** (Distances)**.** The distance between two inputs $x, y \in \{0,1\}^n$ is defined as the number of bits in which they differ.
The distance between an input $x \in \{0,1\}^n$ and a partial assignment $\alpha \colon [n] \to \{0, 1, \star\}$ is defined as the minimum distance between x and any input y agreeing with α.
The distance between two partial assignments $\alpha, \beta \colon [n] \to \{0, 1, \star\}$ is defined as the minimum distance between any input x agreeing with α and any input y agreeing with β.

▶ **Definition 8** (Function Composition)**.** For Boolean functions $f : \{0,1\}^m \to \{0,1\}$ and $g : \{0,1\}^k \to \{0,1\}$ the function $f \circ g : \{0,1\}^{mk} \to \{0,1\}$ is defined on $z \in \{0,1\}^{mk}$ as

$$f \circ g(z) = f(g(z_1, \dots z_k), g(z_{k+1}, \dots, z_{2k}), \dots, g(z_{(m-1)k+1}, \dots, z_{mk}))$$

Properties of function composition were formally studied with respect to sensitivity and block sensitivity (as well as other related measures) in [17, 9]. We note the following two properties, relevant for us.

▶ **Lemma 9.** *[17, 9] For any Boolean functions f and g we have $s(f \circ g) \leq s(f)s(g)$.*

▶ **Definition 10.** [17] For $z \in \{0,1\}$ we say that $f : \{0,1\}^n \to \{0,1\}$ is in z-good form, if (1) $f(z^n) = z$ and (2) $bs(f) = bs(f, z^n)$

▶ **Lemma 11.** *[17] If both f and g are in 0-good form, or if both f and g are in 1-good form, then $bs(f \circ g) \geq bs(f)bs(g)$.*

2.1 Previous Constructions with Quadratic Separation

All previous constructions that achieve quadratic separation between sensitivity and block sensitivity - with the exception of Chakraborty's functions [7] - were based on the following "OR-composition Lemma" first used by Rubinstein [15].

▶ **Lemma 12.** *[15] For any function $g \colon \{0,1\}^m \to \{0,1\}$, we have:*
- $s_0(OR_n \circ g) = ns_0(g)$
- $bs_0(OR_n \circ g) = nbs_0(g)$
- $s_1(OR_n \circ g) = s_1(g)$
- $bs_1(OR_n \circ g) = bs_1(g)$

The quadratic separations of [15, 19, 5, 10] are based on using this lemma and considering functions of the form $f = OR_n \circ g$ for appropriately chosen inner functions g.

Next we briefly describe the previous constructions of functions with quadratic separation.

1. Rubinstein's function [15] Define $g \colon \{0,1\}^{2m} \to \{0,1\}$ as:
   $g(x) = 1$ iff $x_{2j\ -1} = x_{2j} = 1$ for some $j \in [m]$ and $x_i = 0$ for $i \neq 2j-1, 2j$.
   This gives $s_0(g) = 1$, $s_1(g) = bs_1(g) = 2m$, $bs_0(g) = m$.
   Let $f = OR_{2m} \circ g$. Then $s_0(f) = s_1(f) = bs_1(f) = 2m$, $bs_0(f) = 2m^2$, giving $bs(f) = \frac{1}{2}\ s(f)^2$.
2. Virza's function [19] Define $g \colon \{0,1\}^{2m+1} \to \{0,1\}$ as:
   $g(x) = 1$ iff one of the following holds:

1) $\exists j \in [m]$ such that $(x_{2j-1} = x_{2j} = 1)$ and $(x_i = 0 \ \ \forall\, i \neq 2j-1, 2j)$.
2) $(x_{2m+1} = 1)$ and $(x_i = 0 \,\forall\, i \neq 2m+1)$.
This gives $s_0(g) = 1$, $s_1(g) = bs_1(g) = 2m+1$, $bs_0(g) = m+1$.
Let $f = OR_{2m+1} \circ g$. Then $s_0(f) = s_1(f) = bs_1(f) = 2m+1$, $bs_0(f) = (m+1)(2m+1)$.
Therefore, $bs(f) = \frac{1}{2}s(f)^2 + \frac{1}{2}s(f)$.

3. Ambainis and Sun's function [5] Define $g\colon \{0,1\}^{2(2m+1)} \to \{0,1\}$ as:
$g(x) = 1$ iff $\exists j \in [2m+1]$ such that:
1) $x_{2j-1} = x_{2j} = 1$, and
2) For all $i \in [m]$, $x_{2j+2i} = x_{2j-2i} = x_{2j-2i-1} = 0$.
Here, the index of x is taken modulo $(2(2m+1))$ i.e. we index x as if it were laid around a circle.
This gives $s_0(g) = 1$, $s_1(g) = bs_1(g) = 3m+2$, $bs_0(g) = 2m+1$.
Let $f = OR_{3m+2} \circ g$. Then $s_0(f) = s_1(f) = bs_1(f) = 3m+2$, $bs_0(f) = (3m+2)(2m+1)$.
Therefore, $bs(f) = \frac{2}{3}s(f)^2 - \frac{1}{3}s(f)$.

4. Function based on Hamming Code [10]:
Consider the hamming code on $m = 2^r - 1$ bits.
Define $g\colon \{0,1\}^m \to \{0,1\}$ as:
$g(x) = 1$ iff x is a codeword of the hamming code on m bits.
This gives $s_0(g) = 1$, $s_1(g) = bs_1(g) = m$, $bs_0(g) = \frac{m+1}{2}$.
Let $f = OR_m \circ g$. Then, $s_0(f) = s_1(f) = bs_1(f) = m$, $bs_0(f) = \frac{m(m+1)}{2}$. Thus $bs(f) = \frac{1}{2}s(f)^2 + \frac{1}{2}s(f)$.

   Finally, we describe a construction by Chakraborty that does not involve function composition. Another similar construction appeared in [7].

5. Chakraborty's function [7] For integers k, m such that $2 < k < m$ and $2k \mid m$, the function $g_k\colon \{0,1\}^m \to \{0,1\}$ is defined as follows.
For $x = (x_0, \ldots x_{m-1})$, $g_k(x) = 1$ iff $\exists\, i \in \{0, \ldots m-1\}$ such that $x_i = x_{i+1(\mod m)} = 1$ and $x_j = 0$ for all $j \in \{i+2(\mod m), \ldots, i+k-1(\mod m)\}$.
Then, $s_0(g_k) = \frac{2m}{k}$, $s_1(g_k) = k$, $bs_0(g_k) = \frac{m}{2}$ and $bs_1(g_k) = k$.

   Therefore, setting $k = \sqrt{2m}$ gives $s(g_{\sqrt{2m}}) = \sqrt{2m}$ and $bs(g_{\sqrt{2m}}) = \frac{m}{2}$. So we have $bs(g_{\sqrt{2m}}) = \frac{1}{4}s(g_{\sqrt{2m}})^2$.

3 New Building Blocks for Quadratic Separation

Here we define several new functions that we will use as inner or outer functions in various function compositions to obtain quadratic separations.

3.1 A General Framework Based on Certificates

For an odd integer m, consider a set of partial assignments $C = \{c_1, c_2, \ldots c_m\}$ on a set of variables $\mathcal{X}$ with $|\mathcal{X}| = 2m$. We say that the set of partial assignments C is good if it satisfies the following 2 properties:

(a) The distance between any two partial assignments $c_i, c_j \in C$ is at least 3.
(b) Each partial assignment c_i has exactly 2 bits set to 1, $\frac{3}{2}(m-1)$ bits set to 0 and the remaining bits free.

Consider the function g_C defined by a good set of partial assignments. We prove the following lemma for such a function g_C:

▶ **Lemma 13.** *For an odd integer m, and any function $g_C\colon \{0,1\}^{2m} \to \{0,1\}$ defined by a good set of partial assignments C, we have:*
(1) $s_0(g_C) = 1$
(2) $s_1(g_C) = \frac{3m+1}{2}$
(3) $bs_0(g_C) \geq m$

Proof. We first note that the set of partial assignments C also forms a set of 1-certificates for g_C such that every 1-input agrees with exactly one partial assignment from C.

1. $s_0(g_C) = 1$. This follows from property (a) of a good set of partial assignments since: for any 0-input x, there is at most one certificate $c_i \in C$ such that x is at a distance 1 from it.
2. $s_1(g_C) = \frac{3m+1}{2}$. This follows from property (a): Consider a 1-input x agreeing with c_i. The bits fixed by c_i form exactly the set of sensitive bits for f on x, since any two certificates in C are at a distance of at least 3 from each other.
3. $bs_0(g_C) \geq m$. Follows from properties (a),(b): Consider the input 0^{2m} which is a 0-input (due to property (b)).
   Recall that any two certificates c_i, c_j must be at a distance at least 3 from each other. But since each certificate only sets exactly 2 bits to 1, this implies that the bits set to 1 by c_i must be disjoint from the bits set to 1 by c_j, for any $c_i, c_j \in C$.
   Therefore, for the 0-input 0^{2m}, the pair of bits set to 1 by a certificate c_i gives a sensitive block for every $i \in [m]$. All these blocks are mutually disjoint and therefore $bs_0(g_C) \geq m$. ◀

▶ **Theorem 14.** *Consider any function $g_C\colon \{0,1\}^{2m} \to \{0,1\}$ defined by a good set of partial assignments C, for an odd integer m. Then the function $f = OR_{\frac{3m+1}{2}} \circ g_C$ has:*

$$bs(f) = \frac{2}{3}s(f)^2 - \frac{1}{3}s(f).$$

We note that the inner function defined by Ambainis and Sun [5] can be shown to fit into this framework.

We will use Lemma 13 to analyze the functions defined in subsection 3.3.

3.2 Using Finite Field Multiplication

In this subsection, we give constructions of families of functions based on Finite Field Multiplication, which achieve quadratic separation between block sensitivity and sensitivity.

Fix an irreducible polynomial p of degree m in $\mathbb{F}_2[z]$ and consider the representation of the elements of $\mathbb{F}_{2^m}$ as univariate polynomials modulo p.
For $a \in \{0,1\}^m$, we interpret $a = (a_0, \ldots a_{m-1})$ as an element of $\mathbb{F}_{2^m}$ under this representation.

▶ **Definition 15** (Function based on Finite Field Multiplication)**.** The function $g_{FF}\colon \{0,1\}^m \times \{0,1\}^m \to \{0,1\}$ is defined as follows:
$g_{FF}(a,b) = 1$ iff $a \cdot b = c$, where $c \in \mathbb{F}_{2^m}$ is the element represented as $(0, \ldots, 0, 1)$ and multiplication is over the field $\mathbb{F}_{2^m}$.

We prove the following lemma listing the values of sensitivity and block sensitivity for the function g_{FF}:

▶ **Lemma 16.** *For the function* $g_{FF}\colon \{0,1\}^m \times \{0,1\}^m \to \{0,1\}$*, we have:*

- $s_0(g_{FF}) \leq 2$
- $bs_0(g_{FF}) \geq m$
- $s_1(g_{FF}) = 2m$
- $bs_1(g_{FF}) = 2m$

Proof.

- $s_0(g_{FF}) \leq 2$:
  For any non-zero $a \in \mathbb{F}_{2^m}$, there exists a unique $b \in \mathbb{F}_{2^m}$ such that $a \cdot b = (0, \ldots, 0, 1)$ i.e. $g_{FF}(a, b) = 1$. Therefore, for any input $(a, b) \in g_{FF}^{-1}(0)$, at most 1 bit j of a may be flipped to get $a^j \cdot b = (0, 0 \ldots 1)$ i.e. a has at most 1 sensitive bit. Similarly, at most 1 bit of b may be sensitive.
- $s_1(g_{FF}) = 2m$
  Consider any input $(a, b) \in g_{FF}^{-1}(1)$. Flipping any bit of a or b changes the value of the product $a \cdot b$. Therefore every bit of (a, b) is sensitive, giving $s_1(g_{FF}) = 2m$.
- $bs_0(g_{FF}) \geq m$
  Consider the 0-input $a = (0, \ldots 0), b = (0, \ldots 0)$.
  For each $j \in \{0, \ldots m-1\}$, we can flip the pair of bits (a_j, b_{m-1-j}), so that their product becomes $c = (0, \ldots, 0, 1)$. This gives m disjoint sensitive blocks.
- $bs_1(g_{FF}) = 2m$
  This follows since: $2m \geq bs_1(g_{FF}) \geq s_1(g_{FF}) = 2m$ ◀

The following theorem follows from Lemma 16 and the OR-composition Lemma.

▶ **Theorem 17.** *The function* $f = OR_m \circ g_{FF}$ *has:*

$$bs(f) \geq \frac{1}{4} s(f)^2$$

We now modify the function g_{FF} to improve the constant of separation from $\frac{1}{4}$ to $\frac{1}{2}$.

▶ **Definition 18.** The function $g^*_{FF}\colon \{0,1\}^m \times \{0,1\}^m \to \{0,1\}$ is defined as follows: $g^*_{FF}(a, b) = 1$ iff the following two conditions hold:

1. $a \cdot b = c$, where $c \in \mathbb{F}_{2^m}$ is the element represented as $(0, \ldots, 0, 1)$ and multiplication is over the field $\mathbb{F}_{2^m}$
2. $a_0 \oplus a_1 \ldots \oplus a_{m-1} = 1$

▶ **Lemma 19.** *For the function* $g^*_{FF}\colon \{0,1\}^m \times \{0,1\}^m \to \{0,1\}$*, we have:*

- $s_0(g^*_{FF}) = 1$
- $bs_0(g^*_{FF}) \geq m$
- $s_1(g^*_{FF}) = 2m$
- $bs_1(g^*_{FF}) = 2m$

Proof. Note that Conditions 1. and 2. of Definition 18 both have to hold for 1 inputs, and at least one is violated for 0 inputs.

- $s_0(g^*_{FF}) = 1$
  For a 0-input (a, b) which satisfies condition 1., flipping any bit of a or b changes the product $a \cdot b$ and condition 1. is no longer satisfied. Therefore, such a 0-input has no sensitive bit.
  For any 0-input (a, b) which leaves condition 1. unsatisfied, both a and b can have at most one sensitive bit each as observed in the proof of Lemma 16.
  We further note that for any given 0-input (a, b), only one of a or b can have a sensitive bit because condition 2 has to hold for 1-inputs. Therefore, $s_0(g^*_{FF}) = 1$.

- $s_1(g^*_{FF}) = 2m$
  Consider any 1-input (a, b). Flipping any bit of a or b changes the value of the product $a \cdot b$ and condition 1. is no longer satisfied. Therefore every bit of (a, b) is sensitive, giving $s_1(g^*_{FF}) = 2m$.
- $bs_0(g^*_{FF}) \geq m$
  Consider the 0-input $a = (0, \dots 0), b = (0, \dots 0)$.
  For each $j \in \{0, \dots m-1\}$, we can flip the pair of bits (a_j, b_{m-1-j}), so that their product becomes $c = (0, 0 \dots 1)$ to satisfy the first condition. Since a^j has exactly one 1, the second condition is satisfied as well, and $g^*_{FF}(a^j, b^{m-1-j}) = 1$. This gives m disjoint sensitive blocks and therefore, $bs_0(g^*_{FF}) \geq m$.
- $bs_1(g^*_{FF}) = 2m$
  This follows since: $2m \geq bs_1(g^*_{FF}) \geq s_1(g^*_{FF}) = 2m$. ◀

Using Lemma 19 and the OR-composition Lemma gives the following theorem.

▶ **Theorem 20.** *The function $f = OR_{2m} \circ g^*_{FF}$ has:*

$$bs(f) \geq \frac{1}{2} s(f)^2.$$

▶ Remark. We could replace $c = (0, \dots, 0, 1)$ in the above definitions by other field elements and still achieve quadratic separations. In fact using any $c \in \mathbb{F}_{2^m}$, we would get $s_0 \leq 2$ and $s_1 = 2m$ for the inner function. However, we need to choose c carefully to guarantee that bs_0 of the inner function is large enough.

3.3 Using Polynomial Multiplication

We now describe another family of functions similar in essence to the one involving finite field multiplication, but easier to analyze.

Here we consider polynomials over the Integers. For $a \in \{0,1\}^m$, we interpret the bits of $a = (a_0, \dots a_{m-1})$ as the coefficients of a univariate polynomial p_a that is $p_a(z) = a_0 + a_1 z + \dots a_{m-1} z^{m-1}$.

▶ **Definition 21** (Function based on Polynomial Multiplication)**.** The function $g_{poly} \colon \{0,1\}^m \times \{0,1\}^m \to \{0,1\}$ is defined as follows:
$g_{poly}(a, b) = 1$ iff $p_a(z) \cdot p_b(z)$ has a non-zero coefficient for z^{m-1} and has coefficient 0 for z^j for all $j < m-1$.

It is convenient to use the following equivalent definition.

▶ **Definition 22** (Alternative definition)**.** Consider the set of partial assignments $C = \{c_0, c_1 \dots c_{m-1}\}$ on variables (a, b) where $a = (a_0, \dots a_{m-1})$ and $b = (b_0, \dots b_{m-1})$ defined as below:
For every $i \in \{0, \dots m-1\}$,

$$c_i(a_j) = \begin{cases} 1, & \text{if } \ j = i \\ 0, & \text{if } j < i \\ \star, & \text{if } j > i \end{cases}$$

$$c_i(b_j) = \begin{cases} 1, & \text{if } \ j = m-1-i \\ 0, & \text{if } j < m-1-i \\ \star, & \text{if } j > m-1-i \end{cases}$$

Now the function g_{poly} is the function defined by the set of partial assignments C i.e. g_C.

We now analyze this function for its sensitivity and block sensitivity:

▶ **Lemma 23.** *For $g_{poly}\colon \{0,1\}^m \times \{0,1\}^m \to \{0,1\}$, we have:*

- $s_0(g_{poly}) = 2$
- $bs_0(g_{poly}) = m$
- $s_1(g_{poly}) = m+1$
- $bs_1(g_{poly}) = m+1$

Proof.

1. $\boldsymbol{s_0(g_{poly}) = 2}$:
   $s_0(g_{poly}) \leq 2$: Let (a,b) be any 0-input of g_{poly}. Let $i \in \{0,\dots m-1\}$ be the smallest index such that $a_i = 1$ (if it exists - let $i = m$ if it does not) and j be the smallest index such that $b_j = 1$(if it exists - let $j = m$ otherwise). Then, we have two cases:
   Case 1: $i + j > m - 1$. In this case, the only bits which can be flipped to change the value of g_{poly} from 0 to 1 are a_{m-1-i} (unless $i = m$) and b_{m-1-j} (unless $j = m$).
   Case 2: $i + j < m - 1$. Now, the only way to flip a bit and possibly change the value of g_{poly} to 1 is by flipping the bits a_i or b_j.
   $s_0(g_{poly}) \geq 2$: The following 0-input (a,b) achieves $s_0(g_{poly},(a,b)) = 2$:
   Let $a_{m-1} = 1$, $a_i = 0\ \forall i < (m-1)$.
   Similarly, $b_{m-1} = 1$, $b_i = 0\ \forall i < (m-1)$.
   Notice that (a,b) has 2 sensitive bits: a_0 and b_0.
2. $\boldsymbol{s_1(g_{poly}) = m+1}$:
   First, we observe from the alternative definition of g_{poly} that every 1-input of g_{poly} agrees with a certificate c_i from the set C. Therefore, $C_1(g_{poly}) \leq (m+1)$.
   Therefore, $s_1(g_{poly}) \leq (m+1)$.
   Also, note that every two certificates of C are at a distance of at least 2 from each other. Therefore, for any 1-input (a,b) of g_{poly}, each of the $m+1$ bits where it agrees with $c_i \in C$ is sensitive. So $s_1(g_{poly}) = m+1$.
3. $\boldsymbol{bs_0(g_{poly}) = m}$:
   We first prove $bs_0(g_{poly}) \geq m$. Consider the 0-input with $a_i = b_i = 0\ \forall\ i \in \{0,\dots m-1\}$. We can flip the pair of bits a_i, b_{m-1-i} for $i \in \{0,\dots m-1\}$ so that the function changes value from 0 to 1. Therefore, $bs_0(g_{poly}) \geq m$.
   Next we prove $bs_0(g_{poly}) \leq m$. Since $s_0(g_{poly}) = 2$, any 0-input other than the all-0 input can have only at most 2 blocks of size 1 each and all the other blocks must have size at least 2. Therefore, $bs_0(g_{poly}) \leq 2 + (m-2) = m$.
4. $\boldsymbol{bs_1(g_{poly}) = m+1}$:
   As observed before, $C_1(g_{poly}) \leq (m+1)$. Also, $s_1(g_{poly}) = m+1$.
   Since $s_1(g_{poly}) \leq bs_1(g_{poly}) \leq C_1(g_{poly})$, we have $bs_1(g_{poly}) = m+1$. ◀

Lemma 23 and the OR-composition Lemma imply the following theorem.

▶ **Theorem 24.** *Consider $g_{poly}\colon \{0,1\}^m \times \{0,1\}^m \to \{0,1\}$ for any odd integer m. Then the function $f = OR_{\frac{m+1}{2}} \circ g_{poly}$ has:*

$$bs(f) = \frac{1}{2}s(f)^2 - \frac{1}{2}s(f)$$

We modify the above function to improve the constant of separation from $\frac{1}{2}$ to $\frac{2}{3}$.

▶ **Definition 25.** The function $g^*_{poly}\colon \{0,1\}^m \times \{0,1\}^m \to \{0,1\}$, is defined as: $g^*_{poly}(a,b) = 1$ iff all the following conditions are met:

1. $p_a(z)\cdot p_b(z)$ has a non-zero coefficient for z^{m-1} and has coefficient 0 for z^j for all $j < m-1$
2. If j is the smallest index such that $a_j = 1$, then
   $a_i = 0$ for all i such that $i > j$ and $i \oplus j = 1$
3. If k is the smallest index such that $b_k = 1$, then
   $b_i = 0$ for all i such that $i > k$ and $i \oplus k = 0$

It is again helpful to consider an equivalent definition based on certificates.

▶ **Definition 26** (Alternative definition)**.** Consider the set of partial assignments $C' = \{c'_0, c'_1, \dots c'_{m-1}\}$ on variables (a,b) where $a = (a_0, \dots a_{m-1})$ and $b = (b_0, \dots b_{m-1})$ defined as below:
For every $i \in \{0, \dots m-1\}$,

$$c'_i(a_j) = \begin{cases} 1, & \text{if } \ j = i \\ 0, & \text{if } j < i \\ 0, & \text{if } j > i \text{ and } i \oplus j = 1 \\ \star, & \text{if } j > i \text{ and } i \oplus j = 0 \end{cases}$$

$$c'_i(b_j) = \begin{cases} 1, & \text{if } \ j = m-1-i \\ 0, & \text{if } j < m-1-i \\ 0, & \text{if } j > m-1-i \text{ and } (m-1-i) \oplus j = 0 \\ \star, & \text{if } j > m-1-i \text{ and } (m-1-i) \oplus j = 1 \end{cases}$$

Now the function g^*_{poly} is the function defined by the set of partial assignments C' i.e. $g_{C'}$.

▶ **Lemma 27.** *Consider $g^*_{poly}\colon \{0,1\}^m \times \{0,1\}^m \to \{0,1\}$ for any odd integer m. Then*

- $s_0(g^*_{poly}) = 1$
- $bs_0(g^*_{poly}) \geq m$
- $s_1(g^*_{poly}) = \frac{3m+1}{2}$
- $bs_1(g^*_{poly}) = \frac{3m+1}{2}$

Proof. It is clear from the alternative definition of g^*_{poly} that it is defined by a set of good assignments.Therefore, we can use Lemma 13 to prove that:

- $s_0(g^*_{poly}) = 1$
- $bs_0(g^*_{poly}) \geq m$
- $s_1(g^*_{poly}) = \frac{3m+1}{2}$

Furthermore, from the alternative definition of g^*_{poly}, every 1-input has a certificate of size at most $\frac{3m+1}{2}$.
Therefore, $bs_1(g^*_{poly}) \leq C_1(g^*_{poly}) \leq \frac{3m+1}{2}$.
Also, $bs_1(g^*_{poly}) \geq s_1(g^*_{poly}) = \frac{3m+1}{2}$.
Therefore $bs_1(g^*_{poly}) = \frac{3m+1}{2}$. ◀

The following theorem follows from Lemma 27 and the OR-composition Lemma.

▶ **Theorem 28.** *Consider $g^*_{poly}\colon \{0,1\}^m \times \{0,1\}^m \to \{0,1\}$ for any odd integer m. Then the function $f = OR_{\frac{3m+1}{2}} \circ g^*_{poly}$ has:*

$$bs(f) \geq \frac{2}{3}s(f)^2 - \frac{1}{3}s(f).$$

Note that this bound matches the current best quadratic separation of [5].

4 Additional Properties of Function Composition

As we noted in Section 2, properties of function composition have been formally studied in [17, 9] in the context of separating sensitivity and block sensitivity. Here we take a closer look at the effect of function composition on the measures 0-sensitivity, 1-sensitivity, 0-block sensitivity and 1-block sensitivity. These properties provide the tools we need to obtain quadratic separation of both 0-block sensitivity and 1-block sensitivity from sensitivity.

First we define measures to quantify the number of sensitive bits for f on x which are equal to 0 and those that are equal to 1 in x.

▶ **Definition 29.** For a function $f\colon \{0,1\}^n \to \{0,1\}$ and input $x \in \{0,1\}^n$, we define:
$\sigma_1(f,x) = |\{i | x_i = 1 \text{ AND } f(x) \neq f(x^i)\}|$,
$\sigma_0(f,x) = |\{i | x_i = 0 \text{ AND } f(x) \neq f(x^i)\}|$.

We will use the following notation. We index the bits of the input $y \in \{0,1\}^{mn}$ to $f \circ g$ as $y = (y_{11}, y_{12}, \dots y_{1m}, y_{21}, \dots y_{2m}, \dots y_{n1}, \dots y_{nm})$.
We denote by y_i the i-th group of m bits of y, that is $y_i = (y_{i1}, y_{i2}, \dots y_{im})$.

▶ **Lemma 30.** *For any functions $f\colon \{0,1\}^n \to \{0,1\}$ and $g\colon \{0,1\}^m \to \{0,1\}$, we have:*

$$s_0(f \circ g) = \max_{x \in f^{-1}(0)} \{\sigma_0(f,x)s_0(g) + \sigma_1(f,x)s_1(g)\},$$

$$s_1(f \circ g) = \max_{x \in f^{-1}(1)} \{\sigma_0(f,x)s_0(g) + \sigma_1(f,x)s_1(g)\}.$$

Proof. We first prove the first equation. The second equation has an analogous proof.

LHS $\geq$ RHS. Consider the input $a \in f^{-1}(0)$ for which $(\sigma_0(f,a)s_0(g) + \sigma_1(f,a)s_1(g))$ is maximized. Note that $s_0(g), s_1(g)$ don't change for different choices of $a \in f^{-1}(0)$. Now, consider an input $y \in \{0,1\}^{mn}$ such that, $a = (g(y_1), \dots g(y_n))$ and for each $i \in [n]$, if $a_i = g(y_i) = 0$, then $s(g, y_i) = s_0(g)$ and if $a_i = g(y_i) = 1$, then $s(g, y_i) = s_1(g)$. So if $a_i = 0$, we choose as y_i a 0-input of g which achieves the 0-sensitivity of g, and similarly, if $a_i = 1$, we choose as y_i a 1-input of g which achieves the 1-sensitivity of g.
Since $a \in f^{-1}(0)$, y must be a 0-input of $f \circ g$. Therefore, we have

$$s_0(f \circ g) \geq s(f \circ g, y) \geq \sigma_0(f,a)s_0(g) + \sigma_1(f,a)s_1(g).$$

LHS $\leq$ RHS. Consider the input $y \in \{0,1\}^{mn}$ which achieves the 0-sensitivity of $f \circ g$ i.e. $s_0(f \circ g) = s(f \circ g, y)$. Let $g(y_1) = x_1$, $g(y_2) = x_2$ and so on, and let $x = (x_1, x_2 \dots x_n)$.
Consider the expression $(\sigma_0(f,x)s_0(g) + \sigma_1(f,x)s_1(g))$. Now, if a bit y_{ij} of y is sensitive for $f \circ g$, then the bit $x_i = g(y_i)$ must be a sensitive bit for f on x.
Now, consider the set $\mathcal{X}_0$ of indices $i \in [n]$ constructed the following way: i is included in $\mathcal{X}_0$ iff $x_i = g(y_i) = 0$ and there is a bit y_{ij} sensitive for $f \circ g$ on y.
Similarly, we define the set $\mathcal{X}_1$ of indices $i \in [n]$ constructed the following way: i is included in $\mathcal{X}_1$ iff $x_i = g(y_i) = 1$ and there is a bit y_{ij} sensitive for $f \circ g$ on y.
Note that for every $i \in \mathcal{X}_0$, the bit x_i is a 0-bit of x and f is sensitive to the i-th bit on x. So $|\mathcal{X}_0| \leq \sigma_0(f,x)$.
Similarly $|\mathcal{X}_1| \leq \sigma_1(f,x)$.
Now,

$$\begin{aligned} s(f \circ g, y) &= \sum_{i \in \mathcal{X}_0} s(g, y_i) + \sum_{j \in \mathcal{X}_1} s(g, y_j) \\ &\leq \sum_{i \in \mathcal{X}_0} s_0(g) + \sum_{j \in \mathcal{X}_1} s_1(g) \\ &\leq \sigma_0(f,x)s_0(g) + \sigma_1(f,x)s_1(g). \end{aligned}$$

Therefore,

$$s_0(f \circ g) = s(f \circ g, y) \leq \sigma_0(f,x)s_0(g) + \sigma_1(f,x)s_1(g) \leq RHS. \quad \blacktriangleleft$$

To simplify the equations of Lemma 30 (at the cost of being less precise), we define

- $\sigma_0^0(f) \coloneqq \max_{x \in f^{-1}(0)} \sigma_0(f,x)$
- $\sigma_0^1(f) \coloneqq \max_{x \in f^{-1}(1)} \sigma_0(f,x)$
- $\sigma_1^0(f) \coloneqq \max_{x \in f^{-1}(0)} \sigma_1(f,x)$
- $\sigma_1^1(f) \coloneqq \max_{x \in f^{-1}(1)} \sigma_1(f,x)$

Finally, we define:

$\sigma_0(f) \coloneqq \max\{\sigma_0^0(f), \sigma_0^1(f)\}$

$\sigma_1(f) \coloneqq \max\{\sigma_1^0(f), \sigma_1^1(f)\}$

We can now use Lemma 30 to get the following bounds:

▶ **Corollary 31.** *For any functions $f\colon \{0,1\}^n \to \{0,1\}$ and $g\colon \{0,1\}^m \to \{0,1\}$*

$s_0(f \circ g) \leq \sigma_0^0(f)s_0(g) + \sigma_1^0(f)s_1(g)$

$s_1(f \circ g) \leq \sigma_0^1(f)s_0(g) + \sigma_1^1(f)s_1(g)$

Note that the equalities in Lemma 30 change to inequalities in Corollary 31, since the max of $\sigma_0(f,x)$ and $\sigma_1(f,x)$ may be achieved on different inputs among $x \in f^{-1}(0)$ (or among $x \in f^{-1}(1)$, respectively).
We now state some simple observations for these measures.

▶ **Lemma 32.** *For any function $f\colon \{0,1\}^n \to \{0,1\}$, and any input x, we have:*

1. $s(f,x) = \sigma_0(f,x) + \sigma_1(f,x)$
2. $\sigma_0^0(f) \leq s_0(f)$
3. $\sigma_1^0(f) \leq s_0(f)$
4. $\sigma_0^0(f) + \sigma_1^0(f) \geq s_0(f)$
5. $\sigma_0^1(f) \leq s_1(f)$
6. $\sigma_1^1(f) \leq s_1(f)$
7. $\sigma_0^1(f) + \sigma_1^1(f) \geq s_1(f)$

The proof of Lemma 32 is straightforward from the definitions.

Now we present an observation about these measures for monotone functions.

▶ **Lemma 33.** *For any monotone function $f\colon \{0,1\}^n \to \{0,1\}$, we have:*

- $\sigma_0^1(f) = 0$
- $\sigma_1^0(f) = 0$

The proof follows from the definition of monotone functions.

We now consider the effects of function composition on 0- block sensitivity and 1-block sensitivity.

▶ **Lemma 34.** *For any functions $f\colon \{0,1\}^n \to \{0,1\}$ and $g\colon \{0,1\}^m \to \{0,1\}$, we have:*

$bs_0(f \circ g) \geq bs_0(f) \cdot \min\{bs_0(g), bs_1(g)\}$,

$bs_1(f \circ g) \geq bs_1(f) \cdot \min\{bs_0(g), bs_1(g)\}$.

Proof. Consider $x \in \{0,1\}^n$ such that $f(x) = 0$ and $bs_0(f) = bs(f,x)$.
Now, consider input $y \in \{0,1\}^{mn}$ such that, $x = (g(y_1), \ldots g(y_n))$ and for each $i \in [n]$, if $x_i = g(y_i) = 0$, then $bs(g,y_i) = bs_0(g)$ and if $x_i = g(y_i) = 1$, then $bs(g,y_i) = bs_1(g)$. So if $x_i = 0$, we choose as y_i a 0-input of g on which its 0-block sensitivity is achieved, and similarly, if $x_i = 1$, we choose as y_i a 1-input of g on which its 1-block sensitivity is achieved.
Now, we claim that $bs_0(f \circ g, y) \geq bs_0(f) \min\{bs_0(g), bs_1(g)\}$. To see this, let $\rho_1, \rho_2, \ldots, \rho_k$ be the disjoint sensitive blocks for f on x where $k = bs(f,x)$. For each of these sensitive blocks, there are at least $\min\{bs_0(g), bs_1(g)\}$ disjoint blocks of y such that flipping any of them changes the value of $f \circ g$. This gives at least $bs_0(f) \cdot \min\{bs_0(g), bs_1(g)\}$ disjoint sensitive blocks for $f \circ g$ on the input y, and the first equation follows.
The second equation can be proved in an analogous way. ◀

We get a stronger form of Lemma 34 if f satisfies some additional conditions.

▶ **Lemma 35.** *For any functions $f\colon \{0,1\}^n \to \{0,1\}$ and $g\colon \{0,1\}^m \to \{0,1\}$ if f satisfies (1) $f(0^n) = 0$, (2) $bs_0(f) = bs(f,0^n)$, then $bs_0(f \circ g) \geq bs_0(f) \cdot bs_0(g)$ and if f satisfies (1) $f(1^n) = 1$, (2) $bs_1(f) = bs(f,1^n)$, then $bs_1(f \circ g) \geq bs_1(f) \cdot bs_1(g)$*

Proof. Consider input $y \in \{0,1\}^{mn}$ such that, $0^n = (g(y_1), \ldots g(y_n))$ and $bs(g,y_i) = bs_0(g)$ for each $i \in [n]$.

Now, we claim that $bs(f \circ g, y) \geq bs_0(f) bs_0(g)$. To see this, let $\rho_1, \rho_2, \ldots, \rho_k$ be the disjoint sensitive blocks for f on input 0^n, where $k = bs(f,0^n)$. For each of these $k = bs(f,0^n)$ sensitive blocks, there are $bs_0(g)$ disjoint blocks of y that we can flip and change the value of $f \circ g$.
This gives $bs_0(f) \cdot bs_0(g)$ disjoint sensitive blocks for $f \circ g$ on the input y.
The second inequality can be proved analogously. ◀

Comparing the statement of Lemma 35 with Lemma 11 of Tal [17] we note that in the context of 0-block sensitivity and 1-block sensitivity it is enough to require an additional condition for the outer function. On the other hand the condition on the inner function in Lemma 11 of Tal [17] is necessary as illustrated by considering $f = OR_n$ and $g = AND_n$.

Note that the conditions we require are similar to, but slightly different from being in z-good form: It follows from the definition, that if f is in z-good form, then $bs(f) = bs_z(f)$. Our conditions do not require that $bs(f) = bs_z(f)$ for a specific z.

5 Quadratic Separation of both $bs_0(f)$ and $bs_1(f)$ from $s(f)$

We obtain constructions of functions with quadratic separation of both 0-block sensitivity and 1-block sensitivity from sensitivity by considering various compositions of our new building blocks as well as some of the inner functions used in previous quadratic separations.

▶ **Theorem 36.** *Consider $g_{poly}\colon \{0,1\}^m \times \{0,1\}^m \to \{0,1\}$.
Let $f\colon \{0,1\}^{4m^2} \to \{0,1\}$ be defined as $f = g_{poly} \circ g_{poly}$.
Then, we have:*

- $s_0(f) = 2(m-1)$
- $bs_0(f) \geq m^2$
- $s_1(f) = 4(m-1)$
- $bs_1(f) \geq m(m+1)$

Therefore, we have:

$$\min\{bs_0(f), bs_1(f)\} = \Omega(s(f)^2).$$

Proof. In this proof, we refer to g_{poly} by g, and we use the notation $bs_{min}(f) = \min\{bs_0(f), bs_1(f)\}$.

We first prove the following claims about σ-values for g:

▶ **Claim 37.** *For any input $x \in g^{-1}(0)$ exactly one of the following must be true:*
- $\sigma_0(g,x) = s(g,x)$ *and* $\sigma_1(g,x) = 0$
- $\sigma_1(g,x) = s(g,x)$ *and* $\sigma_0(g,x) = 0$

Proof of Claim. For any 0-input $x = (a,b)$ of g, (1) of Lemma 32 states that:
$\sigma_0(g,x) + \sigma_1(g,x) = s(g,x)$.
As in the definition of g_{poly}, consider the polynomials $p_a(z), p_b(z)$. If the lowest degree monomial of $p_a(z)p_b(z)$ with a non-zero coefficient is z^t then, we have 2 cases:
Case 1: $t < m-1$. In this case, no 0-bit of a or b can be sensitive. Therefore, $\sigma_0(g,x) = 0$ and $\sigma_1(g,x) = s(g,x)$.
Case 2: $t > m-1$. In this case, no 1-bit of a or b can be sensitive. Therefore, $\sigma_1(g,x) = 0$ and $\sigma_0(g,x) = s(g,x)$. ◀

▶ **Claim 38.** *For an input $x \in g^{-1}(1)$,*
- $\sigma_0(g,x) = m-1$
- $\sigma_1(g,x) = 2$

Proof of Claim. Recall the alternative definition based on certificates. Any 1-input x of g belongs to a unique subcube given by a certificate $c_i \in C$. Since the subcubes corresponding to different certificates in C are disjoint and at a distance of at least 2 from each other, every bit of x that is fixed by c_i is sensitive.
Since each certificate fixes exactly 2 bits to 1 and $(m-1)$ bits to 0, we have $\sigma_0(g,x) = (m-1)$ and $\sigma_1(g,x) = 2$. ◀

We can now use Lemma 30 to compute the sensitivity of f:
$s_0(f) = 2s_1(g) = 2(m-1)$.
$s_1(f) = (m-1)\cdot 2 + 2\cdot(m-1) = 4(m-1)$
Since $g(0^{2m}) = 0$ and $bs_0(g) = bs(g, 0^{2m})$, we can use Lemma 35 to get:
$bs_0(f) \geq bs_0(g)^2 = m^2$.
We can use Lemma 34 to get:
$bs_1(f) \geq bs_1(g)\cdot\min\{bs_0(g), bs_1(g)\} = m(m+1)$.
Therefore, we have $bs(f) \geq \frac{s(f)^2}{16}$ and $bs_{min}(f) = \Omega(s(f)^2)$. ◀

We prove the following general theorem:

▶ **Theorem 39.** *For functions $f\colon \{0,1\}^n \to \{0,1\}$ and $g\colon \{0,1\}^n \to \{0,1\}$ such that the following conditions hold:*
1. *$\sigma_1(f) = c_1$, where c_1 is some fixed constant*
2. *$s_0(g) = c_2$, where c_2 is some fixed constant*
3. *$bs_0(f), bs_1(f), bs_0(g), bs_1(g) = \theta(n)$*

We have,

$$\min\{bs_0(f\circ g), bs_1(f\circ g)\} = \Omega(s(f\circ g)^2).$$

The proof is straightforward from Corollary 31 and Lemma 34.

This theorem allows us to use various compositions of our new building blocks and some of the inner functions of previous constructions to obtain other functions with both 0-block sensitivity and 1-block sensitivity quadratically larger than sensitivity.

In particular, let f, g be any two functions from the following list of functions: Rubinstein's inner function [15], Virza's inner function [19], Ambainis and Sun's inner function [5], g_{poly}, g^*_{poly}. In addition, we can also let g be g_{FF}, g^*_{FF}, or the inner function of the function based on Hamming Code [10]. Then, $bs_0(f \circ g)$ and $bs_1(f \circ g)$ are both quadratically larger than $s(f \circ g)$.

References

1. Andris Ambainis. Polynomial degree vs. quantum query complexity. *J. Comput. Syst. Sci.*, 72(2):220–238, 2006. doi:10.1016/j.jcss.2005.06.006.
2. Andris Ambainis, Mohammad Bavarian, Yihan Gao, Jieming Mao, Xiaoming Sun, and Song Zuo. Tighter Relations between Sensitivity and Other Complexity Measures. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 101–113, 2014. doi:10.1007/978-3-662-43948-7_9.
3. Andris Ambainis and Krisjanis Prusis. A Tight Lower Bound on Certificate Complexity in Terms of Block Sensitivity and Sensitivity. In *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II*, pages 33–44, 2014. doi:10.1007/978-3-662-44465-8_4.
4. Andris Ambainis, Krišjānis Prūsis, and Jevgēnijs Vihrovs. Sensitivity Versus Certificate Complexity of Boolean Functions. In *Computer Science – Theory and Applications*, pages 16–28, Cham, 2016. doi:10.1007/978-3-319-34171-2_2.
5. Andris Ambainis and Xiaoming Sun. New separation between *s(f)* and *bs(f)*. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:116, 2011. URL: http://eccc.hpi-web.de/report/2011/116.
6. Harry Buhrman and Ronald De Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002. doi:10.1016/S0304-3975(01)00144-X.
7. Sourav Chakraborty. On the Sensitivity of Cyclically-Invariant Boolean Functions. *CoRR*, abs/cs/0501026, 2005. URL: http://arxiv.org/abs/cs/0501026, arXiv:cs/0501026.
8. Stephen A. Cook, Cynthia Dwork, and Rüdiger Reischuk. Upper and Lower Time Bounds for Parallel Random Access Machines without Simultaneous Writes. *SIAM J. Comput.*, 15(1):87–97, 1986. doi:10.1137/0215006.
9. Justin Gilmer, Michael Saks, and Srikanth Srinivasan. Composition limits and separating examples for some boolean function complexity measures. *Combinatorica*, 36(3):265–311, June 2016. doi:10.1007/s00493-014-3189-x.
10. Parikshit Gopalan, Rocco A. Servedio, Avishay Tal, and Avi Wigderson. Degree and Sensitivity: tails of two distributions. *CoRR*, abs/1604.07432, 2016. arXiv:1604.07432.
11. Pooya Hatami, Raghav Kulkarni, and Denis Pankratov. Variations on the Sensitivity Conjecture. *Theory of Computing, Graduate Surveys*, 4:1–27, 2011. doi:10.4086/toc.gs.2011.004.
12. Claire Kenyon and Samuel Kutin. Sensitivity, Block Sensitivity, and L-block Sensitivity of Boolean Functions. *Inf. Comput.*, 189(1):43–53, February 2004. doi:10.1016/j.ic.2002.12.001.
13. Noam Nisan. CREW PRAMs and decision trees. *SIAM J. Comput.*, 20(6):999–1007, 1991. doi:10.1137/0220062.
14. Noam Nisan and Mario Szegedy. On the degree of boolean functions as real polynomials. *Computational Complexity*, 4(4):301–313, December 1994. doi:10.1007/BF01263419.
15. David Rubinstein. Sensitivity vs. block sensitivity of Boolean functions. *Combinatorica*, 15(2):297–299, June 1995. doi:10.1007/BF01200762.

16 Hans-Ulrich Simon. A tight ω(loglog n)-bound on the time for parallel RAM's to compute nondegenerated boolean functions. *Information and Control*, 55(1):102–107, 1982. doi:10.1016/S0019-9958(82)90477-6.

17 Avishay Tal. Properties and applications of boolean function composition. In *Innovations in Theoretical Computer Science, ITCS '13, Berkeley, CA, USA, January 9-12, 2013*, pages 441–454, 2013. doi:10.1145/2422436.2422485.

18 Avishay Tal. Personal communication, 2018.

19 Madars Virza. Sensitivity versus block sensitivity of Boolean functions. *Information Processing Letters*, 111(9):433–435, 2011. doi:10.1016/j.ipl.2011.02.001.

Lambda-Definable Order-3 Tree Functions are Well-Quasi-Ordered

Kazuyuki Asada
Tohoku University, Sendai, Japan
asada@riec.tohoku.ac.jp
https://orcid.org/0000-0001-8782-2119

Naoki Kobayashi
The University of Tokyo, Tokyo, Japan
koba@is.s.u-tokyo.ac.jp

Abstract

Asada and Kobayashi [ICALP 2017] conjectured a higher-order version of Kruskal's tree theorem, and proved a pumping lemma for higher-order languages modulo the conjecture. The conjecture has been proved up to order-2, which implies that Asada and Kobayashi's pumping lemma holds for order-2 tree languages, but remains open for order-3 or higher. In this paper, we prove a variation of the conjecture for order-3. This is sufficient for proving that a variation of the pumping lemma holds for order-3 tree languages (equivalently, for order-4 word languages).

2012 ACM Subject Classification Theory of computation → Lambda calculus

Keywords and phrases higher-order grammar, pumping lemma, Kruskal's tree theorem, well-quasi-ordering, simply-typed lambda calculus

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.14

Related Version A full version of the paper is available at `http://www.riec.tohoku.ac.jp/~asada/papers/fsttcs18.pdf`.

Acknowledgements We would like to thank anonymous referees for useful comments. This work was supported by JSPS Kakenhi 15H05706 and 18K11156.

1 Introduction

Kruskal's tree theorem [7] says that the homeomorphic embedding relation $\preceq^{\mathrm{he}}$ on finite trees is a well-quasi-ordering, i.e., for every infinite sequence of trees $\pi_0, \pi_1, \pi_2, \ldots$, there exist $i < j$ such that $\pi_i \preceq^{\mathrm{he}} \pi_j$. Here, $\pi \preceq^{\mathrm{he}} \pi'$ means that there exists an embedding of the nodes of π to those of π', preserving the labels and the ancestor/descendant relation. Asada and Kobayashi [2] considered a higher-order version $\preceq^{\mathrm{he}}_{\kappa}$ of $\preceq^{\mathrm{he}}$ on simply-typed λ-terms of type κ, and conjectured that $\preceq^{\mathrm{he}}_{\kappa}$ is also a well-quasi-ordering, for every simple type κ. Under the assumption that the conjecture (which we call AK-conjecture) is true, they proved a pumping lemma for higher-order languages (a la higher-order languages in Damm's IO hierarchy [3]), which says that for any order-k tree grammar that generates an infinite language L, there exists a strictly increasing infinite sequence $\pi_0 \prec^{\mathrm{he}} \pi_1 \prec^{\mathrm{he}} \pi_2 \prec^{\mathrm{he}} \cdots$ such that $\pi_i \in L$ and $|\pi_i| \leq \mathbf{exp}_k(ci + d)$, where $\prec^{\mathrm{he}}$ is the strict version of the homeomorphic embedding, c and d are constants that depend on the grammar, and $\mathbf{exp}_k(x)$ is defined by $\mathbf{exp}_0(x) = x$ and $\mathbf{exp}_{k+1}(x) = 2^{\mathbf{exp}_k(x)}$. The pumping lemma can be used to prove that a certain language does not belong to the class of order-k languages. They also proved that the conjecture is

true up to order-2 types, and hence also the pumping lemma for order-2 tree languages and (by the correspondence between tree/word languages [1, 3]) order-3 word languages. The AK-conjecture is still open for order-3 or higher.

In the present paper, we consider a variation of the AK-conjecture (which we call nAK-conjecture), where the homeomorphic embedding relation is replaced by $\preceq^{\#}$, defined by $\pi_1 \preceq^{\#} \pi_2$ if and only if, for every tree constructor a, $\#_a(\pi_1) \leq \#_a(\pi_2)$; here $\#_a(\pi)$ denotes the number of occurrences of a in π. The correctness of the nAK-conjecture would imply the following variation of the pumping lemma: for any order-k tree grammar that generates an infinite language L, there exists a strictly increasing infinite sequence $\pi_0 \prec^{\#} \pi_1 \prec^{\#} \pi_2 \prec^{\#} \cdots$ such that $\pi_i \in L$ and $|\pi_i| \leq \mathbf{exp}_k(ci + d)$. We prove that the nAK-conjecture is true for the order-3 case, i.e., that $\preceq^{\#}_{\kappa}$ (the logical relation on simply-typed λ-terms of type κ, obtained from $\preceq^{\#}$) is a well-quasi-ordering for any type κ of order up to 3. The variation of the pumping lemma above is thus obtained for order-3 tree languages and order-4 word languages. To our knowledge, pumping lemmas were known only for tree (word, resp.) languages of order up to 2 (3, resp.) [2].

To prove the order-3 nAK-conjecture, we define a transformation $(\cdot)^{\natural}$ from order-3 λ-terms to order-2 numeric functions (that are also represented by λ-terms), and prove (i) the transformation reflects the quasi-orderings, i.e., $t_1 \preceq^{\#}_{\kappa} t_2$ if ${t_1}^{\natural} \preceq^{\mathbb{N}} {t_2}^{\natural}$ for a certain quasi-ordering $\preceq^{\mathbb{N}}$ on numeric functions, and (ii) $\preceq^{\mathbb{N}}$ is a well-quasi-ordering.

Related work. We are not aware of directly related work, besides our own previous work [2]. Our reduction from the well-quasi-orderedness of order-3 λ-terms to that of order-2 numeric functions relies on the inexpressiveness of simply-typed λ-terms as (higher-order) tree functions. Zaionc [11, 12, 13] studied the expressive power of simply-typed λ-terms. Pumping lemmas for higher-order languages have been known to be difficult. After Hayashi [5] proved a pumping lemma for indexed languages (i.e. order-2 word languages), it was only in 2017 that a pumping lemma for order-3 word languages was proved [2]. We have further improved the result to obtain a pumping lemma for order-4 word (or, order-3 tree) languages.

The rest of the paper is structured as follows. Section 2 introduces basic definitions. Section 3 explains the nAK-conjecture and the pumping lemma. Section 4 proves the nAK-conjecture up to order-3. Section 5 concludes the paper.

2 Preliminaries

We give basic definitions on λ-terms and quasi-orderings.

2.1 λ-terms and higher-order languages

▶ **Definition 1** (types and terms). The set of *simple types*, ranged over by κ, is given by: $\kappa ::= \mathtt{o} \mid \kappa_1 \to \kappa_2$. The order[1] of a simple type κ, written $\mathtt{order}(\kappa)$ is defined by $\mathtt{order}(\mathtt{o}) = 0$ and $\mathtt{order}(\kappa_1 \to \kappa_2) = \max(\mathtt{order}(\kappa_1) + 1, \mathtt{order}(\kappa_2))$. The type $\mathtt{o}$ describes trees, and $\kappa_1 \to \kappa_2$ describes functions from κ_1 to κ_2. A *(ranked) alphabet* Σ is a map from a finite set of constants (that represent tree constructors) to the set of natural numbers called *arities*. The set of λY^{nd}*-terms*, ranged over by s, t, u, v, is defined by:

$$t ::= x \mid a\, t_1 \cdots t_k \mid t_1\, t_2 \mid \lambda x : \kappa.t \mid Y_\kappa\, t \mid t_1 \oplus t_2$$

[1] For clarity, we use the word *order* for this notion, and *ordering* for relations such as $\leq$, $\preceq^{\mathrm{he}}$, etc.

Here, $x, y, \ldots$ ranges over variables, and a over $dom(\Sigma)$. The term $a\,t_1 \cdots t_k$ (where we require $\Sigma(a) = k$) constructs a tree that has a as the root and (the values of) $t_1, \ldots, t_k$ as children. Y_κ and $\oplus$ represent a fixed-point combinator and a non-deterministic choice, respectively. We often omit the type annotation and just write $\lambda x.t$ and $Y\,t$ for $\lambda x:\kappa.t$ and $Y_\kappa\,t$. A λY^{nd}-term is called: (i) a $\lambda^{\to,\mathrm{nd}}$-*term* if it does not contain Y; (ii) a $\lambda^{\to}$-*term* if it contains neither Y nor $\oplus$; and (iii) an *applicative term* if it contains none of λ-abstractions, Y, and $\oplus$. We often call a $\lambda^{\to}$-term just a *term*. As usual, we identify λY^{nd}-terms up to the α-equivalence, and implicitly apply α-conversions.

A *type environment* Γ is a sequence of type bindings of the form $x:\kappa$ such that Γ contains at most one binding for each variable x. A λY^{nd}-term t has type κ under Γ if $\Gamma \vdash_{\mathtt{ST}} t : \kappa$ is derivable from the following typing rules.

$$\frac{}{\Gamma,\, x:\kappa,\, \Gamma' \vdash_{\mathtt{ST}} x : \kappa} \qquad \frac{\Sigma(a) = k \qquad \Gamma \vdash_{\mathtt{ST}} t_i : \mathtt{o} \ \ (\text{for each } i \in \{1,\ldots,k\})}{\Gamma \vdash_{\mathtt{ST}} a\,t_1 \cdots t_k : \mathtt{o}} \qquad \frac{\Gamma \vdash_{\mathtt{ST}} t : \kappa \to \kappa}{\Gamma \vdash_{\mathtt{ST}} Y_\kappa\,t : \kappa}$$

$$\frac{\Gamma \vdash_{\mathtt{ST}} t_1 : \kappa_2 \to \kappa \qquad \Gamma \vdash_{\mathtt{ST}} t_2 : \kappa_2}{\Gamma \vdash_{\mathtt{ST}} t_1\,t_2 : \kappa} \qquad \frac{\Gamma,\, x:\kappa_1 \vdash_{\mathtt{ST}} t : \kappa_2}{\Gamma \vdash_{\mathtt{ST}} \lambda x:\kappa_1.t : \kappa_1 \to \kappa_2} \qquad \frac{\Gamma \vdash_{\mathtt{ST}} t_1 : \mathtt{o} \qquad \Gamma \vdash_{\mathtt{ST}} t_2 : \mathtt{o}}{\Gamma \vdash_{\mathtt{ST}} t_1 \oplus t_2 : \mathtt{o}}$$

We consider below only well-typed λY^{nd}-terms. Note that given Γ and t, there exists at most one type κ such that $\Gamma \vdash_{\mathtt{ST}} t : \kappa$. We call κ the type of t (with respect to Γ). We often omit "with respect to Γ" if Γ is clear from context. Given a judgment $\Gamma \vdash t : \kappa$, we define $\lambda\Gamma.t$ by: $\lambda\emptyset.t := t$ and $\lambda(\Gamma, x:\kappa').t := \lambda\Gamma.\lambda x.t$. Also we define $\Gamma \to \kappa$ by: $\emptyset \to \kappa := \kappa$ and $(\Gamma, x:\kappa') \to \kappa := \Gamma \to (\kappa' \to \kappa)$; thus we have $\vdash \lambda\Gamma.t : \Gamma \to \kappa$ if $\Gamma \vdash t : \kappa$. Given an alphabet Σ, we write Λ^{Σ} for the set of $\lambda^{\to}$-terms whose constants are taken from Σ. Also we define $\Lambda^{\Sigma}_{\Gamma,\kappa} := \{t \in \Lambda^{\Sigma} \mid \Gamma \vdash t : \kappa\}$ and $\Lambda^{\Sigma}_{\kappa} := \Lambda^{\Sigma}_{\emptyset,\kappa}$.

For a λY^{nd}-term t with a type environment Γ, the *(internal) order* of t (with respect to Γ), written $\mathtt{order}_\Gamma(t)$, is the largest order of the types of subterms of $\lambda\Gamma.t$, and the *external order* of t (with respect to Γ), written $\mathtt{eorder}_\Gamma(t)$, is the order of the type of t with respect to Γ. We often omit Γ when it is clear from context. For example, for $t = (\lambda x:\mathtt{o}.x)\mathtt{e}$, $\mathtt{order}_\emptyset(t) = 1$ and $\mathtt{eorder}_\emptyset(t) = 0$. We define the *size* $|t|$ of a λY^{nd}-term t by: $|x| := 1$, $|a\,t_1 \cdots, t_k| := 1 + |t_1| + \cdots + |t_k|$, $|s\,t| := |s| + |t| + 1$, $|\lambda x.t| := |t| + 1$, $|Y_\kappa\,t| := |t| + 1$ and $|s \oplus t| := |s| + |t| + 1$. We call a λY^{nd}-term t *ground* (with respect to Γ) if $\Gamma \vdash_{\mathtt{ST}} t : \mathtt{o}$. We call t a (finite, Σ-ranked) *tree* if t is a ground closed applicative term (consisting of only constants). We write $\mathbf{Tree}_\Sigma$ for the set of Σ-ranked trees, and use the meta-variable π for a tree. We often write $\overrightarrow{\cdot}$ to denote a sequence (possibly with a condition on the range of the sequence in the superscript). For example, $\overrightarrow{t_i}^{i\le m}$ denotes the sequence $t_1, \ldots, t_m$ of terms, and $[\overrightarrow{t_i/x_i}^{i\le m}]$ denotes the substitution $[t_1/x_1, \ldots, t_m/x_m]$.

We sometimes identify a ranked alphabet $\Sigma = \{a_1 \mapsto r_1, \ldots, a_k \mapsto r_k\}$ with the first-order environment $\Sigma = \{a_1 : \mathtt{o}^{r_1} \to \mathtt{o}, \ldots, a_k : \mathtt{o}^{r_k} \to \mathtt{o}\}$ (assuming an arbitrary fixed linear ordering on Σ).

▶ **Definition 2** (reduction and language)**.** The set of *(call-by-name) evaluation contexts* is defined by:

$$E ::= [\,]\,t_1 \cdots t_k \mid a\,\pi_1 \cdots \pi_i\,E\,t_1 \cdots t_k$$

and the *call-by-name reduction* for (possibly open) ground λY^{nd}-terms is defined by:

$$E[(\lambda x.t)t'] \longrightarrow E[t[t'/x]] \qquad E[Y\,t] \longrightarrow E[t\,(Y\,t)] \qquad E[t_1 \oplus t_2] \longrightarrow E[t_i] \quad (i = 1, 2)$$

where $t[t'/x]$ is the usual capture-avoiding substitution. We write $\longrightarrow^*$ for the reflexive transitive closure of $\longrightarrow$. A *call-by-name normal form* is a ground λY^{nd}-term t such that

$t \not\longrightarrow t'$ for any t'. For a ground closed λY^{nd}-term t, we define the *tree language* $\mathcal{L}(t)$ *generated by* t by $\mathcal{L}(t) := \{\pi \mid t \longrightarrow^* \pi\}$. For a ground closed $\lambda^{\to}$-term t, $\mathcal{L}(t)$ is a singleton set $\{\pi\}$; we write $\mathcal{T}(t)$ for such π and call it *the tree of* t.

In the previous paper [2] we stated the pumping lemma for the notion of a *higher-order grammar*; in this paper, following [8, 9], we use only the formalism by λY^{nd}-terms for simplicity. Since there exist well-known order-preserving and language-preserving transformations between higher-order grammars and ground closed λY^{nd}-terms, we obtain corresponding results on higher-order grammars immediately.

The notion of a word can be seen as a special case of that of a tree:

▶ **Definition 3** (word alphabet)**.** We call a ranked alphabet Σ a *word alphabet* if it has a special nullary constant e and all the other constants have arity 1. For a tree $\pi = a_1(\cdots(a_n\,\mathsf{e})\cdots)$ of a word alphabet, we define $\mathbf{word}(\pi) := a_1 \cdots a_n$, and we define **utree** as the inverse function of **word**, i.e., $\mathbf{utree}(a_1 \cdots a_n) := a_1(\cdots(a_n\mathsf{e}))$. The *word language* generated by a ground closed λY^{nd}-term t over a word alphabet, written $\mathcal{L}_{\mathtt{w}}(t)$, is defined as $\{\mathbf{word}(\pi) \mid \pi \in \mathcal{L}(t)\}$.

A tree language (word language, resp.) over an alphabet (word alphabet, resp.) Σ is called *order-n* if it is generated by some order-n ground closed λY^{nd}-term of Σ; we note that the classes of order-0, order-1, and order-2 word languages coincide with those of regular, context-free, and indexed languages, respectively [10].

2.2 Some quasi-orderings and their logical relation extension

▶ **Definition 4** ((well-)quasi-ordering)**.** A *quasi-ordering* (a.k.a. preorder) on a set A is a binary relation on A that is reflexive and transitive. A *well-quasi-ordering* (*wqo* for short) on a set S is a quasi-ordering $\leq$ on S such that for any infinite sequence $(s_i)_i$ of elements in S there exist j and k such that $j < k$ and $s_j \leq s_k$.

As a general notation, for a quasi-ordering denoted by $\preceq$, we write $\approx$ for the induced equivalence relation (i.e., $x \approx y$ if $x \preceq y$ and $y \preceq x$), and write $\prec$ for the strict version (i.e., $x \prec y$ if $x \preceq y$ and $y \not\preceq x$). Also, for a quasi-ordering denoted by $\leq$, we write $\sim$ for the induced equivalence relation and $<$ for the strict version. We apply these conventions also to notations with superscript/subscript such as $\preceq^a$, $\preceq_b$, $\preceq^a_b$, $\leq^a$, $\leq_b$, and $\leq^a_b$. Further, for any quasi-ordering on the set of trees of a word alphabet, we use the same notation also for the quasi-ordering on the set of words induced through **utree**.

▶ **Definition 5** (logical relation extension)**.** Let Σ be a ranked alphabet. We call $\leq$ a *base quasi-ordering* (with respect to Σ) if $\leq$ is a quasi-ordering on the set $\Lambda^{\Sigma}_{\mathsf{o}}$ modulo $\beta\eta$-equivalence and every constant in Σ is monotonic on $\leq$. We define the *logical relation extension of* $\leq$ as the family $(\leq_\kappa)_\kappa$ of relations $\leq_\kappa$ on the set $\Lambda^{\Sigma}_{\kappa}$ modulo $\beta\eta$-equivalence indexed by simple types κ where $\leq_\kappa$'s are defined by induction on κ as follows:

$$\begin{array}{rll} t_1 \leq_{\mathsf{o}} t_2 & \text{if} & t_1 \leq t_2 \\ t_1 \leq_{\kappa\to\kappa'} t_2 & \text{if} & \text{for any } t_1', t_2', \quad t_1' \leq_\kappa t_2' \implies t_1\,t_1' \leq_{\kappa'} t_2\,t_2'. \end{array}$$

Furthermore we extend the relation to open terms: for $t_1, t_2 \in \Lambda^{\Sigma}_{\Gamma,\kappa}$, we define $t_1 \leq_{\Gamma,\kappa} t_2$ if $\lambda\Gamma.t_1 \leq_{\Gamma\to\kappa} \lambda\Gamma.t_2$. We omit the subscripts of $\leq_\kappa$ and $\leq_{\Gamma,\kappa}$ if there is no confusion.

The next lemma follows immediately from the basic lemma (a.k.a. the abstraction theorem) of logical relations (see the full version for details).

▶ **Lemma 6.** *Let $\leq$ be a base quasi-ordering. Each component $\leq_\kappa$ of the logical relation extension of $\leq$ is a quasi-ordering. Further, $\leq_\kappa$ is the point-wise quasi-ordering:*

$$t_1 \leq_{\kappa\to\kappa'} t_2 \qquad \text{if and only if} \qquad \text{for any } t' \in \Lambda^{\Sigma}_{\kappa},\quad t_1\, t' \leq_{\kappa'} t_2\, t'.$$

Every quasi-ordering for higher-order terms used in this paper is a logical relation extension (of some base quasi-ordering). The next ordering is used in the previous paper [2].

▶ **Definition 7** (homeomorphic embedding)**.** Let Σ be a ranked alphabet. The *homeomorphic embedding* ordering $\preceq^{\mathrm{he},\Sigma}$ between Σ-ranked trees[2] is inductively defined by the following rules:

$$\frac{\pi_i \preceq^{\mathrm{he},\Sigma} \pi'_i \quad (\text{for all } i \leq k) \qquad k = \Sigma(a)}{a\,\pi_1 \cdots \pi_k \preceq^{\mathrm{he},\Sigma} a\,\pi'_1 \cdots \pi'_k} \qquad \frac{\pi \preceq^{\mathrm{he},\Sigma} \pi_i \qquad k = \Sigma(a) > 0 \qquad 1 \leq i \leq k}{\pi \preceq^{\mathrm{he},\Sigma} a\,\pi_1 \cdots \pi_k}$$

We extend the above ordering to a base ordering by: $t_1 \preceq^{\mathrm{he},\Sigma} t_2$ if $\mathcal{T}(t_1) \preceq^{\mathrm{he},\Sigma} \mathcal{T}(t_2)$.

For example, $\mathtt{br}\,\mathtt{a}\,\mathtt{b} \preceq^{\mathrm{he}} \mathtt{br}\,(\mathtt{br}\,\mathtt{a}\,\mathtt{c})\,\mathtt{b}$. The homeomorphic embedding on words is nothing but the (scattered) subsequence ordering. The following is a fundamental result on the homeomorphic embedding:

▶ **Proposition 8** (Kruskal's tree theorem [7])**.** *For any (finite) ranked alphabet Σ, the homeomorphic embedding $\preceq^{\mathrm{he}}$ on Σ-ranked trees is a well-quasi-ordering.*

Also, we often use the Dickson's theorem [6] which says that the product quasi-ordering (component-wise quasi-ordering) of a finite number of wqo's is a wqo.

The next is the quasi-ordering that is used in the theorems in this paper.

▶ **Definition 9** (occurrence-number quasi-ordering)**.** Let Σ be a ranked alphabet. For $a \in \Sigma$ and a Σ-tree π, we define $\#_a(\pi)$ as the number of occurrences of a in π, and extend this to a ground closed $\lambda^{\to}$-term t by $\#_a(t) := \#_a(\mathcal{T}(t))$. Then we define a base quasi-ordering $\preceq^{\#,\Sigma,a}$ by:

$$t_1 \preceq^{\#,\Sigma,a} t_2 \qquad \text{if} \qquad \#_a(t_1) \leq \#_a(t_2).$$

Also we define a base quasi-ordering $\preceq^{\#,\Sigma}$ by:

$$t_1 \preceq^{\#,\Sigma} t_2 \qquad \text{if} \qquad \text{for every } a \in \Sigma,\quad t_1 \preceq^{\#,\Sigma,a} t_2.$$

Note that $\pi \preceq^{\mathrm{he}} \pi'$ implies $\pi \preceq^{\#,\Sigma} \pi'$, shown by induction on the rule of $\preceq^{\mathrm{he}}$; and further $\pi \preceq^{\mathrm{he}}_{\kappa} \pi'$ implies $\pi \preceq^{\#,\Sigma}_{\kappa} \pi'$ for any κ since $\preceq^{\mathrm{he}}_{\kappa}$ and $\preceq^{\#,\Sigma}_{\kappa}$ are point-wise quasi-ordering. Also note that $\preceq^{\#,\Sigma}_{\kappa} = \cap_{a\in\Sigma}(\preceq^{\#,\Sigma,a}_{\kappa})$ for any κ.

The next quasi-ordering is used just in proofs. We write $\Sigma_{\mathbb{N}}$ for the ranked alphabet $\{0 \mapsto 0, 1 \mapsto 0, + \mapsto 2, \times \mapsto 2\}$; we write $+\,t\,t'$ as $t + t'$ and $\times\,t\,t'$ as $t \times t'$. We define a set-theoretical denotational interpretation $[\![-]\!]$ of $\Lambda^{\Sigma_{\mathbb{N}}}$ by: $[\![\mathsf{o}]\!] := \mathbb{N}$, $[\![\kappa \to \kappa']\!]$ is the set of functions from $[\![\kappa]\!]$ to $[\![\kappa']\!]$, $[\![0]\!] := 0$, $[\![1]\!] := 1$, $[\![+]\!](n)(m) := n + m$, and $[\![\times]\!](n)(m) := n \times m$. For $t_1, t_2 \in \Lambda^{\Sigma_{\mathbb{N}}}_{\Gamma,\kappa}$, we write $t_1 =^{[\![]\!]}_{\Gamma,\kappa} t_2$ (or $t_1 =^{[\![]\!]} t_2$) if $[\![t_1]\!] = [\![t_2]\!]$.

▶ **Definition 10** (natural number quasi-ordering)**.** We define a base quasi-ordering $\preceq^{\mathbb{N}}$ on the set $\Lambda^{\Sigma_{\mathbb{N}}}_{\mathsf{o}}$ by:

$$t_1 \preceq^{\mathbb{N}} t_2 \qquad \text{if} \qquad [\![t_1]\!] \leq [\![t_2]\!].$$

[2] In the usual definition, a quasi-ordering on labels (tree constructors) is assumed. Here we fix the quasi-order on labels to the identity relation.

3 Numeric Pumping Lemma for Higher-order Tree Languages

Here we explain the nAK-conjecture and the pumping lemma for higher-order tree languages with respect to $\preceq^{\#,\Sigma}$.

▶ **Conjecture 11** (nAK-conjecture). *For any Σ and κ, $\preceq^{\#,\Sigma}_{\kappa}$ is a well quasi-ordering.*

Our main theorem (Theorem 14) is to show the above conjecture for κ of order up to 3. The above conjecture (and Theorem 14) can be used for the following pumping lemma:

▶ **Theorem 12** (pumping lemma). *Assume that Conjecture 11 holds. Then, for any order-n ground closed λY^{nd}-term t of a ranked alphabet Σ such that $\mathcal{L}(t)$ is infinite, there exist an infinite sequence of trees $\pi_0, \pi_1, \pi_2, \ldots \in \mathcal{L}(t)$, and constants c, d such that:*

(i) $\pi_0 \prec^{\#,\Sigma} \pi_1 \prec^{\#,\Sigma} \pi_2 \prec^{\#,\Sigma} \cdots$, *and*

(ii) $|\pi_i| \leq \mathbf{exp}_n(ci + d)$ *for each* $i \geq 0$.

Furthermore, we can drop the assumption on Conjecture 11 when $n \leq 3$.

The proof of the above theorem is obtained as a simple modification of the proof of the pumping lemma in [2]: see the full version.

▶ Remark. The theorem we prove in the full version is actually slightly stronger than Theorem 12 above, in the following three points (see the full version for details):

(i) As in [2], we relax the assumption of nAK conjecture, so that $\preceq^{\#,\Sigma}_{\kappa}$ need not be the logical relation; any higher-order extension of the base quasi-ordering that is closed under application suffices.

(ii) As in [2], we use actually a weaker conjecture, called the *periodicity*, which requires that, for any $\vdash_{\mathtt{ST}} t : \kappa \to \kappa$ and $\vdash_{\mathtt{ST}} s : \kappa$, there exist $i, j > 0$ such that $t^i\, s \preceq^{\#,\Sigma}_{\kappa} t^{i+j}\, s \preceq^{\#,\Sigma}_{\kappa} t^{i+2j}\, s \preceq^{\#,\Sigma}_{\kappa} \cdots$.

(iii) Whilst Theorem 12 states a pumping lemma on $\preceq^{\#,\Sigma}$, the generalized theorem states a pumping lemma on arbitrary base quasi-ordering with certain conditions, which includes $\preceq^{\#,\Sigma}$ and $\preceq^{\mathrm{he}}$ as instances.

By the correspondence between order-n tree grammars and order-$(n{+}1)$ word grammars [3, 1], we also have:

▶ **Corollary 13** (pumping lemma for word languages). *Assume that Conjecture 11 holds. Then, for any order-n ground closed λY^{nd}-term t of a word alphabet Σ (where $n \geq 1$) such that $\mathcal{L}_{\mathtt{w}}(t)$ is infinite, there exist an infinite sequence of words $w_0, w_1, w_2, \ldots \in \mathcal{L}_{\mathtt{w}}(t)$, and constants c, d such that:*

(i) $w_0 \prec^{\#,\Sigma} w_1 \prec^{\#,\Sigma} w_2 \prec^{\#,\Sigma} \cdots$, *and*

(ii) $|w_i| \leq \mathbf{exp}_{n-1}(ci + d)$ *for each* $i \geq 0$.

Furthermore, we can drop the assumption on Conjecture 11 when $n \leq 4$.

4 Numeric Version of Order-3 Kruskal's Tree Theorem

Here we prove the main theorem (Theorem 14 below), which states that the nAK-conjecture (Conjecture 11) holds for order-3 types. In this whole section, by a *term*, we mean a $\lambda^{\to}$-term, and we never consider a fixed-point combinator nor non-determinism.

4.1 Main theorem

▶ **Theorem 14.** *For any alphabet Σ and any type κ of order up to 3, $\preceq_\kappa^{\#,\Sigma}$ on Λ_κ^Σ is a wqo.*

The theorem above is obtained as a corollary of the following lemma.

▶ **Lemma 15.** *For any alphabet Σ, any $a \in \Sigma$, and any order-2 type environment Γ (i.e., a type environment whose codomain consists of types of order up to 2), the quasi-ordering $\preceq_{\Gamma,\mathsf{o}}^{\#,\Sigma,a}$ on $\Lambda_{\Gamma,\mathsf{o}}^\emptyset$ is a wqo.*

Proof sketch of Theorem 14.

- For Theorem 14, it is sufficient that $\preceq_\kappa^{\#,\Sigma,a}$ on Λ_κ^Σ is a wqo for every $a \in \Sigma$ and κ with $\mathtt{order}(\kappa) \le 3$, because $\preceq_\kappa^{\#,\Sigma} = \cap_{a\in\Sigma}(\preceq_\kappa^{\#,\Sigma,a})$ and well-quasi-orderings are closed under finite intersection.
- For $\preceq_\kappa^{\#,\Sigma,a}$ to be a wqo for every order-3 type κ, it is sufficient that the restriction of $\preceq_\kappa^{\#,\Sigma,a}$ to $\Lambda_\kappa^\emptyset$ (i.e. $\preceq_\kappa^{\#,\Sigma,a} \cap (\Lambda_\kappa^\emptyset \times \Lambda_\kappa^\emptyset)$) is a wqo for every order-3 type κ, because $t_1 \preceq_\kappa^{\#,\Sigma,a} t_2$ holds if $\lambda\Sigma.t_1(\preceq_{\Sigma\to\kappa}^{\#,\Sigma,a} \cap (\Lambda_{\Sigma\to\kappa}^\emptyset \times \Lambda_{\Sigma\to\kappa}^\emptyset))\lambda\Sigma.t_2$, and $\mathtt{order}(\Sigma \to \kappa) \le 3$.
- For $\preceq_\kappa^{\#,\Sigma,a} \cap (\Lambda_\kappa^\emptyset \times \Lambda_\kappa^\emptyset))$ to be a wqo, Lemma 15 is sufficient, because $t_1(\preceq_\kappa^{\#,\Sigma,a} \cap (\Lambda_\kappa^\emptyset \times \Lambda_\kappa^\emptyset))t_2$ holds if $t_1\, z_1 \cdots z_k \preceq_{\Gamma,\mathsf{o}}^{\#,\Sigma,a} t_2\, z_1 \cdots z_k$, where $\kappa = \kappa_1 \to \cdots \to \kappa_k \to \mathsf{o}$ and $\Gamma = z_1 : \kappa_1, \ldots, z_k : \kappa_k$.

See the full version for details. ◀

Henceforth, we fix arbitrary $a_{\mathrm{fix}} \in \Sigma$, and show Lemma 15 for $a = a_{\mathrm{fix}}$. We prove this lemma in two steps: First we give a transformation $(\cdot)^\natural$ from order-3 terms in $\Lambda_{\Gamma,\mathsf{o}}^\emptyset$ (and their type environment Γ) to order-2 terms in $\Lambda_{\Gamma^\natural,\mathsf{o}}^{\Sigma_\mathbb{N}}$ (and to $\Gamma^\natural$) so that it reflects quasi-orderings: $t^\natural \preceq_{\Gamma^\natural,\mathsf{o}}^{\mathbb{N}} t'^\natural$ implies $t \preceq_{\Gamma,\mathsf{o}}^{\#,\Sigma,a_{\mathrm{fix}}} t'$ (Lemma 18). Then we show that $\preceq_{\Gamma^\natural,\mathsf{o}}^{\mathbb{N}}$ on $\Lambda_{\Gamma^\natural,\mathsf{o}}^{\Sigma_\mathbb{N}}$ is a wqo (Lemma 19). From these two results, Lemma 15 follows immediately.

4.2 Transformation from order-3 terms to order-2 terms

The key observation behind the transformation $(\cdot)^\natural$ is as follows. Let s be a closed term of type $\mathsf{o}^m \to \mathsf{o}$ and $t_1, \ldots, t_m$ be closed terms of type o. Then, we have:

$$\#_a(s\, t_1 \cdots t_m) = c_1 \times \#_a(t_1) + \cdots + c_m \times \#_a(t_m) + d$$

for some numbers $c_1, \ldots, c_m, d$ that do not depend on $t_1, \ldots, t_m$. This is because the order-1 function s representable as a $\lambda^\to$-term can copy only arguments, and the number of copies cannot depend on the arguments. Thus, if we are interested only in the number of occurrences of a constant, information about an order-1 function can be represented by a tuple $(c_1, \ldots, c_m, d)$ of numbers (order-0 values, in other words). By lifting this representation to order-3 terms in $\Lambda_{\Gamma,\mathsf{o}}^\emptyset$, we obtain order-2 terms in $\Lambda_{\Gamma^\natural,\mathsf{o}}^{\Sigma_\mathbb{N}}$.

The actual transformation is non-trivial. Let us first fix $\Gamma = \varphi_1 : \kappa_1, \ldots, \varphi_m : \kappa_m, f_1 : \mathsf{o}^{q_1} \to \mathsf{o}, \ldots, f_\ell : \mathsf{o}^{q_\ell} \to \mathsf{o}$. Here, φ_i's are order-2 variables and f_j's are variables of order up to 1. Every element of $\Lambda_{\Gamma,\mathsf{o}}^\emptyset$ can be normalized to a term generated by the following syntax (which we call an *order-3 normal form*):

$$t ::= y \mid f_j \mid t_1\, t_2 \mid \varphi_i\, t_1 \cdots t_k \mid \lambda y.t.$$

Here, y is a local variable of order 0. We require that the order of $\varphi\, t_1 \cdots t_k$ is at most 1. For example, $\varphi : (\mathsf{o} \to \mathsf{o}) \to \mathsf{o} \to \mathsf{o} \to \mathsf{o}$, $f : \mathsf{o} \to \mathsf{o} \to \mathsf{o}$, $x : \mathsf{o} \vdash \lambda y : \mathsf{o}.\ \varphi\,(f\, x)\,((\lambda y' : \mathsf{o}.\ f\, y'\, y')\, y) : \mathsf{o} \to \mathsf{o} \to \mathsf{o}$ is an order-3 normal form. It can be checked by induction that for any order-3 normal form t, $\mathtt{eorder}_\Gamma(t) \le 1$ (with a suitable environment Γ). Since any

long $\beta\eta$-normal form in $\Lambda^{\emptyset}_{\Gamma,\mathsf{o}}$ with $\mathtt{order}(\Gamma \to \mathsf{o}) = 3$ is an order-3 normal form, considering only order-3 normal forms does not lose generality. In the rest of this section, we use the meta-variable t for order-3 normal forms.

We now define the transformation for order-3 normal forms. Given a term $t_0 \in \Lambda^{\emptyset}_{\Gamma,\mathsf{o}}$, we transform the term in a compositional manner, by transforming each subterm t typed by:

$$\varphi_1 : \kappa_1, \dots, \varphi_m : \kappa_m, f_1 : \mathsf{o}^{q_1} \to \mathsf{o}, \dots, f_\ell : \mathsf{o}^{q_\ell} \to \mathsf{o}; y_1 : \mathsf{o}, \dots, y_n : \mathsf{o} \vdash t : \mathsf{o}^r \to \mathsf{o}$$

to a term e with some suitable type environment. Here, $y_1, \dots, y_n$ are order-0 variables that are bound inside t_0 (rather than t), $\mathtt{order}(\kappa_i) = 2$ for $i \le m$, and $q_i \ge 0$ for $i \le \ell$. We call f_i and φ_i *external variables* and y_i an *internal variable*. Note that an external variable f_i can be order-0.

We first explain how variables and environments are transformed.

- The variables $y_1, \dots, y_n$ will just disappear after the transformation.
- For each order-1 variable f_i of type $\mathsf{o}^{q_i} \to \mathsf{o}$, we prepare a tuple of variables $(c_{f_i,1}, \dots, c_{f_i,q_i}, d_{f_i})$. Each $c_{f_i,j}$ expresses how often f_i copies the j-th argument, and d_{f_i} expresses how often a_{fix} occurs in the value of f_i, so that the number of a_{fix} in $f_i\, t_1, \dots, t_{q_i}$ can be represented by $c_{f_i,1} \times \#_{a_{\mathrm{fix}}}(t_1) + \dots + c_{f_i,q_i} \times \#_{a_{\mathrm{fix}}}(t_{q_i}) + d_{f_i}$ (recall the observation given at the beginning of this subsection).
- For each order-2 variable φ_i of type $\kappa_i = (\mathsf{o}^{q_1} \to \mathsf{o}) \to \dots \to (\mathsf{o}^{q_k} \to \mathsf{o}) \to (\mathsf{o}^{q} \to \mathsf{o})$ (where $q_k > 0$), we prepare a tuple of order-1 variables $(g_{\varphi_i,1}, \dots, g_{\varphi_i,q}, h_{\varphi_i}, \hat{h}_{\varphi_i})$. Basically, $g_{\varphi_i,j}$ and h_{φ_i} are analogous to $c_{f_i,j}$ and d_{f_i}, respectively. Given order-1 functions $t_1, \dots, t_k$ whose values are $\vec{u}_1, \dots, \vec{u}_k$ (where each $\vec{u}_\ell$ is a tuple of size $q_\ell + 1$), for each $j \le q$, the function $\varphi_i\, t_1 \cdots t_k$ copies the j-th order-0 argument $g_{\varphi_i,j}(\vec{u}_1, \dots, \vec{u}_k)$ times, and creates $h_{\varphi_i}(\vec{u}_1, \dots, \vec{u}_k)$ copies of the constant a_{fix}. The other function variable $\hat{h}_{\varphi_i}$ is similar to h_{φ_i} but used for counting an internal variable y_j rather than a_{fix}.

For a type environment

$$\Gamma = \varphi_1 : \kappa_1, \dots, \varphi_m : \kappa_m, f_1 : \mathsf{o}^{q_1} \to \mathsf{o}, \dots, f_\ell : \mathsf{o}^{q_\ell} \to \mathsf{o}$$

where $\kappa_i = (\mathsf{o}^{q^i_1} \to \mathsf{o}) \to \dots \to (\mathsf{o}^{q^i_{k_i}} \to \mathsf{o}) \to (\mathsf{o}^{q^i} \to \mathsf{o})$ $(q^i_{k_i} > 0,\ i = 1, \dots, k)$, we define:

$$\Gamma^{\natural} := \overrightarrow{g_{\varphi_i,j}}^{j \le q^i}, h_{\varphi_i}, \hat{h}_{\varphi_i} : \mathsf{o}^{q^i_1+1} \to \dots \to \mathsf{o}^{q^i_{k_i}+1} \to \mathsf{o}^{\ \overrightarrow{\ \ }^{i \le m}}, \overrightarrow{c_{f_i,j}}^{j \le q_i}, d_{f_i} : \mathsf{o}^{\ \overrightarrow{\ \ }^{i \le \ell}}$$

We now define the transformation of terms. A term t such that

$$\varphi_1 : \kappa_1, \dots, \varphi_m : \kappa_m, f_1 : \mathsf{o}^{q_1} \to \mathsf{o}, \dots, f_\ell : \mathsf{o}^{q_\ell} \to \mathsf{o}; y_1 : \mathsf{o}, \dots, y_n : \mathsf{o} \vdash t : \mathsf{o}^r \to \mathsf{o}$$

is transformed to a tuple $(v_1, \dots, v_n; w_1, \dots, w_r; e)$, using the transformation relation

$$\varphi_1{:}\kappa_1, \dots, \varphi_m{:}\kappa_m, f_1{:}\mathsf{o}^{q_1} \to \mathsf{o}, \dots, f_\ell{:}\mathsf{o}^{q_\ell} \to \mathsf{o}; y_1{:}\mathsf{o}, \dots, y_n{:}\mathsf{o} \vdash t \triangleright (v_1, \dots, v_n; w_1, \dots, w_r; e)$$

defined below. Here, each component is constructed from variables $c_{f_i,j}, d_{f_i}, g_{\varphi_i,j}, h_{\varphi_i}, \hat{h}_{\varphi_i}$ above and $\times, +, 0, 1$. The output of the transformation consists of three parts, separated by semicolons: a (possibly empty) sequence $v_1, \dots, v_n$, a (possibly empty) sequence $w_1, \dots, w_r$, and a single element e. The term v_j represents how often y_j is copied, w_j represents how often the j-th argument of t is copied, and e represents how often the constant a_{fix} is copied. The terms v_j and w_j are auxiliary ones for this transformation, and e plays the role of $t^{\natural}$ explained in Section 4.1.

The transformation relation is defined by the following rules, where $\Gamma = \varphi_1 : \kappa_1, \ldots, \varphi_m : \kappa_m, f_1 : \mathsf{o}^{q_1} \to \mathsf{o}, \ldots, f_\ell : \mathsf{o}^{q_\ell} \to \mathsf{o}$ is fixed.

$$\frac{}{\Gamma; y_1 : \mathsf{o}, \ldots, y_n : \mathsf{o} \vdash y_j \triangleright (\underbrace{0, \ldots, 0}_{j-1}, 1, \underbrace{0, \ldots, 0}_{n-j}; ; 0)} \quad \text{(IVar)}$$

$$\frac{}{\Gamma; y_1 : \mathsf{o}, \ldots, y_n : \mathsf{o} \vdash f_i \triangleright (\underbrace{0, \ldots, 0}_{n}; c_{f_i,1}, \ldots, c_{f_i,q_i}; d_{f_i})} \quad \text{(Var)}$$

$$\frac{\begin{array}{c}\Gamma; y_1 : \mathsf{o}, \ldots, y_n : \mathsf{o} \vdash t_1 \triangleright (v_1, \ldots, v_n; w_1, \ldots, w_r; e) \qquad r \geq 1 \\ \Gamma; y_1 : \mathsf{o}, \ldots, y_n : \mathsf{o} \vdash t_2 \triangleright (v'_1, \ldots, v'_n; ; e')\end{array}}{\Gamma; y_1 : \mathsf{o}, \ldots, y_n : \mathsf{o} \vdash t_1 t_2 \triangleright (v_1 + w_1 v'_1, \ldots, v_n + w_1 v'_n; w_2, \ldots, w_r; e + w_1 e')} \quad \text{(App0)}$$

$$\frac{\begin{array}{c}\Gamma; y_1 : \mathsf{o}, \ldots, y_n : \mathsf{o} \vdash t_j \triangleright (\vec{v}_j; \vec{w}_j; e_j) \qquad \vec{u}_j = (\vec{w}_j; e_j) \qquad (\text{for each } j \in \{1, \ldots, k\}) \\ \vec{u'}_{j,j'} = (\vec{w}_j; v_{j,j'}) \qquad (\text{for each } j \in \{1, \ldots, k\} \text{ and } j' \in \{1, \ldots, n\}) \\ k \geq 1 \text{ and the type of } t_k \text{ is order-1}\end{array}}{\begin{array}{l}\Gamma; y_1 : \mathsf{o}, \ldots, y_n : \mathsf{o} \vdash \varphi_i\, t_1 \cdots t_k \triangleright \\ (\hat{h}_{\varphi_i}(\vec{u'}_{1,1}, \ldots, \vec{u'}_{k,1}) \ldots, \hat{h}_{\varphi_i}(\vec{u'}_{1,n}, \ldots, \vec{u'}_{k,n}); \\ \quad g_{\varphi_i,1}(\vec{u}_1, \ldots, \vec{u}_k), \ldots, g_{\varphi_i,q_i}(\vec{u}_1, \ldots, \vec{u}_k);\ h_{\varphi_i}(\vec{u}_1, \ldots, \vec{u}_k))\end{array}} \quad \text{(App1)}$$

$$\frac{\Gamma; y_1 : \mathsf{o}, \ldots, y_n : \mathsf{o}, y_{n+1} : \mathsf{o} \vdash t \triangleright (v_1, \ldots, v_n, v_{n+1}; w_1, \ldots, w_r; e)}{\Gamma; y_1 : \mathsf{o}, \ldots, y_n : \mathsf{o} \vdash \lambda y_{n+1}.t \triangleright (v_1, \ldots, v_n; v_{n+1}, w_1, \ldots, w_r; e)} \quad \text{(Lam)}$$

Rules (IVar) (for internal variables of type o) (Var) (for order-1 variables), and (Lam) should be obvious from the intuition on the tuple and the translation of an environment. Rules (App0) and (App1) are for applications of order-1 and order-2 functions respectively. (Note however that in (App0), t_1 itself may be an application of order-2 function, of the form $\varphi\, t_{1,1} \cdots t_{1,k}$.) In (App0), note that $t_1 t_2$ creates w_1 copies of (the value of) t_2, so that the number of copies of y_i can be calculated by $v_i + w_1 v'_i$, where v_i and v'_i are the numbers of copies created by t_1 and t_2 respectively. Rule (App1) is based on the intuition explained above about the translation of order-2 variables. Note that the same function $\hat{h}_{\varphi_i}$ is used for counting $y_1, \ldots, y_n$; this is because φ_i does not know y_j (in other words, φ_i cannot be instantiated to a term containing y_j as a free variable), so that the information for counting y_j can only be passed through arguments $\vec{u'}_{j,j'}$.

It should be clear that if $\Gamma; y_1 : \mathsf{o}, \ldots, y_n : \mathsf{o} \vdash t \triangleright (v_1, \ldots, v_n; w_1, \ldots, w_r; e)$ then $v_j, w_{j'}, e \in \Lambda^{\Sigma_{\mathbb{N}}}_{\Gamma^\natural, \mathsf{o}}$ and the order of $\Gamma^\natural \to \mathsf{o}$ is no greater than 2.

▶ **Example 16.** Let $\Gamma = \varphi : (\mathsf{o} \to \mathsf{o}) \to \mathsf{o} \to \mathsf{o}, f : \mathsf{o} \to \mathsf{o}$. Then, we have

$$\Gamma^\natural \;=\; g_{\varphi,1}, h_\varphi, \hat{h}_\varphi : \mathsf{o}^2 \to \mathsf{o},\ c_{f,1}, d_f : \mathsf{o}$$

and $t := \lambda y.\varphi(\varphi\, f)\, y$ is transformed to

$$t^\natural \;=\; h_\varphi\big(g_{\varphi,1}(c_{f,1}, d_f), h_\varphi(c_{f,1}, d_f)\big) + g_{\varphi,1}\big(g_{\varphi,1}(c_{f,1}, d_f), h_\varphi(c_{f,1}, d_f)\big) \times 0$$

by the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma; y:\mathsf{o} \vdash f \triangleright (0; c_{f,1}; d_f)}\,(\textsc{Var})}{\Gamma; y:\mathsf{o} \vdash \varphi\, f \triangleright (\hat{h}_\varphi(c_{f,1},0); g_{\varphi,1}(c_{f,1},d_f); h_\varphi(c_{f,1},d_f))}\,(\textsc{App1})}{\Gamma; y:\mathsf{o} \vdash \varphi(\varphi\, f) \triangleright (\hat{h}_\varphi(\vec{u}'); g_{\varphi,1}(\vec{u}); h_\varphi(\vec{u}))}\,(\textsc{App1}) \quad \dfrac{}{\Gamma; y:\mathsf{o} \vdash y \triangleright (1;;0)}\,(\textsc{IVar})}{\Gamma; y:\mathsf{o} \vdash \varphi(\varphi\, f)\, y \triangleright (\hat{h}_\varphi(\vec{u}') + g_{\varphi,1}(\vec{u}) \times 1; ; h_\varphi(\vec{u}) + g_{\varphi,1}(\vec{u}) \times 0)}\,(\textsc{App0})}{\Gamma; \vdash \lambda y.\varphi(\varphi\, f)\, y \triangleright (; \hat{h}_\varphi(\vec{u}') + g_{\varphi,1}(\vec{u}) \times 1; h_\varphi(\vec{u}) + g_{\varphi,1}(\vec{u}) \times 0)}\,(\textsc{Lam})$$

where $\vec{u} = g_{\varphi,1}(c_{f,1},d_f), h_\varphi(c_{f,1},d_f)$ and $\vec{u}' = g_{\varphi,1}(c_{f,1},d_f), \hat{h}_\varphi(c_{f,1},0)$. The terms in the bottom line of the derivation, $\hat{h}_\varphi(\vec{u}') + g_{\varphi,1}(\vec{u}) \times 1$ and $t^\natural = h_\varphi(\vec{u}) + g_{\varphi,1}(\vec{u}) \times 0$, have type o under the environment $\Gamma^\natural$, and $\mathtt{eorder}(\lambda\Gamma^\natural.t^\natural) = \mathtt{order}(\Gamma^\natural \to \mathsf{o}) = 2$.

The next example is a slightly modified one involving an external variable $x:\mathsf{o}$ instead of the internal variable $y:\mathsf{o}$. We have

$$(\Gamma, x:\mathsf{o})^\natural \;=\; \Gamma^\natural,\; d_x:\mathsf{o}$$

and $t' := \varphi(\varphi\, f)\, x$ is transformed to

$$t'^\natural \;=\; h_\varphi\big(g_{\varphi,1}(c_{f,1},d_f), h_\varphi(c_{f,1},d_f)\big) + g_{\varphi,1}\big(g_{\varphi,1}(c_{f,1},d_f), h_\varphi(c_{f,1},d_f)\big) \times d_x$$

by the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, x:\mathsf{o};\vdash f \triangleright (0; c_{f,1}; d_f)}\,(\textsc{Var})}{\Gamma, x:\mathsf{o};\vdash \varphi\, f \triangleright (\hat{h}_\varphi(c_{f,1},0); g_{\varphi,1}(c_{f,1},d_f); h_\varphi(c_{f,1},d_f))}\,(\textsc{App1})}{\Gamma, x:\mathsf{o};\vdash \varphi(\varphi\, f) \triangleright (\hat{h}_\varphi(\vec{u}'); g_{\varphi,1}(\vec{u}); h_\varphi(\vec{u}))}\,(\textsc{App1}) \quad \dfrac{}{\Gamma, x:\mathsf{o};\vdash x \triangleright (0;;d_x)}\,(\textsc{Var})}{\Gamma, x:\mathsf{o};\vdash \varphi(\varphi\, f)\, x \triangleright (\hat{h}_\varphi(\vec{u}') + g_{\varphi,1}(\vec{u}) \times 0; ; h_\varphi(\vec{u}) + g_{\varphi,1}(\vec{u}) \times d_x)}\,(\textsc{App0})$$

where $\vec{u}$ and $\vec{u}'$ are the same as above. ◀

Lemma 17 below says that the transformation preserves the meaning of ground terms. Here we regard constants in Σ as variables of up to order 1, and we define a substitution $\theta_\Sigma^{a_{\mathrm{fix}}}$ by:

$$\theta_\Sigma^{a_{\mathrm{fix}}} \;:=\; [\overrightarrow{1/c_{a,i}}^{a\in\Sigma, i\le \mathtt{ar}(a)},\; 1/d_{a_{\mathrm{fix}}},\; \overrightarrow{0/d_a}^{a\in\Sigma\setminus\{a_{\mathrm{fix}}\}}].$$

(Recall that $a_{\mathrm{fix}} \in \Sigma$ above is the constant arbitrarily fixed at the end of Section 4.1.)

▶ **Lemma 17** (preservation of meaning). *If $\Sigma;\vdash t \triangleright (;;e)$, then we have $\#_{a_{\mathrm{fix}}}(t) = [\![e\theta_\Sigma^{a_{\mathrm{fix}}}]\!]$.*

The above lemma follows from a usual substitution lemma (on internal variables) and a subject reduction property; see the full version for the proof.

The correctness of the transformation is stated as the following lemma.

▶ **Lemma 18** (ordering reflection). *Let: Σ be an alphabet; $a_{\mathrm{fix}} \in \Sigma$; Γ be an environment of the form*

$$\Gamma = \varphi_1:\kappa_1, \ldots, \varphi_m:\kappa_m, f_1:\mathsf{o}^{q_1} \to \mathsf{o}, \ldots, f_\ell:\mathsf{o}^{q_\ell} \to \mathsf{o}$$

where $\mathtt{order}(\kappa_i) = 2$ and $q_i \ge 0$; $t, t' \in \Lambda^{\emptyset}_{\Gamma,\mathsf{o}}$; and

$$\Gamma;\vdash t \triangleright (;;e) \qquad \Gamma;\vdash t' \triangleright (;;e').$$

Then we have:

$$t \preceq^{\#,\Sigma,a_{\mathrm{fix}}}_{\Gamma,\mathsf{o}} t' \qquad \text{if} \qquad e \preceq^{\mathbb{N}}_{\Gamma^\natural,\mathsf{o}} e'.$$

The proof of the above lemma is given in the full version, where we use Lemma 17 and substitution lemmas on external variables.

4.3 $\preceq^{\mathbb{N}}$ on order-2 terms is a wqo

The main goal of this subsection is to prove the following lemma.

▶ **Lemma 19** ($\preceq^{\mathbb{N}}_{\Gamma,\mathsf{o}}$ on order-2 terms is wqo). *For* $\Gamma = f_1 : \mathsf{o}^{q_1} \to \mathsf{o}, \ldots, f_n : \mathsf{o}^{q_n} \to \mathsf{o}$*, the quasi-ordering* $\preceq^{\mathbb{N}}_{\Gamma,\mathsf{o}}$ *on* $\Lambda^{\Sigma_{\mathbb{N}}}_{\Gamma,\mathsf{o}}$ *is a wqo.*

Lemma 15 follows as a corollary of Lemma 19 above and Lemma 18 in the previous subsection:

Proof of Lemma 15. Let $t_0, t_1, \ldots \in \Lambda^{\emptyset}_{\Gamma,\mathsf{o}}$ be an infinite sequence. We have the infinite sequence $e_0, e_1, \ldots \in \Lambda^{\Sigma_{\mathbb{N}}}_{\Gamma^\natural,\mathsf{o}}$ such that $\Gamma;\vdash t_i \triangleright (;;e_i)$, and by Lemma 18, $t_i \preceq^{\#,\Sigma,a_{\mathrm{fix}}}_{\Gamma,\mathsf{o}} t_j$ if $e_i \preceq^{\mathbb{N}}_{\Gamma^\natural,\mathsf{o}} e_j$. By Lemma 19, there indeed exist i, j $(i < j)$ such that $e_i \preceq^{\mathbb{N}}_{\Gamma^\natural,\mathsf{o}} e_j$. Thus, we have $t_i \preceq^{\#,\Sigma,a_{\mathrm{fix}}}_{\Gamma,\mathsf{o}} t_j$ as required. ◀

To prove Lemma 19, we restrict (without loss of generality) $\Lambda^{\Sigma_{\mathbb{N}}}_{\Gamma,\mathsf{o}}$ to the set of β-normal forms (which we call *order-2 polynomials*), generated by the following grammar:

$$P ::= 0 \mid 1 \mid P_1 + P_2 \mid P_1 \times P_2 \mid f\, P_1 \cdots P_q$$

Here, in $f\, P_1 \cdots P_q$, f should have type $\mathsf{o}^q \to \mathsf{o}$. We write $\mathrm{P}^{\mathbb{N}}_2$ for the set of all order-2 polynomials, and write $\mathrm{P}^{\mathbb{N}}_{\Gamma,\mathsf{o}}$ for $\Lambda^{\Sigma_{\mathbb{N}}}_{\Gamma,\mathsf{o}} \cap \mathrm{P}^{\mathbb{N}}_2$. Note that the arity of f may be 0, so that, for example, $f_1(f_2 \times (f_2 + 1)) \in \mathrm{P}^{\mathbb{N}}_{f_1:\mathsf{o}\to\mathsf{o}, f_2:\mathsf{o},\mathsf{o}}$. Thus, for Lemma 19, the following suffices:

▶ **Lemma 20** ($\preceq^{\mathbb{N}}_{\Gamma,\mathsf{o}}$ on order-2 polynomials is wqo). *For* $\Gamma = f_1 : \mathsf{o}^{q_1} \to \mathsf{o}, \ldots, f_n : \mathsf{o}^{q_n} \to \mathsf{o}$*, the quasi-ordering* $\preceq^{\mathbb{N}}_{\Gamma,\mathsf{o}}$ *on* $\mathrm{P}^{\mathbb{N}}_{\Gamma,\mathsf{o}}$ *is a wqo.*

The idea for proving this lemma is as follows:

- An order-2 polynomial is regarded as a tree. Thus, by Kruskal's tree theorem (Proposition 8), the set $\mathrm{P}^{\mathbb{N}}_{\Gamma,\mathsf{o}}$ is well-quasi-ordered with respect to the homeomorphic embedding $\preceq^{\mathrm{he},\Sigma_{\mathbb{N}}\cup\Gamma}_{\mathsf{o}}$. Unfortunately, however, the relation $P_1 \preceq^{\mathrm{he},\Sigma_{\mathbb{N}}\cup\Gamma}_{\mathsf{o}} P_2$ does not necessarily imply $\preceq^{\mathbb{N}}_{\Gamma,\mathsf{o}}$; for example, if $P_1 = 1$ and $P_2 = f_1(1)$, then $P_1 \preceq^{\mathrm{he},\Sigma_{\mathbb{N}}\cup\Gamma}_{\mathsf{o}} P_2$ holds but $P_1 \preceq^{\mathbb{N}}_{\Gamma,\mathsf{o}} P_2$ does not, because f_1 may be instantiated to $\lambda x.0$. Similarly for $P_1 = f_2$ and $P_2 = f_2 \times 0$.
- To address the problem above, we classify the values of $f \in \mathrm{P}^{\mathbb{N}}_{\Gamma,\mathsf{o}}$ (i.e. elements of $\Lambda^{\Sigma_{\mathbb{N}}}_{\mathsf{o}^q\to\mathsf{o}}$) into a finite number of equivalence classes $A^{(1)}, \ldots, A^{(\ell)}$, and use the classification to further normalize order-2 polynomials, so that $P_1 \preceq^{\mathrm{he},\Sigma_{\mathbb{N}}\cup\Gamma}_{\mathsf{o}} P_2$ implies $P_1 \preceq^{\mathbb{N}}_{\Gamma,\mathsf{o}} P_2$ on the normalized polynomials. For example, in the case of $P_1 = 1$ and $P_2 = f_1(1)$ above, the values of f_1 are classified to (i) those that use the argument, (ii) those that return a positive constant without using the argument, and (iii) those that always return 0. We can then normalize $P_2 = f_1(1)$ to $f_1(1)$ (in case (i)), $f_1(0)$ (in case (ii)), and 0 (in case (iii)), respectively. (In case (ii), any argument is replaced with 0, because the argument is irrelevant.) Thus, we can indeed deduce $P_1 \preceq^{\mathbb{N}}_{\Gamma,\mathsf{o}} P_2$ from $P_1 \preceq^{\mathrm{he},\Sigma_{\mathbb{N}}\cup\Gamma}_{\mathsf{o}} P_2$ when the value of f_1 is restricted to just those in (i); and the same holds also for (ii) and (iii). It follows that the restriction of the relation $\preceq^{\mathbb{N}}_{\Gamma,\mathsf{o}}$ to each classification of the values of $f_1, \ldots, f_\ell \in \mathit{dom}(\Gamma)$ is a wqo. Since the number of classifications is finite, by Dickson's theorem (recall the sentence below Proposition 8), $\preceq^{\mathbb{N}}_{\Gamma,\mathsf{o}}$ (which is the intersection of the restrictions of $\preceq^{\mathbb{N}}_{\Gamma,\mathsf{o}}$ to the finite number of classifications) is also a wqo.

We first formalize and justify the reasoning in the last part (using Dickson's theorem).

▶ **Definition 21** (finite case analysis). For $\Gamma = f_1 : \kappa_1, \ldots, f_n : \kappa_n$, we call a *finite case analysis of* Γ a family $(A_i^j)_{i \le n, j \in J_i}$ of sets such that $\Lambda_{\kappa_i}^{\Sigma_\mathbb{N}} = \cup_{j \le J_i} A_i^j$ for each $i \le n$. For $(A_i)_{i \le n}$ such that $A_i \subseteq \Lambda_{\kappa_i}^{\Sigma_\mathbb{N}}$, we define a quasi-ordering $\preceq^{\mathbb{N}}_{\Gamma,(A_i)_i}$ on $\Lambda_{\Gamma,\mathsf{o}}^{\Sigma_\mathbb{N}}$ as follows:

$$t \preceq^{\mathbb{N}}_{\Gamma,(A_i)_i} t' \qquad \Longleftrightarrow \qquad \forall t_1 \in A_1, \ldots, t_n \in A_n.\ [\![t[t_i/f_i]_i]\!] \le [\![t'[t_i/f_i]_i]\!]$$

We often omit the subscript Γ of $\preceq^{\mathbb{N}}_{\Gamma,(A_i)_i}$ and write $\preceq^{\mathbb{N}}_{(A_i)_i}$.

The following lemma follows immediately from the fact that the intersection of a finite number of wqo's is a wqo (which is in turn an immediate corollary of Dickson's theorem). (see the full version for omitted proofs in the rest of this section).

▶ **Lemma 22.** *For $\Gamma = f_1 : \kappa_1, \ldots, f_n : \kappa_n$ and a finite case analysis $(A_i^j)_{i \le n, j \in J_i}$ of Γ, if $\preceq^{\mathbb{N}}_{(A_i^{j_i})_i}$ on $\Lambda_{\Gamma,\mathsf{o}}^{\Sigma_\mathbb{N}}$ is a wqo for any "case" $(j_i)_{i \le n} \in \prod_{i \le n} J_i$, then so is $\preceq^{\mathbb{N}}$ on $\Lambda_{\Gamma,\mathsf{o}}^{\Sigma_\mathbb{N}}$.*

Thus, to prove Lemma 20, it remains to find an appropriate decomposition $\Lambda_{\kappa_i}^{\Sigma_\mathbb{N}} = \cup_{j \le J_i} A_i^j$ (where κ_i is an order-1 type $\mathsf{o}^q \to \mathsf{o}$), and prove that $\preceq^{\mathbb{N}}_{(A_i^{j_i})_i}$ is a wqo.

Henceforth we identify an element of $\Lambda_{\mathsf{o}^q \to \mathsf{o}}^{\Sigma_\mathbb{N}}$ with the corresponding element of the polynomial semi-ring $\mathbb{N}[x_1, \ldots, x_q]$. For example, $\lambda x_1.\lambda x_2.((\lambda y.y)x_1) + x_2 \times x_2$ is identified with the polynomial $x_1 + x_2^2$ (which is obtained by normalizing and omitting λ-abstractions, assuming a fixed ordering of the bound variables). For $t \in \Lambda_{\mathsf{o}^q \to \mathsf{o}}^{\Sigma_\mathbb{N}}$ we write $\mathtt{poly}(t)$ for the corresponding polynomial.

We define the equivalence relation $\sim$ as the least semi-ring congruence relation on $\mathbb{N}[x_1, \ldots, x_q]$ that satisfies (i) $a \sim 1$ if $a > 0$ and (ii) $x_i^j \sim x_i$ if $j > 0$. For example, $2x_1^2x_2 + 3x_1x_2^2 + x_1 + 4 \sim x_1x_2 + x_1 + 1$, and the quotient set $\mathbb{N}[x_1]/\sim$ consists of:

$$[0]_\sim, [1]_\sim, [x_1]_\sim, [x_1 + 1]_\sim,$$

and $\mathbb{N}[x_1, x_2]/\sim$ consists of

$$[0]_\sim, [1]_\sim, [x_1]_\sim, [x_2]_\sim, [x_1x_2]_\sim, [1{+}x_1]_\sim, [1{+}x_2]_\sim, [1{+}x_1x_2]_\sim, [x_1{+}x_2]_\sim, \ldots, [1{+}x_1{+}x_2{+}x_1x_2]_\sim.$$

In general, $\mathcal{P}(\mathcal{P}([q]))$ (where $[q]$ denotes $\{1, \ldots, q\}$ and $\mathcal{P}(X)$ denotes the powerset of X) gives a complete representation of the quotient set $\mathbb{N}[x_1, \ldots, x_q]/\sim$, i.e.,

$$\mathbb{N}[x_1, \ldots, x_q]/\sim \ = \ \left\{ \left[\sum_{\{p_1 < \cdots < p_r\} \in \Phi} x_{p_1} \cdots x_{p_r} \right]_\sim \,\middle|\, \Phi \in \mathcal{P}(\mathcal{P}([q])) \right\}.$$

Through $\mathtt{poly} : \Lambda_{\mathsf{o}^q \to \mathsf{o}}^{\Sigma_\mathbb{N}} \to \mathbb{N}[x_1, \ldots, x_q]$, we can induce an equivalence relation on $\Lambda_{\mathsf{o}^q \to \mathsf{o}}^{\Sigma_\mathbb{N}}$ from $\sim$ on $\mathbb{N}[x_1, \ldots, x_q]$, and let A_q^Φ be the equivalence class corresponding to Φ, i.e.,

$$A_q^\Phi := \left\{ t \in \Lambda_{\mathsf{o}^q \to \mathsf{o}}^{\Sigma_\mathbb{N}} \,\middle|\, \mathtt{poly}(t) \sim \sum_{\{p_1 < \cdots < p_r\} \in \Phi} x_{p_1} \cdots x_{p_r} \right\}. \tag{1}$$

Then we have $\Lambda_{\mathsf{o}^q \to \mathsf{o}}^{\Sigma_\mathbb{N}} = \sqcup_{\Phi \in \mathcal{P}(\mathcal{P}([q]))} A_q^\Phi$. Now, given $\Gamma = f_1 : \mathsf{o}^{q_1} \to \mathsf{o}, \ldots, f_n : \mathsf{o}^{q_n} \to \mathsf{o}$, we have obtained a finite case analysis of Γ as $(A_{q_i}^\Phi)_{i \le n, \Phi \in \mathcal{P}(\mathcal{P}([q_i]))}$; for $(\Phi_i)_i \in \prod_{i \le n} \mathcal{P}(\mathcal{P}([q_i]))$, we write $\preceq^{\mathbb{N}}_{(\Phi_i)_i}$ for $\preceq^{\mathbb{N}}_{(A_{q_i}^{\Phi_i})_i}$. Thus it remains to show that $\preceq^{\mathbb{N}}_{(\Phi_i)_i}$ on $\mathrm{P}^{\mathbb{N}}_{\Gamma,\mathsf{o}}$ is a wqo for each $(\Phi_i)_i \in \prod_{i \le n} \mathcal{P}(\mathcal{P}([q_i]))$.

The following lemma justifies the partition of polynomials based on $\sim$.

▶ **Lemma 23** (zero/positive). *For any $\Gamma = f_1 : \mathsf{o}^{q_1} \to \mathsf{o}, \ldots, f_n : \mathsf{o}^{q_n} \to \mathsf{o}$, $(\Phi_i)_i \in \prod_{i \le n} \mathcal{P}(\mathcal{P}([q_i]))$, and $\Gamma \vdash P : \mathsf{o}$, we have either $P \preceq^{\mathbb{N}}_{(\Phi_i)_i} 0$ or $1 \preceq^{\mathbb{N}}_{(\Phi_i)_i} P$.*

In other words, the lemma above says that, given an order-2 polynomial P, whether $P[t_1/f_1, \dots, t_n/f_n]$ evaluates to 0 or not is solely determined by the equivalence classes $t_1, \dots, t_n$ belong to.

▶ **Example 24.** Let $\Gamma := f : \mathsf{o}^2 \to \mathsf{o}$, and $\Phi := \{\emptyset, \{1,2\}\} \in \mathcal{P}(\mathcal{P}([2]))$, which denotes the equivalence class $[1 + x_1 x_2]_\sim$. We have $1 \preceq^{\mathbb{N}}_{\Phi} f\, P_1\, P_2$ for any P_1 and P_2, since any element of the equivalence class is of the form $a + \cdots$ for some natural number $a \geq 1$.

Based on the property above, we define the rewriting relation $\longrightarrow_{(\Phi_i)_i}$, to simplify order-2 polynomials by replacing (i) subterms that always evaluate to 0, and (ii) arguments of a function that are irrelevant, with 0.

▶ **Definition 25** (rewriting relation and $(\Phi_i)_i$-normal form)**.** For $\Gamma = f_1 : \mathsf{o}^{q_1} \to \mathsf{o}, \dots, f_n : \mathsf{o}^{q_n} \to \mathsf{o}$ and $(\Phi_i)_i \in \prod_{i \leq n} \mathcal{P}(\mathcal{P}([q_i]))$, we define the relation $\longrightarrow^{\circ}_{(\Phi_i)_i}$ by the following two rules.

- $P \longrightarrow^{\circ}_{(\Phi_i)_i} 0$ if $P \preceq^{\mathbb{N}}_{(\Phi_i)_i} 0$ and $P \neq 0$.
- $f_\ell\, P_1 \cdots P_{q_\ell} \longrightarrow^{\circ}_{(\Phi_i)_i} f_\ell\, P_1 \cdots P_{k-1}\, 0\, P_{k+1} \cdots P_{q_\ell}$ if (i) $P_k \neq 0$ and (ii) for all $\phi \in \Phi_\ell$ such that $k \in \phi$, there exists $p \in \phi$ such that $P_p \preceq^{\mathbb{N}}_{(\Phi_i)_i} 0$.

We write $P_0 \longrightarrow_{(\Phi_i)_i} P_1$ if $P_i = E[P_i']$ and $P_0' \longrightarrow^{\circ}_{(\Phi_i)_i} P_1'$ for some E, P_0' and P_1', where the evaluation context E is defined by:

$$E ::= [\,] \mid E + P \mid P + E \mid E \times P \mid P \times E \mid f\, P_1\, \dots\, P_{i-1}\, E\, P_{i+1}\, \dots\, P_q.$$

We call a normal form of $\longrightarrow_{(\Phi_i)_i}$ a $(\Phi_i)_i$*-normal form*.

Intuitively, the condition (ii) in the second rule says that whenever the k-th argument P_k is used by f_ℓ, it occurs only in the form of $P_k \times P_p \times \cdots$ (up to equivalence) and P_p always evaluates to 0; thus, the value of P_k is actually irrelevant.

▶ **Example 26.** We continue Example 24. Recall $\Gamma = f : \mathsf{o}^2 \to \mathsf{o}$ and $\Phi = \{\emptyset, \{1,2\}\}$. Consider the order-2 polynomial $f\, 1\, (1 \times 0)$. It can be rewritten to $f\, 1\, 0$ by using the first rule (and the evaluation context $E = f\, 1\, [\,]$). We can further apply the second rule to obtain $f\, 1\, 0 \longrightarrow_\Phi f\, 0\, 0$, because $k = 1$ satisfies the conditions ((i) and) (ii). In fact, if $1 \in \phi \in \Phi$, then $\phi = \{1,2\}$; hence, the required condition holds for $p = 2$. Note that $f\, 0\, 0$ is a Φ-normal form; the first rule is not applicable, as $f\, 0\, 0 \not\preceq^{\mathbb{N}}_{\Phi} 0$ by the discussion in Example 24.

The following lemma guarantees that any order-2 polynomial can be transformed to at least one equivalent $(\Phi_i)_i$*-normal form*.

▶ **Lemma 27** (existence of normal form)**.**

1. $\longrightarrow_{(\Phi_i)_i}$ *is strongly normalizing.*
2. *If* $P \longrightarrow_{(\Phi_i)_i} P'$ *then* $P \approx^{\mathbb{N}}_{(\Phi_i)_i} P'$.

We can reduce the wqoness of $\preceq^{\mathbb{N}}_{(\Phi_i)_i}$ to that of $\preceq^{\mathrm{he},\Sigma_{\mathbb{N}} \cup \Gamma}_{\mathsf{o}}$ by the following lemma:

▶ **Lemma 28.** *For* $\Gamma = f_1 : \mathsf{o}^{q_1} \to \mathsf{o}, \dots, f_n : \mathsf{o}^{q_n} \to \mathsf{o}$, $(\Phi_i)_i \in \prod_{i \leq n} \mathcal{P}(\mathcal{P}([q_i]))$, *and* $(\Phi_i)_i$*-normal forms* $\Gamma \vdash P', P : \mathsf{o}$, *if* $P' \preceq^{\mathrm{he},\Sigma_{\mathbb{N}} \cup \Gamma}_{\mathsf{o}} P$ *then* $P' \preceq^{\mathbb{N}}_{(\Phi_i)_i} P$.

The proof is given by a simple calculation using Lemma 23 and that the given $(\Phi_i)_i$-normal forms P', P do not satisfy the condition for the rewriting $\longrightarrow_{(\Phi_i)_i}$.

Now we are ready to prove Lemma 20.

Proof of Lemma 20. By Lemma 22, it suffices to show that $\preceq^{\mathbb{N}}_{(\Phi_i)_i}$ on $\mathrm{P}^{\mathbb{N}}_{\Gamma,\mathsf{o}}$ is a wqo for each $(\Phi_i)_i \in \prod_{i\leq n} \mathcal{P}(\mathcal{P}([q_i]))$. By the Kruskal's tree theorem, $\preceq^{\mathrm{he},\Sigma_{\mathbb{N}}\cup\Gamma}_{\mathsf{o}}$ on $\mathrm{P}^{\mathbb{N}}_{\Gamma,\mathsf{o}}$ is a wqo, and hence the sub-ordering $\preceq^{\mathrm{he},\Sigma_{\mathbb{N}}\cup\Gamma}_{\mathsf{o}}$ on the subset

$$\{P \in \mathrm{P}^{\mathbb{N}}_{\Gamma,\mathsf{o}} \mid P \text{ is a } (\Phi_i)_i\text{-normal form}\} \subseteq \mathrm{P}^{\mathbb{N}}_{\Gamma,\mathsf{o}}$$

is a wqo. Therefore by Lemma 28, $\preceq^{\mathbb{N}}_{(\Phi_i)_i}$ on $\{P \in \mathrm{P}^{\mathbb{N}}_{\Gamma,\mathsf{o}} \mid P \text{ is a } (\Phi_i)_i\text{-normal form}\}$ is a wqo. By Lemma 27, $\{P \in \mathrm{P}^{\mathbb{N}}_{\Gamma,\mathsf{o}} \mid P \text{ is a } (\Phi_i)_i\text{-normal form}\}$ and $\mathrm{P}^{\mathbb{N}}_{\Gamma,\mathsf{o}}$ – both modulo $\beta\eta$-equivalence – are isomorphic (with respect to $\preceq^{\mathbb{N}}_{(\Phi_i)_i}$ and $\preceq^{\mathbb{N}}_{(\Phi_i)_i}$); hence $\preceq^{\mathbb{N}}_{(\Phi_i)_i}$ on $\mathrm{P}^{\mathbb{N}}_{\Gamma,\mathsf{o}}$ is a wqo. ◀

5 Conclusion

W have introduced the nAK-conjecture, a weaker version of the AK-conjecture in [2], and proved it up to order 3. We have also proved a pumping lemma for higher-order grammars (which is slightly weaker than the pumping lemma conjectured in [2]) under the assumption that the nAK-conjecture holds. Obvious future work is to show the nAK-conjecture or the original AK-conjecture for arbitrary orders. Finding other applications of the two conjectures (cf. an application of Kruskal's tree theorem to program termination [4]) is also left for future work.

References

1. Kazuyuki Asada and Naoki Kobayashi. On Word and Frontier Languages of Unsafe Higher-Order Grammars. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *LIPIcs*, pages 111:1–111:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.
2. Kazuyuki Asada and Naoki Kobayashi. Pumping Lemma for Higher-order Languages. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 97:1–97:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.97.
3. Werner Damm. The IO- and OI-Hierarchies. *Theor. Comput. Sci.*, 20:95–207, 1982.
4. Nachum Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17(3):279–301, 1982. doi:10.1016/0304-3975(82)90026-3.
5. Takeshi Hayashi. On Derivation Trees of Indexed Grammars –An Extension of the uvwxy-Theorem–. *Publ. RIMS, Kyoto Univ.*, pages 61–92, 1973.
6. Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, 3(1):326–336, 1952.
7. Joseph B. Kruskal. Well-Quasi-Ordering, The Tree Theorem, and Vazsonyi's Conjecture. *Transactions of the American Mathematical Society*, 95(2):210–225, 1960. URL: http://www.jstor.org/stable/1993287.
8. Pawel Parys. Intersection Types and Counting. In Naoki Kobayashi, editor, *Proceedings Eighth Workshop on Intersection Types and Related Systems, ITRS 2016, Porto, Portugal, 26th June 2016.*, volume 242 of *EPTCS*, pages 48–63, 2016. doi:10.4204/EPTCS.242.6.
9. Pawel Parys. The Complexity of the Diagonal Problem for Recursion Schemes. In Satya V. Lokam and R. Ramanujam, editors, *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017, December 11-15, 2017, Kanpur, India*, volume 93 of *LIPIcs*, pages 45:1–45:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.FSTTCS.2017.45.

10 Mitchell Wand. An algebraic formulation of the Chomsky hierarchy. In *Category Theory Applied to Computation and Control*, volume 25 of *LNCS*, pages 209–213. Springer, 1974.

11 Marek Zaionc. Word Operation Definable in the Typed lambda-Calculus. *Theor. Comput. Sci.*, 52:1–14, 1987. doi:10.1016/0304-3975(87)90077-6.

12 Marek Zaionc. On the "lambda"-definable tree operations. In *Algebraic Logic and Universal Algebra in Computer Science, Conference, Ames, Iowa, USA, June 1-4, 1988, Proceedings*, volume 425 of *Lecture Notes in Computer Science*, pages 279–292, 1990.

13 Marek Zaionc. Lambda Representation of Operations Between Different Term Algebras. In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25-30, 1994, Selected Papers*, volume 933 of *Lecture Notes in Computer Science*, pages 91–105. Springer, 1994. doi:10.1007/BFb0022249.

A Hypersequent Calculus with Clusters for Tense Logic over Ordinals

David Baelde
LSV, ENS Paris-Saclay & CNRS & Inria, Université Paris-Saclay, France
baelde@lsv.fr

Anthony Lick
LSV, CNRS & ENS Paris-Saclay, Université Paris-Saclay, France
lick@lsv.fr

Sylvain Schmitz
LSV, ENS Paris-Saclay & CNRS, Université Paris-Saclay, France
schmitz@lsv.fr
https://orcid.org/0000-0002-4101-4308

Abstract

Prior's tense logic forms the core of linear temporal logic, with both past- and future-looking modalities. We present a sound and complete proof system for tense logic over ordinals. Technically, this is a hypersequent system, enriched with an ordering, clusters, and annotations. The system is designed with proof search algorithms in mind, and yields an optimal coNP complexity for the validity problem. It entails a small model property for tense logic over ordinals: every satisfiable formula has a model of order type at most ω^2. It also allows to answer the validity problem for ordinals below or exactly equal to a given one.

2012 ACM Subject Classification Theory of computation → Proof theory, Theory of computation → Modal and temporal logics

Keywords and phrases modal logic, proof system, hypersequent

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.15

Related Version Full paper available from `https://hal.inria.fr/hal-01852077`.

Funding ANR-14-CE28-0005 PRODAQ

1 Introduction

Linear temporal logic has become a staple specification language in verification since its introduction by Pnueli [28]. In its most common form, the logic features an "until" temporal modality and ranges over linear time flows of order type ω, i.e. over infinite words, where it enjoys a PSPACE-complete satisfiability problem [34]. A large number of variants with the same complexity has been motivated and introduced in the literature, notably temporal logics with past modalities [23, 21], ranging over arbitrary ordinals [33, 12], or even – with the Stavi modalities added – over arbitrary linear time flows [10, 32].

Linear temporal logic finds its roots in Prior's tense logic [31, 9], which only featured the strict "past" P and "future" F modalities. This set of modalities is still interesting in its own right, as it is sufficient for many modelling tasks [35], and is known to lead to a slightly easier NP-complete satisfiability problem both over ω [34] and over arbitrary linear time

flows [26]. While linear tense logic is less expressive than FO($<$), the first-order logic over linear orders with unary predicates, it has nevertheless nice characterisations as it captures instead its two-variable fragment FO^2($<$) [13].

In this paper, we investigate tense logic over well-founded linear time flows, i.e. over ordinals, which can be denoted as $\mathbf{K_tL_\ell.3}$ in the taxonomy of modal logics from [6]. We show in particular that

1. the satisfiability problem for $\mathbf{K_tL_\ell.3}$ over the class of ordinals is NP-complete, and that
2. a formula φ of $\mathbf{K_tL_\ell.3}$ has a well-founded linear model if and only if it has a model of order type α for some $\alpha < \omega \cdot (|\varphi| + 2)$; this should be contrasted with the corresponding $\omega^{|\varphi|+2}$ bound proven in [12, Cor. 3.3] for linear temporal logic.

These two results are however just byproducts of our main contribution, which is a sound and complete proof system for $\mathbf{K_tL_\ell.3}$ in which proof search runs in coNP.

All the algorithmic results for tense logic mentioned earlier in this introduction have been obtained via *model-theoretic* techniques, by showing that if a formula has a model, then it has a "small" one, and it is actually possible to proceed similarly for $\mathbf{K_tL_\ell.3}$. However, as the resulting algorithms consist essentially in guessing a model, they are impractical as they are unlikely to avoid the (high) worst case complexity of the problem. In the case of the full linear temporal logic, this has motivated the use of *automata-theoretic* techniques [36, 33, 8, 12], typically by building an at most exponential-sized automaton recognising the set of models of the formula: checking the language non-emptiness of the automaton can then be performed on-the-fly in PSPACE and can rely in practice on a rich algorithmic toolset. However, in the case of tense logic, it is not immediate how to tailor this approach to recover the above NP upper bound, because the automata for tense logic may require exponential-size – over ω, this is a consequence of the proof of [13, Thm. 3]. Finally, if one's interest is to check that a formula φ is valid, neither the model-theoretic nor the automata-theoretic approach yields a "natural" certificate that could be checked by simple independent means.

All these considerations motivate our use of *proof-theoretic* techniques. In their simplest form, those can be Hilbert-style axiomatisations which, in the context of modal logic, allow to characterise valid formulas in a way that is modular with respect to the considered classes of models – incidentally, the name $\mathbf{K_tL_\ell.3}$ refers to its axiomatisation (see Appendix A). However, these systems are not directly amenable to automated reasoning, which is rather achieved through more structured proof systems, the seminal example being Gentzen's sequent calculus. As the latter is often too limited for modal logics, it has been enriched in various ways, using e.g. labelled sequents [25], display calculus [5, 19], nested sequents [18, 7, 30, 29, 22], or hypersequents [2, 14, 20, 15]. These enriched formalisms remain quite modular and sustain extensions simply by adding a few rules. They can be exploited to provide optimal complexity solutions to the validity problem directly by proof search [17, 24, 4, 11, 3], which may sometimes avoid the worst-case complexity of the problem and rely in practice on various heuristics. Finally, this approach obviously yields a proof of validity as a certificate in case of success.

Our proof system for $\mathbf{K_tL_\ell.3}$ is obtained as a natural extension of our earlier work on $\mathbf{K_t4.3}$ [3], using additional insights from Avron's sequent calculus for **KL** [1]. This is satisfying since $\mathbf{K_tL_\ell.3}$ is simply obtained from $\mathbf{K_t4.3}$ – the tense logic of arbitrary linear time flows – by adding well-foundation to the left, i.e. towards the past (see Section 2), and completes the picture as $\mathbf{K_tQ}$ the tense logic of dense linear time flows was also handled in [3]. Specifically, we use the framework of ordered hypersequents *with clusters* introduced in [3] as an elaboration, with terminating proof search, of Indrzejczak's ordered hypersequent calculus for $\mathbf{K_t4.3}$ [15, 16]. Conceptually, re-using the framework required to generalise its semantics.

The new semantics is more uniform, and allows us to provide purely proof-theoretic soundness, completeness, and complexity arguments in Section 3, unlike in [3] where soundness builds on a model-theoretic result from [26].

Furthermore, our proof system is easily shown in Section 4 to also address the more precise problems of validity over all the well-founded linear time flows

- of order type $\beta < \alpha + 1$ for a given α, and
- of order type exactly $\alpha < \omega^2$.

Such a result seems out of reach of axiomatisations, and yields for instance a coNP decision procedure for validity over ω-words. Finally, using the exponential translation of $\mathrm{FO}^2(<)$ into tense logic given in [13, Thm. 2], our results yield an optimal NEXP upper bound for satisfiability of the former over ordinals, which was already known from [27]. But more importantly they yield a proof system for $\mathrm{FO}^2(<)$ over ordinals, which would be challenging to construct directly, because eigenvariables cannot be handled in the usual fashion.

2 Tense Logic over Ordinals

2.1 Syntax

Tense logic features two unary temporal operators, over a countable set Φ of propositional variables, with the following syntax:

$$\varphi ::= \bot \mid p \mid \varphi \supset \varphi \mid \mathsf{G}\,\varphi \mid \mathsf{H}\,\varphi \qquad \text{(where } p \in \Phi\text{)}$$

Formulæ $\mathsf{G}\,\varphi$ and $\mathsf{H}\,\varphi$ are called *modal formulæ*. Intuitively, $\mathsf{G}\,\varphi$ expresses that φ holds "globally" in all future worlds, while $\mathsf{H}\,\varphi$ expresses that φ holds "historically" in all past worlds. Other Boolean connectives may be encoded from $\bot$ and $\supset$, and as usual $\mathsf{F}\,\varphi = \neg\mathsf{G}\,\neg\varphi$ expresses that φ will hold "in the future" and $\mathsf{P}\,\varphi = \neg\mathsf{H}\,\neg\varphi$ that it held "in the past."

2.2 Ordinal Semantics

In the case of $\mathbf{K_tL_\ell.3}$, our formulæ shall be evaluated on Kripke *structures* $\mathfrak{M} = (\alpha, V)$, where α is an ordinal and $V : \Phi \to \wp(\alpha)$ is a valuation of the propositional variables. Recall that an ordinal α is seen set-theoretically as $\{\beta \in \mathrm{Ord} \mid \beta < \alpha\}$. An ordinal is either 0 (the empty linear order), a *limit* ordinal λ (such that for all $\beta < \lambda$ there exists γ with $\beta < \gamma < \lambda$), or a *successor* ordinal $\alpha + 1$.

Given a structure $\mathfrak{M} = (\alpha, V)$, we define the *satisfaction* relation $\mathfrak{M}, \beta \models \varphi$, where $\beta < \alpha$ and φ is a formula, by structural induction on φ:

$$\begin{aligned}
&\mathfrak{M}, \beta \not\models \bot \\
&\mathfrak{M}, \beta \models p && \text{iff } \beta \in V(p) \\
&\mathfrak{M}, \beta \models \varphi \supset \psi && \text{iff if } \mathfrak{M}, \beta \models \varphi \text{ then } \mathfrak{M}, \beta \models \psi \\
&\mathfrak{M}, \beta \models \mathsf{G}\,\varphi && \text{iff } \mathfrak{M}, \gamma \models \varphi \text{ for all } \beta < \gamma < \alpha \\
&\mathfrak{M}, \beta \models \mathsf{H}\,\varphi && \text{iff } \mathfrak{M}, \gamma \models \varphi \text{ for all } \gamma < \beta
\end{aligned}$$

When $\mathfrak{M}, \beta \models \varphi$, we say that $(\mathfrak{M}, \beta)$ is a *model* of φ.

▶ **Example 2.1.** The satisfiable formulæ of $\mathbf{K_tL_\ell.3}$ are strictly contained in the set of formulæ satisfiable in $\mathbf{K_t4.3}$, i.e. over arbitrary linear orders. For instance, the formula $\varphi_0 = \mathsf{P}\,p \wedge \mathsf{H}\,(p \supset \mathsf{P}\,p)$ is satisfiable in $\mathbf{K_t4.3}$ but not in $\mathbf{K_tL_\ell.3}$, because all its models must contain an infinite decreasing sequence of worlds where p is true. Moreover, $\mathbf{K_tL_\ell.3}$

can force models to be of order type greater than ω: for instance, the formula $\varphi_1 = \mathsf{G}(p \supset \mathsf{F}\,p) \wedge \mathsf{G}(\neg p \supset \mathsf{F}\,\neg p) \wedge \mathsf{F}\,\neg p \wedge \mathsf{F}(p \wedge \mathsf{G}\,p)$ forces to have a first infinite sequence of worlds not satisfying p, followed by a second infinite sequence of worlds satisfying p, and all its models (α, V) must have $\alpha \geq \omega \cdot 2$.

3 Hypersequents with Clusters

As is often the case with modal logics, Gentzen's sequent calculus does not provide a rich enough framework to obtain complete proof systems. The extension we consider is to use *hypersequents* [2], which are essentially sets of sequents logically interpreted as a disjunction. Indrzejczak has moved to *ordered* hypersequents [15, 16] (which are lists of hypersequents) to obtain a sound and complete calculus for $\mathbf{K_t4.3}$. We have further enriched the structure of his ordered hypersequents with *clusters* and annotations [3] to obtain a calculus for $\mathbf{K_t4.3}$ for which proof search terminates and, in fact, yields an optimal complexity decision procedure. We keep the same structure in the present work, but significantly adapt the proof rules, annotation mechanism, and even the semantics of hypersequents; we discuss these differences in more depth when concluding in Section 5. It should be noted that, unlike simple hypersequents, hypersequents with clusters do not have a translation as formulæ.

3.1 Annotated Hypersequents with Clusters

A *sequent* (denoted S) is a pair of two finite sets of formulæ, written $\Gamma \vdash \Delta$. It is satisfied in a world γ of a structure $\mathfrak{M}$ if, in that world, the conjunction of the formulæ of Γ implies the disjunction of the formulæ of Δ. In that case, we write $\mathfrak{M}, \gamma \models \Gamma \vdash \Delta$.

We define next the basic structure of our hypersequents, then enrich it with annotations to obtain the hypersequents that we shall work with.

▶ **Definition 3.1** (hypersequent)**.** A *hypersequent* is a list of *cells*, each cell being either a sequent or a non-empty list of sequents called a (syntactic) *cluster*. We shall use the following abstract syntax, where both operators ";" and "$\|$" are associative with unit "$\bullet$":

$$H ::= C \mid H\,;H \qquad \text{(hypersequents)}$$
$$C ::= \bullet \mid S \mid \{Cl\} \qquad \text{(cells)}$$
$$Cl ::= S \mid Cl \parallel Cl \qquad \text{(cluster contents)}$$

Note that this definition allows for empty cells and hypersequents "$\bullet$", but these notational conveniences will never arise in actual proofs – and should not be confused with the empty sequent "$\vdash$". We will see that the order of cells in a hypersequent is semantically relevant, but the order of sequents inside a cluster is not. Nevertheless, assuming an ordering as part of the syntactic structure of clusters is useful in order to refer to specific sequents or positions.

▶ **Definition 3.2.** An *annotated sequent* is a sequent that may be annotated with G formulæ. We simply write $\Gamma \vdash \Delta$ for a sequent carrying no annotation, otherwise we write, e.g., $\Gamma \vdash \Delta\ (\mathsf{G}\,\varphi, \mathsf{G}\,\psi, \ldots)$. Then, *annotated hypersequents* are hypersequents whose sequents are annotated, with the constraint that an annotation may only occur once in an annotated hypersequent, and that φ occurs on the right-hand side of sequents carrying the annotation $\mathsf{G}\,\varphi$. Formally, we can see annotations as partial functions from the set of G formulæ to the set of positions of the hypersequent.

▶ **Example 3.3.** For instance, $\Gamma \vdash \Delta, \varphi\ (\mathsf{G}\,\varphi)\,; \{\Pi \vdash \Sigma, \psi\ (\mathsf{G}\,\psi)\}$ is an annotated hypersequent but $\Gamma \vdash \Delta, \varphi, \psi\ (\mathsf{G}\,\varphi, \mathsf{G}\,\psi)\ ; \{\Pi \vdash \Sigma, \psi\ (\mathsf{G}\,\psi)\}$ is not allowed due to the two occurrences of $(\mathsf{G}\,\psi)$. Finally, $\vdash \bot\ (\mathsf{G}\,p)$ is not an annotated hypersequent as it fails the second condition.

Annotations will impact the semantics of hypersequents: intuitively, counter-models should attach to sequents annotated with $(\mathsf{G}\,\varphi)$ a world or set of worlds that invalidates φ and is (in a sense that will be made clear below) "rightmost" for that property.

3.2 Semantics

The semantics of an ordered hypersequent with clusters relies on a notion of embedding which we define next, building on a view of hypersequents as partially ordered structures.

▶ **Definition 3.4** (partial order of a hypersequent)**.** Let H be a hypersequent containing n sequents, counting both the sequents found directly in its cells and those in its clusters. In this context, any $i \in [1; n]$ is called a *position* of H, and we write $H(i)$ for the i-th sequent of H. We define a partial order $\precsim$ on the positions of H by setting $i \precsim j$ if and only if either the i-th and j-th sequents are in the same cluster, or the i-th sequent is in a cell that lies strictly to the left of the cell of the j-th sequent. We write $i \prec j$ when $i \precsim j$ but $j \not\precsim i$, i.e. j lies strictly to the right of i in H. We write $i \sim j$ when $i \precsim j \precsim i$. Finally, the *domain* of H is defined as $\mathrm{dom}(H) = ([1; n], \precsim)$; note that empty cells are ignored in $\mathrm{dom}(H)$.

While a hypersequent is syntactically a finite partial order, its semantics will refer to a linear well-founded order, obtained by "bulldozing" its clusters into copies of ω. The resulting order type is the object of the next definition.

▶ **Definition 3.5** (order type)**.** Let H be a hypersequent. We define its *order type* $o(H)$ by induction on its structure: for cells, $o(\bullet) = 0$, $o(S) = 1$, and $o(\{Cl\}) = \omega$, and for hypersequents, $o(H_1\,;H_2) = o(H_1) + o(H_2)$. Thus, $o(H) = \omega \cdot k + m$ where k is the number of clusters in H and m the number of non-empty cells to the right of the rightmost cluster.

▶ **Definition 3.6** (embedding)**.** Let H be an annotated hypersequent and α an ordinal. We say that $\mu : \mathrm{dom}(H) \to \alpha + 1 \setminus \{0\}$ is an *embedding* of H into α, written $H \hookrightarrow_\mu \alpha$, if:

- for all $i, j \in \mathrm{dom}(H)$, $i \prec j$ implies $\mu(i) < \mu(j)$ and $i \sim j$ implies $\mu(i) = \mu(j)$; and
- for all $i \in \mathrm{dom}(H)$, i is in a cluster if and only if $\mu(i)$ is a limit ordinal.

Observe that, if $H \hookrightarrow_\mu \alpha$, then $o(H) < \alpha + 1$.

▶ **Definition 3.7** (semantics)**.** Let $\mathfrak{M} = (\alpha, V)$ be a structure, H a hypersequent, and μ an embedding $H \hookrightarrow_\mu \alpha$. We say that μ is *annotation-respecting* if, for all φ and i such that $H(i)$ carries the annotation $(\mathsf{G}\,\varphi)$ and for all $\gamma < \alpha$ such that $\mathfrak{M}, \gamma \models \neg\varphi$, we have $\gamma < \mu(i)$.

We say that $(\mathfrak{M}, \mu)$ is a *model* of H, written $\mathfrak{M}, \mu \models H$, if μ is annotation-respecting and there exists a position i of H and an ordinal $\beta < \mu(i)$ such that for all γ with $\beta \leq \gamma < \mu(i)$ we have $\mathfrak{M}, \gamma \models H(i)$.

Following this definition, we say that a hypersequent is *valid* if for any $\mathfrak{M} = (\alpha, V)$ and annotation-respecting $H \hookrightarrow_\mu \alpha$, we have $\mathfrak{M}, \mu \models H$. A formula φ is valid in the usual sense (i.e., satisfied in every world of every ordinal structure) if and only if the hypersequent $\vdash \varphi$ is valid in our sense.

If a hypersequent H is not valid, then it has a *counter-model*, that is a structure $\mathfrak{M} = (\alpha, V)$ and an annotation-respecting embedding $H \hookrightarrow_\mu \alpha$ such that, for every $i \in \mathrm{dom}(H)$ and $\beta < \mu(i)$, there exists γ with $\beta \leq \gamma < \mu(i)$ such that $\mathfrak{M}, \gamma \not\models H(i)$. For the positions

$$(\mathsf{ax})\ \frac{}{H\,[\varphi, \Gamma \vdash \Delta, \varphi]} \qquad \frac{H\,[\varphi \supset \psi, \Gamma \vdash \Delta, \varphi] \qquad H\,[\varphi \supset \psi, \psi, \Gamma \vdash \Delta]}{H\,[\varphi \supset \psi, \Gamma \vdash \Delta]}\ (\supset \vdash)$$

$$(\bot)\ \frac{}{H\,[\Gamma, \bot \vdash \Delta]} \qquad \frac{H\,[\varphi, \Gamma \vdash \Delta, \psi, \varphi \supset \psi]}{H\,[\Gamma \vdash \Delta, \varphi \supset \psi]}\ (\vdash \supset)$$

Figure 1 Propositional rules of $\mathbf{HK_tL_\ell.3}$.

$i \in \mathrm{dom}(H)$ that are not in clusters, $\mu(i)$ is a successor ordinal $\gamma + 1$ and this amounts to asking that $\mathfrak{M}, \gamma \not\models H(i)$. When i is in a cluster, the condition implies the existence of an infinite increasing sequence $(\gamma_j)_j$ of ordinals with limit $\mu(i) = \sup_j \gamma_j$ such that $\mathfrak{M}, \gamma_j \not\models H(i)$ for all j.

3.3 Proof System

We now present our proof system for $\mathbf{K_tL_\ell.3}$, called $\mathbf{HK_tL_\ell.3}$. This system deals with annotated hypersequents; from now on, we simply call sequents and hypersequents their annotated versions. The rules of $\mathbf{HK_tL_\ell.3}$ are given in Figures 1 to 3: the first group includes the usual propositional rules, the second deals with modalities, and the last one with annotations. The figures make use of some notations which we explain next, before commenting on the rule definitions themselves.

Notations. First, we use hypersequents with *holes*. One-placeholder hypersequents, cells, and clusters are defined by the following syntax:

$$H\,[] ::= H\,;C\,[]\,;H \qquad C\,[] ::= \star \mid \{\, Cl[]\,\} \qquad Cl[] ::= Cl_\bullet \parallel \star \parallel Cl_\bullet \qquad Cl_\bullet ::= \bullet \mid Cl$$

Two-placeholder cells and hypersequents have two holes identified by $\star_1$ and $\star_2$:

$$H\,[]\,[] ::= H\,;C\,[]\,[]\,;H \mid H[\star_1]\,;H[\star_2] \quad C\,[]\,[] ::= \{\, Cl[\star_1] \parallel Cl[\star_2]\,\} \mid \{\, Cl[\star_2] \parallel Cl[\star_1]\,\}$$

As usual, $C\,[S]$ (resp. $C\,[Cl]$) denotes the same cell with S (resp. Cl) substituted for $\star$; two-placeholder cells and hypersequents with holes behave similarly. In terms of the partial orders underlying hypersequents with two holes, observe that the positions i and j associated resp. to $\star_1$ and $\star_2$ are such that $i \precsim j$.

Second, we do not write explicitly the annotations that sequents may carry in rule applications. These annotations are implicitly the same in a conclusion sequent and the corresponding sequents in premises, or updated by adding the explicit annotation; freshly created sequents always have an explicit annotation. Annotations can prevent a rule application if the addition of an annotation would break the single-annotation constraint.

Third, we use a convenient notation for *enriching* a sequent: if S is a sequent $\Gamma \vdash \Delta$ (A), then $S \ltimes (\Gamma' \vdash \Delta'\ (A'))$ is the sequent $\Gamma, \Gamma' \vdash \Delta, \Delta'$ (A, A'). Moreover, we sometimes need to enrich an arbitrary sequent of a cluster $\{Cl\}$ with a sequent S; then $\{Cl\} \ltimes S$ denotes the cluster with its leftmost sequent enriched.

Rules. We now comment on the definition of our rules. The propositional rules of Figure 1 are straightforward: they are the usual ones applied to an arbitrary sequent of the hypersequent. The left modal rules of Figure 2 should not be surprising. For instance, in $(\mathsf{G}\vdash)$, if the

$$(\mathsf{G}\vdash)\ \frac{H\,[\mathsf{G}\,\varphi, \Gamma \vdash \Delta]\,[\varphi, \mathsf{G}\,\varphi, \Pi \vdash \Sigma]}{H\,[\mathsf{G}\,\varphi, \Gamma \vdash \Delta]\,[\Pi \vdash \Sigma]} \qquad \frac{H_1; \{Cl_\bullet \parallel \varphi, \mathsf{G}\,\varphi, \Gamma \vdash \Delta \parallel Cl'_\bullet\}; H_2}{H_1; \{Cl_\bullet \parallel \mathsf{G}\,\varphi, \Gamma \vdash \Delta \parallel Cl'_\bullet\}; H_2}\ (\{\mathsf{G}\vdash\})$$

$$(\mathsf{H}\vdash)\ \frac{H\,[\varphi, \mathsf{H}\,\varphi, \Pi \vdash \Sigma]\,[\mathsf{H}\,\varphi, \Gamma \vdash \Delta]}{H\,[\Pi \vdash \Sigma]\,[\mathsf{H}\,\varphi, \Gamma \vdash \Delta]} \qquad \frac{H_1; \{Cl_\bullet \parallel \varphi, \mathsf{H}\,\varphi, \Gamma \vdash \Delta \parallel Cl'_\bullet\}; H_2}{H_1; \{Cl_\bullet \parallel \mathsf{H}\,\varphi, \Gamma \vdash \Delta \parallel Cl'_\bullet\}; H_2}\ (\{\mathsf{H}\vdash\})$$

$$\frac{\begin{array}{ll} H_1\,;C\,[\Gamma \vdash \Delta, \mathsf{G}\,\varphi]\,; \vdash \varphi\ (\mathsf{G}\,\varphi)\,; C'\,; H_2 & \\ H_1\,;C\,[\Gamma \vdash \Delta, \mathsf{G}\,\varphi]\,; \{\vdash \varphi\ (\mathsf{G}\,\varphi)\}\,; C'\,; H_2 & \\ H_1\,;C\,[\Gamma \vdash \Delta, \mathsf{G}\,\varphi \parallel \vdash \varphi\ (\mathsf{G}\,\varphi)]\,; C'\,; H_2 & \text{if } C \neq \star \\ H_1\,;C\,[\Gamma \vdash \Delta, \mathsf{G}\,\varphi]\,; C' \ltimes (\vdash \mathsf{G}\,\varphi)\,; H_2 & \text{if } C' \neq \bullet \\ H_1\,;C\,[\Gamma \vdash \Delta, \mathsf{G}\,\varphi]\,; C' \ltimes (\vdash \varphi\ (\mathsf{G}\,\varphi))\,; H_2 & \text{if } C' \neq \bullet \text{ and } C' \neq \{Cl\} \end{array}}{H_1\,;C\,[\Gamma \vdash \Delta, \mathsf{G}\,\varphi]\,; C'\,; H_2}\ (\vdash\mathsf{G})$$

$$\frac{\begin{array}{ll} H_2\,;C'\,; \mathsf{H}\,\varphi \vdash \varphi\,;C\,[\Gamma \vdash \Delta, \mathsf{H}\,\varphi]\,; H_1 & \\ H_2\,;C' \ltimes (\vdash \mathsf{H}\,\varphi)\,;C\,[\Gamma \vdash \Delta, \mathsf{H}\,\varphi]\,; H_1 & \text{if } C' \neq \bullet \\ H_2\,;C' \ltimes (\mathsf{H}\,\varphi \vdash \varphi)\,;C\,[\Gamma \vdash \Delta, \mathsf{H}\,\varphi]\,; H_1 & \text{if } C' \neq \bullet \text{ and } C' \neq \{Cl\} \end{array}}{H_2\,;C'\,;C\,[\Gamma \vdash \Delta, \mathsf{H}\,\varphi]\,; H_1}\ (\vdash\mathsf{H})$$

Figure 2 Modal rules of $\mathbf{HK_tL_\ell.3}$. In $(\vdash\mathsf{G})$ and $(\vdash\mathsf{H})$, we allow $C' = \bullet$ only when $H_2 = \bullet$.

$$((\mathsf{G}))\ \frac{}{H_1\,[\Gamma \vdash \Delta\ (\mathsf{G}\,\varphi)]\,; H_2\,[\Pi \vdash \Sigma, \mathsf{G}\,\varphi]} \qquad \frac{}{H_1\,; \Gamma \vdash \Delta, \mathsf{G}\,\varphi\ (\mathsf{G}\,\varphi)\,; H_2}\ (\{(\mathsf{G})\})$$

$$((\bar{\mathsf{G}}))\ \frac{H_1\,[\Gamma \vdash \Delta\ (\mathsf{G}\,\varphi)]\,; H_2\,[\Pi, \varphi \vdash \Sigma]}{H_1\,[\Gamma \vdash \Delta\ (\mathsf{G}\,\varphi)]\,; H_2\,[\Pi \vdash \Sigma]}$$

Figure 3 Annotation rules of $\mathbf{HK_tL_\ell.3}$.

conclusion has a counter-model, then $\mathsf{G}\,\varphi$ holds at some ordinal and thus both φ and $\mathsf{G}\,\varphi$ must also hold at strictly greater ordinals. The rule also applies to two distinct sequents inside the same cluster; the soundness proof below shows how this is covered in detail. The $(\{\mathsf{G}\vdash\})$ rule allows to proceed in the same way inside a cluster when the sequent "further to the right" is the original sequent itself, something that our notations do not allow in $(\mathsf{G}\vdash)$. Finally, $(\mathsf{H}\vdash)$ and $(\{\mathsf{H}\vdash\})$ are symmetric to the two previous rules.

The rules $(\vdash\mathsf{G})$ and $(\vdash\mathsf{H})$ are the most complex ones. We shall not try to justify their soundness at this point, but simply make a few remarks that are important to understand their definition. First, these rules are the only ones that may introduce new cells in hypersequents. In the case of $(\vdash\mathsf{G})$, new cells are annotated with the principal formula $\mathsf{G}\,\varphi$, which prevents another application of $(\vdash\mathsf{G})$ on $\mathsf{G}\,\varphi$ (otherwise a premise would carry this annotation at two positions). Second, the principal cell $C\,[\Gamma \vdash \Delta, \mathsf{G}\,\varphi]$ in $(\vdash\mathsf{G})$ may be the rightmost cell of the conclusion hypersequent, in which case both C' and H_2 are empty, and the rule has two or three premises depending on whether the principal cell is a cluster or not. When the principal cell is not rightmost, then C' is not allowed to be empty, and the rule has one or two extra premises depending on whether C' is a cluster or not. The situation is symmetric for $(\vdash\mathsf{H})$.

Finally, the special rules of Figure 3 are, again, best explained through the soundness proof: they correspond to situations that can be ruled out or simplified by taking into account

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{S_1\,;\,p\vdash p\;(\mathsf{G}\,p)\,;\{\mathsf{G}\,p,p\vdash\;(\mathsf{G}\,\varphi)\}}(\mathsf{ax}) \qquad \dfrac{}{S_1;\vdash p,\mathsf{G}\,p\;(\mathsf{G}\,p)\,;\{\mathsf{G}\,p,p\vdash\;(\mathsf{G}\,\varphi)\}}(\{(\mathsf{G})\})}{S_1\,;\,\mathsf{G}\,p\supset p\vdash p\;(\mathsf{G}\,p)\,;\{\mathsf{G}\,p,p\vdash\;(\mathsf{G}\,\varphi)\}}(\supset\vdash)}{\ldots\quad \mathsf{G}\,(\mathsf{G}\,p\supset p),\mathsf{G}\,(p\supset(\mathsf{G}\,(p\supset\bot)\supset\bot))\vdash\mathsf{G}\,p,\mathsf{G}\,\varphi\,;\vdash p\;(\mathsf{G}\,p)\,;\{\mathsf{G}\,p,p\vdash\;(\mathsf{G}\,\varphi)\}}(\mathsf{G}\vdash)\quad\ldots}(\vdash\mathsf{G})}{\ldots\quad \mathsf{G}\,(\mathsf{G}\,p\supset p),\mathsf{G}\,(p\supset(\mathsf{G}\,(p\supset\bot)\supset\bot))\vdash\mathsf{G}\,p,\mathsf{G}\,\varphi\,;\{\mathsf{G}\,p,p\vdash\;(\mathsf{G}\,\varphi)\}}(\vdash\mathsf{G})$$
$$\mathsf{G}\,(\mathsf{G}\,p\supset p),\mathsf{G}\,(p\supset(\mathsf{G}\,(p\supset\bot)\supset\bot))\vdash\mathsf{G}\,p,\mathsf{G}\,(p\supset\bot\vee(\mathsf{G}\,p\supset\bot))$$

Figure 4 In a proof of S_1 (Example 3.9) branches with a non-cluster cell for $(\mathsf{G}\,p)$ are provable.

the annotation-respecting nature of our semantics. These rules are important to be able to extract counter-models from proof search failures (i.e., sequents on which no rule applies).

Examples. We have designed our rules so that they are all *invertible*: by keeping in premises all the formulæ from the conclusion, we ensure that validity is never lost by applying a rule; this will be shown formally in Proposition 3.11. In practice, keeping all formulæ can be unnecessarily heavy. Fortunately, it is easy to see that the following weakening rules are admissible:

$$(\text{weak}\vdash)\;\dfrac{H\,[\Gamma\vdash\Delta]}{H\,[\Gamma,\varphi\vdash\Delta]}\qquad\dfrac{H\,[\Gamma\vdash\Delta]}{H\,[\Gamma\vdash\varphi,\Delta]}\;(\vdash\text{ weak})$$

▶ **Example 3.8.** The formula $\varphi_0=\mathsf{P}\,p\wedge\mathsf{H}\,(p\supset\mathsf{P}\,p)$ from Example 2.1 is not satisfiable in $\mathbf{K_tL_\ell.3}$, so the dual sequent $S_0=\mathsf{H}\,(p\supset(\mathsf{H}\,(p\supset\bot)\supset\bot))\vdash\mathsf{H}\,(p\supset\bot)$ is valid. Here is indeed a proof tree, with implicit uses of propositional and weakening rules, and principal formulæ shown in orange.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{H}\,(p\supset\bot),p\vdash p\,;S_0}(\mathsf{ax}) \qquad \dfrac{}{\mathsf{H}\,(p\supset\bot),p\vdash\mathsf{H}\,(p\supset\bot)\,;S_0}(\mathsf{ax})}{p\supset(\mathsf{H}\,(p\supset\bot)\supset\bot),\mathsf{H}\,(p\supset\bot),p\vdash\,;S_0}(\supset\vdash)}{\mathsf{H}\,(p\supset\bot),p\vdash\,;\mathsf{H}\,(p\supset(\mathsf{H}\,(p\supset\bot)\supset\bot))\vdash\mathsf{H}\,(p\supset\bot)}(\mathsf{H}\vdash)}{\mathsf{H}\,(p\supset(\mathsf{H}\,(p\supset\bot)\supset\bot))\vdash\mathsf{H}\,(p\supset\bot)}(\vdash\mathsf{H})$$

▶ **Example 3.9.** Since $\varphi_1=\mathsf{G}\,(p\supset\mathsf{F}\,p)\wedge\mathsf{G}\,(\neg p\supset\mathsf{F}\,\neg p)\wedge\mathsf{F}\,\neg p\wedge\mathsf{F}\,(p\wedge\mathsf{G}\,p)$ from Example 2.1 is satisfiable, its dual sequent $S_1=\mathsf{G}\,(\mathsf{G}\,p\supset p),\mathsf{G}\,(p\supset(\mathsf{G}\,(p\supset\bot)\supset\bot))\vdash\mathsf{G}\,p,\mathsf{G}\,\varphi$ where $\varphi=p\supset\bot\vee(\mathsf{G}\,p\supset\bot)$ is invalid, although with no counter-models below $\omega\cdot 2$.

In our calculus, proof search for S_1 will succeed on branches not considering at least two clusters; we show in Figure 4 one such branch, with implicit uses of propositional and weakening rules, and principal formulæ shown in orange.

However, proof search will fail on the branch shown in Figure 5, which corresponds to the counter-model described in Section 2.

3.4 Soundness

▶ **Proposition 3.10.** *The rules of* $\mathbf{HK_tL_\ell.3}$ *are sound: if the premises of a rule instance are valid, then so is its conclusion.*

Proof. We show the contrapositive: considering an application of a rule with a conclusion hypersequent H and a counter-model $(\mathfrak{M},\mu)$ of H with $\mathfrak{M}=(\alpha,V)$ and $H\hookrightarrow_\mu\alpha$, we provide a counter-model of one of the premises (or a contradiction when there is no premise).

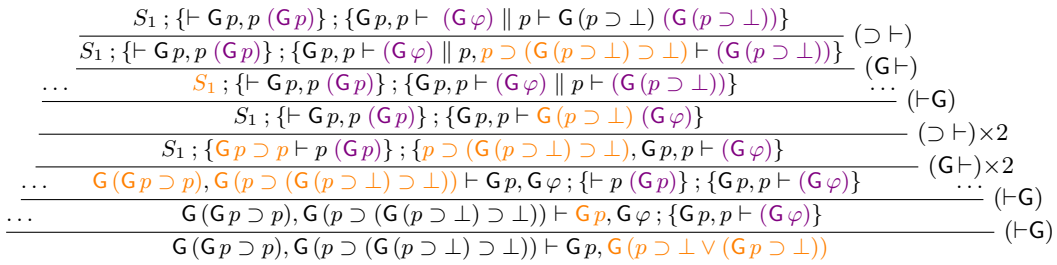


Figure 5 A failed branch in the proof of S_1 (Example 3.9).

Since we will often have to extend an embedding with a value for a new position, we define $\mu + (i \mapsto \alpha)$ as the mapping μ' such that $\mu'(i) = \alpha$, $\mu'(k) = \mu(k)$ for $k < i$ and $\mu'(k+1) = \mu(k)$ for $k \geq i$ in the domain of μ.

A full proof is given in Appendix B, and we cover here only a few key cases. Consider first an application of $(\vdash\mathsf{G})$ with $\Gamma \vdash \Delta, \mathsf{G}\,\varphi$ at position i, when $C'; H_2$ is empty. For any $\beta_i < \mu(i)$ there exists γ_i with $\beta_i \leq \gamma_i < \mu(i)$ such that $\mathfrak{M}, \gamma_i \not\models H(i)$, hence there also exists $\gamma_i' > \gamma_i$ such that $\mathfrak{M}, \gamma_i' \not\models \varphi$. Let γ be the least ordinal that contains all such γ_i'. We have that $\mu(i) \leq \gamma$.

- If $\mu(i) = \gamma$, then $\mu(i)$ must be a limit ordinal. Hence $C \neq \star$ and the third premise H_3' is available. We construct a counter-model $(\mathfrak{M}, \mu')$ for it by taking $\mu' = \mu + (k \mapsto \gamma)$, where $k = i + 1$ is the new position in H_3'. Indeed, we have that for any $\beta' < \mu'(k)$ there exists γ' with $\beta' \leq \gamma' < \mu'(k)$ and $\mathfrak{M}, \gamma' \not\models \varphi$ (the inequality can even be made strict). Moreover, the annotation is respected by definition of γ: there cannot be any $\lambda \geq \gamma$ such that $\mathfrak{M}, \lambda \not\models \varphi$.
- Otherwise we conclude by observing that $(\mathfrak{M}, \mu')$ is a counter-model of one of the first two premises with $\mu' = \mu + (k \mapsto \gamma)$ where k is the newly created position. We check that μ' is monotone, because $\mu(i) < \gamma$. If γ is a successor ordinal, $(\mathfrak{M}, \mu')$ is a counter-model of the first premise simply because the predecessor of γ invalidates φ and the annotation is respected; both hold by construction. If γ is a limit ordinal we have a counter-model $(\mathfrak{M}, \mu')$ of the second premise: we do have that for any $\beta' < \mu'(k)$ there exists γ' with $\beta' \leq \gamma' < \mu'(k)$ that invalidates φ, and the annotation is respected by construction.

When $C'; H_2$ is not empty, we need to consider whether γ is less than the ordinal to which the positions of C' are mapped by μ, and use the last two premises when it is not the case.

The case of $(\vdash\mathsf{H})$ is similar, but simpler in that we can take $\gamma = \lambda + 1$ where λ is the least ordinal such that $\mathfrak{M}, \lambda \not\models \varphi$. Finally, annotation rules (Figure 3) rely on the annotation-respecting condition on μ: informally, φ cannot be falsified at an ordinal beyond $\mu(i)$ when i carries the annotation $(\mathsf{G}\,\varphi)$, thus the conclusions of $((\mathsf{G}))$ and $(\{(\mathsf{G})\})$ cannot have counter-models, and φ must be satisfied at ordinals corresponding to $\Pi \vdash \Sigma$ for $((\bar{\mathsf{G}}))$. ◀

3.5 Completeness and Complexity

As in [3], completeness is a by-product of the very simple proof-search behaviour of our calculus. As we shall see, all the rules are invertible and proof search branches are polynomially bounded, as long as obvious pitfalls are avoided in the search strategy. Thus it is useless to backtrack during proof-search. Moreover, proof attempts result in finite (polynomial depth) partial proofs, whose unjustified leaves yield counter-models that amount (by invertibility) to counter-models of the conclusion. Hence the completeness of our calculus. We detail this argument below, and its corollary: proof-search yields an optimal coNP procedure for validity.

$$\dfrac{\dfrac{\dfrac{(\mathsf{ax})}{\mathsf{H}\,a, a \vdash a \,;\, \mathsf{H}\,b \vdash b \,;\, \mathsf{H}\,a \vdash a \,;\, \vdash \mathsf{H}\,a, \mathsf{H}\,b}}{\mathsf{H}\,a \vdash a \,;\, \mathsf{H}\,b \vdash b \,;\, \mathsf{H}\,a \vdash a \,;\, \vdash \mathsf{H}\,a, \mathsf{H}\,b}\,(\mathsf{H}\vdash) \qquad \dfrac{b, \mathsf{H}\,a \vdash a \,;\, \mathsf{H}\,b \vdash b, \mathsf{H}\,a \,;\, \vdash \mathsf{H}\,a, \mathsf{H}\,b}{\mathsf{H}\,a \vdash a \,;\, \mathsf{H}\,b \vdash b, \mathsf{H}\,a \,;\, \vdash \mathsf{H}\,a, \mathsf{H}\,b}\,(\mathsf{H}\vdash) \quad \cdots}{\mathsf{H}\,a \vdash a \,;\, \mathsf{H}\,b \vdash b \,;\, \vdash \mathsf{H}\,a, \mathsf{H}\,b \qquad \cdots}\,(\vdash\mathsf{H})}{\mathsf{H}\,a \vdash a \,;\, \vdash \mathsf{H}\,a, \mathsf{H}\,b}\,(\vdash\mathsf{H})}{\vdash \mathsf{H}\,a, \mathsf{H}\,b}\,(\vdash\mathsf{H})$$

Figure 6 Proof search with a failure hypersequent and an immediately provable hypersequent.

▶ **Proposition 3.11** (invertibility). *In any rule instance, if a premise has a counter-model, then so does its conclusion.*

Proof. Considering a rule instance with a counter-model $(\mathfrak{M}, \mu)$ of a premise H, we build a counter-model $(\mathfrak{M}, \mu')$ of the conclusion H'. Depending on the rule that is applied, H and H' will either have exactly the same structure, or H will have a new cell. Accordingly, we take μ' to be the restriction of μ to the positions of H' (and adapt it accordingly for the positions that have been shifted). It is indeed a proper annotation-respecting embedding of H' into $\mathfrak{M}$. It is then easy to see that $(\mathfrak{M}, \mu')$ is a counter-model of H', since any sequent $H'(i)$ is contained in the corresponding sequent $H(j)$: $\mathfrak{M}, \mu(j) \not\models H(j)$ implies $\mathfrak{M}, \mu'(i) \not\models H'(i)$. ◀

We characterise next the proof attempts that we consider for proof search, and show how to extract counter-models when such attempts fail.

▶ **Definition 3.12.** We say that a sequent is *immediately provable* if it is provable by an application of $(\mathsf{H}\vdash)$ or $(\{\mathsf{H}\vdash\})$ followed by (ax). We call *partial proof* a finite derivation tree whose internal nodes correspond to rule applications, but whose leaves may be unjustified hypersequents, and that satisfies two conditions: any rule application should be such that all premises differ from the conclusion; immediately provable sequents must be proven through (ax) and $(\mathsf{H}\vdash)$ or $(\{\mathsf{H}\vdash\})$. Finally, we call *failure hypersequent* a hypersequent that can only be the conclusion of a rule instance when it is also one of its premises.

Obviously, a hypersequent has a proof if and only if it has a partial proof without unjustified leaves. The two conditions on partial proofs amount to a simple proof search strategy that avoids loops. The second one addresses specifically loops arising from repeated applications of $(\vdash\mathsf{H})$, in branches where several new cells are created for the same $\mathsf{H}\,\varphi$ formula: this results in two cells of the form $\Gamma, \mathsf{H}\,\varphi \vdash \varphi, \Delta$ and thus in an immediately provable hypersequent. This is seen, for example, in the first premise of the third application of rule $(\vdash\mathsf{H})$ in Figure 6. Finally, failure hypersequents correspond to points where proof search is stuck, as with the unjustified hypersequent of Figure 6. We show next that such hypersequents are invalid.

▶ **Proposition 3.13.** *Any failure hypersequent H has a counter-model.*

Proof sketch, details in Appendix B. We construct a counter-model over $\alpha = o(H)$, taking μ as the only possible embedding, notably satisfying $\mu(i) = \omega \cdot k$ if i belongs to the k-th cluster of H and $\mu(i) = \omega \cdot k + m$ if i is the m-th cell between the k-th and the next cluster (if any). We can then take a function $\mathrm{pos} : \alpha \to \mathrm{dom}(H)$ which maps worlds $\beta < \alpha$ to positions of H in a way that respects the partial order induced by H. For a position i that does not belong to a cluster, $\mathrm{pos}(\beta) = i$ if and only if β is the predecessor of $\mu(i)$. A position i appearing in a cluster must correspond to an infinite sequence of ordinals of limit $\mu(i)$, so that for all $i \sim j$ and β, if $\mathrm{pos}(\beta) = i$ then there exists γ with $\beta < \gamma < \mu(i) = \mu(j)$ such that $\mathrm{pos}(\gamma) = j$; informally, this ensures that positions i and j inside a cluster are

"infinitely interleaved" within $\mu(i) = \mu(j)$. For example, for the unjustified hypersequent of Figure 5, we could set $\mathrm{pos}(0) = 1$, $\mathrm{pos}(i) = 2$ for all other $i < \omega$, $\mathrm{pos}(\omega + 2j) = 3$ and $\mathrm{pos}(\omega + 2j + 1) = 4$ for all $j \geq 0$. We finally define the valuation $V : \Phi \to \wp(\alpha)$ by $V(p) = \{\beta < \alpha \mid \exists \Gamma, \Delta \,.\, H(\mathrm{pos}(\beta)) = (p, \Gamma \vdash \Delta)\}$ and let $\mathfrak{M} = (\alpha, V)$.

We claim that $\mathfrak{M}, \gamma \not\models H(\mathrm{pos}(\gamma))$ for all $\gamma < \alpha$: we prove by induction on ψ that, if ψ appears in the left-hand (resp. right-hand) side of $H(\mathrm{pos}(\gamma))$, then $\mathfrak{M}, \gamma \vDash \psi$ (resp. $\mathfrak{M}, \gamma \nvDash \psi$). Most cases follow a standard argument, we only detail the one where $\psi = \mathsf{G}\,\varphi$ occurs on the right of $H(\mathrm{pos}(\gamma))$. Since ($\vdash\mathsf{G}$) does not apply, an annotation must already exist for $\mathsf{G}\,\varphi$. By rules ((G)) and ({(G)}) this annotation must be on a position i such that $\mathrm{pos}(\gamma) \precsim i$. By definition of annotations, φ occurs on the right of $H(i)$. Hence, there exists $\gamma' > \gamma$ such that $i = \mathrm{pos}(\gamma')$, and $\mathfrak{M}, \gamma' \nvDash \varphi$, thus $\mathfrak{M}, \gamma \nvDash \mathsf{G}\,\varphi$.

From there we can check that $H \hookrightarrow_\mu \alpha$, and the rule ($\bar{(\mathsf{G})}$) enforces that μ is annotation-respecting. It is then easy to conclude that $(\mathfrak{M}, \mu)$ is a counter-model of H. ◀

We now turn to establishing that proof search terminates, and always produces branches of polynomial length. For a hypersequent H, let $\mathsf{len}(H)$ be its number of sequents (i.e., the size of $\mathrm{dom}(H)$), and $|H|$ the number of distinct subformulæ occurring in H.

▶ **Lemma 3.14** (small branch property). *For any partial proof of a hypersequent H, any branch of the proof is of length at most $2(|H| + \mathsf{len}(H) + 1) \cdot |H|$.*

Proof. Let H be a hypersequent, $\mathcal{P}$ a partial proof of it, and B a branch of $\mathcal{P}$. Remark that the number of positions in hypersequents of β is bounded by $|H| + \mathsf{len}(H) + 1$: we have at most $\mathsf{len}(H)$ positions initially, and a new position may only be created once per modal formula among at most $|H|$ formulæ plus possibly one more (overall) to create an immediately provable hypersequent. This is by definition of the annotation system for G formulæ, and because a second cell created by ($\vdash\mathsf{H}$) on the same $\mathsf{H}\,\varphi$ would belong to an immediately provable sequent. Any rule application adds some subformulæ among $|H|$ to the left or to the right of the turnstile at a position among $|H| + \mathsf{len}(H) + 1$, hence with $2(|H| + \mathsf{len}(H) + 1) \cdot |H|$ choices. Thus B is of length at most $2(|H| + \mathsf{len}(H) + 1) \cdot |H|$. ◀

We conclude that $\mathbf{HK_tL_\ell.3}$ is complete, and also enjoys optimal complexity proof search.

▶ **Theorem 3.15** (completeness). *Every valid hypersequent H has a proof in $\mathbf{HK_tL_\ell.3}$.*

Proof. Assume that H is not provable. Consider a partial proof $\mathcal{P}$ of H that cannot be expanded any more: its leaves cannot be obtained as the conclusion of a rule instance. Such a partial proof exists by Lemma 3.14. Any unjustified leaf of that partial proof has a counter-model by Proposition 3.13, and by invertibility it is also a counter-model of H. ◀

▶ **Proposition 3.16.** *Proof search in $\mathbf{HK_tL_\ell.3}$ is in* coNP.

Proof. Proof search can be implemented in an alternating Turing machine maintaining the current hypersequent on its tape, where existential states choose which rule to apply (and how) and universal states choose a premise of the rule. By Lemma 3.14, the computation branches are of length bounded by a polynomial. By Proposition 3.11, the non-deterministic choices in existential states can be replaced by arbitrary deterministic choices, thus the resulting Turing machine has only universal states, hence is in coNP. ◀

4 Logic on Given Ordinals

We have designed a proof system that is sound and complete for $\mathbf{K_tL_\ell.3}$, and enjoys optimal complexity proof search. We now show that this system can easily be enriched to obtain decision procedures not only for tense logic over arbitrary ordinals, but also for tense logic over specific ordinals. We first observe that the logic can only distinguish ordinals up to ω^2, which should be contrasted with [12]. Then we show how to capture validity over ordinals below some $\omega \cdot k + m$, and finally how to reason over a specific ordinal of this form.

▶ **Proposition 4.1** (small model property)**.** *If a hypersequent H has a counter-model, then it has a counter-model of order type $\alpha < \omega \cdot (|H| + \mathsf{len}(H) + 1)$.*

Proof. This is a corollary of Theorem 3.15. By the proof of Lemma 3.14, the hypersequents in a failure hypersequent – which are not immediately provable – have at most $|H| + \mathsf{len}(H)$ non-empty cells. The counter-model extracted in Proposition 3.13 from a failure hypersequent H' is over $o(H') < \omega \cdot (|H| + \mathsf{len}(H) + 1)$. A counter-model for H is then obtained by Proposition 3.11, with a different embedding but the same structure. ◀

In particular, for a formula φ, the hypersequent $H = \vdash \varphi$ has $|H| = |\varphi|$ and $\mathsf{len}(H) = 1$, hence the $\omega \cdot (|\varphi| + 2)$ bound announced in the introduction.

Next we observe that we can easily enrich our calculus to obtain a proof system for tense logic over ordinals below a certain type α.

▶ **Proposition 4.2.** *Let α be an ordinal. The proof system $\mathbf{HK_tL_\ell.3}$ enriched with the following axiom is sound and complete for tense logic over ordinals $\beta \leq \alpha$:*

$$\frac{}{H}\ (\mathsf{ord}_\alpha)\quad \text{if } o(H) > \alpha$$

Proof. The soundness argument for the rules of $\mathbf{HK_tL_\ell.3}$ (Proposition 3.10) carries over to the restricted semantics, since the underlying structure (and ordinal) is never modified in the argument. Conversely, the completeness argument of Theorem 3.15 can be strengthened because, thanks to the new rule, we can guarantee that any failure hypersequent H is such that $o(H) \leq \alpha$, hence the extracted counter-model is also below this bound. ◀

▶ **Example 4.3.** When extending $\mathbf{HK_tL_\ell.3}$ to check for validity below ω, the failing branch of Figure 5 can be completed, as well as the other failing branches since they all involve hypersequents of order type $\omega \cdot 2$, and S_1 becomes provable.

We finally show how to capture validity at a fixed ordinal $\alpha < \omega^2$. The basic idea is to start with a hypersequent H such that $o(H) = \alpha = \omega \cdot k + m$ for some finite k and m, and take rule (ord_α) to forbid larger ordinals. The only catch is that we should check that the formula of interest in valid in all possible positions. Let us write $\{\vdash\}^k$ for $\{\vdash\}; \cdots ; \{\vdash\}$ with k clusters containing the empty sequent, and $(\vdash)^m$ for $\vdash; \cdots ; \vdash$ with m cells containing the empty sequent.

▶ **Proposition 4.4.** *The formula φ is valid in all structures of order type exactly $\alpha = \omega \cdot k + m$ if and only if $\mathbf{HK_tL_\ell.3}$ extended with (ord_α) proves all hypersequents of the form*

$$\{\vdash\}^{k_1} ; \vdash \varphi ; \{\vdash\}^{k_2} ; (\vdash)^m \quad \text{and} \quad \{\vdash\}^{k} ; (\vdash)^{m_1} ; \vdash \varphi ; (\vdash)^{m_2}$$

where $k_1 + k_2 = k$, $k_2 > 0$ and $m_1 + m_2 = m - 1$. In other words, one must consider all hypersequents H containing one sequent $\vdash \varphi$ and otherwise only empty sequents, and such that $o(H) = \omega \cdot k + m$.

For instance, when $k = m = 0$, φ vacuously holds in all worlds of $(0, V)$. When $k = 0$ and $m = 1$ we are checking $\vdash \varphi$ only, and (ord_α) closes any branch where a new cell is created, rendering modal formulæ trivially true. When $k = 1$ and $m = 0$ we are checking $\vdash \varphi \,;\{\vdash\}$.

Proof. If φ holds in all worlds of all structures of the form (α, V) for some V, the hypersequents are valid and thus provable in $\mathbf{HK_tL_\ell.3}$ with (ord_α). We prove the converse by contradiction. Assume that all the hypersequents hold and $\mathfrak{M}, \beta \not\models \varphi$ for some $\mathfrak{M} = (\alpha, V)$ and $\beta < \alpha$. If $\omega \cdot k_1 \leq \beta < \omega \cdot (k_1 + 1)$ with $k_1 + 1 \leq k$ we can build an embedding to obtain a counter-model of the first kind of sequent. Otherwise, $\omega \cdot k \leq \beta < \omega \cdot k + m$ and we derive a counter-model of the second kind of sequent. ◀

▶ **Example 4.5.** Consider the formula $\mathsf{G}\,\varphi$ for $\varphi = \mathsf{G}\,\bot \supset \bot$. We cannot prove $\mathsf{G}\,\varphi$ in general, since this formula is not satisfied over finite ordinals, as witnessed by the following partial proof and its failure hypersequent (in the left branch) corresponding to a counter-model over the ordinal 2:

$$\dfrac{(\vdash \supset)\ \dfrac{\vdash \mathsf{G}\,\varphi\,;\ \mathsf{G}\,\bot \vdash \bot, \varphi\ (\mathsf{G}\,\varphi)}{\vdash \mathsf{G}\,\varphi;\ \vdash \varphi\ (\mathsf{G}\,\varphi)} \qquad \dfrac{\dfrac{\dfrac{}{\vdash \mathsf{G}\,\varphi\,;\{\ \mathsf{G}\,\bot, \bot \vdash \bot, \varphi\ (\mathsf{G}\,\varphi)\}}(\bot)}{\vdash \mathsf{G}\,\varphi\,;\{\ \mathsf{G}\,\bot \vdash \bot, \varphi\ (\mathsf{G}\,\varphi)\}}(\{\mathsf{G}\vdash\})}{\vdash \mathsf{G}\,\varphi\,;\{\vdash \varphi\ (\mathsf{G}\,\varphi)\}}(\vdash \supset)}}{\vdash \mathsf{G}\,\varphi}(\vdash \mathsf{G})$$

According to Proposition 4.4, over $\alpha = \omega$, i.e., $k = 1$ and $m = 0$, we need to prove $\vdash \mathsf{G}\,\varphi\,;\{\vdash\}$ in $\mathbf{HK_tL_\ell.3}$ extended with (ord_ω), for which the presence of the cluster will be crucial. The extra rule (ord_ω) is actually not necessary in this case, but simplifies the proof. We start with an application of $(\vdash\mathsf{G})$, this time with three premises:

$$\dfrac{\vdash \mathsf{G}\,\varphi;\ \vdash \varphi\ (\mathsf{G}\,\varphi)\,;\{\vdash\} \qquad \vdash \mathsf{G}\,\varphi\,;\{\vdash \varphi\ (\mathsf{G}\,\varphi)\}\,;\{\vdash\} \qquad \vdash \mathsf{G}\,\varphi\,;\{\vdash \mathsf{G}\,\varphi\}}{\vdash \mathsf{G}\,\varphi\,;\{\vdash\}}(\vdash \mathsf{G})$$

The first premise is derived as follows:

$$\dfrac{\dfrac{\dfrac{}{\vdash \mathsf{G}\,\varphi;\ \mathsf{G}\,\bot \vdash \bot, \varphi\ (\mathsf{G}\,\varphi)\,;\{\bot \vdash\}}(\bot)}{\vdash \mathsf{G}\,\varphi;\ \mathsf{G}\,\bot \vdash \bot, \varphi\ (\mathsf{G}\,\varphi)\,;\{\vdash\}}(\mathsf{G}\vdash)}{\vdash \mathsf{G}\,\varphi;\ \vdash \varphi\ (\mathsf{G}\,\varphi)\,;\{\vdash\}}(\vdash \supset)$$

The middle premise can simply be discharged by (ord_ω). For the last premise, we use $(\vdash\mathsf{G})$ inside the cluster, which yields three premises: $\vdash \mathsf{G}\,\varphi\,;\{\vdash \mathsf{G}\,\varphi\}\,;\vdash \varphi\ (\mathsf{G}\,\varphi)$ and $\vdash \mathsf{G}\,\varphi\,;\{\vdash \mathsf{G}\,\varphi\}\,;\{\vdash \varphi\ (\mathsf{G}\,\varphi)\}$ are discharged by (ord_ω), while the last one is derived as follows:

$$\dfrac{\dfrac{\dfrac{}{\vdash \mathsf{G}\,\varphi\,;\{\bot \vdash \mathsf{G}\,\varphi \parallel \mathsf{G}\,\bot \vdash \bot, \varphi\ (\mathsf{G}\,\varphi)\}}(\bot)}{\vdash \mathsf{G}\,\varphi\,;\{\vdash \mathsf{G}\,\varphi \parallel \mathsf{G}\,\bot \vdash \bot, \varphi\ (\mathsf{G}\,\varphi)\}}(\mathsf{G}\vdash)}{\vdash \mathsf{G}\,\varphi\,;\{\vdash \mathsf{G}\,\varphi \parallel \vdash \varphi\ (\mathsf{G}\,\varphi)\}}(\vdash \supset)$$

5 Related Work and Conclusion

We have designed the first proof system for $\mathbf{K_tL_\ell.3}$, i.e. tense logic over ordinals. Thanks to Indrzejczak's ordered hypersequents [15], enriched with clusters and annotations as in [3], our system enjoys optimal complexity proof search, allows to derive small model properties, and can be extended into a proof system for variants of the logic over bounded or fixed ordinals.

Our ($\vdash \mathsf{H}$) rule is broadly related to the rule that Avron uses in his system for **KL** [1]. Unlike Avron, we cannot work with standard sequents due to the presence of converse modalities. In turn, this allows us to consider a somewhat simpler right introduction rule for H, which does not have to take into account $\mathsf{H}\,\Gamma$ antecedents as they will remain available in the principal cell when a new one is created.

The system most closely related to $\mathbf{HK_tL_\ell.3}$ is obviously the calculus for $\mathbf{K_t4.3}$ [3] in which we introduced the notions of clusters and annotations. These were inspired by the small model property of $\mathbf{K_t4.3}$ [26], and it is notable that we could put them to work in the considerably richer setting of $\mathbf{K_tL_\ell.3}$; it is the main technical contribution of the present paper. In retrospect, we believe that it is possible to present the semantics of $\mathbf{HK_tL_\ell.3}$ hypersequents as a particular case of $\mathbf{HK_t4.3}$ hypersequents: the semantics $\mu(i)$ of a position in a cluster would be infinite to the left and right for $\mathbf{HK_t4.3}$, but only infinite to the right for $\mathbf{HK_tL_\ell.3}$. This shift of perspective, together with the addition of rule ($(\bar{\mathsf{G}})$), allows to get rid of the somewhat awkward use of different semantics for the soundness and completeness of $\mathbf{HK_t4.3}$. It also frees the proof-theoretic development from the small model property; in fact, proof theory then allows to derive the small model property just as precisely. Of course, there are also fundamental differences between $\mathbf{HK_tL_\ell.3}$ and $\mathbf{HK_t4.3}$: well-foundedness allows us to take $\mathsf{H}\,\varphi$ assumptions in rule ($\vdash \mathsf{H}$), which renders ($\mathsf{H}\,\varphi$) annotations useless; this benefit of well-foundedness for proof search is usual [1, 4].

References

1 Arnon Avron. On Modal Systems Having Arithmetical Interpretations. *Journal of Symbolic Logic*, 49(3):935–942, 1984. doi:10.2307/2274147.

2 Arnon Avron. Hypersequents, logical consequence and intermediate logics for concurrency. *Annals of Mathematics and Artificial Intelligence*, 4(3–4):225–248, 1991. doi:10.1007/BF01531058.

3 David Baelde, Anthony Lick, and Sylvain Schmitz. A Hypersequent Calculus with Clusters for Linear Frames. In *Proc. AiML 2018*, volume 12 of *Advances in Modal Logic*, pages 36–55. College Publications, 2018. To appear. https://hal.inria.fr/hal-01756126.

4 David Baelde, Simon Lunel, and Sylvain Schmitz. A Sequent Calculus for a Modal Logic on Finite Data Trees. In *Proc. CSL 2016*, volume 62 of *Leibniz International Proceedings in Informatics*, pages 32:1–32:16. LZI, 2016. doi:10.4230/LIPIcs.CSL.2016.32.

5 Nuel D. Belnap. Display logic. *Journal of Philosophical Logic*, 11(4):375–417, 1982.

6 Patrick Blackburn, Marteen de Rijke, and Yde Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2001.

7 Kai Brünnler. Deep sequent systems for modal logic. *Archiv für Mathematische Logik und Grundlagenforschung*, 48(6):551–577, 2009. doi:10.1007/s00153-009-0137-3.

8 Véronique Bruyère and Olivier Carton. Automata on Linear Orderings. *J. Comput. Syst. Sci.*, 73(1):1–24, February 2007. doi:10.1016/j.jcss.2006.10.009.

9 Nino B. Cocchiarella. *Tense and Modal Logic: a Study in the Topology of Temporal Reference*. PhD thesis, University of California, Los Angeles, 1965.

10 Julien Cristau. Automata and temporal logic over arbitrary linear time. In *Proc. FSTTCS 2009*, volume 4 of *Leibniz International Proceedings in Informatics*, pages 133–144. LZI, 2009. doi:10.4230/LIPIcs.FSTTCS.2009.2313.

11 Anupam Das and Damien Pous. A Cut-Free Cyclic Proof System for Kleene Algebra. In *Proc. Tableaux 2017*, volume 10501 of *Lecture Notes in Computer Science*, pages 261–277. Springer, 2017. doi:10.1007/978-3-319-66902-1_16.

12 Stephane Demri and Alexander Rabinovich. The complexity of linear-time temporal logic over the class of ordinals. *Logical Methods in Computer Science*, 6(4), 2010. doi:10.2168/LMCS-6(4:9)2010.

13 Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-Order Logic with Two Variables and Unary Temporal Logic. *Information and Computation*, 179(2):279–295, 2002. doi:10.1006/inco.2001.2953.

14 Andrzej Indrzejczak. Cut-free hypersequent calculus for S4.3. *Bulletin of the Section of Logic*, 41(1/2):89–104, 2012.

15 Andrzej Indrzejczak. Linear Time in Hypersequent Framework. *Bulletin of Symbolic Logic*, 22(1):121–144, 2016. doi:10.1017/bsl.2016.2.

16 Andrzej Indrzejczak. Cut Elimination Theorem for Non-Commutative Hypersequent Calculus. *Bulletin of the Section of Logic*, 46(1/2):135–149, 2017. doi:10.18778/0138-0680.46.1.2.10.

17 Max Kanovich. The multiplicative fragment of linear logic is NP-complete. Technical Report X-91-13, Institute for Language, Logic, and Information, 1991.

18 Ryo Kashima. Cut-free sequent calculi for some tense logics. *Studia Logica*, 53(1):119–135, 1994. doi:10.1007/BF01053026.

19 Marcus Kracht. Power and weakness of the modal display calculus. In *Proof Theory of Modal Logic*, pages 93–121. Springer, 1996.

20 Hidenori Kurokawa. Hypersequent Calculi for Modal Logics Extending S4. In *JSAI-isAI 2013*, volume 8417 of *Lecture Notes in Computer Science*, pages 51–68. Springer, 2014. doi:10.1007/978-3-319-10061-6_4.

21 François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. Temporal logic with forgettable past. In *Proc. LICS 2002*, 2002. doi:10.1109/LICS.2002.1029846.

22 Björn Lellmann. Linear Nested Sequents, 2-Sequents and Hypersequents. In *Proc. Tableaux 2015*, volume 9323 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 2015. doi:10.1007/978-3-319-24312-2_10.

23 Orna Lichtenstein, Amir Pnueli, and Lenore D. Zuck. The glory of the past. In *Proc. Workshop on Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218. Springer, 1985. doi:10.1007/3-540-15648-8_16.

24 Patrick Lincoln, John Mitchell, Andre Scedrov, and Natarajan Shankar. Decision problems for propositional linear logic. *Annals of Pure and Applied Logic*, 56(1–3):239–311, 1992. doi:10.1016/0168-0072(92)90075-B.

25 Sara Negri. Proof Analysis in Modal Logic. *Journal of Philosophical Logic*, 34(5):507–544, 2005. doi:10.1007/s10992-005-2267-3.

26 Hiroakira Ono and Akira Nakamura. On the size of refutation Kripke models for some linear modal and tense logics. *Studia Logica*, 39(4):325–333, 1980. doi:10.1007/BF00713542.

27 Martin Otto. Two Variable First-Order Logic Over Ordered Domains. *Journal of Symbolic Logic*, 66(2):685–702, 2001. doi:10.2307/2695037.

28 Amir Pnueli. The temporal logic of programs. In *FOCS 1977*, pages 46–57. IEEE, 1977. doi:10.1109/SFCS.1977.32.

29 Francesca Poggiolesi. A purely syntactic and cut-free sequent calculus for the modal logic of provability. *The Review of Symbolic Logic*, 2(4):593–611, 2009. doi:10.1017/S1755020309990244.

30 Francesca Poggiolesi. The Method of Tree-Hypersequents for Modal Propositional Logic. In *Proc. Trends in Logic IV*, volume 28 of *Trends in Logic*, pages 31–51. Springer, 2009. doi:10.1007/978-1-4020-9084-4_3.

31 Arthur N. Prior. *Time and Modality*. Oxford University Press, 1957.

32 Alexander Rabinovich. Temporal logics over linear time domains are in PSPACE. *Information and Computation*, 210:40–67, 2012. doi:10.1016/j.ic.2011.11.002.

33 Gareth S. Rohde. *Alternating Automata and the Temporal Logic of Ordinals*. PhD thesis, University of Illinois at Urbana-Champaign, 1997.

34 A. Prasad Sistla and Edmund M. Clarke. The Complexity of Propositional Linear Temporal Logics. *J. ACM*, 32(3):733–749, 1985. doi:10.1145/3828.3837.

35 A. Prasad Sistla and Lenore D. Zuck. Reasoning in a Restricted Temporal Logic. *Information and Computation*, 102(2):167–195, 1993. doi:10.1006/inco.1993.1006.

36 Moshe Y Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *LICS 1986*, pages 322–331. IEEE, 1986.

A Axiomatisation

For reference, the logic $\mathbf{K_tL_\ell.3}$ can also be defined as the set of theorems generated by necessitation, modus ponens and substitution from classical tautologies and the following axioms [6, Ch. 4]:

$$\mathsf{G}(p \supset q) \supset (\mathsf{G}\,p \supset \mathsf{G}\,q) \qquad (\mathbf{K_r})$$

$$\mathsf{H}(p \supset q) \supset (\mathsf{H}\,p \supset \mathsf{H}\,q) \qquad (\mathbf{K_\ell})$$

$$p \supset \mathsf{G}\,\mathsf{P}\,p \qquad (\mathbf{t_r})$$

$$p \supset \mathsf{H}\,\mathsf{F}\,p \qquad (\mathbf{t_\ell})$$

$$\mathsf{F}\,p \wedge \mathsf{F}\,q \supset \mathsf{F}(p \wedge \mathsf{F}\,q) \vee \mathsf{F}(p \wedge q) \vee \mathsf{F}(q \wedge \mathsf{F}\,p) \qquad (\mathbf{.3_r})$$

$$\mathsf{P}\,p \wedge \mathsf{P}\,q \supset \mathsf{P}(p \wedge \mathsf{P}\,q) \vee \mathsf{P}(p \wedge q) \vee \mathsf{P}(q \wedge \mathsf{P}\,p) \qquad (\mathbf{.3_\ell})$$

$$\mathsf{H}(\mathsf{H}\,\phi \supset \phi) \supset \mathsf{H}\,\phi \qquad (\mathbf{L_\ell})$$

The first two axioms are simply the Kripke schema, given for each modality. Next we find the **t** axioms, which force the two modalities to be converses of each other. The canonical models of the *trichotomy* axioms **.3** have accessibility relationships that are non-branching to the left and to the right. Finally, the axiom $(\mathbf{L_\ell})$ of Gödel-Löb ensures that the models are transitive and well-founded to the left.

B Detailed Proofs

▶ **Proposition 3.10.** *The rules of* $\mathbf{HK_tL_\ell.3}$ *are sound: if the premises of a rule instance are valid, then so is its conclusion.*

Proof. We show the contrapositive: considering an application of a rule with a conclusion hypersequent H and a counter-model $(\mathfrak{M}, \mu)$ of H with $\mathfrak{M} = (\alpha, V)$ and $H \hookrightarrow_\mu \alpha$ an annotation-respecting embedding, we provide a counter-model of one of the premises (or a contradiction when there is no premise).

Since we will often have to extend an embedding with a value for a new position, we define $\mu + (i \mapsto \alpha)$ as the mapping μ' such that $\mu'(i) = \alpha$, $\mu'(k) = \mu(k)$ for $k < i$ and $\mu'(k+1) = \mu(k)$ for $k \geq i$ in the domain of μ.

The case of propositional rules (Figure 1) is immediate: The usual reasoning applies to the principal sequent, and the same embedding is used to obtain a counter-model of one of the premises.

Next we turn to the modal rules of Figure 2:

- Consider the case of $(\mathsf{G}\vdash)$, applied with $\mathsf{G}\,\varphi, \Gamma \vdash \Delta$ at position i and $\Pi \vdash \Sigma$ at position j such that $i \precsim j$. Remark that the rule ensures that $i \neq j$, but we do not need this assumption to justify it. We show that $(\mathfrak{M}, \mu)$ is an annotation-respecting counter-model

of the premise H', concentrating on the only difference with H, at position j. For clarity we distinguish two cases:

- When $i \prec j$, we also have $\mu(i) < \mu(j)$. Since $(\mathfrak{M}, \mu)$ is a counter-model of H, by taking an arbitrary $\beta_i < \mu(i)$ we obtain γ_i such that $\beta_i \leq \gamma_i < \mu(i)$ such that $\mathfrak{M}, \gamma_i \not\models H(i)$. In particular, $\mathfrak{M}, \gamma_i \models \mathsf{G}\,\varphi$. Now, considering an arbitrary $\beta < \mu(j)$ we need to exhibit γ such that $\beta \leq \gamma < \mu(j)$ and $\mathfrak{M}, \gamma \not\models H'(j)$. By taking $\beta_j = \max(\beta, \mu(i)) < \mu(j)$ we obtain γ_j such that $\beta_j \leq \gamma_j < \mu(j)$ and $\mathfrak{M}, \gamma_j \not\models H(j)$. Furthermore, since $\gamma_i < \mu(i) \leq \beta_j \leq \gamma_j$, we also have $\mathfrak{M}, \gamma_j \models \varphi$ and $\mathfrak{M}, \gamma_j \models \mathsf{G}\,\varphi$, hence $\mathfrak{M}, \gamma_j \not\models H'(j)$.
- When $i \sim j$ we have that $\mu(i) = \mu(j)$ and it is a limit ordinal because we are considering positions in a cluster. Consider an arbitrary $\beta < \mu(i)$. There exists γ_i such that $\beta \leq \gamma_i < \mu(i)$ and $\mathfrak{M}, \gamma_i \not\models H(i)$. Because $\mu(i)$ is a limit ordinal, $\gamma_i + 1 < \mu(i) = \mu(j)$. Again, there exists γ_j such that $\gamma_i + 1 \leq \gamma_j < \mu(j)$ and $\mathfrak{M}, \gamma_j \not\models H(j)$. But, since $\gamma_i < \gamma_j$ we also have that γ_j satisfies φ and $\mathsf{G}\,\varphi$, hence $\mathfrak{M}, \gamma_j \not\models H'(j)$.

- The case of rule $(\{\mathsf{G}\vdash\})$ is covered by the second part of the previous argument, by taking $i = j$. Indeed, we have $i \sim i$ when $(\{\mathsf{G}\vdash\})$ applies at position i.
- Consider now an application of rule $(\mathsf{H}\vdash)$ with $\Pi \vdash \Sigma$ at position i and $\mathsf{H}\,\varphi, \Gamma \vdash \Delta$ at j. We have $i \precsim j$, hence $\mu(i) \leq \mu(j)$. Consider an arbitrary $\beta < \mu(i)$. There exists γ_i such that $\beta \leq \gamma_i < \mu(i)$ and $\mathfrak{M}, \gamma_i \not\models H(i)$. We claim, as before, that there exists γ_j such that $\gamma_i < \gamma_j < \mu(j)$ and $\mathfrak{M}, \gamma_j \not\models H(j)$. Indeed, if $\mu(i) < \mu(j)$ then there exists γ_j with $\mu(i) \leq \gamma_j < \mu(j)$ that falsifies $H(j)$. Otherwise $\mu(i) = \mu(j)$ but then this must be a limit ordinal and, by considering $\gamma_i + 1 < \mu(i) = \mu(j)$ we obtain $\gamma_i < \gamma_j < \mu(j)$ that invalidates $H(j)$. Having $\mathfrak{M}, \gamma_j \not\models H(j)$, we also have $\mathfrak{M}, \gamma_j \models \mathsf{H}\,\varphi$. Thus γ_i satisfies φ and $\mathsf{H}\,\varphi$, and $\mathfrak{M}, \gamma_i \not\models H'(i)$ as needed.
- The case of $(\{\mathsf{H}\vdash\})$ is covered by the previous argument.
- Consider an application of $(\vdash\mathsf{G})$ with $\Gamma \vdash \Delta, \mathsf{G}\,\varphi$ at position i. For any $\beta_i < \mu(i)$ there exists γ_i with $\beta_i \leq \gamma_i < \mu(i)$ such that $\mathfrak{M}, \gamma_i \not\models H(i)$, and thus $\mathfrak{M}, \gamma_i \not\models \mathsf{G}\,\varphi$. Hence there also exists $\gamma_i' > \gamma_i$ such that $\mathfrak{M}, \gamma_i' \not\models \varphi$. Let γ be the least ordinal that contains all such γ_i'. We have that $\mu(i) \leq \gamma$.
  We now distinguish several cases regarding γ. When $C'; H_2$ is not empty let j be the first position of the conclusion hypersequent that is in C'.

  - If $\mu(i) = \gamma$, then $\mu(i)$ must be a limit ordinal. Hence $C \neq \star$ and the third premise H_3' is available. We construct a counter-model $(\mathfrak{M}, \mu')$ for it by taking $\mu' = \mu + (k \mapsto \gamma)$, where $k = i + 1$ is the new position in H_3'. Indeed, we have that for any $\beta' < \mu'(k)$ there exists γ' with $\beta' \leq \gamma' < \mu'(k)$ and $\mathfrak{M}, \gamma' \not\models \varphi$ (the inequality can even be made strict). Moreover, the annotation is respected by definition of γ: there cannot be any $\lambda \geq \gamma$ such that $\mathfrak{M}, \lambda \not\models \varphi$.
  - If $C'\,; H_2$ is empty, or $\gamma < \mu(j)$, we conclude by observing that $(\mathfrak{M}, \mu')$ is a counter-model of one of the first two premises with $\mu' = \mu + (k \mapsto \gamma)$ where k is the position of the new cell in these premises. We check that μ' is monotone, because $\mu(i) < \gamma$, and $\gamma < \mu(j)$ when it is defined. If γ is a successor ordinal, $(\mathfrak{M}, \mu')$ is a counter-model of the first premise simply because the predecessor of γ invalidates φ and the annotation is respected; both hold by construction. If γ is a limit ordinal we have a counter-model $(\mathfrak{M}, \mu')$ of the second premise: we do have that for any $\beta' < \mu'(k)$ there exists γ' with $\beta' \leq \gamma' < \mu'(k)$ that invalidates φ, and the annotation is respected by construction.
  - Otherwise $\mu(j) \leq \gamma$.
    * If $\mu(j) < \gamma$, we obtain a counter-model $(\mathfrak{M}, \mu)$ of the fourth premise H_4'. We check it for the only position whose sequent has changed between H and H_4', that is position

j. Take any $\beta_j < \mu(j)$. We know that there exists γ_j with $\beta_j \leq \gamma_j < \mu(j)$ such that $\mathfrak{M}, \gamma_j \not\models H(j)$. But, since $\gamma_j < \mu(j) < \gamma$, there exists δ such that $\gamma_j < \delta < \gamma$ and $\mathfrak{M}, \delta \not\models \varphi$. Thus $\mathfrak{M}, \gamma_j \not\models \mathsf{G}\varphi$, and $\mathfrak{M}, \gamma_j \not\models H'_4(j)$.

- If $\mu(j) = \gamma$ and is a limit ordinal, we also obtain a counter-model $(\mathfrak{M}, \mu)$ of the fourth premise. This time, for any $\beta_j < \mu(j)$, we know that there exists γ_j with $\beta_j \leq \gamma_j < \mu(j)$ such that $\mathfrak{M}, \gamma_j \not\models H(j)$. But, since $\gamma_j < \gamma$ and γ is a limit ordinal, there still exists δ such that $\gamma_j < \delta < \gamma$ and $\mathfrak{M}, \delta \not\models \varphi$. Thus $\mathfrak{M}, \gamma_j \not\models \mathsf{G}\varphi$, and $\mathfrak{M}, \gamma_j \not\models H'_4(j)$.
- Finally, if $\mu(j) = \gamma$ and is not a limit ordinal, then the position j is not in a cluster, so the fifth premise is available. We claim that it admits $(\mathfrak{M}, \mu)$ as a counter-model. Let θ be the predecessor of $\gamma = \theta + 1$, which satisfies $\mathfrak{M}, \theta \not\models \varphi$ by definition of γ. Since $(\mathfrak{M}, \mu)$ is a counter-model of H we also have $\mathfrak{M}, \theta \not\models H(j)$. This allows us to conclude, together with the fact that, as before, the new annotation is respected by definition of γ (there cannot be any $\lambda \geq \gamma$ such that $\mathfrak{M}, \lambda \not\models \varphi$).

- Finally we consider an application of rule $(\vdash\mathsf{H})$ with $\Gamma \vdash \Delta, \mathsf{H}\varphi$ at position i. Let j be the first position of C', if it exists. For any $\beta_i < \mu(i)$ there exists γ_i with $\beta_i \leq \gamma_i < \mu(i)$ that invalidates $H(i)$, thus there exists $\gamma'_i < \gamma_i < \mu(i)$ such that $\mathfrak{M}, \gamma'_i \not\models \varphi$. Let γ be the successor of the least ordinal among all such γ'_i. We have $\gamma < \mu(i)$.
  - If $H_2; C'$ is empty, or $\mu(j) < \gamma$, then $(\mathfrak{M}, \mu')$ is a counter-model of the first premise with $\mu' = \mu + (k \mapsto \gamma)$ where k is the new position in that premise. We do have that the predecessor of γ satisfies $\mathsf{H}\varphi$ (by minimality) but not φ (by definition). Moreover, μ' is indeed annotation-respecting.
  - If $\mu(j) = \gamma$ then C' cannot be a cluster, because γ is a successor. In that case $(\mathfrak{M}, \mu)$ directly yields a counter-model of the third premise.
  - Otherwise $\gamma < \mu(j)$ and $(\mathfrak{M}, \mu)$ is a counter-model of the second premise.

We finally consider the case of annotation rules (Figure 3):

- Consider an application of $((\mathsf{G}))$, with $H(i) = \Gamma \vdash \Delta\ (\mathsf{G}\varphi)$ and $H(j) = \Pi \vdash \Sigma, \mathsf{G}\varphi$, and $i \prec j$. By definition of an embedding, we have $\mu(i) < \mu(j)$. Since $(\mathfrak{M}, \mu)$ is a counter-model of H, there exists γ_j such that $\mu(i) \leq \gamma_j < \mu(j)$ and $\mathfrak{M}, \gamma_j \not\models H(j)$. There also exists $\gamma_i < \mu(i)$ such that $\mathfrak{M}, \gamma_i \not\models H(i)$. Hence there exists $\gamma' > \gamma_j$ such that $\mathfrak{M}, \gamma' \not\models \varphi$. A fortiori, $\gamma' > \gamma_i$, so μ does not respect the annotation on i, contradiction.
- Consider an application of $(\{(\mathsf{G})\})$ with $H(i) = \Gamma \vdash \Delta, \mathsf{G}\varphi\ (\mathsf{G}\varphi)$. For any $\beta < \mu(i)$ there exists γ with $\beta < \gamma$ such that $\mathfrak{M}, \gamma \not\models \varphi$. Because μ is annotation-respecting, we must have $\gamma < \mu(i)$, thus $\mu(i)$ is a limit ordinal. This contradicts the fact that i is not in a cluster.
- Consider an application of $((\bar{\mathsf{G}}))$ with $\Gamma \vdash \Delta\ (\mathsf{G}\varphi)$ at position i and $\Pi \vdash \Sigma$ at position j, with $i \prec j$. Since μ is annotation-respecting we have that, for all $\lambda \geq \mu(i)$, $\mathfrak{M}, \lambda \models \varphi$. Hence $(\mathfrak{M}, \mu)$ is a counter-model of the premise. ◀

▶ **Proposition 3.13.** *Any failure hypersequent H has a counter-model.*

Proof. Let $\alpha = o(H)$. We define $\mu : \mathrm{dom}(H) \to \alpha + 1 \setminus \{0\}$ as follows:

$\mu(i) = m$	if i is the m-th cell of H and appears before its first cluster;
$\mu(i) = \omega \cdot k$	if i belongs to the k-th cluster of H;
$\mu(i) = \omega \cdot k + m$	if i is the m-th cell between the k-th and the next cluster (if any).

Now let $\mathrm{pos} : \alpha \to \mathrm{dom}(H)$ be a function such that:

(a) $\forall \beta < \beta' < \alpha,\ \mathrm{pos}(\beta) \precsim \mathrm{pos}(\beta')$
(b) $\forall \beta < \alpha,\ \forall i \in \mathrm{dom}(H),\ \beta < \mu(i) \Leftrightarrow (\mathrm{pos}(\beta) \precsim i \text{ or } \mathrm{pos}(\beta) = i)$
(c) $\forall \beta < \alpha,\ \forall i \in \mathrm{dom}(H),\ \mathrm{pos}(\beta) \precsim i \Rightarrow \exists \beta < \gamma < \mu(i),\ i = \mathrm{pos}(\gamma)$

There always exists one such function. Its choice is quite constrained due to the definitions of α and μ. Positions i that are not in a cluster will be such that $i = \mathrm{pos}(\beta)$ for a single β, typically the predecessor of $\mu(i)$. A position i appearing in a cluster must correspond to an infinite sequence of ordinals of limit $\mu(i)$, so that for all $i \sim j$ and β, if $\mathrm{pos}(\beta) = i$ then there exists γ with $\beta < \gamma < \mu(i) = \mu(j)$ such that $\mathrm{pos}(\gamma) = j$; informally, this ensures that positions i and j inside a cluster are "infinitely interleaved" within $\mu(i) = \mu(j)$.

We finally define a valuation $V : \Phi \to \wp(\alpha)$ by $V(p) = \{\beta < \alpha \mid \exists \Gamma, \Delta\,.\, H(\mathrm{pos}(\beta)) = (p, \Gamma \vdash \Delta)\}$ and let $\mathfrak{M} = (\alpha, V)$. We now claim that $\mathfrak{M}, \gamma \not\models H(\mathrm{pos}(\gamma))$ for all $\gamma < \alpha$: we prove by induction on ψ that, if ψ appears in the left-hand (resp. right-hand) side of the turnstile in $H(\mathrm{pos}(\gamma))$, then $\mathfrak{M}, \gamma \vDash \psi$ (resp. $\mathfrak{M}, \gamma \nvDash \psi$).

- If ψ is an atom $p \in \Phi$ the results follow by definition of V, and because (ax) does not apply to H. The propositional cases are obtained by induction hypothesis, because the corresponding rules of Figure 1 have already been applied.
- The cases of modal formulæ on the left-hand side are similar, we only detail that of H. If $\psi = \mathsf{H}\varphi$ occurs on the left-hand side of $H(\mathrm{pos}(\gamma))$ then by $(\mathsf{H}\vdash)$ and $(\{\mathsf{H}\vdash\})$, the formula φ must occur on the left-hand side of any $H(i)$ with $i \precsim \mathrm{pos}(\gamma)$. Moreover, for all $\gamma' < \gamma$, we have $\mathrm{pos}(\gamma') \precsim \mathrm{pos}(\gamma)$ by a, so $\mathfrak{M}, \gamma' \vDash \varphi$, and thus $\mathfrak{M}, \gamma \vDash \psi$.
- Assume that $\psi = \mathsf{G}\varphi$ occurs on the right of $H(\mathrm{pos}(\gamma))$. Since $(\vdash\mathsf{G})$ does not apply, an annotation must already exist for $\mathsf{G}\varphi$. By rules (G) and $(\{\mathsf{G}\})$ this annotation must be on a position i such that $\mathrm{pos}(\gamma) \precsim i$. By definition of annotations, φ occurs on the right of $H(i)$. By c, there exists $\gamma' > \gamma$ such that $i = \mathrm{pos}(\gamma')$. We then have $\mathfrak{M}, \gamma' \nvDash \varphi$, thus $\mathfrak{M}, \gamma \nvDash \mathsf{G}\varphi$.
- Assume finally that $\psi = \mathsf{H}\varphi$ occurs on the right of $H(\mathrm{pos}(\gamma))$. We prove by a sub-induction on $\mathrm{pos}(\gamma)$ that $\mathfrak{M}, \gamma \not\models \mathsf{H}\varphi$. Since $(\vdash\mathsf{H})$ does not apply, and since the first premise necessarily differs from the conclusion, it must be that there is a cell C' preceding the cell that contains $\mathrm{pos}(\gamma)$, and that the last two premises (if available) would coincide with H. Let i be the first position in C'. Take an arbitrary $\lambda < \mu(i)$ such that $\mathrm{pos}(\lambda) = i$ (such a λ always exists, thanks to b and c instantiated with $\beta = 0$). Since $i \prec \mathrm{pos}(\gamma)$ it must be that $\lambda < \gamma$. As noted above, we have either that $\mathsf{H}\varphi$ belongs to the right-hand side of $H(i)$, or that φ belongs to its left-hand side. In the first case, we obtain $\mathfrak{M}, \lambda \not\models \mathsf{H}\varphi$ by induction hypothesis on $i < \mathrm{pos}(\gamma)$. In the second case we directly have $\mathfrak{M}, \lambda \not\models \varphi$. We conclude either way that $\mathfrak{M}, \gamma \not\models \mathsf{H}\varphi$.

We can check that $H \hookrightarrow_\mu \alpha$: the conditions of Definition 3.6 hold by construction.

We must also check that μ is annotation-respecting. Assume that $H(i)$ carries the annotation $(\mathsf{G}\varphi)$, and that there is a world $\mathfrak{M}, \beta \not\models \varphi$. Let $j = \mathrm{pos}(\beta)$. If $j \precsim i$, then by c there exists γ with $\beta < \gamma < \mu(i)$ such that $i = \mathrm{pos}(\gamma)$, so $\beta < \mu(i)$ as expected. If $i \prec j$, then by the rule $(\bar{\mathsf{G}})$ φ occurs on the left of $H(j)$, contradicting $\mathfrak{M}, \beta \not\models \varphi$. Otherwise, $i = j$ and by b we have $\beta < \mu(\mathrm{pos}(\beta)) = \mu(i)$ as expected.

Finally, $(\mathfrak{M}, \mu)$ is a counter-model of H. Indeed, for all $i \in \mathrm{dom}(H)$ and $\beta < \mu(i)$ there exists γ with $\beta \leq \gamma < \mu(i)$ such that $\mathrm{pos}(\gamma) = i$, and hence $\mathfrak{M}, \gamma \not\models H(i)$: if $\mathrm{pos}(\beta) = i$, we can take $\gamma = \beta$, else b enforces $\mathrm{pos}(\beta) \precsim i$, and c provides one such γ. ◀

Büchi Good-for-Games Automata Are Efficiently Recognizable

Marc Bagnol[1]
LIP, École Normale Supérieure, Lyon, France

Denis Kuperberg
CNRS, LIP, École Normale Supérieure, Lyon, France

Abstract

Good-for-Games (GFG) automata offer a compromise between deterministic and nondeterministic automata. They can resolve nondeterministic choices in a step-by-step fashion, without needing any information about the remaining suffix of the word. These automata can be used to solve games with ω-regular conditions, and in particular were introduced as a tool to solve Church's synthesis problem. We focus here on the problem of recognizing Büchi GFG automata, that we call *Büchi GFGness problem*: given a nondeterministic Büchi automaton, is it GFG? We show that this problem can be decided in P, and more precisely in $O(n^4m^2|\Sigma|^2)$, where n is the number of states, m the number of transitions and $|\Sigma|$ is the size of the alphabet. We conjecture that a very similar algorithm solves the problem in polynomial time for any fixed parity acceptance condition.

2012 ACM Subject Classification Theory of computation → Models of computation, Theory of computation → Formal languages and automata theory

Keywords and phrases Büchi, automata, games, polynomial time, nondeterminism

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.16

1 Introduction

The fundamental difference between determinism and nondeterminism is one of the deep questions asked by theoretical computer science. The P versus NP problem is an emblematic example of the fact that many basic questions about the power of nondeterminism are still not well-understood. In this work, we investigate an automaton model that offers a middle ground between determinism and nondeterminism, while retaining some advantages of both paradigms in the framework of automata theory. Although this model was introduced as a tool to solve a specific problem – Church's synthesis – we believe that it is a natural stepping stone on our way to get a better understanding of the power of nondeterminism in automata theory.

We will start by mentioning the historical motivation for the model of Good-for-Games (shortly GFG) automata. One of the classical problems of automata theory is synthesis – given a specification, decide if there exists a system that fulfils it and if there is, automatically construct one. The problem was posed by Church [5] and solved positively by Büchi and Landweber [3] for the case of ω-regular specifications. Henzinger and Piterman [10] have proposed the model of GFG automata, that can be seen as a weakening of determinism while still preserving soundness and completeness when solving the synthesis problem. An automaton is GFG if there exists a strategy that resolves the nondeterministic choices, by

[1] This work has been funded by the European Research Council (ERC) under the European Union's Horizon 2020 programme (CoVeCe, grant agreement No 678157).

taking into account only the prefix of the input ω-word read so far. The strategy is supposed to construct an accepting run of the automaton whenever an ω-word from the language is given. The notion of GFG automata was independently discovered in [6] under the name history-determinism, in the more general framework of regular cost functions. It turns out that deterministic cost automata have strictly smaller expressive power than nondeterministic ones and therefore history-determinism is used whenever a sequential model is needed.

We emphasize the fact that although the model of GFG automata requires the existence of a strategy resolving the nondeterminism, this strategy is not used in algorithms but only in proofs. Therefore, it is not a part of the size of the input in computations based on GFG automata. The model of GFG automata offers a compromise between determinism and nondeterminism: in particular, as deterministic automata, it preverses soundness of composition with games and trees [10, 1], while as nondeterministic automata, it can exhibit exponential succinctness compared to deterministic automata [15]. Properties of GFG automata are currently being actively investigated, and most of what we know about them has been uncovered only very recently, with several important questions still open. A brief history of recent advances in the understanding of GFG automata is given in Section 1.1 "Related works".

A major challenge in the understanding of GFG automata is to be able to decide efficiently whether an input automaton is GFG. If $\mathcal{C}$ is an accepting condition, for instance $\mathcal{C} \in \{\text{Büchi}, \text{coBüchi}, \text{Parity}\}$, we call $\mathcal{C}$ *GFGness problem* the following decision problem:

Input: A nondeterministic automaton $\mathcal{A}$ with accepting condition $\mathcal{C}$
Output: Is $\mathcal{A}$ a GFG automaton ?

This problem has been posed in [10], where an EXPTIME algorithm is given for the general case of parity automata. The algorithm makes use of a deterministic automaton for $\mathbf{L}(\mathcal{A})$, which can be built in exponential time. The problem is further studied in [15], where the following results are obtained:

- The coBüchi GFGness problem is in P.
- The Büchi GFGness problem is in NP.
- In general, the $\mathcal{C}$ GFGness problem is at least as hard as solving games with winning condition $\mathcal{C}$. This is tight for automata accepting all infinite words.

The precise complexity of the GFGness problem for Büchi and all higher parity conditions remained open. In particular, even for parity conditions using only 3 ranks, the only known upper bound is EXPTIME. In [14], an incremental algorithm to build GFG automata is given. This algorithm uses as a subroutine an algorithm deciding the GFGness problem. This gives an additional motivation to pinpoint the complexity of the GFGness problem, as it is a bottleneck of the algorithm from [14].

In this work, we tackle the Büchi case, and we show that the Büchi GFGness problem is in P. More precisely, we show that for a Büchi automaton $\mathcal{A}$ on alphabet Σ with n states and m transitions, we can decide whether $\mathcal{A}$ is GFG in $O(n^4m^2|\Sigma|^2)$. We do so by reducing the GFGness problem to a game where 3 tokens move in $\mathcal{A}$. The correctness of the reduction is showed using an intermediate construction using doubly exponentially many tokens.

1.1 Related Works

In the survey [7] two important results about GFG automata over finite words are mentioned: first that every GFG automaton over finite words contains an equivalent deterministic subautomaton, second that the GFGness problem is in P for automata on finite words.

Additionally, a conjecture stating that every parity GFG automaton over ω-words contains an equivalent deterministic subautomaton is posed. In [1], examples were given of Büchi and coBüchi GFG automata which do not contain any equivalent deterministic subautomaton. Moreover, a link between GFG and tree automata was established: an automaton for a language L of ω-words is GFG if and only if its infinite tree version accepts the language of trees that have all their branches in L. Experimental evaluation of GFG automata and their applications to stochastic problems were discussed in [13]. In [15], it is shown that for co-Büchi automata (and all higher parity conditions), GFG automata can be exponentially more succinct than deterministic ones. For Büchi automata, this gap is not exponential, and only a quadratic upper bound is known. Typeness properties of GFG automata are established in [2], as well as complexities for changing between several acceptance conditions. In [14], the model of GFG automata is generalized to the notion of width of a nondeterministic automaton, GFG automata corresponding to width 1, and an incremental algorithm is given to build GFG automata from nondeterministic automata. The games with tokens we define in the present work are very similar in spirit to the k-simulation games introduced in [9]. However, our games cannot be seen directly as particular instances of k-simulation games, as the specific dynamics of the games are different.

2 Definitions

We will use Σ to denote a finite alphabet. The empty word is denoted ε. If $i \leq j$, the set $\{ i, i+1, i+2, \dots, j \}$ is denoted $[i, j]$. The cardinal of a set X is denoted $|X|$. If $u \in \Sigma^*$ is a word and $L \subseteq \Sigma^*$ is a language, the left quotient of L by u is $u^{-1}L = \{ v \in \Sigma^* \mid uv \in L \}$.

2.1 Automata

A nondeterministic automaton $\mathcal{A}$ is a tuple $(Q, \Sigma, q_0, \Delta, F)$ where Q is the set of states, Σ is a finite alphabet, $q_0 \in Q$ is the initial state, $\Delta : Q \times \Sigma \to 2^Q$ is the transition function, and $F \subseteq Q$ is the set of accepting states. We will often write $p \xrightarrow{a} q$ or $p \xrightarrow{a} q \in \Delta$ to signify that $q \in \Delta(p, a)$, *i.e.* there is a transition from p to q labelled by a. If for all (p, a) in $Q \times \Sigma$, $\Delta(p, a) \neq \emptyset$, we say that the automaton is *complete*. In the following, we will assume that all automata are complete, by adding a rejecting sink state $\bot$ if needed. If for all $(p, a) \in Q \times \Sigma$, $|\Delta(p, a)| = 1$, we say that $\mathcal{A}$ is *deterministic*. If $u = a_1 a_2 \dots$ is an infinite word of Σ^ω, a run of $\mathcal{A}$ on u is a sequence $q_0 q_1 q_2 \dots$ such that for all $i > 0$, we have $q_i \in \Delta(q_{i-1}, a_i)$. A run is said to be *Büchi accepting* if it contains infinitely many accepting states, and *coBüchi accepting* if it contains finitely many non-accepting states. Automata on infinite words will be called Büchi and coBüchi automata, to specify their acceptance condition. Finally, we define the *parity condition* on infinite runs: each state q has a rank $\mathrm{rk}(q) \in \mathbb{N}$, and an infinite run is accepting if the highest rank appearing infinitely often is even. An automaton on infinite words using this acceptance condition is a *parity automaton*. The language of an automaton $\mathcal{A}$, denoted $\mathbf{L}(\mathcal{A})$, is the set of words on which the automaton $\mathcal{A}$ has an accepting run. If p is a state of $\mathcal{A}$, the language accepted by $\mathcal{A}$ with p as initial state will be denoted $\mathbf{L}(p)$. A language is called *ω-regular* if it is recognized by a nondeterministic Büchi automaton, or equivalently by a deterministic parity automaton. Two automata are said *equivalent* if they recognise the same language.

An automaton $\mathcal{A}$ is *Good-for-Games* (GFG) if there exists a function $\sigma : \Sigma^* \to Q$ (called *GFG strategy*) that resolves the nondeterminism of $\mathcal{A}$ depending only on the prefix of the input word read so far: over every word $u = a_1 a_2 a_3 \dots$ (finite or infinite depending on the

type of automaton considered), the sequence of states $\sigma(\varepsilon)\sigma(a_1)\sigma(a_1a_2)\sigma(a_1a_2a_3)\dots$ is a run of $\mathcal{A}$ on u, and it is accepting whenever $u \in \mathbf{L}(\mathcal{A})$. For instance every deterministic automaton is GFG. See [1] for more introductory material and examples on GFG automata.

2.2 Games

A *game* $\mathbf{G} = (V_E, V_A, v_I, E, W)$ of infinite duration between two players Eve (Player E) and Adam (Player A) consists of: a finite set of *positions* V being a disjoint union of V_E and V_A; an *initial position* $v_I \in V$; a set of *edges* $E \subseteq V \times V$; and a *winning condition* $W \subseteq V^\omega$.

A *play* is an infinite sequence of positions $v_0v_1v_2\cdots \in V^\omega$ such that $v_0 = v_I$ and for all $n \in \mathbb{N}$, $(v_n, v_{n+1}) \in E$. A play $\pi \in V^\omega$ is *winning* for Eve if it belongs to W. Otherwise π is *winning* for Adam.

A *strategy* for Eve (resp. Adam) is a function $\sigma_E : V^* \times V_E \to V$ (resp. $\sigma_A : V^* \times V_A \to V$) describing which edge should be played given the history of the play $u \in V^*$ and the current position $v \in V$. A strategy has to obey the edge relation, *i.e.* there has to be an edge in E from v to $\sigma_P(u, v)$. A play π is *consistent* with a strategy σ_P of a player $P \in \{E, A\}$ if for every n such that $\pi(n) \in V_P$ we have $\pi(n+1) = \sigma_P(v_0 \dots v_{n-1}, v_n)$.

A strategy for Eve (resp. Adam) is *positional* if it does not use the history of the play, *i.e.* it is a function $V_E \to V$ (resp. $V_A \to V$).

We say that a strategy σ_P of a player P is *winning* if every play consistent with σ_P is winning for player P. In this case, we say that P *wins* the game $\mathbf{G}$.

A game is *positionally determined* if one of the players has a positional winning strategy in the game. A game is *half-positionally determined* if whenever Eve wins, she has a positional winning strategy.

A *finite-memory strategy* for Eve is a tuple (M, m_0, σ_M, upd) where

- M is a finite set called the *memory*, and $m_0 \in M$ is the *initial memory state*.
- σ_M is a function $M \times V_E \to V$,
- upd is a function $M \times V \to M$ called the *update function*.

Such a tuple induces a strategy $\sigma_E : V^* \times V_E \to V$ for Eve in the original sense as follows. First, the function $upd^* : V^* \to M$ is defined by $upd^*(\varepsilon) = m_0$, and if $(\vec{u}, v) \in V^* \times V$, $upd^*(\vec{u} \cdot v) = upd(upd^*(\vec{u}), v)$. We can now define σ_E by $\sigma_E(\vec{u} \cdot v) = \sigma_M(upd^*(\vec{u}), v)$.

A game is *finite-memory determined* if one of the players has a finite-memory winning strategy.

▶ **Remark 1.** *In the rest of the paper, for readability purposes, we will define games in a slightly more informal manner. Namely we will allow sequences of moves of Eve and Adam going through implicit states in the game. Note that it is always possible to come back to the formal version defined in this section. We will also use examples of automata where the acceptance condition is defined on transitions rather than on states, for clarity purposes.*

2.2.1 Winning Conditions

A parity game is a game where W is a parity condition, *i.e.* where every position v has rank $\mathrm{rk}(v) \in \mathbb{N}$, and the winning set W consists of infinite words for which the maximal rank appearing infinitely often is even. The *degree* of a parity game is the number of ranks used in its parity condition.

We will use the following results on parity games:

▶ **Theorem 2** ([8, 12, 4]). *Parity games are positionally determined, and can be solved in* QuasiP.

▶ **Theorem 3** ([16, 11, 17]). *Parity games of fixed degree can be solved in polynomial time. In particular, parity games of degree* 3 *with* n *positions and* m *edges can be solved in* $O(n \cdot m)$.

If the winning set W is an ω-regular language of V^ω, we say that the game is *ω-regular*.

▶ **Theorem 4** ([3]). *ω-regular games are finite-memory determined.*

Solving an ω-regular game $G = (V, E)$ can be costly: the classical procedure from [3] is to build a deterministic automaton $\mathcal{D}$ recognizing W, and building a new game $G \circ \mathcal{D}$ of size $|V| \times |\mathcal{D}|$ with winning condition inherited from the acceptance condition of $\mathcal{D}$. Thus, the idea is to simplify the winning condition of the game at the expense of a blowup in the number of positions.

▶ **Remark 5.** *The original motivation for GFG automata [10] is that it is sufficient for the correctness of this algorithm to take $\mathcal{D}$ GFG instead of deterministic. This also explains the name "good-for-games" introduced in [10]. Remarkably, $\mathcal{D}$ can be used in this algorithm without any knowledge of the GFG strategy witnessing that $\mathcal{D}$ is GFG: as long as such a strategy exists, $\mathcal{D}$ can be used in place of a deterministic automaton to solve any ω-regular G with winning condition W.*

3 Game Characterization of GFG Automata

The main goal of this paper is to give an efficient decision procedure for the Büchi GFGness problem. In order to decide whether an input automaton is GFG, it is natural to replace the abstract definition from Section 2.1 with a more operational one.

3.1 The GFG Game

If $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$ is a nondeterministic Büchi automaton recognizing a language L, let us define the *GFG game* $\mathbf{G}_{\mathrm{GFG}}(\mathcal{A})$ on $\mathcal{A}$. The game is played on arena Q, starting from position q_0. Each round, from position p:

1. Adam chooses a letter $a \in \Sigma$,
2. Eve chooses a transition $p \xrightarrow{a} p'$,
3. the position of the game moves to p'.

The winning condition is the following: Eve wins if either the word $u = a_1 a_2 a_3 \dots$ chosen by Adam is not in $\mathbf{L}(\mathcal{A})$, or if the run $\rho = p_0 p_1 p_2 \dots$ she constructed is accepting (*i.e.* there are infinitely many i such that $p_i \in F$).

The GFG game actually corresponds to the original definition of GFG automata in [10]: an automaton $\mathcal{A}$ is GFG if and only if Eve wins $\mathbf{G}_{\mathrm{GFG}}(\mathcal{A})$. It is shown in [1] that the definition we gave in Section 2.1 for GFG automata is equivalent.

3.2 Solving the GFG Game

Notice that $\mathbf{G}_{\mathrm{GFG}}(\mathcal{A})$ is an ω-regular game. Therefore, by Remark 5, in order to solve it we need a GFG automaton for the language $W = \{(u, \rho) \in A^\omega \times Q^\omega \mid u \notin L \text{ or } \rho \text{ Büchi accepting}\}$. The Büchi condition can be recognized easily by a deterministic 2-state automaton, but for the $u \notin L$ part, we need a GFG automaton for the complement of L. A GFG automaton for L would also do, since we can consider the game where the roles of

the players are reversed, thereby complementing the accepting condition. Thus, in order to decide whether an automaton for L is GFG, we seem to need a GFG automaton for L. In [15], this approach is actually used for the coBüchi GFGness problem: an auxiliary GFG automaton for L is computed, allowing to decide whether the original input automaton is itself GFG.

In the present work, we will circumvent this issue, and instead consider relaxations of the GFG game $\mathbf{G}_{\mathrm{GFG}}(\mathcal{A})$, called *token games* that aim at mimicking the GFG game while enjoying a simpler winning condition.

4 Token Games

Suppose we have fixed a Büchi automaton $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$ for the rest of this section. We define associated token games that will help deciding whether $\mathcal{A}$ is GFG.

4.1 First Attempt: the Game $\mathbf{G}_1$

As seen in Section 3.2, the difficulty of solving the GFG game $\mathbf{G}_{\mathrm{GFG}}(\mathcal{A})$ comes from the fact that $\mathbf{L}(\mathcal{A})$ appears in the winning condition of $\mathbf{G}_{\mathrm{GFG}}(\mathcal{A})$.

A natural attempt to circumvent this difficulty is to replace the condition "$u \notin \mathbf{L}(\mathcal{A})$" by "Adam cannot build an accepting run of $\mathcal{A}$ on u". This would simplify the winning condition, turning it into a boolean combination of Büchi conditions, thus making the game solvable in polynomial time by Theorem 3.

We therefore define $\mathbf{G}_1(\mathcal{A})$ as a modification of $\mathbf{G}_{\mathrm{GFG}}(\mathcal{A})$, where in addition to choosing letters, Adam must additionally build a run witnessing that $u \in \mathbf{L}(\mathcal{A})$. If he fails to do so, Eve automatically wins the game. Therefore, we can view a play as Adam choosing letters, and both Eve and Adam possessing a token, and moving it in the automaton in order to build an accepting run. Here is a formal definition of $\mathbf{G}_1(\mathcal{A})$:

▶ **Definition 6** ($\mathbf{G}_1(\mathcal{A})$). *We define the game $\mathbf{G}_1(\mathcal{A})$ as follows. The game is played on arena Q^2, starting from (q_0, q_0). Each turn, from position (p, q):*

1. *Adam chooses a letter $a \in \Sigma$,*
2. *Eve chooses a transition $p \xrightarrow{a} p'$,*
3. *Adam chooses a transition $q \xrightarrow{a} q'$,*
4. *The game moves to position (p', q').*

Eve wins the game if either the run $\rho = p_0 p_1 \dots$ she chose is accepting, or the run $\lambda = q_0 q_1 \dots$ chosen by Adam is rejecting.

Notice that at each turn, Eve must choose a transition before Adam does. This is not abritrary, as the other way around would trivialize the game: if Adam chooses $q \xrightarrow{a} q'$ before Eve chooses $p \xrightarrow{a} p'$, then Eve can simply copy all choices of Adam, and will always win $\mathbf{G}_1(\mathcal{A})$ even if $\mathcal{A}$ is not GFG.

▶ **Lemma 7.** *If $\mathcal{A}$ is GFG, then Eve wins $\mathbf{G}_1(\mathcal{A})$.*

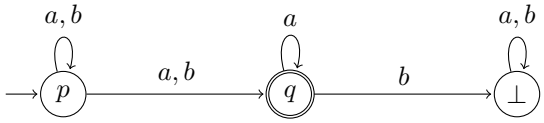
Proof. If Eve has a winning strategy in $\mathbf{G}_{\mathrm{GFG}}(\mathcal{A})$, she can simply use the same strategy in $\mathbf{G}_1(\mathcal{A})$, ignoring the second component of the position. If the run λ built by Adam is accepting, this means the word u that has been played is in $\mathbf{L}(\mathcal{A})$, and therefore by definition of the winning condition of $\mathbf{G}_{\mathrm{GFG}}(\mathcal{A})$, the run ρ built by Eve is accepting. ◀

The hope behind the definition of $\mathbf{G}_1(\mathcal{A})$ is that if Eve wins, then she can win without using the extra information given by the second component of the position. This would

make $\mathbf{G}_1(\mathcal{A})$ equivalent to $\mathbf{G}_{\mathrm{GFG}}(\mathcal{A})$, and allow us to decide the Büchi GFGness problem in polynomial time by Theorem 3.

Unfortunately, we can show that the converse of Lemma 7 does not hold: there is a Büchi automaton $\mathcal{B}$ such that Eve wins $\mathbf{G}_1(\mathcal{B})$ but $\mathcal{B}$ is not GFG.

Indeed, let $\mathcal{B}$ be the following automaton, recognizing $\mathbf{L}(\mathcal{B}) = (a+b)^*a^\omega$ (the accepting state is drawn with a double circle):



▶ **Lemma 8.** *The automaton $\mathcal{B}$ is not GFG, but Eve wins $\mathbf{G}_1(\mathcal{B})$.*

Proof. Adam wins $\mathbf{G}_{\mathrm{GFG}}(\mathcal{B})$ with the following strategy: play the letter a until Eve decides to move to the state q (if she never moves, she fails to build an accepting run for a^ω which is accepted by the automaton), then play ba^ω from there; Eve is forced into state $\bot$ and cannot build an accepting run for a word of the form $a^m ba^\omega$ which is accepted by the automaton.

On the other hand, the strategy for Eve to win $\mathbf{G}_1(\mathcal{A})$ is to simply go where the token of Adam currently is. ◀

This means that in general, if $\mathcal{A}$ is a Büchi automaton, $\mathbf{G}_1(\mathcal{A})$ is not a good enough approximation of $\mathbf{G}_{\mathrm{GFG}}(\mathcal{A})$ and does not characterize GFGness of $\mathcal{A}$.

4.2 Allowing More Tokens

Since the game $\mathbf{G}_1(\mathcal{A})$ is too easy for Eve compared to the GFG game, it is natural to try to make the game harder for Eve, by allowing Adam to build several runs in parallel, some of them being allowed to fail as long as one accepts. Indeed, it is sufficient that one accepting run exists in order to guarantee that the input word chosen by Adam is in $\mathbf{L}(\mathcal{A})$.

The game $\mathbf{G}_k(\mathcal{A})$ can be summed up as follows: Adam chooses a word, Eve moves a token in the automaton while Adam moves k tokens. After ω moves, if one of Adam's tokens followed an accepting run, then Eve's token must also have followed an accepting run. We note simply $\mathbf{G}_k$ instead of $\mathbf{G}_k(\mathcal{A})$ in the rest of the document, in order to lighten notations. Below is a formal definition of the game $\mathbf{G}_k$.

▶ **Definition 9 ($\mathbf{G}_k$).** *For any integer $k \geq 2$, we define the game $\mathbf{G}_k$ as follows. The game is played on arena Q^{k+1}, starting from $(q_0, q_0, \dots, q_0)$. Each turn, from position $(p, q_1, \dots, q_k)$:*

1. *Adam chooses a letter $a \in \Sigma$,*
2. *Eve chooses a transition $p \xrightarrow{a} p'$,*
3. *Adam chooses transitions $q_1 \xrightarrow{a} q'_1, \dots, q_k \xrightarrow{a} q'_k$,*
4. *The game moves to position $(p', q'_1, \dots, q'_k)$.*

Eve wins the game if either the run $\rho = p_0 p_1 \dots$ she chose is accepting, or all runs $\lambda_1, \dots, \lambda_k$ chosen by Adam are rejecting.

▶ **Lemma 10.** *$\mathbf{G}_k$ can be seen as a parity game with 3 parities.*

Proof. We define a new parity condition on $\mathbf{G}_k$ as follows:

$$\mathrm{rk}(p, q_1, \dots q_k) = \begin{cases} 2 \text{ if } p \in F \\ 1 \text{ if } p \notin F \text{ and } q_i \in F \text{ for some } i \\ 0 \text{ otherwise.} \end{cases}$$

A play is won by Eve in $\mathbf{G}_k$ iff ρ is accepting or all runs $\lambda_1, \dots, \lambda_k$ are rejecting iff the play contains infinitely many positions with rank 2 or finitely many positions with rank 1 iff Eve wins according to the new parity condition. Notice that we use the fact that if infinitely many positions have rank 1, then there is an $i \in [1, k]$ such that the $(1+i)^{\text{th}}$ component of the position (corresponding to the i^{th} token of Adam) is in F infinitely many times. ◀

By Theorem 2, this means that $\mathbf{G}_k$ is positionally determined for all k. We will now turn to the particular case where $k = 2$, and obtain a precise upper bound on the complexity of solving $\mathbf{G}_2$. Let n be the number of states of $\mathcal{A}$ and m its number of transitions.

▶ **Lemma 11.** *An explicit version of $\mathbf{G}_2$ has $O(n^3|\Sigma|)$ positions and $O(nm^2|\Sigma|)$ edges.*

Proof. We can define an explicit version of $\mathbf{G}_2$, where the last component specifies which player owns the positions. Positions in this game are $V = \{(v_0, \text{Adam})\} \cup \big(Q^3 \times \Sigma \times \{\text{Eve}, \text{Adam}\}\big)$, so $|V|$ is in $O(n^3|\Sigma|)$. Edges are

$$\begin{aligned} E = &\{(v_0, \text{Adam}) \to (q_0, q_0, a, \text{Eve}) \mid a \in \Sigma\} \\ &\cup \{(p, q_1, q_2, a, \text{Eve}) \to (p', q_1, q_2, a, \text{Adam}) \mid p \xrightarrow{a} p' \in \Delta, (q_1, q_2) \in Q^2\} \\ &\cup \{(p, q_1, q_2, a, \text{Adam}) \to (p, q_1', q_2', b, \text{Eve}) \mid p \in Q, q_1 \xrightarrow{a} q_1' \in \Delta, q_2 \xrightarrow{a} q_2' \in \Delta, b \in \Sigma\}. \end{aligned}$$

We obtain $|E| = |\Sigma| + n^2m + nm^2|\Sigma|$. Since we assume our automata to be complete, $n \leq m$ and $|E|$ is in $O(nm^2)|\Sigma|)$. ◀

By combining Theorem 3, Lemma 10 and Lemma 11, we obtain the following result:

▶ **Theorem 12.** *G_2 can be solved in $O(n^4m^2|\Sigma|^2)$.*

4.3 Some Consequences of Winning $\mathbf{G_2}$

The main result of the present work will be that Eve winning $\mathbf{G}_2$ on $\mathcal{A}$ is equivalent to $\mathcal{A}$ being GFG. One direction is actually trivial:

▶ **Proposition 13.** *If $\mathcal{A}$ is GFG, then Eve has a winning strategy for all $\mathbf{G}_k$.*

Proof. Eve can ignore Adam's tokens and play her GFG strategy. If the word played by Adam is in $\mathbf{L}(\mathcal{A})$ she will build an accepting run, if not Adam will not be able to build one with any of his tokens. ◀

One of the key steps of the proof is to show that if Eve wins against 2 tokens for Adam, she can actually win against any number of tokens.

▶ **Theorem 14.** *If Eve wins $\mathbf{G}_2$ then she wins $\mathbf{G}_k$ for any k.*

Proof. Let σ_2 be a positional winning strategy for Eve in $\mathbf{G}_2$. We proceed by induction on k, the idea being that σ_{k+1} will be obtained by having σ_k play against the first k tokens and then σ_2 against the last token and the output of σ_k. More precisely:

If Eve wins $\mathbf{G}_k$, by Theorem 2 and Lemma 10, she has a winning positional strategy $\sigma_k : Q^{k+1} \times \Sigma \to Q$ in $\mathbf{G}_k$. Define the finite-memory strategy $\sigma'_{k+1} = (M, m_0, \mu, upd)$ where

- the memory M is the set of states Q, and the initial memory state m_0 is q_0
- the update function is $upd(m, (p, q_1, \dots, q_{k+1})) = \sigma_k(m, q_1, \dots, q_k)$
- $\mu(m, (p, q_1, \dots, q_{k+1})) = \sigma_2(p, m, q_{k+1})$ picks the move actually played by Eve

In a play using σ'_{k+1}, the memory m takes the value of σ_k playing against $q_1, \dots, q_k$ so if any of these tokens follows an accepting run, then so will m. The moves of Eve are chosen by playing σ_2 against m and q_{k+1} so that if either of these follow an accepting run, so will Eve's token. In the end, if any of Adam's tokens follows an accepting run, then either q_{k+1} or m does as well and therefore, by correctness of σ_2, the strategy σ'_{k+1} is winning for Eve in $\mathbf{G}_{k+1}$. Because $\mathbf{G}_{k+1}$ is a parity game, there exists another winning strategy σ_{k+1} for Eve that is positional. Since Eve wins $\mathbf{G}_2$, she wins $\mathbf{G}_k$ for all $k \geq 2$ by induction. ◀

Winning $\mathbf{G}_2$ also implies an important property regarding residuals, that will be key in the proof of our main theorem.

▶ **Definition 15** (residual automaton). *A transition $p \xrightarrow{a} q$ is called* residual *if $\mathbf{L}(q) = a^{-1}\mathbf{L}(p)$ (remember that only $\mathbf{L}(q) \subseteq a^{-1}\mathbf{L}(p)$ holds in general). An automaton is* residual *if all its transitions are residual. Given an automaton $\mathcal{A}$, we define the associated residual automaton $\mathcal{A}_r$ as $\mathcal{A}$ where all non-residual transitions have been removed. An automaton is* pre-residual *if it accepts the same language as its residual automaton.*

▶ **Lemma 16.** *If Eve wins $\mathbf{G}_2$ on $\mathcal{A}$, we have:*

- *$\mathcal{A}$ is pre-residual,* i.e. $\mathbf{L}(A) = \mathbf{L}(\mathcal{A}_r)$
- *Eve wins $\mathbf{G}_2$ on $\mathcal{A}_r$*
- *If $\mathcal{A}_r$ is GFG, then $\mathcal{A}$ is GFG*

Proof. Assume that $\mathcal{A}$ is not pre-residual, *i.e.* $\mathbf{L}(\mathcal{A}) \neq \mathbf{L}(\mathcal{A}_r)$. Since $\mathcal{A}_r$ is obtained from $\mathcal{A}$ by removing transitions, we always have $\mathbf{L}(\mathcal{A}_r) \subseteq \mathbf{L}(\mathcal{A})$. So there is $u \in \mathbf{L}(A) \setminus \mathbf{L}(\mathcal{A}_r)$, *i.e.* any accepting run for u must take a non-residual transition at some point. Then Adam can win $\mathbf{G}_2$ in the following way: play the letters of u and have the first token follow Eve's token, and the second one follow an accepting run for u. If Eve never takes any non-residual transition, she cannot build an accepting run for u and loses; if she eventually takes a non-residual transition $p \xrightarrow{a} q$, then Adam picks another transition $p \xrightarrow{a} q'$ such that there is $v \in \mathbf{L}(q') \setminus \mathbf{L}(q)$, move the first token to q' and start playing the letters of v from there. Adam can build an accepting run for v from q' with the first token, while Eve is unable to do so from q. Therefore, this is a winning strategy for Adam in $\mathbf{G}_2$, a contradiction.

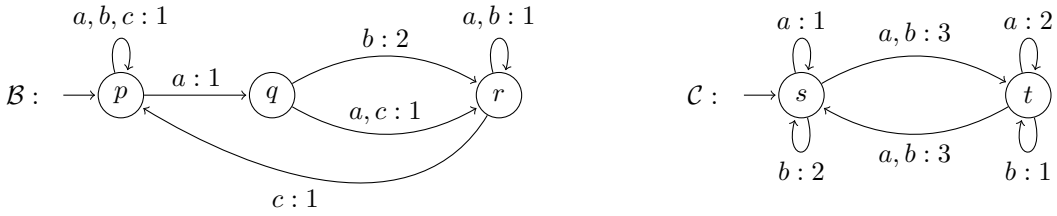
For the second property, we show that if Eve wins $\mathbf{G}_2(\mathcal{A})$ with a strategy σ_2, then σ_2 is actually well-defined and winning for $\mathbf{G}_2(\mathcal{A}_r)$. First note that any reachable position (p, q, r) of $\mathbf{G}_2(\mathcal{A}_r)$ has the property that $\mathbf{L}(p) = \mathbf{L}(q) = \mathbf{L}(r)$ since the initial position is (q_0, q_0, q_0) and only residual transitions can be taken. But in a position (p, q, r) such that $\mathbf{L}(p) = \mathbf{L}(q) = \mathbf{L}(r)$, σ_2 cannot pick a non-residual transition $p \xrightarrow{a} p'$, otherwise Adam can start playing a word that is not in $\mathbf{L}(p')$ and win the game. So σ_2 is a valid strategy to play in $\mathbf{G}_2(\mathcal{A}_r)$. Moreover, any play of $\mathbf{G}_2(\mathcal{A}_r)$ is in particular a play of $\mathbf{G}_2(\mathcal{A})$, and we showed that $\mathbf{L}(\mathcal{A}) = \mathbf{L}(\mathcal{A}_r)$, so σ_2 is a winning strategy in $\mathbf{G}_2(\mathcal{A}_r)$.

Finally, suppose Eve has a GFG strategy σ for $\mathcal{A}_r$. Then this strategy is also well-defined on $\mathcal{A}$ and wins the GFG game because $\mathbf{L}(\mathcal{A}) = \mathbf{L}(\mathcal{A}_r)$. ◀

▶ **Remark 17.** *Any GFG automaton is pre-residual, but the converse does not hold.*

The proof that any GFG automaton is pre-residual is stated in the appendix of [15]. In our setting, it is a corollary of Proposition 13 together with the first item of Lemma 16.

We give two counter-examples for the converse: a Büchi automaton $\mathcal{B}$ on $\Sigma = \{a, b, c\}$ with $\mathbf{L}(\mathcal{B}) = (\Sigma^* ab \Sigma^* c)^\omega$, and a $\{1, 2, 3\}$-parity automaton $\mathcal{C}$ accepting $(a + b)^\omega$. In both cases, we label transitions with parity ranks.



Automata $\mathcal{B}$ and $\mathcal{C}$ are pre-residual, and in fact residual: all their states accept the language of the automaton, so all transitions are residual. However, they are not GFG: we can give a winning strategy for Adam in the GFG game in both cases.[2]

For $\mathcal{B}$, Adam first plays a. If Eve goes to q, then Adam plays abc, bringing Eve back to p. If Eve stays in p, then Adam plays bc, leaving Eve in p. Repeating this process leads Adam to build a word of $L(\mathcal{B})$, while preventing Eve from seeing any Büchi transition.

For $\mathcal{C}$, Adam can play a whenever Eve is in s and b whenever she is in t.

5 Deciding GFGness

Before we get to the sequence of results leading to the proof of our main result, let us quickly outline the approach.

We already know that if Eve is winning G_2 then she wins G_k for any k (Theorem 14) and the main idea is to find a k for which she will be able to move k tokens so that at least one follows an accepting run, and then play σ_k against these virtual tokens. We can note that by simply splitting tokens at any nondeterministic choice, she will be able to explore all the possible runs, and as k grows bigger she can keep doing it for a longer time. The results of subsection 5.1 (specifically Theorem 19) essentially guarantee there is a k large enough so that following this approach, she will eventually reach accepting states.

It then remains to use this to precisely formulate Eve's strategy to win against an hypothetic winning strategy for Adam in the GFG game, reaching a contradiction (Theorem 20).

5.1 Powerset Automaton

We will assume here that $\mathcal{A}$ is residual. We review a few properties of the powerset automaton that will be useful in our setting.

▶ **Definition 18.** *Given a residual Büchi automaton $\mathcal{A} = (Q, q_0, \Delta, F)$ we define the powerset automaton of $\mathcal{A}$, $2^{\mathcal{A}} = (2^Q, \Sigma, \{q_0\}, \Delta', F')$ where*

- $\Delta'(\mathbf{q}, a) = \bigcup_{p \in \mathbf{q}} \Delta(p, a)$
- $\mathbf{q} \in F'$ *when there is* $q \in \mathbf{q}$ *such that* $q \in F$

Note that it is well known that as such $2^{\mathcal{A}}$ does not necessarily recognize the same language as $\mathcal{A}$. However, it is always true that $\mathbf{L}(\mathcal{A}) \subseteq \mathbf{L}(2^{\mathcal{A}})$. More precisely, for any state $p \in Q$, we have $\mathbf{L}(p) \subseteq \mathbf{L}(\{p\})$. This property will be sufficient for our purpose. Let us write $\mathbf{q}\bullet w$ for the sequence of states visited by $2^{\mathcal{A}}$ when reading w from state $\mathbf{q}$.

The following lemma will be crucial in the proof of the main theorem: it tells us that if Adam is choosing letters according to a finite-memory winning strategy for the GFG game,

[2] Actually, $\mathcal{B}$ has a stronger property: Eve could not win the GFG game even if she had k tokens instead of one, for any k.

then the number of turns before being able to see an accepting state while reading these letters is bounded. This bound will allow to follow all the possible runs up to that accepting state with a finite number of tokens.

▶ **Lemma 19.** *If τ is a finite-memory winning strategy for Adam in $\mathbf{G}_{\mathrm{GFG}}(\mathcal{A})$, then there exists an integer K_τ such that: if w is a sequence of letters of length K_τ chosen by τ from a state p_0 (reached by playing τ from the starting position), and q is any state with $\mathbf{L}(p_0) = \mathbf{L}(q)$ then $\{q\}\bullet w$ contains an accepting state.*

Proof. Let M be the memory of τ and let $K_\tau = |Q \times M \times 2^Q|$. Let $w = a_1 a_2 \dots a_{K_\tau}$ be a word of length K_τ that can be played by τ in $\mathbf{G}_{\mathrm{GFG}}(\mathcal{A})$ from a position p_0 reachable in $\mathbf{G}_{\mathrm{GFG}}(\mathcal{A})$ with some memory m_0. Consider the sequence

$$(p_0, m_0, \{q\}) \xrightarrow{a_1} (p_1, m_1, \mathbf{q}_1) \xrightarrow{a_2} \cdots \xrightarrow{a_{K_\tau}} (p_{K_\tau}, m_{K_\tau}, \mathbf{q}_{K_\tau})$$

where the p_i's describe the states of $\mathcal{A}$ in this play, the m_i's are Adam's memory states, and the $\mathbf{q}_i$'s are the states of $2^{\mathcal{A}}$ reached upon reading the letters of w starting from $\{q\}$. By choice of K_τ, there must be $i < j$ such that $(p_i, m_i, \mathbf{q}_i) = (p_j, m_j, \mathbf{q}_j)$. This means that there is a prefix uv of w such that Eve can force the strategy τ to play uv^ω from $(p_0, m_0, \{q\})$, while guaranteeing that on the suffix v^ω, the run of $2^{\mathcal{A}}$ (corresponding to the third component) repeats the same cycle C from $\mathbf{q}_i$ to $\mathbf{q}_j = \mathbf{q}_i$.

Because τ is winning for Adam, and $\mathcal{A}$ is residual, we must have $uv^\omega \in \mathbf{L}(p_0) = \mathbf{L}(q)$. Since $\mathbf{L}(q) \subseteq \mathbf{L}(\{q\})$, we have $uv^\omega \in \mathbf{L}(\{q\})$, and therefore the cycle C must contain an accepting state of $2^{\mathcal{A}}$. Since the cycle C is present in $\{q\}\bullet w$, this concludes the proof. ◀

5.2 Two Tokens Are Enough

▶ **Theorem 20.** *If Eve wins $\mathbf{G}_2$ on a residual automaton $\mathcal{A}$, then $\mathcal{A}$ is GFG.*

Proof. Assume by contradiction that Eve wins $\mathbf{G}_2$ but Adam wins $\mathbf{G}_{\mathrm{GFG}}(\mathcal{A})$. By Theorem 4, he can do so with a finite-memory strategy τ (with memory of size exponential in $|Q|$). Let $K = K_\tau$ given by Theorem 19, and $c = \max\{|\Delta(p, a)| \mid p \in Q, a \in \Sigma\}$ be the degree of nondeterminism of $\mathcal{A}$. Let $N = c^K$ and $T = N \cdot |Q|$, so that when moving T tokens on $\mathcal{A}$, at least one state will hold N or more tokens at any given time. Recall that by Theorem 14, Eve has a positional winning strategy $\sigma_T : Q^{T+1} \times \Sigma \to Q$ in $\mathbf{G}_T$. Notice that T is doubly exponential in $|Q|$.

We will now define a finite-memory strategy $\sigma = (M, m_0, \sigma_M, upd)$ for Eve in $\mathbf{G}_{\mathrm{GFG}}(\mathcal{A})$. The strategy σ will be defined according to the following intuition: Eve plays against τ by simulating T tokens moving in $\mathcal{A}$, and chooses her actual moves in $\mathbf{G}_{\mathrm{GFG}}(\mathcal{A})$ by playing σ_T against these virtual tokens. The memory M of σ is Q^T, and its initial memory state is $m_0 = (q_0, \dots, q_0)$. We now describe the update function upd of σ. This amounts to giving a strategy for moving T tokens in $\mathcal{A}$, when letters are given by the opponent step-by-step. We will consider that some tokens are *active* and the others are *passive*. Tokens are moved according to the following rules:

- Initially, the T tokens are in q_0, and are all active.
- At each nondeterministic choice, active tokens are divided evenly between possible successors.
- Passive tokens are moved arbitrarily.
- If an accepting state is reached by some token, then choose a state p containing at least N tokens, and set the tokens in this state to active, and all others to passive. We call this a *reset point* p.

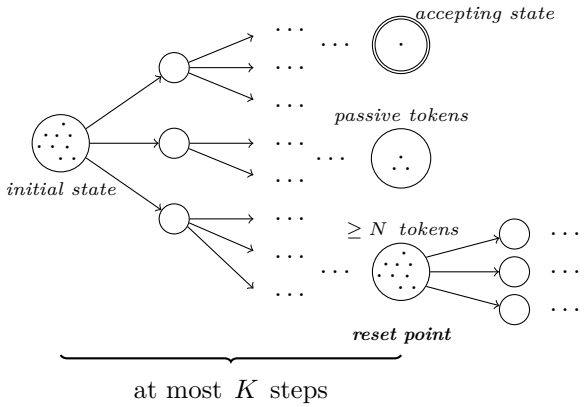


Figure 1 Illustrating Eve's strategy for moving the T tokens.

We will also consider that the initial position is a reset point. An illustration of this update strategy is given in Figure 1.

Finally, we define $\sigma_M : M \times (Q \times \Sigma) \to Q$ by $\sigma_M(\mathbf{q}, p, a) = \sigma_T(p, \mathbf{q}, a)$.

We can now consider the play ρ of σ against τ in $\mathbf{G}_{\mathrm{GFG}}(\mathcal{A})$. Let $p_i, \mathbf{q}_i, a_i$ be respectively the state of Q, the memory state of σ, and the letter played by τ after i moves in ρ. Notice that since $\mathcal{A}$ is residual, for all i and $q \in \mathbf{q}_i$, we have $\mathbf{L}(p_i) = \mathbf{L}(q)$. This allows us to use Theorem 19 in the following.

We first show that there are infinitely many i such that $\mathbf{q}_i$ contains an accepting state. Consider p a reset point in the play. Starting from at least $N = c^K$ active tokens in p, and dividing them evenly at each step, the update strategy can cover all states reached by $2^{\mathcal{A}}$ from $\{p\}$ with some active tokens during K steps. By Theorem 19, the memory will reach an accepting state within these K steps, and can therefore restart at another reset point without ever running out of active tokens. This shows that there are infinitely many i such that $\mathbf{q}_i$ contains an accepting state. Since M is a finite tuple, there is one of its a components j such that the j^{th} coordinate of $\mathbf{q}_i$ is accepting for infinitely many i. By correctness of σ_T, we obtain that there are infinitely many i such that p_i is accepting. This implies that the play ρ of $\mathbf{G}_{\mathrm{GFG}}(\mathcal{A})$ is won by Eve, a contradiction with the assumption that τ is winning for Adam. ◀

▶ **Corollary 21.** *On any Büchi automaton $\mathcal{A}$, Eve wins $\mathbf{G}_2$ if and only if $\mathcal{A}$ is GFG.*

Proof. A consequence of Theorem 16 and the above theorem: if Eve wins $\mathbf{G}_2$ on $\mathcal{A}$, then she also does on $\mathcal{A}_r$, which implies that $\mathcal{A}_r$ is GFG, and therefore $\mathcal{A}$ is GFG as well. We already saw the other direction in Theorem 13. ◀

By Theorem 12, we can now state our main result:

▶ **Theorem 22.** *The Büchi GFGness problem is in* P*, and more precisely in $O(n^4 m^2 |\Sigma|^2)$.*

▶ **Remark 23.** *Let us discuss briefly the possible extension of this proof to other parity cases.*

On one hand Theorem 14 is true regardless of the acceptance condition (the proof does not rely on the automaton being Büchi), which is quite promising. But on the other, the adaptation of Theorem 19 proves problematic, and without this lemma it seems difficult to find a way to move T virtual tokens so that at least one of them follows an accepting run, which we rely on critically in the proof of Theorem 20. Already in the coBüchi case a substitute

technique is missing, although our current work focuses on using some of the techniques from [15] to prove Theorem 20 in this case, hoping this will eventually lead to a technique working for any parity condition.

Conclusion

We showed that the Büchi GFGness problem can be decided in P, by introducing new techniques using token games. While it seems that our proof cannot be directly used to solve efficiently the parity GFGness problem, the game $\mathbf{G}_2$ could still be relevant in this more general setting. We did not find any example of a non-GFG parity automaton $\mathcal{A}$ such that Eve wins $\mathbf{G}_2(\mathcal{A})$, so in our opinion it is plausible that Eve wins $\mathbf{G}_2(\mathcal{A})$ if and only if $\mathcal{A}$ is GFG for any parity automaton $\mathcal{A}$. Since for any fixed acceptance condition (for instance parity condition of fixed degree), the game $\mathbf{G}_2$ can be solved in P, this would put the GFGness problem in P for any fixed acceptance condition, with an algorithm that is already known.

References

1. Udi Boker, Denis Kuperberg, Orna Kupferman, and Michał Skrzypczak. Nondeterminism in the Presence of a Diverse or Unknown Future. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, pages 89–100, 2013.
2. Udi Boker, Orna Kupferman, and Michał Skrzypczak. How Deterministic are Good-For-Games Automata? In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017, December 11-15, 2017, Kanpur, India*, pages 18:1–18:14, 2017.
3. J. Richard Buchi and Lawrence H. Landweber. Solving Sequential Conditions by Finite-State Strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.
4. Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 252–263, 2017.
5. Alonzo Church. Application of Recursive Arithmetic to the Problem of Circuit Synthesis. *Journal of Symbolic Logic*, 28(4):289–290, 1963.
6. Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *Automata, languages and programming. Part II*, volume 5556 of *Lecture Notes in Comput. Sci.*, pages 139–150, Berlin, 2009. Springer.
7. Thomas Colcombet. Forms of Determinism for Automata (Invited Talk). In *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France*, pages 1–23, 2012.
8. E. Allen Emerson and Charanjit S. Jutla. Tree Automata, Mu-Calculus and Determinacy (Extended Abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 368–377, 1991.
9. Kousha Etessami. A Hierarchy of Polynomial-Time Computable Simulations for Automata. In *CONCUR 2002 - Concurrency Theory, 13th International Conference, Brno, Czech Republic, August 20-23, 2002, Proceedings*, pages 131–144, 2002.
10. Thomas A. Henzinger and Nir Piterman. Solving Games Without Determinization. In *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, pages 395–410, 2006.

11 Marcin Jurdziński. Small Progress Measures for Solving Parity Games. In Horst Reichel and Sophie Tison, editors, *STACS 2000*, pages 290–301, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

12 Nils Klarlund. Progress Measures, Immediate Determinacy, and a Subset Construction for Tree Automata. *Ann. Pure Appl. Logic*, 69(2-3):243–268, 1994.

13 Joachim Klein, David Müller, Christel Baier, and Sascha Klüppelholz. Are Good-for-Games Automata Good for Probabilistic Model Checking? In *Language and Automata Theory and Applications - 8th International Conference, LATA 2014, Madrid, Spain, March 10-14, 2014. Proceedings*, pages 453–465, 2014.

14 Denis Kuperberg and Anirban Majumdar. Width of Non-deterministic Automata. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, pages 47:1–47:14, 2018.

15 Denis Kuperberg and Michał Skrzypczak. On Determinisation of Good-for-Games Automata. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, pages 299–310, 2015.

16 Robert McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993.

17 Sven Schewe. Solving parity games in big steps. *Journal of Computer and System Sciences*, 84:243–262, 2017.

Popular Matchings in Complete Graphs

Ágnes Cseh[1]
Hungarian Academy of Sciences, Budapest, Hungary
cseh.agnes@krtk.mta.hu

Telikepalli Kavitha[2]
Tata Institute of Fundamental Research, Mumbai, India
kavitha@tcs.tifr.res.in

Abstract

Our input is a complete graph $G = (V, E)$ on n vertices where each vertex has a strict ranking of all other vertices in G. The goal is to construct a matching in G that is "globally stable" or *popular*. A matching M is popular if M does not lose a head-to-head election against any matching M': here each vertex casts a vote for the matching in $\{M, M'\}$ where it gets a better assignment. Popular matchings need not exist in the given instance G and the popular matching problem is to decide whether one exists or not. The popular matching problem in G is easy to solve for odd n. Surprisingly, the problem becomes NP-hard for even n, as we show here.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases popular matching, complete graph, complexity, linear programming

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.17

Related Version A full version of the paper is available at [9], `https://arxiv.org/pdf/1807.01112.pdf`.

Acknowledgements Thanks to Chien-Chung Huang for asking us about the complexity of the popular roommates problem with complete preference lists.

1 Introduction

Consider a complete graph $G = (V, E)$ on n vertices where each vertex ranks all other vertices in a strict order of preference. Such a graph is called a *roommates* instance with complete preferences. The problem of computing a stable matching in G is a classical and well-studied problem. Recall that a matching M is stable if there is no *blocking pair* with respect to M, i.e., a pair (u, v) where both u and v prefer each other to their respective assignments in M.

Stable matchings need not always exist in a roommates instance. For example, the instance given in Fig. 1 on 4 vertices d_0, d_1, d_2, d_3 has no stable matching. (Here d_0's top choice is d_1, then d_2, and finally d_3, and so on.)

Irving [17] gave an efficient algorithm to decide if G admits a stable matching or not. In this paper we consider a notion that is more relaxed than stability: this is the notion of *popularity*. For any vertex u, a ranking over neighbors can be extended naturally to a

[1] Supported by the Cooperation of Excellences Grant (KEP-6/2017), by the Ministry of Human Resources under its New National Excellence Programme (ÚNKP-18-4-BME-331), the Hungarian Academy of Sciences under its Momentum Programme (LP2016-3/2016), its János Bolyai Research Fellowship, and OTKA grant K128611.
[2] This work was done while visiting the Hungarian Academy of Sciences, Budapest.

d_0 : $d_1 > d_2 > d_3$
d_1 : $d_2 > d_3 > d_0$
d_2 : $d_3 > d_1 > d_0$
d_3 : $d_1 > d_2 > d_0$

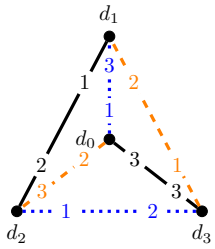


Figure 1 An instance that admits two popular matchings – marked by dotted blue and dashed orange edges – but no stable matching. The preference list of each vertex is illustrated by the numbers on its edges: a lower number indicates a more preferred neighbor.

ranking over matchings as follows: u prefers matching M to matching M' if (i) u is matched in M and unmatched in M' or (ii) u is matched in both and u prefers $M(u)$ to $M'(u)$. For any two matchings M and M', let $\phi(M, M')$ be the number of vertices that prefer M to M'.

▶ **Definition 1.** Let M be any matching in G. M is *popular* if $\phi(M, M') \geq \phi(M', M)$ for every matching M' in G.

Suppose an election is held between M and M' where each vertex casts a vote for the matching that it prefers. So $\phi(M, M')$ (similarly, $\phi(M', M)$) is the number of votes for M (resp., M'). A popular matching M never loses an election to another matching M' since $\phi(M, M') \geq \phi(M', M)$: thus it is a weak *Condorcet winner* [6, 1] in the corresponding voting instance. So popularity can be regarded as a natural notion of "global stability".

The notion of popularity was first introduced in bipartite graphs in 1975 by Gärdenfors – popular matchings always exist in bipartite graphs since stable matchings always exist here [11] and every stable matching is popular [12]. The proof that every stable matching is popular holds in non-bipartite graphs as well [5]; in fact, it is easy to show that every stable matching is a min-size popular matching [14]. Relaxing the constraint of stability to popularity allows us to find globally stable matchings that may exist in instances that do not admit stable matchings; moreover, even when stable matchings exist, there may be popular matchings that achieve more "social good" (such as larger size) in many applications.

Observe that the instance in Fig. 1 has two popular matchings: $M_1 = \{(d_0, d_1), (d_2, d_3)\}$ and $M_2 = \{(d_0, d_2), (d_1, d_3)\}$. However as was the case with stable matchings, popular matchings also need not always exist in the given instance G. The *popular roommates problem* is to decide if G admits a popular matching or not. When the graph is not complete, it is known that the popular roommates problem is NP-hard [10, 13]. Here we are interested in the complexity of the popular matching problem when the input instance is complete.

Interestingly, several popular matching problems that are intractable in bipartite graphs become tractable in *complete bipartite* graphs. The min-cost popular matching problem in bipartite graphs is such a problem – this is NP-hard in a bipartite graph with incomplete lists [10], however it can be solved in polynomial time in a bipartite graph with complete lists [8]. The difference is due to the fact that while there is no efficient description of the convex hull of all popular matchings in a general bipartite graph, this polytope has a compact extended formulation in a complete bipartite graph.

It is a simple observation (see Section 2) that when n is *odd*, a matching in a complete graph G on n vertices is popular only if it is stable. Since there is an efficient algorithm to decide if G admits a stable matching or not, the popular roommates problem in a complete graph G can be efficiently solved when n is odd. We show the following result here.

▶ **Theorem 2.** *Let G be a complete graph on n vertices, where n is even. The problem of deciding whether G admits a popular matching or not is NP-hard.*

So the popular roommates problem with complete preference lists is NP-hard for even n while it is easy to solve for odd n. Note that the popular roommates problem is non-trivial for every $n \geq 5$, i.e., there are both "yes instances" and "no instances" of size n. It is rare and unusual for a natural decision problem in combinatorial optimization to be efficiently solvable when n has one parity and become NP-hard when n has the other parity. We are not aware of any natural optimization problem on graphs that is non-trivially tractable when the cardinality of the vertex set has one parity, which becomes intractable for the other parity.

1.1 Background and related work

The first polynomial time algorithm for the stable roommates problem was by Irving [17] in 1985. Roommates instances that admit stable matchings were characterized in [25]. New polynomial time algorithms for the stable roommates problem were given in [24, 26].

Algorithmic questions for popular matchings in bipartite graphs have been well-studied in the last decade [3, 8, 14, 16, 18, 19, 20]. Not much was known on popular matchings in non-bipartite graphs. Biró et al. [3] proved that validating whether a given matching is popular can be done in polynomial time, even when ties are present in the preference lists. It was shown in [15] that every roommates instance $G = (V, E)$ admits a matching with *unpopularity factor* $O(\log |V|)$ and that it is NP-hard to compute a least unpopularity factor matching. It was shown in [16] that computing a max-weight popular matching in a roommates instance with edge weights is NP-hard, and more recently, that computing a max-size popular matching in a roommates instance is NP-hard [21].

The complexity of the popular roommates problem was open for several years [3, 7, 15, 16, 22] and two independent NP-hardness proofs [10, 13] of this problem were announced very recently. Interestingly, both these hardness proofs need "incomplete preference lists", i.e., the underlying graph is *not* complete. The reduction in [13] is from a variant of the vertex cover problem called the *partitioned vertex cover* problem and we discuss the reduction in [10] in Section 1.2 below. So the complexity status of the popular roommates problem in a complete graph was an open problem and we resolve it here.

Computational hardness for instances with complete lists has been investigated in various matching problems under preferences. An example is the three-sided stable matching problem with cyclic preferences: this involves three groups of participants, say, men, women, and dogs, where dogs have weakly ordered preferences over men only, men have preferences over women only, and finally, women only list the dogs. If these preferences are allowed to be incomplete, the problem of finding a *weakly* stable matching is known to be NP-complete [4]. It is one of the most intriguing open questions in stable matchings [22, 27] as to whether the same problem becomes tractable when lists are complete.

1.2 Techniques

The 1-in-3 SAT problem is a well-known NP-hard problem [23]: it consists of a 3-SAT formula ϕ with no negated literals and the problem is to find a truth assignment to the variables in ϕ such that every clause has exactly one variable set to `true`. We show a polynomial time reduction from 1-in-3 SAT to the popular roommates problem with complete lists.

Our construction is based on the reduction in [10] that proved the NP-hardness of the popular roommates problem. However there are several differences between our reduction and the reduction in [10]. The reduction in [10] considered a popular matching problem in

bipartite graphs called the "exclusive popular set" problem and showed it to be NP-hard – when preference lists are complete, this problem can be easily solved. Thus the reduction in [10] needs incomplete preference lists.

The exclusive popular set problem asks if there is a popular matching in the given bipartite graph where the set of matched vertices is S, for a given even-sized subset S. A key step in the reduction in [10] from this problem in bipartite graphs to the popular matching problem in non-bipartite graphs merges all vertices outside S into a single node. Thus the total number of vertices in the non-bipartite graph used in [10] is *odd*. Moreover, the fact that popular matchings always exist in bipartite graphs is crucially used in this reduction. However in our setting, the whole problem is to decide if *any* popular matching exists in the given graph – thus there are no popular matchings that "always exist" here.

The reduction in [10] primarily uses the LP framework of popular matchings in bipartite graphs from [18, 19, 21] to analyze the structure of popular matchings in their instance. The LP framework characterizing popular matchings in non-bipartite graphs is more complex [21], so we use the combinatorial characterization of popular matchings [14] in terms of forbidden alternating paths/cycles to show that any popular matching in our instance will yield a 1-in-3 satisfying assignment for ϕ. To show the converse, we use a dual certificate similar to the one used in [10] to prove the popularity of the matching that we construct using a 1-in-3 satisfying assignment for ϕ.

2 Preliminaries

Let M be any matching in $G = (V, E)$. For any pair $(u, v) \notin M$, define $\mathsf{vote}_u(v, M)$ as follows: (here $M(u)$ is u's partner in M and $M(u) = \mathsf{null}$ if u is unmatched in M)

$$\mathsf{vote}_u(v, M) = \begin{cases} + & \text{if } u \text{ prefers } v \text{ to } M(u); \\ - & \text{if } u \text{ prefers } M(u) \text{ to } v. \end{cases}$$

Label every edge (u, v) that does not belong to M by the pair $(\mathsf{vote}_u(v, M), \mathsf{vote}_v(u, M))$. Thus every non-matching edge has a label in $\{(\pm, \pm)\}$. Note that an edge is labeled $(+, +)$ if and only if it is a blocking edge to M. Let G_M be the subgraph of G obtained by deleting edges labeled $(-, -)$ from G. The following theorem characterizes popular matchings in G.

▶ **Theorem 3** ([14]). *M is popular in G if and only if G_M does not contain any of the following with respect to M:*

(1) *an alternating cycle with a $(+, +)$ edge;*

(2) *an alternating path with two distinct $(+, +)$ edges;*

(3) *an alternating path with a $(+, +)$ edge and an unmatched vertex as an endpoint.*

Using the above characterization, it can be easily checked whether a given matching is popular or not [14]. Thus our NP-hardness result implies that the popular roommates problem is NP-complete.

When n is odd. Recall the claim made in Section 1 that when n is odd, every popular matching in G has to be stable. A simple proof of this statement is included below.

▶ **Observation 4** ([2]). *Let G be a complete graph on n vertices, where n is odd. Any popular matching in G has to be stable.*

x_i : $y_i > y'_i > D > \dots$
x'_i : $y_i > y'_i > D > \dots$
y_i : $x_i > x'_i > D > \dots$
y'_i : $x_i > x'_i > D > \dots$

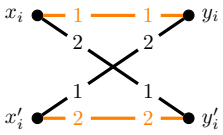


Figure 2 The variable gadget in level 1.

Proof. Suppose not. Let M be a popular matching in G that is not stable. So there is a blocking edge (u, v) to M. Because n is odd, we know that there is an unmatched vertex. If one of u, v is unmatched, then the edge (u, v) is a forbidden alternating path for popularity (by Theorem 3, part (3)). So let the unmatched vertex be $x \notin \{u, v\}$.

Then the path x - $(M(u), u)$ - $(v, M(v))$ with respect to M is again a forbidden alternating path for popularity (by Theorem 3, part (3)). Thus M is not a popular matching. ◀

3 The graph G

Recall that ϕ is the input formula to 1-in-3 SAT. The graph G that we construct here consists of gadgets in 4 levels along with 2 special gadgets that we will call the D-gadget and Z-gadget. Gadgets in level 1 correspond to variables in the formula ϕ while gadgets in levels 0, 2, and 3 correspond to clauses in ϕ. Variants of the gadgets in levels 0-3 and the D-gadget were used in [10] while the Z-gadget is new.

We will now describe these gadgets: along with a figure, we provide the preference lists of vertices in this gadget. The tail of each list consists of all vertices not listed yet, in an arbitrary order. Even though the preference lists are complete, the structure of the gadgets and the preference lists will ensure that inter-gadget edges will not belong to any popular matching, as we will show in Section 4.

The D-gadget. The D-gadget is on 4 vertices d_0, d_1, d_2, d_3 and the preference lists of these vertices are as given in Fig. 1 with all vertices outside the D-gadget at the tail of each list (in an arbitrary order). Recall that this gadget admits no stable matching.

We describe gadgets from level 1 first, then levels 0, 2, 3, and finally, the Z-gadget. The stable matchings within the gadgets are highlighted by colors in the figures. The gray elements in the preference lists denote vertices that are outside this gadget. We will assume that D in a preference list stands for $d_0 > d_1 > d_2 > d_3$.

Level 1. For each variable X_i in the formula ϕ, we construct a gadget on four vertices as shown in Fig. 2. The bottom vertices x'_i and y'_i will be preferred by some vertices in level 0 to vertices in their own gadget, while the top vertices x_i and y_i will be preferred by some vertices in level 2 to vertices in their own gadget. All four vertices in a level 1 gadget prefer to be matched among themselves, along the four edges drawn than be matched to any other vertex in the graph. This gadget has a unique stable matching $\{(x_i, y_i), (x'_i, y'_i)\}$.

Level 0. To each clause $c = X_i \vee X_j \vee X_k$ in the formula ϕ, we create *6 gadgets* in level 0. One of these can be seen in Fig. 3. The top two vertices, i.e. a^c_1 and b^c_1, rank y'_j and x'_k in level 1, as their respective second choices. Recall that indices j and k are well-defined in the

$a_1 : b_1 > y'_j > b_2 > D > \dots$
$a_2 : b_2 > b_1 > D > \dots$
$b_1 : a_2 > x'_k > a_1 > D > \dots$
$b_2 : a_1 > a_2 > D > \dots$

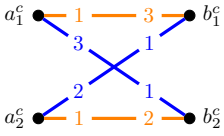


Figure 3 A clause gadget in level 0.

$p_0 : q_0 > q_2 > D > \dots$
$p_1 : q_1 > q_2 > D > \dots$
$p_2 : q_0 > y_j > q_1 > q_2 > D > \dots$
$q_0 : p_0 > p_2 > D > \dots$
$q_1 : p_1 > p_2 > D > \dots$
$q_2 : p_1 > x_k > p_0 > p_2 > D > \dots$

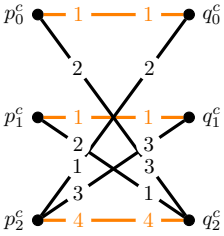


Figure 4 A clause gadget in level 2.

clause $c = X_i \vee X_j \vee X_k$. Within this level 0 gadget on $a_1^c, b_1^c, a_2^c, b_2^c$, both $\{(a_1^c, b_1^c), (a_2^c, b_2^c)\}$ and $\{(a_1^c, b_2^c), (a_2^c, b_1^c)\}$ are stable matchings. In the preference lists below (and also for gadgets in levels 2 and 3), we have omitted the superscript c in their lists for the sake of readability.

The gadget on vertices $\{a_3^c, a_4^c, b_3^c, b_4^c\}$ is built analogously: the vertex a_3^c ranks y'_k as its second choice, while b_3^c ranks x'_i second. In the third gadget, the vertex a_5^c ranks y'_i second, while b_5^c ranks x'_j second. Observe the shift in i, j, k indices as second choices for vertices a_1^c, a_3^c, a_5^c (and similarly, for b_1^c, b_3^c, b_5^c).

The fourth, fifth and sixth gadgets are analogous to the first, second, and third gadgets, respectively, but there is a slight twist. More precisely, the preferences of $a_1'^c, a_2'^c, b_1'^c, b_2'^c$ in the fourth gadget are analogous to the preferences in Fig. 3, except that $a_1'^c$ ranks y'_k second, while $b_1'^c$ ranks x'_j second. Similarly, the second choice of $a_3'^c$ is y'_i, the second choice of $b_3'^c$ is x'_k, and finally, $a_5'^c$ ranks y'_j second, while $b_5'^c$ ranks x'_i second. Observe the change in *orientation* of the indices i, j, k as second choice neighbors when comparing the first three level 0 gadgets of c with its last three level 0 gadgets. This will be important to us later.

Level 2. To each clause $c = X_i \vee X_j \vee X_k$ in the formula ϕ, we create *6 gadgets* in level 2. The first gadget in level 2 is on vertices $p_0^c, p_1^c, p_2^c, q_0^c, q_1^c, q_2^c$ and their preference lists are described in Fig. 4. Note that p_2^c ranks y_j from level 1 as its second choice, while q_2^c ranks x_k from level 1 second.

The second gadget in level 2 is on vertices $p_3^c, p_4^c, p_5^c, q_3^c, q_4^c, q_5^c$ and it is built analogously. That is, p_3^c and q_3^c are each other's top choices and similarly, p_4^c and q_4^c are each other's top choices, and so on. The preference list of p_5^c is $q_3^c > y_k > q_4^c > q_5^c > D > \dots$ and the preference list of q_5^c is $p_4^c > x_i > p_3^c > p_5^c > D > \dots$

The third gadget in level 2 is on vertices $p_6^c, p_7^c, p_8^c, q_6^c, q_7^c, q_8^c$ and it is built analogously. In particular, the preference list of p_8^c is $q_6^c > y_i > q_7^c > q_8^c > D > \dots$ and the preference list of q_8^c is $p_7^c > x_j > p_6^c > p_8^c > D > \dots$

The fourth gadget in level 2 is on vertices $p_0'^c, p_1'^c, p_2'^c, q_0'^c, q_1'^c, q_2'^c$ and it is totally analogous to the first gadget in level 2. That is, $p_0'^c$ and $q_0'^c$ are each other's top choices and similarly, $p_1'^c$ and $q_1'^c$ are each other's top choices, and so on. In particular, the preference list of $p_2'^c$ is $q_0'^c > y_j > q_1'^c > q_2'^c > D > \dots$ and the preference list of $q_2'^c$ is $p_1'^c > x_k > p_0'^c > p_2'^c > D > \dots$

s_0 : $t_1 > q_0 > t_2 > q_3 > t_3 > D > t_0 > \dots$
t_0 : $s_3 > p_7 > s_2 > p_4 > s_1 > D > s_0 > \dots$
s_1 : $t_1 > t_0 > D > \dots$
t_1 : $s_1 > s_0 > D > \dots$
s_2 : $t_2 > t_0 > D > \dots$
t_2 : $s_2 > s_0 > D > \dots$
s_3 : $t_3 > t_0 > D > \dots$
t_3 : $s_3 > s_0 > D > \dots$

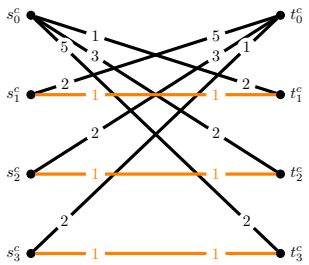


Figure 5 A clause gadget in level 3.

z_0 : $z_4 > z_5 > \cup_i \{x_i, y_i\} > \cup_{c,i} \{p^c_{3i+1}, q^c_{3i}, p'^c_{3i+1}, q'^c_{3i}\} > \cup_{c,i} \{a^c_i, b^c_i, a'^c_i, b'^c_i\} > z_1 > z_2 > z_3 > D > \dots$
z_1 : $z_5 > z_4 > \cup_i \{x_i, y_i\} > \cup_{c,i} \{p^c_{3i+1}, q^c_{3i}, p'^c_{3i+1}, q'^c_{3i}\} > \cup_{c,i} \{a^c_i, b^c_i, a'^c_i, b'^c_i\} > z_0 > z_3 > z_2 > D > \dots$
z_2 : $z_0 > z_1 > z_3 > z_4 > z_5 > D > \dots$
z_3 : $z_1 > z_0 > z_2 > z_5 > z_4 > D > \dots$
z_4 : $z_2 > z_3 > z_5 > z_0 > z_1 > D > \dots$
z_5 : $z_3 > z_2 > z_4 > z_1 > z_0 > D > \dots$

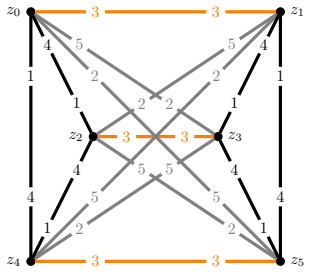


Figure 6 The Z-gadget.

Similarly, the fifth gadget in level 2 is on vertices $p'^c_3, p'^c_4, p'^c_5, q'^c_3, q'^c_4, q'^c_5$ and it is totally analogous to the second gadget in level 2. Also, the sixth gadget in level 2 is on vertices $p'^c_6, p'^c_7, p'^c_8, q'^c_6, q'^c_7, q'^c_8$ and it is totally analogous to the third gadget in level 2.

Level 3. To each clause $c = X_i \vee X_j \vee X_k$ in the formula ϕ, we create *2 gadgets* in level 3. The first gadget is on vertices $s^c_0, s^c_1, s^c_2, s^c_3, t^c_0, t^c_1, t^c_2, t^c_3$ and the preference lists of these vertices are described in Fig. 5.

The second gadget in level 3 is on $s'^c_0, s'^c_1, s'^c_2, s'^c_3, t'^c_0, t'^c_1, t'^c_2, t'^c_3$ and their preference lists are absolutely analogous to the preference lists of the first gadget in level 3.

The Z-gadget. The Z-gadget is on 6 vertices $z_0, z_1, z_2, z_3, z_4, z_5$ and the preference lists of these vertices are given in Fig. 6. The vertices in a set stand for all these vertices in an arbitrary order. For example, $\cup_i \{x_i, y_i\}$ denotes all the "top" vertices belonging to variable gadgets in an arbitrary order.

Note that G is a complete graph on an even number of vertices and so every popular matching in G has to be a perfect matching.

4 Popular edges in G

Call an edge e in G *popular* if there is a popular matching M in G such that $e \in M$. In this section we identify edges that are not popular and show that every popular edge is an *intra-gadget* edge, connecting two vertices of the same gadget. All missing proofs are in the full version of our paper on the arxiv [9].

▶ **Lemma 5.** *For any clause c, no popular matching in G can match s^c_0 (similarly, t^c_0) to a neighbor worse than t^c_0 (resp., s^c_0). An analogous statement holds for s'^c_0 and t'^c_0.*

▶ **Lemma 6.** *Every popular matching matches the vertices in the D-gadget among themselves.*

The gadget D admits 2 popular matchings: $\{(d_0, d_1), (d_2, d_3)\}$ and $\{(d_0, d_2), (d_1, d_3)\}$. So if M is a popular matching then either $\{(d_0, d_1), (d_2, d_3)\} \subset M$ or $\{(d_0, d_2), (d_1, d_3)\} \subset M$.

▶ **Lemma 7.** *Let (u, v) be an edge in G where both u and v prefer d_0 to each other. Then (u, v) cannot be a popular edge.*

▶ **Corollary 8.** *The edges (s_0^c, t_0^c) and $(s_0'^c, t_0'^c)$ are not popular edges for any clause c.*

Corollary 8 follows from Lemma 7 by setting u and v to s_0^c and t_0^c (similarly, $s_0'^c$ and $t_0'^c$), respectively. Let us call u a level i vertex if u belongs to a level i gadget.

▶ **Lemma 9.** *No edge between a level i vertex and a level $i+1$ vertex is popular, for $0 \leq i \leq 2$.*

▶ **Lemma 10.** *All popular matchings match the 6 vertices of the Z-gadget among themselves.*

Proof. Let M be any popular matching in G. It follows from Lemma 7 that M has to pair each of z_2, z_3, z_4, and z_5 to a vertex in the Z-gadget. Let us now show that z_0 also has to be matched within the Z-gadget. Then it immediately follows that z_1 also has to be matched within the Z-gadget. We have the following 3 cases:

(1) Suppose z_0 is matched in M to a level 0 neighbor, say b_1^c. Then (a_1^c, b_1^c) is a blocking edge to M. Lemmas 6, 7, and 9 ensure that a_1^c is either matched to z_1 or to b_2^c. We investigate these two cases below.
- $(a_1^c, z_1) \in M$: Here both z_0 and z_1 are matched to vertices they prefer to all their neighbors inside the Z-gadget, except for z_4 and z_5. We know that z_4 and z_5 must be matched inside the Z-gadget. There are 3 subcases and in each case there is an alternating cycle in G_M with a blocking edge (a_1^c, b_1^c): a contradiction to M's popularity (by Theorem 3).
  - $(z_4, z_2) \in M$: the alternating cycle is $(b_1^c, z_0) \overset{(+,-)}{-} (z_4, z_2) \overset{(+,-)}{-} (z_1, a_1^c) \overset{(+,+)}{-} (b_1^c, z_0)$.
  - $(z_4, z_3) \in M$: the alternating cycle is $(b_1^c, z_0) \overset{(+,-)}{-} (z_4, z_3) \overset{(+,-)}{-} (z_1, a_1^c) \overset{(+,+)}{-} (b_1^c, z_0)$.
  - $(z_4, z_5) \in M$: the alternating cycle is $(b_1^c, z_0) \overset{(+,-)}{-} (z_4, z_5) \overset{(-,+)}{-} (z_1, a_1^c) \overset{(+,+)}{-} (b_1^c, z_0)$.
- $(a_1^c, b_2^c) \in M$: Lemmas 6, 7, and 9 ensure that a_2^c is matched to z_1 (recall that M is perfect). This leads to the same 3 subcases as above, except that instead of the edge (z_1, a_1^c), there is the path $(z_1, a_2^c) - (b_2^c, a_1^c)$ in G_M: here (a_2^c, b_2^c) is labeled $(+, -)$.

(2) Suppose z_0 is matched in M to a level 1 neighbor, say y_i.
This case is similar to the previous case. Here the edge (x_i, y_i) becomes the blocking edge to M. It follows from Lemmas 6, 7, and 9 that x_i is either matched to z_1 or to y_i'. The latter case leaves x_i' unmatched and the subcases that arise in the former case are analogous to the ones in case (1).

(3) Suppose z_0 is matched in M to a level 2 neighbor, say q_0^c.
It follows from Lemmas 6, 7, and 9 that $(p_0^c, q_2^c), (p_2^c, q_1^c)$, and (p_1^c, z_1) are in M. Consider the alternating path $(z_0, q_0^c) - (p_2^c, q_1^c) - (p_1^c, z_1)$: it has two blocking edges (p_2^c, q_0^c) and (p_1^c, q_1^c). This is again a contradiction to M's popularity.

Recall that Lemma 6 showed that all vertices of D must be matched within the gadget. Thus z_0 cannot be matched to a vertex in the D-gadget. The case where z_0 is matched in M to a level 3 neighbor does not arise as such an edge would violate Lemma 7. This finishes our proof that any popular matching M matches the 6 vertices of the Z-gadget among themselves. ◀

It follows from Lemmas 6, 7, 9, and 10 that every popular edge is an intra-gadget edge. Lemma 11 (proof in [9]) shows that there is only one possibility for a popular matching within the Z-gadget. Thus every popular matching in G contains $(z_0, z_1), (z_2, z_3), (z_4, z_5)$.

▶ **Lemma 11.** *The only popular matching inside the Z-gadget is $\{(z_0, z_1), (z_2, z_3), (z_4, z_5)\}$.*

5 Stable states versus unstable states

In this section we will show how to obtain a 1-in-3 satisfying assignment for the input ϕ from any popular matching in G. The following definition will be useful to us.

▶ **Definition 12.** A gadget A in $G = (V, E)$ is said to be in *unstable state* with respect to matching M if there is a blocking edge $(u, v) \in V(A) \times V(A)$ with respect to M. If there is no such blocking edge to M then we say A is in *stable state* with respect to M.

In Figures 2-6 depicting our gadgets, corresponding to matchings that consist of colored edges within the gadget, the relevant gadget is in *stable* state. A level 1 gadget in unstable state will encode the corresponding variable being set to true while a level 1 gadget in stable state will encode the corresponding variable being set to false. We will now analyze what gadgets are in stable/unstable state with respect to any popular matching M in G. This will lead to the proof that for any clause c, exactly one of the level 1 gadgets corresponding to the 3 variables in c is in unstable state.

▶ **Lemma 13.** *For any clause c, the following statements hold:*
- *all its 6 level 0 gadgets are in stable state with respect to M;*
- *both its level 3 gadgets in G are in unstable state with respect to M.*

Proof. Consider a level 0 gadget corresponding to clause c, say the one on vertices $a_1^c, b_1^c, a_2^c, b_2^c$. Lemmas 6, 7, 9, and 10 imply that either $\{(a_1^c, b_1^c), (a_2^c, b_2^c)\} \subset M$ or $\{(a_1^c, b_2^c), (a_2^c, b_1^c)\} \subset M$. Thus there is no blocking edge within this gadget. As this holds for every level 0 gadget corresponding to c and for every clause c, the first part of the lemma follows.

We will now prove the second part of the lemma. Since M is a perfect matching, the vertices s_0^c, t_0^c (also $s_0'^c, t_0'^c$) have to be matched in M, for all clauses c. It follows from Lemmas 6 and 7 that both s_0^c and t_0^c (similarly, $s_0'^c$ and $t_0'^c$) have to be matched to neighbors that are better than d_0. Lemma 9 showed that there is no popular edge between a level 3 vertex and a level 2 vertex. Thus s_0^c is matched to t_i^c for some $i \in \{1, 2, 3\}$.

If s_0^c is matched to t_i^c then s_i^c has to be matched to t_0^c – otherwise Lemma 7 would be violated by s_i^c and its partner. So (s_i^c, t_i^c) blocks M and this holds for every clause c. Similarly, there is a blocking edge $(s_i'^c, t_i'^c)$ for some $i \in \{1, 2, 3\}$ for every clause c. ◀

▶ **Lemma 14.** *For any clause c, at least* one *of the following two conditions has to hold:*
- *two or more of its first three level 2 gadgets are in unstable state with respect to M;*
- *two or more of its last three level 2 gadgets are in unstable state with respect to M.*

The proof of Lemma 14 is given in [9]. Recall that there are three level 1 gadgets associated with any clause c: these gadgets correspond to the three variables in c.

▶ **Lemma 15.** *Let $c = X_i \vee X_j \vee X_k$. At least one of the level 1 gadgets corresponding to X_i, X_j, X_k is in unstable state with respect to M.*

Proof. Suppose not. That is, assume that for some clause c, all three of its level 1 gadgets are in stable state. Let $c = X_i \vee X_j \vee X_k$. So (x_r, y_r) and (x'_r, y'_r) are in M for all $r \in \{i, j, k\}$.

We know from Lemma 14 that either two or more of the *first* three level 2 gadgets corresponding to c are in unstable state with respect to M or two or more of the *last* three level 2 gadgets corresponding to c are in unstable state with respect to M. Assume without loss of generality that the first and second gadgets, i.e., those on p_i^c, q_i^c, for $0 \leq i \leq 5$, are in unstable state with respect to M.

We know from our lemmas in Section 4 that there is no popular edge across gadgets. Thus M matches the 6 vertices of a level 2 gadget with each other. In particular, it follows from Lemma 7 that for the level 2 gadget on p_i^c, q_i^c for $i = 0, 1, 2$, we have (i) $(p_0^c, q_0^c), (p_1^c, q_1^c), (p_2^c, q_2^c)$ in M or (ii) $(p_0^c, q_2^c), (p_1^c, q_1^c), (p_2^c, q_0^c)$ in M or (iii) $(p_0^c, q_0^c), (p_1^c, q_2^c), (p_2^c, q_1^c)$ in M.

There are two unstable states for each level 2 gadget, i.e., either (ii) or (iii) above for the gadget on p_i^c, q_i^c for $i = 0, 1, 2$. A level 2 gadget can be in either of these two unstable states in M – without loss of generality assume that M contains $(p_0^c, q_0^c), (p_1^c, q_2^c), (p_2^c, q_1^c)$ and $(p_3^c, q_5^c), (p_4^c, q_4^c), (p_5^c, q_3^c)$. Observe that p_2^c likes y_j more than q_1^c and similarly, q_5^c likes x_i more than p_3^c. Consider the following alternating path ρ with respect to M:

$$(q_2^c, p_1^c) \overset{(+,+)}{-} (q_1^c, p_2^c) \overset{(+,-)}{-} (y_j, x_j) \overset{(-,+)}{-} (z_0, z_1) \overset{(+,-)}{-} (y_i, x_i) \overset{(-,+)}{-} (q_5^c, p_3^c) \overset{(+,+)}{-} (q_3^c, p_5^c).$$

Note that M has to contain (z_0, z_1) (by Lemma 11). Observe that ρ is an alternating path in G_M with *two* blocking edges (p_1^c, q_1^c) and (p_3^c, q_3^c). This is a contradiction to M's popularity (by Theorem 3) and the lemma follows. ◀

We can also show that (see [9]) *at most one* of the level 1 gadgets corresponding to X_i, X_j, X_k is in unstable state with respect to M. So *exactly one* of the level 1 gadgets corresponding to X_i, X_j, X_k is in unstable state with respect to M. This allows us to set a 1-in-3 satisfying assignment to instance ϕ. For each variable X_i in ϕ do:

– if the gadget corresponding to X_i is in *unstable* state then set $X_i = \mathsf{true}$ else set $X_i = \mathsf{false}$.

It follows from our above discussion that this is a 1-in-3 satisfying assignment for ϕ. We have thus shown the following result.

▶ **Theorem 16.** *If G admits a popular matching then ϕ has a 1-in-3 satisfying assignment.*

6 The converse

We will now show the converse of Theorem 16, i.e., if ϕ has a 1-in-3 satisfying assignment S then G admits a popular matching. We will use S to construct a popular matching M in G as follows. To begin with, $M = \emptyset$.

Level 1. For each variable X_i do:
- if X_i is set to true in S then add (x_i, y_i') and (x_i', y_i) to M;
- else add (x_i, y_i) and (x_i', y_i') to M.

For each clause $c = X_i \vee X_j \vee X_k$, we know that exactly one of X_i, X_j, X_k is set to true in S. Assume without loss of generality that $X_k = \mathsf{true}$ in S. For the level 0, 2, and 3 gadgets corresponding to c, we do as follows:

Level 0. Recall that there are *six* level 0 gadgets that correspond to c. For the first 3 gadgets (these are on vertices a_i^c, b_i^c for $i = 1, \ldots, 6$) do:
- include $(a_1^c, b_2^c), (a_2^c, b_1^c)$ from the first gadget;
- include $(a_3^c, b_3^c), (a_4^c, b_4^c)$ from the second gadget;
- choose either $(a_5^c, b_5^c), (a_6^c, b_6^c)$ or $(a_5^c, b_6^c), (a_6^c, b_5^c)$ from the third gadget.

Observe that since the third variable X_k of c was set to be true, cross edges are fixed in the first gadget (see Fig. 3), while the other stable matching (horizontal edges) is chosen in the second gadget.

For the fourth and fifth gadgets, we will do exactly the opposite. Also, it will not matter which stable pair of edges is chosen from the third and sixth gadgets. So for the last 3 level 0 gadgets corresponding to c (these are on vertices a'^c_i, b'^c_i for $i = 1, \ldots, 6$) do:

- include $(a'^c_1, b'^c_1), (a'^c_2, b'^c_2)$ from the fourth gadget;
- include $(a'^c_3, b'^c_4), (a'^c_4, b'^c_3)$ from the fifth gadget.
- choose either $(a'^c_5, b'^c_5), (a'^c_6, b'^c_6)$ or $(a'^c_5, b'^c_6), (a'^c_6, b'^c_5)$ from the sixth gadget.

Level 2. Recall that there are *six* level 2 gadgets that correspond to c. For the first 3 gadgets (these are on vertices p^c_i, q^c_i for $i = 0, \ldots, 8$) do:

- include $(p^c_0, q^c_2), (p^c_1, q^c_1), (p^c_2, q^c_0)$ from the first gadget
- include $(p^c_3, q^c_3), (p^c_4, q^c_5), (p^c_5, q^c_4)$ from the second gadget
- include $(p^c_6, q^c_6), (p^c_7, q^c_7), (p^c_8, q^c_8)$ from the third gadget

In the first three gadgets, because $X_k =$ true, the third one is set to parallel edges, reaching the stable state, while the first one is blocked by the top horizontal edge and the second one is blocked by the middle horizontal edge. Include isomorphic edges (to the above ones) from the last three level 2 gadgets corresponding to c, i.e., include $(p'^c_0, q'^c_2), (p'^c_1, q'^c_1), (p'^c_2, q'^c_0)$ from the fourth gadget, and so on. On this level, the last three gadgets mimic the matching edges from the first three gadgets, unlike in level 0.

Level 3. For the first level 3 gadget corresponding to c do:

- include $(s^c_0, t^c_3), (s^c_1, t^c_1), (s^c_2, t^c_2), (s^c_3, t^c_0)$ in M.

Since the third variable in c was set to be true, the vertices s^c_0 and t^c_0 are matched to t^c_3 and s^c_3, respectively – thus the bottom horizontal edge (s^c_3, t^c_3) blocks M. Include isomorphic edges (to the above ones) for the second level 3 gadget corresponding to c, i.e., include $(s'^c_0, t'^c_3), (s'^c_1, t'^c_1), (s'^c_2, t'^c_2), (s'^c_3, t'^c_0)$ in M. Once again, the second gadget mimics the matching edges on the first gadget.

***Z*-gadget and *D*-gadget.** Finally include the edges $(z_0, z_1), (z_2, z_3), (z_4, z_5)$ from the Z-gadget in M. By Lemma 11, every popular matching in G has to include these edges. Also include $(d_0, d_1), (d_2, d_3)$ from the D-gadget in M.

6.1 The popularity of M

We will now prove the popularity of the above matching M via the LP framework of popular matchings initiated in [18] for bipartite graphs. This framework generalizes to provide a sufficient condition for popularity in non-bipartite graphs [10]. This involves showing a witness $\vec{\alpha} \in \{0, \pm 1\}^{|V|}$ such that $\vec{\alpha}$ is a *certificate* of M's popularity. In order to define the constraints that $\vec{\alpha}$ has to satisfy so as to certify M's popularity, let us define an edge weight function w_M as follows.

For any edge (u, v) in G do:

- if (u, v) is labeled $(-, -)$ then set $w_M(u, v) = -2$;
- if (u, v) is labeled $(+, +)$ then set $w_M(u, v) = 2$;
- else set $w_M(u, v) = 0$. (So $w_M(e) = 0$ for all $e \in M$.)

Let N be any perfect matching in G. It is easy to see from the definition of the edge weight function w_M that $w_M(N) = \phi(N, M) - \phi(M, N)$.

Let the max-weight perfect fractional matching LP in the graph G with edge weight function w_M be our primal LP. This is LP1 defined below.

$$\text{maximize} \sum_{e \in E} w_M(e) x_e \qquad \text{(LP1)}$$

subject to

$$\sum_{e \in \delta(u)} x_e \;=\; 1 \;\; \forall\, u \in V \qquad \text{and} \qquad x_e \geq 0 \;\; \forall\, e \in E.$$

If the primal optimal value is at most 0 then $w_M(N) \leq 0$ for all perfect matchings N in G, i.e., $\phi(N, M) \leq \phi(M, N)$. This means $\phi(M', M) \leq \phi(M, M')$ for *all* matchings M' in G, since G is a complete graph on an even number of vertices (so $M' \subseteq$ some perfect matching). That is, M is a popular matching in G.

Consider the LP that is dual to LP1. This is LP2 given below in variables α_u, where $u \in V$.

$$\text{minimize} \sum_{u \in V} \alpha_u \qquad \text{(LP2)}$$

subject to

$$\alpha_u + \alpha_v \;\geq\; w_M(u, v) \;\; \forall\, (u, v) \in E.$$

If we show a dual feasible solution $\vec{\alpha}$ such that $\sum_{u \in V} \alpha_u = 0$ then the primal optimal value is at most 0, i.e., M is a popular matching.

In order to prove the popularity of M, we define $\vec{\alpha}$ as follows. For each variable X_r do:

- if X_r was set to true then set $\alpha_{x_r} = \alpha_{y_r} = 1$ and $\alpha_{x'_r} = \alpha_{y'_r} = -1$;
- else set $\alpha_{x_r} = \alpha_{y_r} = \alpha_{x'_r} = \alpha_{y'_r} = 0$.

Let clause $c = X_i \vee X_j \vee X_k$. Recall that we assumed that $X_i = X_j =$ false and $X_k =$ true. For the vertices in clauses corresponding to c, we will set α-values as follows.

- For every level 0 vertex v do: set $\alpha_v = 0$.
- For the first three level 2 gadgets corresponding to c do:
  - set $\alpha_{p_0^c} = \alpha_{q_0^c} = 1$, $\alpha_{p_1^c} = 1, \alpha_{q_1^c} = -1$, and $\alpha_{p_2^c} = \alpha_{q_2^c} = -1$;
  - set $\alpha_{p_3^c} = -1, \alpha_{q_3^c} = 1$, $\alpha_{p_4^c} = \alpha_{q_4^c} = 1$, and $\alpha_{p_5^c} = \alpha_{q_5^c} = -1$;
  - set $\alpha_{p_6^c} = \alpha_{q_6^c} = \alpha_{p_7^c} = \alpha_{q_7^c} = \alpha_{p_8^c} = \alpha_{q_8^c} = 0$.

The setting of α-values is analogous for vertices in the last three level 2 gadgets corresponding to c. For the first level 3 gadget corresponding to c do:

- set $\alpha_{s_0^c} = \alpha_{t_0^c} = -1$, $\alpha_{s_1^c} = -1, \alpha_{t_1^c} = 1$, $\alpha_{s_2^c} = -1$, $\alpha_{t_2^c} = 1$, and $\alpha_{s_3^c} = \alpha_{t_3^c} = 1$.

The setting of α-values is analogous for vertices in the other level 3 gadget corresponding to c. For the z-vertices do: set $\alpha_u = 0$ for all $u \in \{z_0, \ldots, z_5\}$. For the d-vertices do:

- set $\alpha_{d_0} = \alpha_{d_2} = -1$ and $\alpha_{d_1} = \alpha_{d_3} = 1$.

Properties of $\vec{\alpha}$. For every $(u, v) \in M$, either $\alpha_u = \alpha_v = 0$ or $\{\alpha_u, \alpha_v\} = \{-1, 1\}$; so $\alpha_u + \alpha_v = 0$. Since M is a perfect matching, we have $\sum_{u \in V} \alpha_u = 0$. The claims stated below (proofs are in [9]) show that $\vec{\alpha}$ is a feasible solution to LP2. This will prove the popularity of M.

We need to show that every edge (u, v) is *covered*, i.e., $\alpha_u + \alpha_v \geq w_M(u, v)$. We have already observed that for any $(u, v) \in M$, $\alpha_u + \alpha_v = 0 = w_M(u, v)$.

▶ **Claim 17.** *Let (u, v) be a blocking edge to M. Then $\alpha_u + \alpha_v = 2 = w_M(u, v)$.*

▶ **Claim 18.** *Let (u, v) be an intra-gadget edge that is non-blocking. Then $\alpha_u + \alpha_v \geq w_M(u, v)$.*

▶ **Claim 19.** *Let (u, v) be any inter-gadget edge. Then $\alpha_u + \alpha_v \geq w_M(u, v)$.*

Thus we have shown the following theorem.

▶ **Theorem 20.** *If ϕ has a 1-in-3 satisfying assignment then G admits a popular matching.*

Theorem 2 stated in Section 1 follows from Theorems 16 and 20. Thus the popular matching problem in a roommates instance on n vertices with complete preference lists is NP-hard for even n.

References

1 Condorcet method. Accessed 3 October 2018. URL: `https://en.wikipedia.org/wiki/Condorcet_method`.

2 C.-C. Huang. Personal communication.

3 P. Biró, R.W. Irving, and D.F. Manlove. Popular matchings in the Marriage and Roommates problems. In *Proceedings of CIAC '10: the 7th International Conference on Algorithms and Complexity*, volume 6078 of *Lecture Notes in Computer Science*, pages 97–108. Springer, 2010.

4 P. Biró and E. McDermid. Three-sided stable matchings with cyclic preferences. *Algorithmica*, 58(1):5–18, 2010.

5 K.S. Chung. On the Existence of Stable Roommate Matchings. *Games and Economic Behavior*, 33(2):206–230, 2000.

6 M.-J.-A.-N. de C. (Marquis de) Condorcet. *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*. L'Imprimerie Royale, 1785.

7 Ágnes Cseh. Popular matchings. *Trends in Computational Social Choice*, page 105, 2017.

8 Ágnes Cseh and Telikepalli Kavitha. Popular edges and dominant matchings. *Mathematical Programming*, pages 1–21, 2017. online first.

9 Ágnes Cseh and Telikepalli Kavitha. Popular Matchings in Complete Graphs. *CoRR*, abs/1807.01112, 2018.

10 Yuri Faenza, Telikepalli Kavitha, Vladlena Powers, and Xingyu Zhang. Popular Matchings and Limits to Tractability. *arXiv preprint arXiv:1805.11473*, 2018.

11 D. Gale and L.S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.

12 P. Gärdenfors. Match making: assignments based on bilateral preferences. *Behavioural Science*, 20:166–173, 1975.

13 Sushmita Gupta, Pranabendu Misra, Saket Saurabh, and Meirav Zehavi. Popular Matching in Roommates Setting is NP-hard. *arXiv preprint arXiv:1803.09370*, 2018.

14 C.-C. Huang and T. Kavitha. Popular Matchings in the Stable Marriage Problem. In *Proceedings of ICALP '11: the 38th International Colloquium on Automata, Languages and Programming*, volume 6755 of *Lecture Notes in Computer Science*, pages 666–677. Springer, 2011.

15 C.-C. Huang and T. Kavitha. Near-popular matchings in the Roommates problem. *SIAM Journal on Discrete Mathematics*, 27(1):43–62, 2013.

16 Chien-Chung Huang and Telikepalli Kavitha. Popularity, mixed matchings, and self-duality. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2294–2310. Society for Industrial and Applied Mathematics, 2017.

17 R.W. Irving. An efficient algorithm for the "stable roommates" problem. *Journal of Algorithms*, 6:577–595, 1985.

18 T. Kavitha, J. Mestre, and M. Nasre. Popular Mixed Matchings. In *Proceedings of ICALP '09: the 36th International Colloquium on Automata, Languages and Programming*, volume 5555 of *Lecture Notes in Computer Science*, pages 574–584. Springer, 2009.

19 Telikepalli Kavitha. A size-popularity tradeoff in the stable marriage problem. *SIAM Journal on Computing*, 43(1):52–71, 2014.

20 Telikepalli Kavitha. Popular half-integral matchings. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 55. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

21 Telikepalli Kavitha. Max-size popular matchings and extensions. *arXiv preprint arXiv:1802.07440*, 2018.

22 David F. Manlove. *Algorithmics of Matching Under Preferences*, volume 2 of *Series on Theoretical Computer Science*. WorldScientific, 2013. doi:10.1142/8591.

23 Thomas J Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226. ACM, 1978.

24 A. Subramanian. A New Approach to Stable Matching Problems. *SIAM Journal on Computing*, 23(4):671–700, 1994.

25 J.J.M. Tan. A necessary and sufficient condition for the existence of a complete stable matching. *Journal of Algorithms*, 12:154–178, 1991.

26 C.-P. Teo and J. Sethuraman. The geometry of fractional stable matchings and its applications. *Mathematics of Operations Research*, 23(4):874–891, 1998.

27 Gerhard J Woeginger. Core stability in hedonic coalition formation. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 33–50. Springer, 2013.

Graph Pattern Polynomials

Markus Bläser
Department of Computer Science, Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
mblaeser@cs.uni-saarland.de

Balagopal Komarath
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
bkomarath@cs.uni-saarland.de

Karteek Sreenivasaiah[1]
Department of Computer Science and Engineering, Indian Institute of Technology Hyderabad, India
karteek@iith.ac.in

Abstract

Given a *host* graph G and a *pattern* graph H, the induced subgraph isomorphism problem is to decide whether G contains an induced subgraph that is isomorphic to H. We study the time complexity of induced subgraph isomorphism problems when the pattern graph is fixed. Nešetřil and Poljak gave an $O(n^{k\omega})$ time algorithm that decides the induced subgraph isomorphism problem for *any* $3k$ vertex pattern graph (the universal algorithm), where ω is the matrix multiplication exponent. Improvements are not known for *any* infinite pattern family.

Algorithms faster than the universal algorithm are known only for a finite number of pattern graphs. In this paper, we show that there exists infinitely many pattern graphs for which the induced subgraph isomorphism problem has algorithms faster than the universal algorithm.

Our algorithm works by reducing the pattern detection problem into a multilinear term detection problem on special classes of polynomials called graph pattern polynomials. We show that many of the existing algorithms including the universal algorithm can also be described in terms of such a reduction. We formalize this class of algorithms by defining graph pattern polynomial families and defining a notion of reduction between these polynomial families. The reduction also allows us to argue about relative hardness of various graph pattern detection problems within this framework. We show that solving the induced subgraph isomorphism for any pattern graph that contains a k-clique is at least as hard detecting k-cliques. An equivalent theorem is not known in the general case.

In the full version of this paper, we obtain new algorithms for P_5 and C_5 that are optimal under reasonable hardness assumptions. We also use this method to derive new combinatorial algorithms – algorithms that do not use fast matrix multiplication – for paths and cycles. We also show why graph homomorphisms play a major role in algorithms for subgraph isomorphism problems. Using this, we show that the arithmetic circuit complexity of the graph homomorphism polynomial for $K_k - e$ (The k-clique with an edge removed) is related to the complexity of many subgraph isomorphism problems. This generalizes and unifies many existing results.

2012 ACM Subject Classification Theory of computation → Probabilistic computation, Theory of computation → Problems, reductions and completeness

Keywords and phrases algorithms, induced subgraph detection, algebraic framework

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.18

[1] Part of this work was done while the author was at Saarland University, Saarbrücken, Germany.

Related Version The full version of this paper is available on arXiv: `https://arxiv.org/abs/1809.08858`.

Acknowledgements The authors thank Cornelius Brand and Holger Dell for helpful discussions during the early parts of this work. The authors also thank the anonymous reviewers for comments that helped improve the presentation in the paper.

1 Introduction

The *induced subgraph isomorphism problem* asks, given simple and undirected graphs G and H, whether there is an induced subgraph of G that is isomorphic to H. The graph G is called the host graph and the graph H is called the pattern graph. This problem is NP-complete (See [8], problem [GT21]). If the pattern graph H is fixed, there is a simple $O(n^{|V(H)|})$ time algorithm to decide the induced subgraph isomorphism problem for H. We study the time complexity of the induced subgraph isomorphism problem for fixed pattern graphs on the Word-RAM model.

The earliest non-trivial algorithm for this problem was given by Itai and Rodeh [9] who showed that the number of triangles can be computed in $O(n^\omega)$ time on n-vertex graphs, where ω is the exponent of matrix multiplication. Later, Nešetřil and Poljak[11] generalized this algorithm to count K_{3k} in $O(n^{k\omega})$ time, where K_{3k} is the clique on $3k$ vertices. Eisenbrand and Grandoni [3] extended this algorithm further to count K_{3k+j} for $j \in \{0, 1, 2\}$ using rectangular matrix multiplication in $O(n^{\omega(k+\lceil j/2\rceil, k, k+\lfloor j/2\rfloor)})$ time. Here $\omega(i, j, k)$ denotes the exponent of the running time of matrix multiplication when multiplying an $i \times j$ matrix with a $j \times k$ matrix. It is known that detecting/counting any k-vertex pattern is easier than detecting/counting K_k. Therefore, these algorithms are called "universal" algorithms.

Our Contributions

Algorithms that improve the universal algorithm for specific pattern graphs are only known for small fixed values of k. For example, the induced subgraph isomorphism problem for P_4 can be solved in $O(n + m)$ time [1] and all 4-vertex graphs other than K_4 can be detected in $O(n^\omega)$ time [14]. In Section 5, we give the first algorithm that detects infinitely many pattern graphs faster than the universal algorithm.

Our algorithm works by reducing the induced subgraph isomorphism problem into detecting multilinear terms in a related polynomial. This idea has been previously used by many authors (See [13], [10], [5], and [7] for its application to subgraph isomorphism problems) to solve combinatorial problems efficiently. A major contribution of our work is a general framework that can describe many existing algorithms for subgraph isomorphism problems. We show that graph pattern[2] detection problems can be reformulated as the problem of detecting multilinear terms in special classes of polynomials called *graph pattern polynomials* (Defined in Section 4).

We also define a notion of reduction between these polynomials that allows us to argue about the relative hardness of the graph pattern detection problems. It is known that detecting an induced path on $2k$ vertices is at least as hard as detecting a K_k [6]. Intuitively,

[2] Examples of graph patterns include subgraph isomorphisms, induced subgraph isomorphisms, and graph homomorphisms

any pattern graph H that contains a K_k (or equivalently, an independent set on k vertices) should be as hard to detect as a K_k. But this is not known. In Section 6, we show that the graph pattern polynomial for K_k can be reduced to the polynomial that corresponds to the induced subgraph isomorphism problem for H for any H that contains a K_k. This shows that if we can obtain better algorithms for H using our framework, then we obtain better algorithms for K_k. We show that all existing algorithms for induced subgraph isomorphism problems can be either described using our framework or we can obtain algorithms with matching running times using our framework. Therefore, these reductions can be viewed as showing the limitations of current methods for solving subgraph isomorphism problems.

In Section 7, we discuss the results in the full version of this paper. In Section 3, we show how to use graph pattern polynomials to obtain a linear-time algorithm for detecting paths on four vertices.

2 Preliminaries

For a polynomial f, we use $deg(f)$ to denote the degree of f. A monomial is called multilinear, if every variable in it has degree at most one. We use $ML(f)$ to denote the multilinear part of f, that is, the sum of all multilinear monomials in f. An arithmetic circuit computing a polynomial $P \in K[x_1, \ldots, x_n]$ is a circuit with $+$, $\times$ gates where the input gates are labelled by variables or constants from the underlying field and one gate is designated as the output gate. The size of an arithmetic circuit is the number of wires in the circuit. For indeterminates $x_1, \ldots, x_n$ and a set $S = \{s_1, \ldots, s_p\} \subseteq \{1, \ldots, n\}$ of indices, we write x_S to denote the product $x_{s_1} \cdots x_{s_p}$.

An induced subgraph isomorphism from H to G is an injective function $\phi : V(H) \overset{ind}{\mapsto} V(G)$ such that $\{u, v\} \in E(H) \iff \{\phi(u), \phi(v)\} \in E(G)$. Any function from $V(H)$ to $V(G)$ can be extended to unordered pairs of vertices of H as $\phi(\{u, v\}) = \{\phi(u), \phi(v)\}$. A subgraph isomorphism from H to G is an injective function $\phi : V(H) \overset{sub}{\mapsto} V(G)$ such that $\{u, v\} \in E(H) \implies \{\phi(u), \phi(v)\} \in E(G)$. Two subgraph isomorphisms or induced subgraph isomorphisms are considered different only if the set of edges in the image are different. A graph homomorphism from H to G is a function $\phi : V(H) \overset{hom}{\mapsto} V(G)$ such that $\{u, v\} \in E(H) \implies \{\phi(u), \phi(v)\} \in E(G)$. Unlike isomorphisms, we consider two distinct functions that yield the same set of edges in the image as distinct graph homomorphisms. We define $\phi(S) = \{\phi(s) : s \in S\}$.

We write $H \sqsubseteq H'$ ($H \sqsupseteq H'$) to specify that H is a subgraph (supergraph) of H'. The number $tw(H)$ stands for the treewidth of H. We denote the number of automorphisms of H by $\#aut(H)$. The graph K_n is the complete graph on n vertices labelled using $[n]$. We use the fact that $\#aut(H) = 1$ for almost all graphs in many of our results. In this paper, we will frequently consider graphs where vertices are labelled by tuples. A vertex (i, p) is said to have *label* i and *colour* p. An edge $\{(i_1, p_1), (i_2, p_2)\}$ has *label* $\{i_1, i_2\}$ and *colour* $\{p_1, p_2\}$. We will sometimes write this edge as $(\{i_1, i_2\}, \{p_1, p_2\})$. Note that both $\{(i_1, p_1), (i_2, p_2)\}$ and $\{(i_2, p_1), (i_1, p_2)\}$ are written as $(\{i_1, i_2\}, \{p_1, p_2\})$. But the context should make it clear which edge is being rewritten.

We will often use the following short forms to denote specific pattern graphs:

K_ℓ :	A clique on ℓ vertices	I_ℓ :	An independent set on ℓ vertices
$K_\ell - e$:	A K_ℓ with an edge removed	$K_\ell + e$:	A K_ℓ and one more edge on $\ell + 1$ vertices
P_ℓ :	A Path on ℓ vertices	C_ℓ :	A cycle on ℓ vertices

3 A Motivating Example: Induced-P_4 Isomorphism

In this section, we sketch a one-sided error, randomized $O(n^2)$ time algorithm for the induced subgraph isomorphism problem for P_4 to illustrate the techniques used to derive algorithms in this paper.

We start by giving an algorithm for the subgraph isomorphism problem for P_4. Consider the following polynomial:

$$N_{P_4,n} = \sum_{(p,q,r,s):p<s} y_p y_q y_r y_s x_{\{p,q\}} x_{\{q,r\}} x_{\{r,s\}}$$

where the summation is over all quadruples over $[n]$ where all four elements are distinct. Each of the y variables corresponds to a vertex of a possible P_4 and the x variables correspond to the edges. Hence each monomial in the above polynomial corresponds naturally to a P_4 on the vertices p, q, r, s chosen in the summation. The condition $p < s$ ensures that each path has exactly one monomial corresponding to it.

Given an n-vertex host graph G and an arithmetic circuit for $N_{P_4,n}$, we can construct an arithmetic circuit for the polynomial $N_{P_4,n}(G)$ on the y variables obtained by substituting $x_e = 0$ when $e \notin E(G)$ and $x_e = 1$ when $e \in E(G)$. The polynomial $N_{P_4,n}(G)$ can be written as $\sum_X a_X y_X$ where the summation is over all four vertex subsets X of $V(G)$ and a_X is the number of P_4s in the induced subgraph $G[X]$. Therefore, we can decide whether G has a subgraph isomorphic to P_4 by testing whether $N_{P_4,n}(G)$ is identically 0. Since the degree of this polynomial is a constant k, this can be done in time linear in the size of the arithmetic circuit computing $N_{P_4,n}$.

However, we do not know how to construct a $O(n^2)$ size arithmetic circuit for $N_{P_4,n}$. Instead, we construct a $O(n^2)$ size arithmetic circuit for the following polynomial called the walk polynomial:

$$Hom_{P_4,n} = \sum_{\phi:P_4 \overset{hom}{\mapsto} K_n} \prod_{v \in V(P_4)} z_{v,\phi(v)} y_{\phi(v)} \prod_{e \in E(P_4)} x_{\phi(e)}$$

Similar to $N_{P_4,n}$, the y and x variables correspond to vertices and edges respectively. The z variables play the role of fixing the mapping from P_4 to K_n that is chosen in the summation. This polynomial is also called the homomorphism polynomial for P_4 because its terms are in one-to-one correspondence with graph homomorphisms from P_4 to K_n. As before, we consider the polynomial $Hom_{P_4,n}(G)$ obtained by substituting for the x variables appropriately. The crucial observation is that $Hom_{P_4,n}(G)$ contains a multilinear term if and only if $N_{P_4,n}(G)$ is not identically zero. This is because the multilinear terms of $Hom_{P_4,n}$ correspond to injective homomorphisms from P_4 which in turn correspond to subgraph isomorphisms from P_4. More specifically, each P_4 corresponds to two injective homomorphisms from P_4 since P_4 has two automorphisms. Therefore, we can test whether G has a subgraph isomorphic to P_4 by testing whether $Hom_{P_4,n}(G)$ has a multilinear term. It is known that the polynomial $p_4 = Hom_{P_4,n}$ has $O(n^2)$ size circuits using the following inductive construction:

$$\begin{aligned} p_{1,v} &= y_v, v \in [n] \\ p_{i+1,v} &= z_{i+1,v} \sum_{u \in [n]} p_{i,u} y_v x_{\{u,v\}}, v \in [n], i \geq 1 \\ p_4 &= \sum_{v \in [n]} p_{4,v} \end{aligned}$$

The above construction can be extended to construct p_k for any k and not just $k = 4$. This method is used in [13] to obtain an $O(2^k(n+m))$ time algorithm for the subgraph isomorphism problem for P_k.

In fact, the above method works for any pattern graph H. Extend the definitions above to define $N_{H,n}$ and $Hom_{H,n}$ in the natural fashion. Then, we can test whether an n-vertex graph G has a subgraph isomorphic to H by testing whether $N_{H,n}(G)$ is identically zero which in turn can be done by testing whether $Hom_{H,n}(G)$ has a multilinear term. Therefore, the complexity of subgraph isomorphism problem for any pattern H is as easy as constructing the homomophism polynomial for H. This method is used by Fomin et. al. [7] to obtain efficient algorithms for subgraph isomorphism problems.

We now turn our attention to the induced subgraph isomorphism problem for P_4. We note that the induced subgraph isomorphism problem for P_k is much harder than the subgraph isomorphism problem for P_k. The subgraph isomorphism problem for P_k has a linear time algorithm as seen above but the induced subgraph isomorphism problem for P_k cannot have $n^{o(k)}$ time algorithms unless $\mathsf{FPT} = \mathsf{W}[1]$. We start by considering the polynomial:

$$I_{P_4,n} = \sum_{(p,q,r,s):p<s} y_p y_q y_r y_s x_{\{p,q\}} x_{\{q,r\}} x_{\{r,s\}} (1 - x_{\{p,r\}})(1 - x_{\{p,s\}})(1 - x_{\{q,s\}})$$

The polynomial $I_{P_4,n}(G)$ can be written as $\sum_X y_X$ where the summation is over all four vertex subsets of $V(G)$ that induces a P_4. Notice that all coefficents are 1 because there can be at most 1 induced-P_4 on any four vertex subset. By expanding terms of the form $1 - x_*$ in the above polynomial, we observe that we can rewrite $I_{P_4,n}$ as follows:

$$I_{P_4,n} = N_{P_4,n} - 4N_{C_4,n} - 2N_{K_3+e,n} + 6N_{K_4-e,n} + 12N_{K_4,n}$$

Since the coefficients in $I_{P_4,n}(G)$ are all 0 or 1, it is sufficient to check whether $I_{P_4,n}(G) \pmod 2$ is non-zero to test whether $I_{P_4,n}(G)$ is non-zero. From the above equation, we can see that $I_{P_4,n} = N_{P_4,n} \pmod 2$. Therefore, instead of working with $I_{P_4,n} \pmod 2$, we can work with $N_{P_4,n} \pmod 2$. We have already seen that we can use $Hom_{P_4,n}(G)$ to test whether $N_{P_4,n}(G)$ is non-zero. However, this is not sufficient to solve induced subgraph isomorphism. We want to detect whether $N_{P_4,n}(G)$ is non-zero modulo 2. Therefore, the multilinear terms of $Hom_{P_4,n}(G)$ has to be in one-to-one correspondence with the terms of $N_{P_4,n}(G)$. We have to divide the polynomial $Hom_{P_4,n}(G)$ by 2 before testing for the existence of multilinear terms modulo 2. However, since we are working over a field of characteristic 2, this division is not possible. We work around this problem by starting with $Hom_{P_4,n'}$ for n' slightly larger than n and we show that this enables the "division" by 2.

The reader may have observed that instead of the homomorphism polynomial, we could have taken any polynomial f for which the multilinear terms of $f(G)$ are in one-to-one correspondence with $N_{P_4,n}(G)$. This observation leads to the definition of a notion of reduction between polynomials. Informally, $f \preceq g$ if detecting multilinear terms in $f(G)$ is as easy as detecting multilinear terms in $g(G)$. Additionally, for the evaluation $f(G)$ to be well-defined, the polynomial f must have some special structure. We call such polynomials graph pattern polynomials.

On first glance, it appears hard to generalize this algorithm for P_4 to sparse pattern graphs on an arbitrary number of vertices (For example, P_k) because we have to argue about the coefficients of many N_* polynomials in the expansion. On the other hand, if we consider the pattern graph K_k, we have $I_{K_k} = Hom_{K_k}$. In this paper, we show that for many graph patterns sparser than K_k, the induced subgraph isomorphism problem is as easy as constructing arithmetic circuits for homomorphism polynomials for those patterns (or patterns that are only slightly denser).

4 Graph pattern polynomial families

We will consider polynomial families $f = (f_n)$ of the following form: Each f_n will be a polynomial in variables $y_1, \ldots, y_n$, the vertex variables, and variables $x_1, \ldots, x_{\binom{n}{2}}$, the edge variables, and at most linear in n number of additional variables.The degree of each f_n will usually be constant.

The (not necessarily induced) subgraph isomorphism polynomial family $N_H = (N_{H,n})_{n\geq 0}$ for a fixed pattern graph H on k vertices and ℓ edges is a family of multilinear polynomials of degree $k + \ell$. The n^{th} polynomial in the family, defined over the vertex set $[n]$, is the polynomial on $n + \binom{n}{2}$ variables given by (1):

$$N_{H,n} = \sum_{\phi: V(H) \overset{sub}{\mapsto} V(K_n)} y_{\phi(V(H))} x_{\phi(E(H))} \tag{1}$$

When context is clear, we will often omit the subscript n and simply write N_H. Given a (host) graph G on n vertices, we can substitute values for the edge variables of $N_{H,n}$ depending on the edges of G ($x_e = 1$ if $e \in E(G)$ and $x_e = 0$ otherwise) to obtain a polynomial $N_{H,n}(G)$ on the vertex variables. The monomials of this polynomial are in one-to-one correspondence with the H-subgraphs of G. i.e., a term $a y_{v_1} \cdots y_{v_k}$, where a is a positive integer, indicates that there are a subgraphs isomorphic to H in G on the vertices $v_1, \ldots, v_k$. Therefore, to detect if there is an H-subgraph in G, we only have to test whether $N_{H,n}(G)$ has a multilinear term.

The induced subgraph isomorphism polynomial family $I_H = (I_{H,n})_{n\geq 0}$ for a pattern graph H over the vertex set $[n]$ is defined in (2).

$$I_{H,n} = \sum_{\phi: V(H) \overset{ind}{\mapsto} V(K_n)} y_{\phi(V(H))} x_{\phi(E(H))} \prod_{e \notin E(H)} (1 - x_{\phi(e)}) \tag{2}$$

If we substitute the edge variables of $I_{H,n}$ using a host graph G on n vertices, then the monomials of the resulting polynomial $I_{H,n}(G)$ on the vertex variables are in one-to-one correspondence with the induced H-subgraphs of G. In particular, all monomials have coefficient 0 or 1 because there can be at most one induced copy of H on a set of k vertices. This implies that to test if there is an induced H-subgraph in G, we only have to test whether $I_{H,n}(G)$ has a multilinear term and we can even do this modulo p for any prime p. Also, note that I_H is simply $I_{\overline{H}}$ where all the edge variables x_e are replaced by $1 - x_e$.

The homomorphism polynomial family $Hom_H = (Hom_{H,n})_{n\geq 0}$ for pattern graph H over the vertex set $[n]$ is defined in (3).

$$Hom_{H,n} = \sum_{\phi: V(H) \overset{hom}{\mapsto} V(K_n)} \prod_{v \in V(H)} z_{v,\phi(v)} y_{\phi(v)} \prod_{e \in E(H)} x_{\phi(e)} \tag{3}$$

The variables $z_{a,v}$'s are called the *homomorphism variables*. They keep track how the vertices of H are mapped by the different homomorphisms in the summation. We note that the size of the arithmetic circuit computing $Hom_{H,n}$ is independent of the labelling chosen to define the homomorphism polynomial. The arithmetic circuit complexity of such homomorphism polynomials, with respect to properties of the pattern graph, has been studied in [4].

The induced subgraph isomorphism polynomial for any graph H and subgraph isomorphism polynomials for supergraphs of H are related as follows:

$$I_{H,n} = \sum_{H' \supseteq H} (-1)^{e(H')-e(H)} \#sub(H, H') N_{H',n} \tag{4}$$

Here $e(H)$ is the number of edges in H and $\#sub(H, H')$ is the number of times H appears as a subgraph in H'. The sum is taken over all supergraphs H' of H having the same vertex set as H. Equation 4 is used by Curticapean, Dell, and Marx [2] in the context of counting subgraph isomorphisms.

For any fixed pattern graph H, the degree of polynomial families N_H, I_H, and Hom_H are bounded by a constant depending only on the size of H. Such polynomial families are called constant-degree polynomial families.

▶ **Definition 4.1.** A constant-degree polynomial family $f = (f_n)$ is called a *graph pattern* polynomial family if the n^{th} polynomial in the family has n vertex variables, $\binom{n}{2}$ edge variables, and at most cn other variables, where c is a constant, and every non-multilinear term of f_n has at least one non-edge variable of degree greater than 1.

It is easy to verify that I_H, N_H, and Hom_H are all graph pattern polynomial families. For a graph pattern polynomial f, we denote by $f(G)$ the polynomial obtained by substituting $x_e = 0$ if $e \notin E(G)$ and $x_e = 1$ if $e \in E(G)$ for all edge variables x_e. Note that for any graph pattern polynomial f, we have $ML(f(G)) = ML(f)(G)$. This is because any non-multilinear term in f has to remain non-multilinear or become 0 after this substitution.

▶ **Definition 4.2.**

1. A constant degree polynomial family $f = (f_n)$ has circuits of size $s(n)$ if there is a sequence of arithmetic circuits (C_n) such that C_n computes f_n and has size at most $s(n)$.
2. f has uniform $s(n)$-size circuits, if on input n, we can construct C_n in time $O(s(n))$ on a Word-RAM.[3]

We now define a notion of reducibility among graph pattern polynomials. Informally, if $f \preceq g$, then we detecting whether $f_n(G)$ has a multilinear term is as easy as constructing an arithmetic circuit for g_n for all n. First, we define a notion of substitution families that preserves the semantic structure of graph pattern polynomials.

▶ **Definition 4.3.** A *substitution family* $\sigma = (\sigma_n)$ is a family of mappings

$$\sigma_n : \{y_1, \ldots, y_n, x_1, \ldots, x_{\binom{n}{2}}, u_1, \ldots, u_{m(n)}\} \to K[y_1, \ldots, y_{n'}, x_1, \ldots, x_{\binom{n'}{2}}, v_1, \ldots, v_{r(n)}]$$

mapping variables to polynomials such that:

1. σ maps vertex variables to constant-degree monomials containing one or more vertex variables or other variables, and no edge variables.
2. σ maps edge variables to polynomials with constant-size circuits containing at most one edge variable and no vertex variables.
3. σ maps other variables to constant-degree monomials containing no vertex or edge variables and at least one other variable.

σ_n naturally extends to $K[y_1, \ldots, y_n, x_1, \ldots, x_{\binom{n}{2}}, u_1, \ldots, u_{m(n)}]$.

For the reduction to be useful in deriving algorithms, the substitution has to be easily computable. This leads us to the following definition.

▶ **Definition 4.4.** A substitution family $\sigma = (\sigma_n)$ is *constant-time computable* if given n and a variable z in the domain of σ_n, we can compute $\sigma_n(z)$ in constant-time on a Word-RAM. (Note that an encoding of any z fits into on cell of memory.)

[3] Since we are dealing with fine-grained complexity, we have to be precise with the encoding of the circuit. We assume an encoding such that evaluating the circuit is linear time and substituting for variables with polynomials represented by circuits is constant-time.

Finally, we define our notion of reduction.

▶ **Definition 4.5.** Let $f = (f_n)$ and $g = (g_n)$ be graph pattern polynomial families. Then f is reducible to g, denoted $f \preceq g$, via a constant time computable substitution family $\sigma = (\sigma_n)$ if for all n there is an $m = O(n)$ and $q = O(1)$ such that

1. $\sigma_m(ML(g_m))$ is a graph pattern polynomial and
2. $ML(\sigma_m(g_m)) = v_{[q]}ML(f_n)$. (Recall that $v_{[q]} = v_1 \cdots v_q$.)

For any prime p, we say that $f \preceq g \pmod p$ if there exists an $f' = f \pmod p$ such that $f' \preceq g$.

Property 1 of Definition 4.5 and Properties 1 and 3 of Definition 4.3 imply that $\sigma_m(g_m)$ is a graph pattern polynomial because Properties 1 and 3 of Definition 4.3 ensure that non-multilinear terms remain so after the substitution. It is easy to see that $\preceq$ is reflexive via the identity substitution. We can also assume w.l.o.g. that the variables $v_1, \ldots, v_q$ are fresh variables introduced by the substitution family σ.

What is the difference between $\sigma_m(ML(g_m))$ and $ML(\sigma_m(g_m))$ in the Definition 4.5? Every monomial in $ML(\sigma_m(g_m))$ also appears in $\sigma_m(ML(g_m))$, however, the latter may contain further monomials that are not multilinear.

It is easy to see that $\preceq$ is reflexive via the identity substitution. It can be shown that $\preceq$ is transitive by composing substitutions.

We conclude this section by mentioning how to obtain efficient algorithms using $\preceq$. Efficient algorithms are known (See [10]) for detecting multilinear terms of *small* degree with non-zero coefficient modulo primes.

▶ **Theorem 4.6.** *Let k be any constant and let p be any prime. Given an arithmetic circuit of size s, there is a randomized, one-sided error $O(s)$-time algorithm to detect whether the polynomial computed by the circuit has a multilinear term of degree at most k with non-zero modulo p coefficient.*

An important algorithmic consequence of reducibility is stated in Proposition 4.7. This proposition is used to derive algorithms for induced subgraph isomorphism problems in this paper.

▶ **Proposition 4.7.** *Let p be any prime. Let f and g be graph pattern polynomial families. Let $s(n)$ be a polynomially-bounded function. If $f \preceq g \pmod p$ and g has size uniform $s(n)$-size arithmetic circuits, then we can test whether $f_n(G)$ has a multilinear term with non-zero coefficient modulo p in $O(s(n))$ (randomized one-sided error) time for any n-vertex graph G.*

5 Pattern graphs easier than cliques

In this section, we describe a family H_{3k} of pattern graphs such that the induced subgraph isomorphism problem for H_{3k} has an $O(n^{\omega(k,k-1,k)})$ time algorithm when $k = 2^\ell, \ell \geq 1$. Note that for the currently known best algorithms for fast matrix multiplication, we have $\omega(k, k-1, k) < k\omega$. Therefore, these pattern graphs are strictly easier to detect than cliques.

The pattern graph H_{3k} is defined on $3k$ vertices and we consider the canonical labelling of H_{3k} where there is a $(3k-1)$-clique on vertices $\{1, \ldots, 3k-1\}$ and the vertex $3k$ is adjacent to the vertices $\{1, \ldots, 2k-1\}$.

▶ **Lemma 5.1.** $I_{H_{3k}} = N_{H_{3k}} \pmod 2$ *when* $k = 2^\ell, \ell \geq 1$

Proof. We show that the number of times H_{3k} is contained in any of its proper supergraphs is even if k is a power of 2. The graph K_{3k} contains $3k\binom{3k-1}{2k-1}$ copies of H_{3k}. This number is even when k is even. The graph $K_{3k} - e$ contains $2\binom{3k-2}{2k-1}$ copies of H_{3k}. This number is always even. The remaining proper supergraphs of H_{3k} are the graphs $K_{3k-1} + (2k+i)e$, i.e., a $(3k-1)$-clique with $2k+i$ edges to a single vertex, for $0 \leq i < k-2$. There are $m_i = \binom{2k+i}{2k-1}$ copies of the graph H_{3k} in these supergraphs. We observe that the numbers m_i are even when $k = 2^\ell, \ell \geq 1$ by Lucas' theorem. Lucas' theorem states that $\binom{p}{q}$ is even if and only if in the binary representation of p and q, there exists some bit position i such that $q_i = 1$ and $p_i = 0$. To see why m_i is even, observe that in the binary representation of $2k-1$, all bits 0 through ℓ are 1 and in the binary representation of $2k+i, 0 \leq i < k-2$, at least one of those bits is 0. ◀

▶ **Lemma 5.2.** $N_{H_{3k}} \preceq Hom_{H_{3k}}$

Proof. We start with $Hom_{H_{3k}}$ over the vertex set $[n] \times [3k]$ and apply the following substitution.

$$\sigma(z_{a,(v,a)}) = z_a \tag{1}$$
$$\sigma(z_{a,(v,b)}) = z_a^2, a \neq b \tag{2}$$
$$\sigma(y_{(v,a)}) = y_v \tag{3}$$
$$\sigma(x_{(u,a),(v,b)}) = 0, \text{if } a,b \in \{1,\ldots,2k-1\} \text{ and } a < b \text{ and } u > v \tag{4}$$
$$\sigma(x_{(u,a),(v,b)}) = 0, \text{if } a,b \in \{2k,\ldots,3k-1\} \text{ and } a < b \text{ and } u > v \tag{5}$$
$$\sigma(x_{(u,a),(v,b)}) = x_{\{u,v\}}, \text{otherwise} \tag{6}$$

Rule 3 ensures that in any surviving monomial, all vertices have distinct labels. Rule 4 ensures that the vertices coloured $1,\ldots,2k-1$ are in increasing order and Rule 5 ensures that the vertices coloured $2k,\ldots,3k-1$ are in increasing order.

Consider an H_{3k} labelled using $[n]$ where the vertices in the $(3k-1)$-clique are labelled $v_1,\ldots,v_{3k-1}$ and the remaining vertex is labelled v_{3k} which is connected to $v_1 < \ldots < v_{2k-1}$. Also, $v_{2k} < \ldots < v_{3k-1}$. We claim that the monomial corresponding to this labelled H_{3k} (say m) is uniquely generated by the monomial $m' = \prod_{1\leq i\leq 3k} z_{i,(v_i,i)} w$ in $Hom_{H_{3k}}$. Note that the vertices and edges in the image of the homomorphism is determined by the map $i \mapsto (v_i, i)$. The monomial w is simply the product of these vertex and edge variables. It is easy to see that this monomial yields the required monomial under the above substitution. The uniqueness is proved as follows: observe that in any monomial m'' in $Hom_{H_{3k}}$ that generates m, the vertex coloured $3k$ must be v_{3k}. This implies that the vertices coloured $1,\ldots,2k-1$ must be the set $\{v_1,\ldots,v_{2k-1}\}$. Rule 4 ensures that vertex coloured i must be v_i for $1 \leq i \leq 2k-1$. Similarly, the vertices coloured $2k,\ldots,3k-1$ must be the set $\{v_{2k},\ldots,v_{3k-1}\}$ and Rule 5 ensures that vertex coloured i must be v_i for $2k \leq i \leq 3k-1$ as well. But then the monomials m' and m'' are the same. ◀

▶ **Lemma 5.3.** *$Hom_{H_{3k}}$ can be computed by arithmetic circuits of size $O(n^{\omega(k,k-1,k)})$ for $k > 1$.*

Proof. Consider H_{3k} labelled as before. We define the sets $S_{1,k,2k,3k-1} = \{1,\ldots,k,2k\ldots,3k-1\}$, $S_{k+1,3k-1} = \{k+1,\ldots,3k-1\}$, $S_{k+1,2k-1} = \{k+1,\ldots,2k-1\}$, and $S_{1,2k-1} = \{1,\ldots,2k-1\}$. We also define the tuples $V_{1,k} = (v_1,\ldots,v_k)$, $V_{2k,3k-1} = (v_{2k},\ldots,v_{3k-1})$, and $V_{k+1,2k-1} = (v_{k+1},\ldots,v_{2k-1})$ for any set v_i of $3k-1$ distinct vertex labels. The algorithm

also uses the matrices defined below. The dimensions of each matrix are specified as the superscript. All other entries of the matrix are 0. Notice that all entries are constant-sized monomials.

$$A^{n^k \times n^k}_{V_{1,k},V_{2k,3k-1}} = \left(\prod_{i \in S_{1,k,2k,3k-1}} z_{i,v_i} y_{v_i}\right)\left(\prod_{\substack{i,j \in S_{1,k,2k,3k-1} \\ i \neq j}} x_{\{v_i,v_j\}}\right)$$

$$B^{n^k \times n^{k-1}}_{V_{2k,3k-1},V_{k+1,2k-1}} = \left(\prod_{i \in S_{k+1,2k-1}} z_{i,v_i} y_{v_i}\right)\left(\prod_{\substack{i \in S_{k+1,3k-1} \\ j \in S_{k+1,2k-1} \\ i \neq j}} x_{\{v_i,v_j\}}\right)$$

$$C^{n^{k-1} \times n^k}_{V_{k+1,2k-1},V_{1,k}} = x_{\{(v_i,i)_{i \in S_{1,2k-1}}\}} \prod_{\substack{i \in S_{k+1,2k-1} \\ j \in [k] \\ i \neq j}} x_{\{v_i,v_j\}}$$

$$D^{n^k \times n}_{V_{1,k},v_{3k}} = z_{3k,v_{3k}} y_{v_{3k}} \prod_{i \in [k]} x_{\{v_i,v_{3k}\}}$$

$$E^{n \times n^{k-1}}_{v_{3k},V_{k+1,2k-1}} = \prod_{i \in S_{k+1,2k-1}} x_{\{v_i,v_{3k}\}}$$

Compute the matrix products ABC and DE. Replace the n^{2k-1} variables $x_{\{(v_i,i)_{i \in S_3}\}}$ with $(DE)_{V_{1,k},V_{k+1,2k-1}}$. The required polynomial is then just

$$Hom_{H_{3k}} = \sum_{(v_1,\ldots,v_k)} (ABC)_{(v_1,\ldots,v_k),(v_1,\ldots,v_k)}$$

Consider a homomorphism of H_{3k} defined as $\phi : i \mapsto u_i$. The monomial corresponding to this homomorphism is uniquely generated as follows. Let U_* be defined similarly to the tuples V_*. Set $v_i = u_i$ for $i \in [k]$ in the summation and consider the monomial generated by the product $A_{U_{1,k},U_{2k,3k-1}} B_{U_{2k,3k-1},U_{k+1,2k-1}} C_{U_{k+1,2k-1},U_{1,k}}$ after replacing the variable $x_{\{(u_i,i)_{i \in S_3}\}}$ by $(DE)_{U_{1,k},U_{k+1,2k-1}}$ taking the monomial $D_{U_{1,k},u_{3k}} E_{u_{3k},U_{k+1,2k-1}}$ from that entry. It is easy to verify that this generates the required monomial. For uniqueness, observe that this is the only way to generate the required product of the homomorphism variables.

Computing ABC can be done using $O(n^{\omega(k,k-1,k)})$ size circuits. Computing DE can be done using $O(n^{\omega(k,1,k-1)})$ size circuits. The top level sum contributes $O(n^k)$ gates. This proves the lemma. ◀

We conclude this section by stating our main theorem.

▶ **Theorem 5.4.** *The induced subgraph isomorphism problem for H_{3k} has an $O(n^{\omega(k,k-1,k)})$ time algorithm when $k = 2^\ell, \ell \geq 1$.*

6 Lower Bounds for Pattern Graphs with Cliques

Since we can obtain algorithms for induced subgraph isomorphism problems that match the known best algorithms using reductions between graph pattern polynomials, we can interpret the reduction $f \preceq g$ as evidence that detecting the graph pattern corresponding to g is harder than detecting f. It is known that the induced subgraph isomorphism problem for P_{2k} is harder than that for K_k. In general, one would think that detecting any graph H

that contains K_k as a subgraph would be at least as hard as detecting K_k. However, this is known only when H has a K_k that is vertex disjoint from all other K_k in H. The following theorem shows that we can drop this restriction when working with pattern polynomials.

▶ **Theorem 6.1.** *If H contains a k-clique or a k-independent set, then $I_{K_k} \preceq I_H$.*

Proof. We will prove the statement when H contains a k-clique. The other part follows because if H contains a k-independent set, then the graph $\overline{H}$ contains a k-clique and $I_{K_k} \preceq I_{\overline{H}} \preceq I_H$.

Fix a labelling of H where the vertices of a k-clique are labelled using $[k]$ and the remaining vertices are labelled $k+1, \ldots, k+\ell$. Consider the polynomial I_H over the vertex set $([n] \times [k]) \cup \{(n+i, k+i) : 1 \leq i \leq \ell\}$ and apply the following substitution.

$$\sigma(y_{(i,p)}) = \begin{cases} y_i u_p & \text{if } i \in [n] \text{ and } p \in [k] \\ u_p & \text{otherwise} \end{cases} \tag{1}$$

$$\sigma(x_{\{(i_1,p_1),(i_2,p_2)\}}) = \begin{cases} x_{\{i_1,i_2\}} & \text{if } \{p_1,p_2\} \in E(K_k) \text{ and } p_1 < p_2 \text{ and } i_1 < i_2 \\ 1 & \text{if } \{p_1,p_2\} \in E(H) \setminus E(K_k) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Consider a k-clique on the vertices $i_1, \ldots, i_k \in [n]$ on an n-vertex graph where $i_1 < \cdots < i_k$. The monomial in I_{K_k} corresponding to this clique is generated uniquely from the monomial $y_{(i_1,1)} \cdots y_{(i_k,k)} \prod_i y_{(n+i,k+i)} x_{\{(i_1,1),(i_2,2)\}} \cdots x_{\{(i_{k-1},k-1),(i_k,k)\}} w$ in I_H, where w is the product of all edge variables corresponding to edges in H but not in K_k. Note that Rules 1 and 2 ensure that in any surviving monomial, the labels and colours of all vertices are distinct and the colours of the edges must be the same as $E(H)$. The product w is determined by $i_1, \ldots, i_k$. This proves that $ML(\sigma(I_H)) = u_{[k+\ell]} ML(I_{K_k})$. It is easy to verify that the substitution satisfies the other properties. ◀

Using reductions between graph pattern polynomial families, it is possible to give evidence for many "natural" relative hardness results. We see these hardness results as showing the limitations of current methods for solving induced subgraph isomorphism problems.

7 Discussion

Are there patterns other than the pattern in Theorem 5.4 for which we can use homomorphism polynomials of graphs sparser than K_k for solving the induced subgraph isomorphism problem? In the full version of this paper, we show that we can obtain better algorithms for paths and cycles using our method. More specifically, we show that the induced subgraph isomorphism problems for P_5 and C_5 can be done in $O(n^\omega)$ time which is optimal assuming the optimality of triangle detection. We also show how to speed up algorithms for P_k and C_k when $k \leq 9$.

An interesting class of algorithms for induced subgraph isomorphism problems are the so called combinatorial algorithms – algorithms that do not use fast matrix multiplication. The best combinatorial algorithm known for k-cliques is the trivial $O(n^k)$ time algorithm. Contrary to general algorithms, we know that many patterns have improved combinatorial algorithms. For example, Virginia Williams [12] showed that there is a $O(n^{k-1})$ time combinatorial algorithm for the induced subgraph isomorphism problem for $K_k - e$. In fact, we show that ,from existing results, one can obtain combinatorial algorithms running in time $O(n^{k-1})$ for all patterns except K_k and I_k. Furthermore, for P_k and C_k we show that we can obtain new combinatorial algorithms running in time $O(n^{k-2})$.

In the full version of the paper, we show that the complexity of many pattern detection and counting problems can be linked to the circuit complexity of homomorphism polynomials for $K_k - e$. We show that if there are $O(n^{f(k)})$ size circuits for Hom_{K_k-e}, then:

1. The induced subgraph isomorphism problem for any k-vertex pattern other than K_k, I_k can be solved in $O(n^{f(k)})$ time. This shows that the induced subgraph isomorphism problem for any k-vertex pattern has a $O(n^{k-1})$ time combinatorial algorithm. This also shows that when $k \leq 9$, all patterns other than K_k, I_k have faster algorithms.
2. The number of subgraphs isomorphic to any k-vertex pattern can be counted in $O(n^{f(k)})$ time.
3. If we can count the number of induced subgraphs isomorphic to some k-vertex pattern in $O(t(n))$ time, then we can count all k-vertex patterns in $O(n^{f(k)} + t(n))$ time. This implies that for $k \leq 9$, improved algorithms for counting any k-vertex pattern will improve algorithms for counting k-cliques.

We also explain why homomorphism polynomials feature prominently in many results related to subgraph isomorphism. We show that for any pattern H, if there exists a family of polynomials such that $N_H \preceq f$, then the size complexity of Hom_H is at most the size complexity of f. Therefore, in a concrete sense, homomorphism polynomials are the best graph pattern polynomial families for subgraph isomorphism problems.

We also use reductions between graph pattern polynomial families similar to Theorem 6.1 to show many lower bounds that seem natural but are not known for general algorithms.

1. For almost all pattern graphs H, the induced subgraph isomorphism problem for H is harder than the subgraph isomorphism problem for H (A randomized reduction is to just randomly delete edges from the graph).
2. For almost all pattern graphs H, the subgraph isomorphism problem for H is easier than subgraph isomorphism problems for any supergraph of H.

Note however that we do not know whether these lower bounds imply general algorithmic hardness. But we believe that these results show the limitations of existing methods for solving subgraph isomorphism problems.

References

1 D. Corneil, Y. Perl, and L. Stewart. A Linear Recognition Algorithm for Cographs. *SIAM Journal on Computing*, 14(4):926–934, 1985. doi:10.1137/0214065.

2 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 210–223. ACM, 2017. doi:10.1145/3055399.3055502.

3 Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theoretical Computer Science*, 326(1):57–67, 2004. doi:10.1016/j.tcs.2004.05.009.

4 Christian Engels. Dichotomy Theorems for Homomorphism Polynomials of Graph Classes. *J. Graph Algorithms Appl.*, 20(1):3–22, 2016.

5 Peter Floderus, Miroslaw Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Detecting and Counting Small Pattern Graphs. *SIAM J. Discrete Math.*, 29(3):1322–1339, 2015. doi:10.1137/140978211.

6 Peter Floderus, Miroslaw Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Induced subgraph isomorphism: Are some patterns substantially easier than others? *Theor. Comput. Sci.*, 605:119–128, 2015. doi:10.1016/j.tcs.2015.09.001.

7 Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, Saket Saurabh, and B. V. Raghavendra Rao. Faster algorithms for finding and counting subgraphs. *J. Comput. Syst. Sci.*, 78(3):698–706, 2012. doi:10.1016/j.jcss.2011.10.001.

8 Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

9 Alon Itai and Michael Rodeh. Finding a Minimum Circuit in a Graph. *SIAM Journal on Computing*, 7(4):413–423, 1978. doi:10.1137/0207033.

10 Ioannis Koutis and Ryan Williams. LIMITS and applications of group algebras for parameterized problems. *ACM Trans. Algorithms*, 12(3):31:1–31:18, 2016. doi:10.1145/2885499.

11 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 026(2):415–419, 1985. URL: http://eudml.org/doc/17394.

12 Virginia Vassilevska. *Efficient Algorithms for Path Problems in Weighted Graphs*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, August 2008.

13 Ryan Williams. Finding paths of length k in $O^*(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, 2009. doi:10.1016/j.ipl.2008.11.004.

14 Virginia Vassilevska Williams, Joshua R. Wang, Richard Ryan Williams, and Huacheng Yu. Finding Four-Node Subgraphs in Triangle Time. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1671–1680. SIAM, 2015. doi:10.1137/1.9781611973730.111.

Shortest k-Disjoint Paths via Determinants

Samir Datta[1]
Chennai Mathematical Institute and UMI ReLaX, Chennai, India
sdatta@cmi.ac.in

Siddharth Iyer[2]
University of Washington, Seattle, USA
sviyer97@gmail.com

Raghav Kulkarni
Chennai Mathematical Institute, Chennai, India
kulraghav@gmail.com

Anish Mukherjee[3]
Chennai Mathematical Institute, Chennai, India
anish@cmi.ac.in

Abstract

The well-known k-disjoint path problem (k-DPP) asks for pairwise vertex-disjoint paths between k specified pairs of vertices (s_i, t_i) in a given graph, if they exist. The decision version of the shortest k-DPP asks for the length of the shortest (in terms of total length) such paths. Similarly, the search and counting versions ask for one such and the number of such shortest set of paths, respectively.

We restrict attention to the shortest k-DPP instances on undirected planar graphs where all sources and sinks lie on a single face or on a pair of faces. We provide efficient sequential and parallel algorithms for the search versions of the problem answering one of the main open questions raised by Colin de Verdière and Schrijver [13] for the general one-face problem. We do so by providing a randomised NC^2 algorithm along with an $O(n^{\omega/2})$ time randomised sequential algorithm, for any fixed k. We also obtain deterministic algorithms with similar resource bounds for the counting and search versions. In contrast, previously, only the sequential complexity of decision and search versions of the "well-ordered" case has been studied. For the one-face case, sequential versions of our routines have better running times for constantly many terminals.

The algorithms are based on a bijection between a shortest k-tuple of disjoint paths in the given graph and cycle covers in a related digraph. This allows us to non-trivially modify established techniques relating counting cycle covers to the determinant. We further need to do a controlled inclusion-exclusion to produce a polynomial sum of determinants such that all "bad" cycle covers cancel out in the sum allowing us to count "pure" cycle covers.

2012 ACM Subject Classification Theory of computation → Parallel algorithms

Keywords and phrases disjoint paths, planar graph, parallel algorithm, cycle cover, determinant, inclusion-exclusion

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.19

Related Version https://arxiv.org/abs/1802.01338

[1] The author was partially funded by a grant from Infosys foundation and SERB grant MTR/2017/000480.
[2] The work was done while the author was a student at Birla Institute of Technology and Science, India.
[3] The author was partially supported by a grant from Infosys foundation and TCS PhD fellowship.

1 Introduction

1.1 The k-disjoint paths problem

The k-Disjoint Path Problem, denoted by k-DPP, is a well-studied problem in algorithmic graph theory with many applications in transportation networks, VLSI-design and most notably in the algorithmic graph minor theory (see for instance [19] and references therein). The k-DPP can be formally defined as follows: Given a (directed/undirected) graph $G = (V, E)$ together with k specified pairs of terminal vertices (s_i, t_i) for $i \in [k]$, find k pairwise vertex-disjoint paths P_i from s_i to t_i, if they exist. One may similarly define an edge-disjoint variant (EDPP) of the problem. We will mainly focus on the vertex-disjoint variant in this paper though several of our results are translated to an edge-disjoint variant as well. The Shortest k-DPP asks to find k pairwise vertex-disjoint paths of minimum total length. We consider the following variants of Shortest k-DPP:

1. Decision: given w, decide if there is a set of k-disjoint paths of length at most w.
2. Construction/Search: construct one set of shortest k-disjoint paths.
3. Counting: count the number of shortest k-disjoint paths.

1.2 Finding k-disjoint paths: Historical overview

The existence as well as construction versions of k-DPP are well-studied in general as well as planar graphs. The problem in general directed graphs is NP-hard even for $k = 2$ [16]. It is one of Karp's NP-hard problems [18] (when k is part of the input) and remains so when restricted to undirected planar graphs [20] and [22] extends this to EDPP as well. In fact, EDPP remains NP-hard even on planar undirected graphs when all the terminals lie on a single face [29].The problem of finding two disjoint paths, one of which is a shortest path, is also NP-hard [14].

The existence of a One/Two-Face k-DPP was studied in [24] as part of the celebrated *Graph Minors* series. This was extended (for fixed k) to graphs on a surface [25] and general undirected graphs [26] in later publications in the same series [26]. A solution to this problem was central to the Graph Minors Project and adds to the importance of the corresponding optimization version. Even when k is part of the input, Suzuki et al. [30] gave linear time and $O(n \log n)$ time algorithms for the One-Face and the Two-Face case, respectively and [31] gave NC algorithms for both. In directed graphs, for fixed k polynomial time algorithms are known when the graph is either planar [28] or acyclic [16].

Though there are recent exciting works on planar restrictions of the problem (e.g. [7]) and even on grid graphs where all the terminals lie on the outer-face [9], the One-Face or Two-Face setting might appear on first-look to be a bit restrictive. However, the One-Face setting occurs naturally in the context of routing problems for VLSI circuits where the graph is a two dimensional grid and all the terminals lie on the outer face. In Relaxations of the One-Face setting become intractable, e.g., "only all source-terminals on one face" is hard to even approximate under a reasonable complexity assumption (NP $\neq$ quasi-P [8]).

1.3 Shortest k-DPP: Related work

The optimization problem is considerably harder. A version of the problem is called *length-bounded* DPP, where each of the path need to have length bounded by some integer ℓ. This problem is NP-hard in the strong sense even in the One-Face case for unbounded k [34]. For the shortest k-DPP, where we want to minimise the sum of the lengths of the paths, very few instances are known to be solvable in polynomial time. For general undirected graphs, very

recently, Björklund and Husfeldt [3] have shown that shortest 2-DPP admits a randomised polynomial time algorithm. The deterministic polynomial time bound for the same – to this date – remains an intriguing open question.

For planar graphs, Colin de Verdière and Schrijver [13] and Kobayashi and Sommer [19] give polynomial time algorithms for shortest k-DPP in some special cases. An $O(kn \log n)$ time algorithm is given in [13] for the case when the sources are incident on one face and sinks on another. In [19] an $O(n^4 \log n)$ time and $O(n^3 \log n)$ time algorithm is given when the terminal vertices are on one face for $k \leq 3$ or on two faces for $k = 2$, respectively. For arbitrary k, linear time algorithm is known for bounded tree-width graphs [27]. Polynomial time algorithms are also known through reducing the problems to the minimum cost flow problem when all the sources (or sinks) coincides or when the terminal vertices lie on a face in the (parallel) order $s_1, s_2, \ldots, s_k, t_k, \ldots, t_2, t_1$ [34].

In [13] the authors ask about the existence of a polynomial time algorithm provided all the terminals are on a common face, for which we give an efficient deterministic algorithm for $k = O(\log n)$. The only progress on this was made by Borradaile et al. [5] where an $O(kn^5)$ time algorithm is presented when corresponding sources and sinks are in series on the boundary of a common face and more recently, by Erickson and Wang [15] who give an $O(n^6)$ time algorithm for $k = 4$. All the previous One-Face planar results are strictly more restrictive or orthogonal to our setting and our sequential algorithms are more efficient (for fixed k). We are able to tackle the counting version that is typically harder than the decision version. Also, to the best of our knowledge, none of the previous works have addressed the parallel complexity of these problems. Very recently, Björklund and Husfeldt [4] presented an algorithm for the $k = 2$ case in max-degree 3 planar graphs with no restriction on the placement of the terminals. Interestingly, like our algorithms, their algorithm also uses determinants (with some additional techniques) to count the solutions.

1.4 Our results and techniques

▶ **Theorem 1.** *Given an undirected planar graph with k pairs of source and sink terminals on the boundary of a common face we can count all shortest k-disjoint paths between the terminals in $O(4^k n^{\omega/2+1})$ time.*

Here $\omega < 2.373$ is the matrix multiplication constant. We also get efficient randomised algorithm (through isolation *a la* [23] and matrix inversion) and deterministic algorithm (using the counting procedure as an oracle) to construct a witness.

▶ **Theorem 2.** *Given an undirected planar graph with k pairs of source and sink terminals on the boundary of a common face, finding a shortest set of k-disjoint paths between the terminals is in randomised $O(4^k n^{\omega/2})$ time and in deterministic $O(4^k n^{\omega/2+2})$ time, respectively.*

The counting algorithm is based on computing several determinants in parallel along with a large matrix inversion which, for k logarithmic in n, can be computed using NC (efficient-parallel) algorithms, i.e., using uniform circuits of polynomial size and polylogarithmic depth. Hence we also get the following result.

▶ **Theorem 3.** *Given an undirected planar graph with k pairs of source and sink terminals on the boundary of a common face and k logarithmic in n, we can count all shortest k-disjoint paths between the terminals in* NC.

From the randomised procedure of Theorem 2 we also get a randomized NC (RNC) algorithm to construct a witness. Our algorithms work for weighted graphs where each edge is assigned a weight which is polynomially bounded in the number of vertices. All our results also hold

Table 1 Summary of Results. The dependence on k and n of our results (in **bold**) is emphasized. Note that ω is the matrix multiplication constant.

Problem	Variant	Sequential		Parallel
		Deterministic	Randomised	
One-Face General	Decision	$\mathbf{4^k n^{\omega/2}}$		**NC**
	Counting	$\mathbf{4^k n^{\omega/2+1}}$		**NC**
	Search	$\mathbf{4^k n^{\omega/2+2}}$	$\mathbf{4^k n^{\omega/2}}$	**RNC**
Two-Face Parallel	Decision	$\mathbf{kn^{\omega}}$		**NC**
	Counting	$\mathbf{kn^{\omega+1}}$		**NC**
	Search	$\mathbf{kn^{\omega+2}}$ ($kn \log n$[13])	$\mathbf{kn^{\omega}}$	**RNC**

for the case when all the source vertices lie on a single face and the sinks on another, with an extra $n^{\omega/2}$ factor blow up in the sequential runtime. Though a more efficient algorithm for the search version is known from [13], we provide an efficient parallel algorithm which is also able to count. Our algorithms extend to a variant of the edge-disjoint version of the problem (for decision and search) by known reductions to the vertex disjoint case. We obtain running times independent of k when the terminal vertices on the faces are in parallel order. We summarize our main results in Table 1. The proof of Theorem 1 depends on the following ideas:

- An injection from k disjoint paths to cycle covers in a related graph for the general case.
- The injection above reduces to a bijection in the parallel case. (Lemma 29)
- An identity involving telescoping sums to simplify the count of k-disjoint paths. (Lemma 16)

We sketch these ideas in more detail below.

Proof Sketch

Throughout the following sketch we talk about pairings which are essentially a collection of k source-sink pairs, though not necessarily the same one which was specified in the input. We refer to this input pairing by M_0.

1. **One-Face Case.** We first convert the given undirected planar graph into a directed one such that each set of disjoint paths between the source-sink pairs in M_0 corresponds to directed cycle covers (Lemma 5). In this process, we might introduce "bad" cycle covers corresponding to pairings of terminals which are not required and they need to be cancelled out. Each "bad" cycle cover which was included, can be mapped to a unique pairing, say M_1. Since the "bad" cycle cover occurs in M_0 as well as M_1 we can cancel it out by adding or subtracting the determinant of M_1 from M_0. However, M_1 can introduce further "bad" cycle covers which again need to be cancelled. We show that all the "bad" cycle covers like this can be cancelled by adding or subtracting determinants exactly like in an inclusion-exclusion formula over a DAG (Lemma 16). This process terminates with the so called "parallel" pairings (where the correspondence between k-disjoint paths and cycle covers with k non-trivial cycles is a bijection) (Lemma 29).
2. **Counting.** The cycle covers in a graph can be counted by a determinant - more precisely, we have a univariate polynomial which is the determinant of some matrix such that every cycle cover corresponds with one monomial in the determinant expansion. Since the "bad" cycle covers cancel out in the inclusion-exclusion, the coefficient of the least degree term gives the correct count of the shortest cycle covers in M_0 which can then be extracted out by interpolation.

3. **Two-Face Case.** The inclusion-exclusion formula exploited the topology of the One-Face case which is not present in the Two-Face case. Here, this approach breaks down as the pairings can not be put together as a DAG. We resolve this for a special case when all sources are on one face and all sinks are on the other by using a topological artifice to prune out pairings which cause cycles. For the Two-Face case, we need the number of cycle covers with a certain winding number modulo k. This can be read off from the monomial with the appropriate exponent in the determinant polynomial.

Main Technical Contribution

Our main technical ingredient here is the Cancellation Lemma 16 that makes it possible to reduce the count of disjoint paths to signed counts over a larger set in such a way that the spurious terms cancel out. This reduces the count of disjoint paths to the determinant. To the best of our knowledge this is the first time a variant of the disjoint path problem has been reduced to the determinant, a parallelizable quantity (in contrast [3] reduce 2-DPP to the Permanent modulo 4 for which no parallel algorithm is known).

1.5 Organization

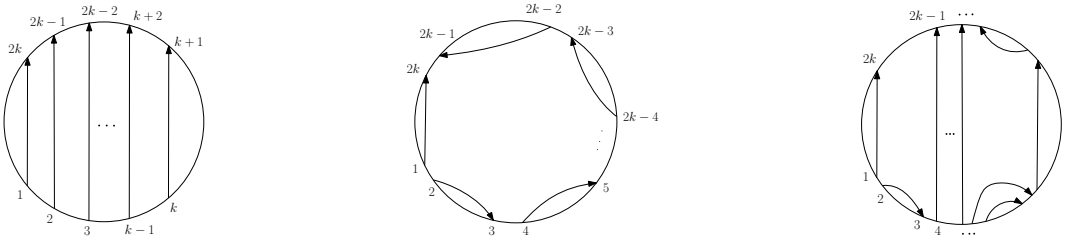
We recall some preliminaries in Section 2 and describe the connection between k-disjoint paths and the determinant in Section 3. In Section 4 we discuss the general One-Face case and in Section 5 the parallel Two-Face case. We extend our results for shortest k-DPP to a variant of shortest k-EDPP in Section 6. We conclude in Section 7 with some open ends.

2 Preliminaries

An *embedding* of a graph $G = (V, E)$ into the plane is a mapping from V to different points of $\mathbb{R}^2$, and from E to internally disjoint simple curves in $\mathbb{R}^2$ such that the endpoints of the image of $(u, v) \in E$ are the images of vertices $u, v \in V$. If such an embedding exists then G is planar. The faces of an embedded planar graph G are the maximal connected components of $\mathbb{R}^2$ that are disjoint from the image of G. We can find a planar embedding in logspace using [2, 12]. In this paper we assume G to be an embedded planar graph. We say that a set of k terminal pairs $\{(s_i, t_i) : i \in [k]\}$ is *One-Face* if the terminals all occur on a single face F. They are in parallel order if the pairs occur in the order $s_1, s_2, \ldots, s_k, t_k, \ldots, t_2, t_1$ on the facial boundary and in serial order if they occur in the order $s_1, t_1, s_2, t_2, \ldots, s_k, t_k$. Otherwise they are said to be in general order. If all the k terminal pairs occur on two faces F_1 and F_2, we call it *Two-Face*. Here they are in parallel order if the sources $s_1, s_2, \ldots, s_k$ occur on one face and all the sinks $t_1, t_2, \ldots, t_k$, are on another. Though conventionally the face containing the terminals is drawn as the outer (infinite) face, for the ease of exposition here we consider it to be *bounded*. The region inside the face (including the face boundary) is a closed set and the graph is embedded on the other side of the face, which is an open set.

Recall that a cycle cover is a collection of directed vertex-disjoint cycles incident on every vertex in the graph. Our proofs go through by reducing the problems to counting/isolating cycle covers. Since the determinant of the adjacency matrix of a graph is the signed sum of its cycle covers, we can count the lightest cycle covers by ensuring that all such cycle covers get the same sign. Similarly, isolating one lightest cycle cover enables us to extract it via determinant computations. We note the following seemingly innocuous but important fact:

▶ **Fact 4** (see e.g. [21]). *The sign of a permutation $\pi \in S_n$ equals $(-1)^{n+c}$ where c is the number of cycles in the cycle decomposition of π.*



(a) Parallel. **(b)** Serial. **(c)** General.

Figure 1 (a) Parallel. (b) Serial. (c) General Terminal Orderings.

3 Disjoint paths, cycle covers and determinant

We first describe a basic graph modification step using which we can show connections between cycle covers and shortest k-DPP. In the rest of the paper, we will first perform the modification before applying our algorithms.

Modification Step. Let G be an undirected graph with $2k$ terminal vertices. We add k new *special* vertices $r_1, \ldots, r_k$ to get a new graph G' and let A be the corresponding adjacency matrix. We add unit weight self loops to all non-special vertices and weigh the rest of the edges of G' by x. The terminals are paired together into k disjoint ordered pairs. We refer to the i^{th} pair as (s_i, t_i), where s_i is the source and t_i is the sink. For each terminal pair i, we add directed edges of unit weight from the sink t_i to r_i and from r_i to the source s_i. By slightly abusing the terminology we refer to these pair of edges (essentially a directed path of length two) together as a *demand edge*. These k demand edges together defines the *input pairing*. In general any set of k demand edges between the terminals (not necessarily directed from the sources to the sinks) that do not share any endpoints defines a *pairing* which essentially gives a bijection between two equal sized partitions of the $2k$ terminals (e.g. in the input pairing each t_i maps to s_i). Let the resulting mixed graph, containing both directed and undirected edges, be H. It can be thought of as a directed graph where each undirected edge corresponds to a pair of directed edges oriented in the opposite directions. Let B be the resultant weighted adjacency matrix corresponding to H and it can be written as $D + xA$ where D is the matrix with 1's for non-special vertices and zeroes for special ones on the diagonal and 1's for the newly added subdivided demand edges as off diagonal entries. There is a bijection between cycle covers in the graph and monomials[4] in the determinant $det(D + xA)$. Each cycle cover in turn consists of disjoint cycles which are one of three types:

1. consisting alternately of paths between two terminals and demand edges.
2. a non-trivial cycle avoiding all terminals.
3. a trivial cycle i.e. a self loop.

Thus every cycle cover contains a set of k disjoint paths. Further any collection of k disjoint paths between the terminals (not necessarily in the specified pairing) can be *extended* using the edges on the uncovered vertices (by the paths) in at least one way to a cycle cover of the above type.

Finally we have extensions of "pure" k-disjoint paths (which are between a designated set of pairs of terminals), which are in bijection with a *subset* of all cycle covers. We call the corresponding set of cycle covers *pure cycle covers*. This bijection carries over to some monomials (the so called *pure monomials*) of the determinant. Thus we obtain the following:

[4] Here we think of the entries of the matrix as formal variables and many such monomials combine to give a term.

▶ **Lemma 5.** *Let $B = D + xA$ as above. The non-zero monomials in $det(B)$ are in bijection with the cycle covers in the graph H and every cycle cover in H is also an extension of a k-disjoint path in G. This bijection also applies to the subset of "pure" k-disjoint paths to yield, so called pure cycle covers and pure monomials. Moreover, the bijection preserves the degree of a monomial as the length of the cycle cover it is mapped to.*

Let's focus on the terms that correspond to minimum length pure cycle covers. Then these terms have the same exponent ℓ, the length of this shortest pure cycle cover. This is also the least exponent amongst all the pure monomials occurring in the determinant. Notice that their sign is the same. To see this, consider the sign given by $(-1)^{n+c}$ (see Fact 4) where n is the number of vertices and c the number of cycles in the cycle cover. The number of non self-loop cycles is k, the minimum number of cycles needed to cover all the vertices without self loops and equalling the number of source sink pairs. Notice that any extra cycles can be replaced by self loops yielding a cycle cover of strictly smaller length hence will not figure in the minimum exponent term. The number of self loops is therefore $n - \ell$. Hence the total number of cycles is $k + n - \ell$ for each of these terms hence the sign is $(-1)^{k-\ell}$ which is independent of the specific shortest cycle cover under consideration.

▶ **Lemma 6.** *The shortest pure cycle covers all have the same sign.*

Notice that ultimately we want to cancel out all monomials which are not pure. In the One-Face case described in Section 4 we show how to do this in the Cancellation Lemma 16. In the Two-Face case, we cannot do this in general but by measuring how paths wind around the faces, we can characterize the cycle covers which we wish to obtain (see Theorem 26).

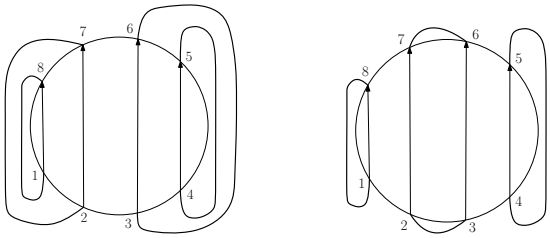
4 Disjoint Paths on One Face: The General Case

In the Appendix B we consider an important special case - when all demands are in parallel and now we proceed to the more general case. We consider an embedding of an undirected planar graph G with all the terminal vertices on a single face in some arbitrary order. The primary idea is, given graph G to construct a sequence of graphs $\mathcal{H}$ so that in the signed sum of the determinants of the graphs in $\mathcal{H}$ the uncancelled minimum weight cycle covers are in bijection with the shortest k-disjoint paths of G.

Notation and Modification. Let $s_1, \ldots, s_k$ and $t_1, \ldots, t_k$ be the source and the sink vertices respectively, incident on a face F in some arbitrary order. Consider the graph G_{M_0} obtained by applying the modification step in Section 3 on G with respect to the input pairing M_0 such that each special vertex r_i is placed inside F and so are the edges $(t_i, r_i), (r_i, s_i)$. Label the terminals in the counter clockwise order by $\{1, 2, \ldots, 2k\}$ and let $\ell(t)$ denote the label of terminal t. A demand edge (u, v) is said to be forward if $\ell(u) < \ell(v)$ and reverse otherwise. For any pairing M if the edges of M are forward we declare the pairing to be in *standard* form.

4.1 Pure Cycle Covers

We define *pure cycle covers* of a graph to be cycle covers in which each non-trivial cycle (cycles that are not self-loops) either avoids all terminals or consists only of a terminal and its mate, where the mate of a terminal is specified in the pairing under consideration. In other words, in a pure cycle cover no two terminal pairs are part of the same cycle. Let the graph obtained by deleting all vertices and edges strictly outside F in G_{M_0} be $\hat{G}_{M_0}$.



(a) Compatible Pairing: $\langle M, M \rangle$.

(b) Incompatible Pairing: $\langle M, M' \rangle$.

Figure 2 Compatible and Incompatible Pairings where $M = \{(1,8),(2,7),(3,6),(4,5)\}(\mathbf{len}(M) = 16)$ and $M' = \{(1,8),(2,3),(4,5),(6,7)\}(\mathbf{len}(M') = 10)$.

Though F does not remain a face in G_{M_0}, it is a cycle nonetheless. If two edges in $\hat{G}_{M_0}$ cross then the paths joining corresponding endpoints outside F in G_{M_0} will also cross. So the terminals cannot interlace (see Definition 8), because otherwise there is no solution. A bit more formally, the following is a consequence of the fact that two cycles in the plane must cross each other an even number of times. Notice that the following condition is necessary but not sufficient.

▶ **Observation 7.** *Unless $\hat{G}_{M_0}$ is outerplanar there is no pure cycle cover in G.*

4.2 Cancelling Bad Cycle Covers

▶ **Definition 8.** Consider two forward demand edges $h_1 = (u_1, v_1)$ and $h_2 = (u_2, v_2)$. We say h_1 and h_2 are in series if either both endpoints of h_1 are smaller than both the endpoints of h_2 or vice-versa. If however, the sources of h_1 and h_2 are smaller than the corresponding sinks then the demands could be in parallel or interlacing with each other as follows.

1. Parallel: either $\ell(u_1) < \ell(u_2) < \ell(v_2) < \ell(v_1)$ or $\ell(u_2) < \ell(u_1) < \ell(v_1) < \ell(v_2)$.
2. Interlacing: either $\ell(u_1) < \ell(u_2) < \ell(v_1) < \ell(v_2)$ or $\ell(u_2) < \ell(u_1) < \ell(v_2) < \ell(v_1)$.

We don't use interlacing demands in the One-face case. The concept is needed in Section 5.

▶ **Definition 9.** An *ordered* pair $\langle M, M' \rangle$ of pairings is compatible if, when we orient M in the standard form then there is a way to orient M' such that the union of the two directed edge sets forms a set of directed cycles. We refer to this set of directed cycles as $M \cup M'$.

See Figure 2 for an example. Let $\langle M, M' \rangle$ form a compatible pair. We call the edges of M as internal edges (drawn inside the face) and those of M' as external edges (drawn outside).

▶ **Lemma 10.** *Compatibility is reflexive and antisymmetric i.e. $\langle M, M \rangle$ is always compatible and for $M \neq M'$ if $\langle M, M' \rangle$ is compatible then $\langle M', M \rangle$ isn't.*

Proof. $\langle M, M \rangle$ is always a compatible pair as for any pairing M inside just put M outside with demand edges directed in the opposite direction. Antisymmetry follows from Lemma 12. ◀

▶ **Definition 11.** Define $\mathbf{len}(u, v) = \ell(v) - \ell(u)$ for every demand edge (u, v). Let $\mathbf{len}(M)$ be the sum of lengths of demand edges of M when the pairing M is placed inside and $\mathbf{len}(\vec{M})$ be the sum of lengths of the demand edges when the pairing comes with directions not necessarily in the standard form.

For external demand edges $\mathbf{len}(u, v)$ may be negative, but for internal edges $\mathbf{len}(u, v)$ is positive since the internal demand edges are always drawn with $\ell(u) < \ell(v)$. Call a standard pairing to be the parallel pairing if for each demand edge (u, v) we have $\ell(u) + \ell(v) = 2k + 1$.

Similarly we have the serial pairing where for each demand edge (u, v) we have $\ell(v) - \ell(u) = 1$. Then notice that $\mathbf{len}(M)$ achieves the maximum value when M is the parallel pairing and achieves the minimum value in the case when M is the serial pairing.

▶ **Lemma 12.** *If $\langle M, M' \rangle$ is a compatible pair and $M \neq M'$ then $\mathbf{len}(M) < \mathbf{len}(M')$.*

Proof. It suffices to prove this for a non-trivial cycle C in $M \cup M'$. Let the edges of the cycle C be partitioned into A, A' according to which one is inside. We have $\mathbf{len}(A) + \mathbf{len}(\vec{A'}) = 0$ where $\vec{A'}$ is A' oriented according to the orientation of M' when placed outside (because each vertex of C occurs with opposite sign in $\mathbf{len}(A)$ and $\mathbf{len}(\vec{A'})$. Notice that to go from $\vec{A'}$ to A' we need to convert the reverse edges to forward edges, which increases the absolute value of $\mathbf{len}(\vec{A'})$). Since in absolute value A and $\vec{A'}$ have the same length, the lemma follows. ◀

A set of disjoint paths R in G between a collection of pairs of terminals which form a pairing M is called a *routing*. We say that R *corresponds* to M in this case i.e. the mapping between the terminals is given by the routing R. For pairings M, M' let $W(\langle M, M' \rangle)$ denote the weighted signed sum of all cycle covers consisting of the pairing M inside the face in forward direction and routing R' that correspond to the pairing M', outside the face. Note that the cycle covers are computed on the mixed graph G_M. It follows immediately that $W(\langle M, M \rangle)$ denotes the weighted sum of all pure cycle covers of M.

▶ **Observation 13.** *$W(\langle M, M' \rangle)$ will be zero unless $\langle M, M' \rangle$ is a compatible pair.*

Also notice that the cycle cover has an arbitrary set of (disjoint) cycles covering vertices not lying on the routing in the sense that we may cover such vertices by non self-loops. Let's abbreviate $W(\langle M, * \rangle) = \sum_{M' : M' \text{ is a pairing}} W(\langle M, M' \rangle)$. From Lemma 12 and Observation 13 we have that:

▶ **Proposition 14.** $W(\langle M, * \rangle) = \sum_{M' : \mathbf{len}(M') > \mathbf{len}(M) \vee M' = M} W(\langle M, M' \rangle)$

▶ **Proposition 15.** *For a compatible pair $\langle M, M' \rangle$, $W(\langle M, M' \rangle) = (-1)^{k - c_{M,M'}} W(\langle M', M' \rangle)$ where $c_{M,M'}$ is the number of cycles passing through at least one demand edge in the union $M \cup M'$ (and k the total number of terminal pairs and equals the number of cycles in $\langle M', M' \rangle$).*

Proof. Notice that the paths belonging to the routing R' are the same in both $\langle M, M' \rangle$ and $\langle M', M' \rangle$. Thereafter it is an immediate consequence of the assumption that the number of cycles in $M \cup M'$ is $c_{M,M'} + k'$ (where k' is the the number of cycles avoiding all terminals in $\langle M, M' \rangle$), in $M' \cup M'$ is $k + k'$ (because the number of cycles avoiding all terminals is the same in both $\langle M, M' \rangle$ and $\langle M', M' \rangle$) and of Fact 4. ◀

Thus by plugging in the values from Proposition 15 in Proposition 14 and rearranging, we get the main result of this section (see example in Subsection 4.4):

▶ **Lemma 16** (Cancellation Lemma)**.** *Let $\mathcal{M}_M$ be the set of pairings M' compatible with M such that $M \neq M'$. Then,*

$$W(\langle M, M \rangle) = W(\langle M, * \rangle) + \sum_{M' : \mathcal{M}_M} (-1)^{k + c_{M,M'} + 1} W(\langle M', M' \rangle).$$

For a given pairing M_0, we are interested in the least order term in $W(\langle M_0, M_0 \rangle)$. From Lemma 5 we know that for any pairing M, there is a bijection between shortest k-DPP of M and the lightest pure cycle covers of M. Moreover, from Lemma 6 we know that all the

lightest pure cycle covers of M occur with the same sign and exponent in the determinant and hence also in the $W(\langle M, M\rangle)$ polynomial. Therefore, the coefficient of the least order term in $W(\langle M_0, M_0\rangle)$ gives us the count of the shortest k-DPP of M_0. We illustrate this with an example in Subsection 4.4. We can now apply Lemma 16 to prove Theorems 1 and 3.

4.3 Proof of The Main Theorems

▶ **Theorem** (Theorem 1 Restated). *Given an undirected planar graph with k pairs of source and sink terminals on the boundary of a common face we can count all shortest k-disjoint paths between the terminals in $O(4^k n^{\omega/2+1})$ time.*

Proof. The Cancellation Lemma 16 allows us to cancel out all cycle covers that are not pure (i.e. those which do not correspond to the input terminal pairing M_0) and replace them by a signed sum of $W(\langle M, *\rangle)$ for various pairings. This process terminates with $W(\langle P, P\rangle)$ where P is the unique parallel pairing. Moreover, the replacement can be done in time linear in the total number of possible terms since each pairing will be considered at most once. Observe that there are at most 4^k different pairings possible (since they correspond to outerplanar matchings, see Observation 7 which are bounded in number by the Catalan number $\frac{1}{k+1}\binom{2k}{<}4^k$ see e.g. [17]). We obtain the count itself by evaluating the polynomial obtained by the signed sum of determinants at $O(n)$ distict points followed by interpolation (see Fact 28). This accounts for a blow-up of $O(n)$ in the running time. We know that the determinant of an $n \times n$ matrix which corresponds to the adjacency matrix of some planar graph, can be computed in time $O(n^{\omega/2})$ [36] where $\omega < 2.373$ is the matrix multiplication constant. ◀

Observe that for the decision version of the shortest k-DPP, it suffices to check whether the polynomial obtained by the signed sum of determinants is non-zero or not.

▶ **Theorem** (Theorem 3 Restated). *Given an undirected planar graph with k pairs of source and sink terminals on the boundary of a common face and k logarithmic in n, we can count all shortest k-disjoint paths between the terminals in* NC.

Proof. Lemma 16 gives us a formula using which one can isolate the pure cycle covers of M by adding to $W(\langle M, *\rangle)$ (obtained by computing the determinant) an appropriately signed sum of pure cycle covers of all pairings $M' \neq M$ such that M' is compatible with M. Observe that Lemma 12 allows us to order all such pairings M' (according to the **len**() metric) in the form of a poset. We can build a matrix C (of size $4^k \times 4^k$) indexed by M, M' and containing zero if $\langle M, M'\rangle$ is not a compatible pair and the sign with which $W(\langle M, M'\rangle)$ occurs in the expression for $W(\langle M, *\rangle)$, otherwise. Since there is a partial order on the pairings (from Lemma 12) this matrix which represents a system of linear equations $Cx = b$ is upper triangular. Here C is the compatibility matrix above and entries of column vector b are $W(\langle M, *\rangle)$. Also along the diagonal we have ± 1's because $W(\langle M, M\rangle)$ always occurs in the expression for $W(\langle M, *\rangle)$. Thus the determinant of C is ± 1 and in particular, C is invertible. We can invert the matrix to get the count in $O(k^2 + \log^2 n)$ parallel time using $4^{O(k)} n^{O(1)}$ processors [10], hence in NC^2 for $k = O(\log n)$. ◀

▶ **Theorem** (Theorem 2 Restated). *Given an undirected planar graph with k pairs of source and sink terminals on the boundary of a common face, finding a shortest set of k-disjoint paths between the terminals is in randomised $O(4^k n^{\omega/2})$ time and in deterministic $O(4^k n^{\omega/2+2})$ time, respectively.*

First we describe a simple deterministic algorithm followed by a randomised algorithm as well as an RNC procedure for search. These together completes the proof of Theorem 2. Using the proof ideas from Theorem 1 and Theorem 3 we can also count the solutions for the Two-face parallel case (see Section 5) in time $O(kn^{\omega+1})$ as well as in NC. Hence the following procedures also work in the Two-face parallel case giving an $O(kn^{\omega+2})$ time deterministic algorithm, an $O(kn^{\omega})$ time randomised algorithm along with an RNC algorithm.

A deterministic search algorithm. Let C_{tot} be the count of total number of shortest k-disjoint paths in G. For every edge $e \in G$ we remove e and count the remaining number of shortest k-disjoint paths using the sequential counting procedure above as oracle. Let $C_{\bar{e}}$ be this count. If $C_{\bar{e}} > 0$, we proceed with the graph $G \setminus e$ since the graph still has a shortest k-disjoint path. If $C_{\bar{e}} = 0$ then every existing shortest k-disjoint paths contains the edge e so keep e in G and proceed with the next edge. Let H be the final graph obtained.

▶ **Claim 17.** *The graph H is a valid shortest k-disjoint path.*

Proof. It is easy to see that all the edges in H are part of a shortest k-disjoint path. Notice that all the edges are part of a single shortest k-disjoint paths since otherwise we could remove that edge, say e^* and will have $C_{\bar{e^*}} > 0$ in H and therefore also in the graph G at the time e^* was under consideration, contradicting that e^* was retained. ◀

Since for each edge we spend $O(4^k n^{\omega/2+1})$ time, the total search time is $O(4^k n^{\omega/2+2})$.

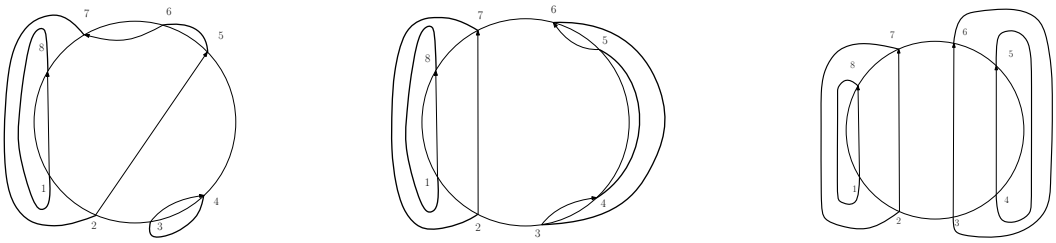
A randomised search algorithm. For the construction of shortest k-DPP we use the following Isolation lemma introduced by Mulmuley, Vazirani, and Vazirani [23]. It is a simple but powerful lemma that crucially uses randomness:

▶ **Lemma 18** (Isolation Lemma). *Given a non-empty $\mathcal{F} \subseteq 2^{[m]}$, if one assigns for each $i \in [m]$, $w_i \in [2m]$ uniformly at random then with probability at least half, the minimum weight subset of in $\mathcal{F}$ is unique; where the weight of a subset S is $\sum_{i \in S} w_i$.*

▶ **Lemma 19.** *A solution to the shortest One-Face k-DPP can be constructed in randomised $O(4^k n^{\omega/2})$ time.*

Proof. First we introduce small random weights in the lower order bits of the edges of the graph G (i.e. give weights like $4n^2 + r_e$ to edge e). Using Lemma 18 these are isolating for the set of k-disjoint paths between the designated vertices, with high probability. In other words the coefficient of least degree monomial equals ± 1 in the isolating case. At the same time the ordering of unequal weight paths is preserved. This is because the sum of the lower order bits cannot interfere with the higher order bits of the monomial which represent the length of the corresponding k-disjoint path.

Let the monomial with minimum exponent be x^w. Our counting algorithms works for the weighted case as explained in the remark in Subsection A.2. Borrowing notation from the previous part we can compute $C_{\bar{e}}$ in parallel for each edge under the small random weights above. If the weight is indeed isolating, we will obtain the least degree monomial in $C_{\bar{e}}$ will be x^w exactly when e does not belong to the isolated shortest k-disjoint paths. Thus with probability at least half we will obtain a set of shortest k-disjoint paths. When the assignment is not isolating the set of edges which lie on some shortest k-disjoint path will not form a k-disjoint path itself so we will know for sure that the random assignment was not isolating. For $k = O(\log n)$ this also gives an RNC algorithm using the NC algorithm for counting from Theorem 3 as subroutine.



(a) (b) (c)

Figure 3 An Example (a) $M_1 \cup M_2$ (b) $M_2 \cup M_3$ (c) $\langle M_3, * \rangle = M_3 \cup M_3$.

We give a randomised sequential algorithm for the problem running in time $O(4^k n^{\omega/2})$ using the idea of inverting a matrix in order to find a witness for perfect matching described in [23]. They use it in the parallel setting but we apply it in the sequential case also. Essentially we need to compute all the $O(n)$ many $C_{\bar{e}}$'s in $O(n^{\omega/2})$ time. Notice that $C - C_{\bar{e}}$ will be the weighted count for the k-disjoint paths that contain the edge e. This is precisely the co-factor of the entry (u, v) where $e = (u, v)$ and since all co-factors can be computed in $O(n^{\omega/2})$ time we are done. ◀

4.4 An Example of the One-Face Case

Let $M_1 = \{(1,8),(2,5),(3,4),(6,7)\}$ be the input pairing. M_1 is compatible with a routing, say R_2, whose corresponding pairing is $M_2 = \{(1,8),(2,7),(3,4),(5,6)\}$. We consider the pairing M_2 then which is compatible with another routing, say R_3 and the corresponding pairing be $M_3 = \{(1,8),(2,7),(3,6),(4,5)\}$. Since M_3 is in parallel configuration, from Lemma 29 the only routing compatible with M_3 corresponds to M_3 itself and the recursion stops. We illustrate this in Figure 3. From the above discussion, we have the following sequence of equations.

$$\begin{aligned}
W(\langle M_1, M_1 \rangle) &= W(\langle M_1, * \rangle) - W(\langle M_1, M_2 \rangle) \\
W(\langle M_1, M_2 \rangle) &= -W(\langle M_2, M_2 \rangle) \\
W(\langle M_2, M_2 \rangle) &= W(\langle M_2, * \rangle) - W(\langle M_2, M_3 \rangle) \\
W(\langle M_2, M_3 \rangle) &= -W(\langle M_3, M_3 \rangle) \\
W(\langle M_3, * \rangle) &= W(\langle M_3, M_3 \rangle)
\end{aligned}$$

After substitutions we get the following formula,

$$W(\langle M_1, M_1 \rangle) = W(\langle M_1, * \rangle) + W(\langle M_2, * \rangle) + W(\langle M_3, * \rangle)$$

5 Disjoint Paths on Two faces: The parallel case

In this section, we solve the shortest k-DPP on planar graphs such that all terminals lie on two faces, say f_1, f_2 in some embedding of the graph and all the demands are directed from one face to another. The key difference between the One-Face case and the Two-Face case is that the compatibility relation in the Two-Face case is not antisymmetric. Consequently, the pairings in the Two-Face case cannot directly be put together as a DAG (see Figure 4) and we are unable to perform an inclusion-exclusion (like in Lemma 16).

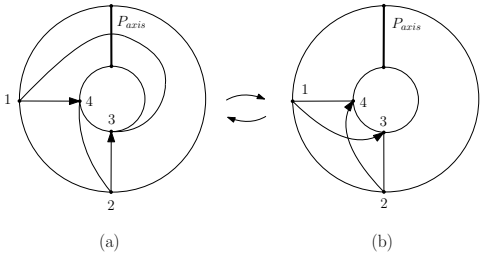


Figure 4 The presence of two faces allows routings of two pairings to be present in the determinant of each other like in this example. P_{axis} is a path between the two faces. (a) shows two parallel demands on two faces and (b) shows a different configuration for the two parallel demands. Notice that one of the two paths necessarily needs to cross the axis in order to obtain (b) from (a), whereas to obtain the pure cycle cover of (a) both paths must cross the axis equal number of times.

Notation and Modification. We connect f_1, f_2 by a path P_{axis} in the (directed)[5] dual graph G^*. We consider the corresponding primal arcs of P_{axis} which are directed from f_1 to f_2 (in the dual) and weigh them by y. Without loss of generality, we can assume that these arcs are counter clockwise as seen from P_{axis}. Similarly, the primal arcs of P_{axis} which are directed from f_2 to f_1(in the dual) are weighed by y^{-1}. According to our convention, these arcs are clockwise as seen from P_{axis}. We number the terminals of the graph in the following manner. Take the face f_2 and start labeling the terminals in a counter-clockwise manner starting from the vertex immediately to the left of P_{axis} as $1, 2, \ldots, k$ and then label the terminals of f_1 again in a counter-clockwise manner starting from the vertex immediately to the right of the dual path as $k+1, \ldots, 2k$. Here the directions "left" and "right" are chosen with respect to P_{axis} in the plane and are used consistently. For any terminal s, $\ell(s)$ describes the label associated with s. We now apply the modification step in Section 3 and direct the demand edges forward. Throughout this section, we fix a pairing M such that each demand edge of M has one terminal on either face. We refer to these types of demand edges as *cross demand* edges and denote them by CD_M. Clearly, $|\mathsf{CD}_M| = k$.

5.1 Pure Cycle Covers

Like in Subsection 4.1 *pure cycle covers* are defined to be cycle covers CC, such that each cycle in CC which contains a terminal also contains the corresponding mate of that terminal and no other terminal. We distribute the terminals of the cross demands(CD_M) evenly on the faces f_1 and f_2 at intervals of $\frac{2\pi}{|\mathsf{CD}_M|}$. For convenience sake, assume that the graph is embedded such that P_{axis} is a radial line. Our proofs go through even if this is not the case simply by accounting for the angle between the endpoints of the axis. The other terminals, vertices and edges of G are embedded such that the graph is planar. We begin with Lemma 20 from [24] which will be useful to analyze the Two-Face k-DPP. In their notation, the two faces having terminals are C_1, C_2 with C_1 inside C_2 in the embedding of G. See Appendix A.3 for details.

▶ **Lemma 20** (Quoted from Section 5 [24]). *We represent the surface on which C_1, C_2 are drawn by $\sigma = \{(r, \theta) : 1 \leq r \leq 2, 0 \leq \theta \leq 2\pi\}$. Let $f : [0, 1] \to \sigma$ be continuous. Then it has finite* winding number *$\theta(f)$ defined intuitively as $\frac{1}{2\pi}$ times the the total angle turned through (measured counterclockwise) by the line OX, where O is the origin, $X = f(x)$, and x ranges*

[5] By directed dual graph we mean the dual graph of G where edges are bi-directed (like in the primal).

from 0 to 1. Let $\mathcal{L}$ be a set of k paths (from C_1 to C_2) drawn on σ, pairwise disjoint. We call such a set $\mathcal{L}$ a linkage. If $\mathcal{L}$ is a linkage then clearly $\theta(P)$ is constant for $P \in \mathcal{L}$, and we denote this common value by $\theta(\mathcal{L})$.

Claim 21, while not being crucial in the analysis, still helps us understand how the demand edges occur in the parallel Two-Face case.

▶ **Claim 21.** *Any three demand edges in CD_M cannot interlace with each other.*

Proof. Assume that the claim does not hold for three demand edges $h_1, h_2, h_3 \in \mathsf{CD}_M$ such that $l(s_1) < l(s_2) < l(s_3)$. Since all three edges interlace, we have that $l(t_1) > l(t_2) > l(t_3)$. If this is the case, we show that M cannot support a pure cycle cover, say CC. Let C_1, C_2, C_3 be the cycles of CC including the demand edges h_1, h_2, h_3 respectively. Since the cycle cover is pure, there exist disjoint paths, say P_1, P_2, P_3, between the endpoints of the three demand edges. Also consider the paths P_4, P_5 which are comprised of the edges of f_1 from s_1 to s_3 via s_2 and t_1 to t_3 without using t_2. Paths P_1, P_3, P_4, P_5 form a cycle in the graph with s_2 inside and t_2 outside it. Therefore, P_2 must intersect either P_1 or P_3 which gives a contradiction. ◀

We say that a cycle cover CC *effectively crosses the axis x* times if the total number of times the paths in CC cross P_{axis} counter-clockwise is x more than the total number of times they cross it in the clockwise direction. We abbreviate this by $\mathsf{AxisCross}_{M,CC}$. We now show that for any pure cycle cover CC the value of $\mathsf{AxisCross}_{M,CC}$ (modulo $|\mathsf{CD}_M|$)must be a constant independent of the cycle cover itself (Lemma 23).

▶ **Observation 22.** *If P is any path (on the plane) in G such that $\theta(P) = 2\pi$ then P effectively crosses the axis exactly once in the counter-clockwise direction. Similarly, when $\theta(P) = -2\pi$ then P effectively crosses the axis exactly once in the clockwise direction.*

Proof (Sketch). We know that θ is a continuous function and its evaluations at the start and end of P are zero and 2π respectively. By the intermediate value theorem, it follows that on some point of P, θ takes on the value θ_0 where θ_0 which is the angle between the start of P and any point on P_{axis}. Since the direction of measurement is counter-clockwise, we conclude that P must cross P_{axis} exactly once in the counter-clockwise direction. The second part of the statement follows analogously with the only difference being that the direction of traversal of P must be clockwise in order to obtain a negative value of $\theta(P)$. ◀

▶ **Lemma 23.** *Assuming $\mathsf{CD}_M \neq \emptyset$, for any pure cycle cover CC, there exists a fixed integer $\mathsf{O_M} \in \{0, 1, \ldots, |\mathsf{CD_M}| - 1\}$ (independent of CC) such that $\mathsf{AxisCross}_{M,CC} = \omega|\mathsf{CD}_M| + \mathsf{O_M}$ where $\omega \in \mathbb{Z}$.*

Proof. We only have to show that the cross demands must contribute to $\mathsf{AxisCross}_{M,CC}$ by an amount of $\omega|\mathsf{CD}_M| + \mathsf{O_M}$. As CC is a pure cycle cover, we know from Lemma 20 that each path between a terminal pair traverses the same angle, say $\theta = 2\pi\omega$ for some integer ω. Since each path traverses the same angle, each source terminal is routed to its corresponding sink terminal which is shifted by an angle of $\theta_0 \in [0, 2\pi)$ and therefore, θ_0 can be written as $2\pi\frac{\mathsf{O_M}}{\mathsf{CD}_M}$ where $\mathsf{O_M} \in \{0, 1, \ldots, |\mathsf{CD_M} - 1|\}$ is the common offset. Observe that the offset is dependent only on the pairing M and is not related to the cycle cover. Summing this angle for all demand edges in CD_M, the total angle traversed by the corresponding paths in CC is simply $\theta|\mathsf{CD}_M| = 2\pi\omega|\mathsf{CD}_M| + 2\pi\mathsf{O_M}$. From Observation 22 every time an angle of 2π is covered, we effectively cross the axis exactly once. Thus the value of $\mathsf{AxisCross}$ due to the demands in CD_M is $\omega|\mathsf{CD}_M| + \mathsf{O_M}$. ◀

5.2 Pruning Bad Cycle Covers

As a consequence of the topology of the One-Face case, the compatibility relation for pairings is antisymmetric and therefore a straightforward inclusion-exclusion is enough to cancel all the "bad" cycle covers. In the Two-Face case, there may exist a set of compatible pairings which yield routings of each other in the determinant, thus making it impossible to cancel bad cycle covers. Therefore, we must make distinction between compatible pairings which yield pure cycle covers and the ones which yield bad cycle covers.

▶ **Definition 24** (Compatibility & **M**-Compatibility)**.** Consider two pairings M, M'. We say that M' is compatible with M if there exists a routing R' yielding a pure cycle cover for M', which when combined with the demand edges of M, forms a cycle cover, denoted by $CC_{R'}$. Moreover, if $CC_{R'}$ satisfies the following property, we say M' is **M**-compatible for M.

$$\mathsf{AxisCross}_{M,CC_{R'}} \equiv \mathsf{O_M} (\text{mod } |\mathsf{CD_M}|) \qquad \text{(Modular Property)}$$

From Lemma 23, it is clear that M is **M**-compatible with itself. We now show that any other $M' \neq M$ is not **M**-compatible with M.

▶ **Lemma 25.** *For any routing R' corresponding to a pairing M' such that $M' \neq M$,*

$$\mathsf{AxisCross}_{M,CC_{R'}} \not\equiv \mathsf{O_M} (\text{mod } |\mathsf{CD_M}|)$$

Proof. Let $\{P_1, P_2, \ldots, P_k\}$ be k disjoint paths in the routing R'. Next, we use Lemma 20 to say that each path in the set must have the same angle as seen from the center of the concentric faces. Since the routing does not lead to a pure cycle cover of M, each source terminal is routed to a sink terminal which is shifted by an angle of $\theta'_0 \in [0, 2\pi)$ and therefore, θ'_0 can be written as $2\pi \frac{\mathsf{O}_{R'}}{\mathsf{CD}_M}$ where $\mathsf{O}_{R'} \in \{0, 1, \ldots, |\mathsf{CD_M} - 1|\} \backslash \{\mathsf{O_M}\}$ is the common offset that each path traverses. Notice that pure cycle covers will have an offset of $\mathsf{O_M} \neq \mathsf{O}_{R'}$ since in the pure case, the offset between the source and sink must be different from that of the offset of $O_{R'}$, otherwise R' would be a pure cycle cover. Therefore,

$$\theta(P_1) = \theta(P_2) = \ldots = \theta(P_k) = 2\omega\pi + \theta'_0 \tag{1}$$

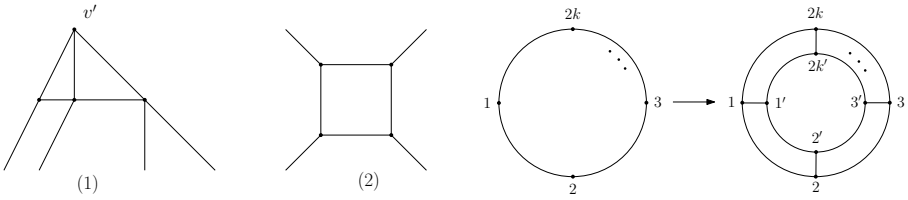
$$\implies \theta(\bigcup_{i=1}^{k} P_i) = 2\pi(\omega.|CD_M| + \mathsf{O}_{R'}) \tag{2}$$

From Observation 22 every time an angle of 2π is covered, we effectively cross the axis exactly once. Thus the value of $\mathsf{AxisCross}$ due to the routing R' is $2\omega|\mathsf{CD}_M| + \mathsf{O}_{R'}$. Since, $\mathsf{O}_{R'} \not\equiv \mathsf{O_M} \bmod |\mathsf{CD_M}|$, we conclude that R' does not satisfy (Modular Property). ◀

▶ **Theorem 26.** *Let M, M' be two Two-Face pairings such that M' is* **M***-compatible for M. Then it must be the case that $M = M'$.*

Theorem 26 is a consequence of Lemma 23 and Lemma 25.

Using the proof ideas of Theorem 1 and Theorem 3 in addition to the following, we can also do counting in the Two-Face parallel case in time $O(kn^{\omega+1})$ as well as in NC. Unlike the One-Face case, here the graph might not remain planar after the modification step and the determinant computation takes $O(n^{\omega})$ time [1]. Also, here we have a bivariate polynomial and we need to discard the terms in the determinant polynomial whose exponent in y is not equivalent to $\mathsf{O_M}$ modulo k. In order to do this, we can evaluate the polynomial at each one of the k^{th} roots of unity and sum each of the k polynomials obtained by the k evaluations. We describe this in Appendix A.4 in more detail. After discarding the unwanted terms in the determinant polynomial we extract the monomial with the smallest exponent in x to obtain the shortest pure cycle covers.



(a) (1): Arbitrary degree to degree ≤ 4, (2): degree 4 to degree 3.

(b) Terminal degree Reduction.

Figure 5 Degree Reduction Gadgets.

6 Edge disjoint paths

We define *planar* k-EDPP to be the problem of finding k edge disjoint paths in a planar graph G between terminal pairs when, the demand edges can be *embedded* in G such that planarity is preserved. We show how to transfer results for k vertex disjoint paths to k edge disjoint paths in undirected graphs using gadgets in Figure 5 borrowed from [22].

▶ **Lemma 27.** *Decision, Search for One-Face planar k-EDPP reduces to One-Face k-DPP.*

Proof (Sketch). The reduction is performed in three steps. First we reduce the degrees of terminals by using the gadget in Figure 5(b) to at most three. Next, we use the gadget in Figure 5(a)(1) to reduce the degree of any vertex which is not a terminal to at most four. After each application of this gadget the degree of the vertex reduces by one. A parallel implementation of this procedure would first expand every vertex into an, at most ternary tree and then replace each node by the gadget. We then reduce the degrees to at most three by using the gadget in Figure 5(a)(2). Notice that since the demand edges can be embedded in a planar manner on the designated face, the disjoint paths can only cross each other an even number of times and hence the for every shortest EDPP we will always be able to find a corresponding shortest DPP after using the gadget in Figure 5(a)(2). It must also be noted that path lengths will not be preserved, however, we can give any new edges introduced in the gadgets zero additive weight. This can be achieved by simply not weighing the new edges by the indeterminate x in the graph modification step. Finally, observe that two paths in a graph with maximum degree three are vertex disjoint iff they are edge disjoint. ◀

▶ Remark. Since counts are not preserved in the gadget reduction, we do not have an NC-bound for counting k-EDPP's.

7 Conclusion and Open Ends

We have reduced some planar versions of the shortest k-DPP to computing determinants. This technique has the advantage of being simple and parallelisable while remaining sequentially competitive. Is it possible to solve the Two-Face case with an arbitrary distribution of the demand edges while obtaining similar complexity bounds? The more general question of extending our result to the case when the terminals are on some fixed f many faces also remains open. For the One-Face case, can we make the dependence on k from exponential to polynomial or even quasipolynomial? Also, what about extending our result to planar graphs or even $K_{3,3}$-free or K_5 free graphs or to graphs on surfaces. Can one de-randomize our algorithm to get deterministic NC bound for the construction? It will be interesting if one can show lower bounds or hardness results for these problems.

References

1. Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
2. Eric Allender and Meena Mahajan. The complexity of planarity testing. *Inf. Comput.*, 189(1):117–134, 2004.
3. Andreas Björklund and Thore Husfeldt. Shortest Two Disjoint Paths in Polynomial Time. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP, Proceedings, Part I*, pages 211–222, 2014.
4. Andreas Björklund and Thore Husfeldt. Counting Shortest Two Disjoint Paths in Cubic Planar Graphs with an NC Algorithm. In *29th International Symposium on Algorithms and Computation, ISAAC*, 2018.
5. Glencora Borradaile, Amir Nayyeri, and Farzad Zafarani. Towards Single Face Shortest Vertex-Disjoint Paths in Undirected Planar Graphs. In *23rd Annual European Symposium on Algorithms, ESA*, pages 227–238, 2015.
6. Jin-yi Cai and D. Sivakumar. Resolution of Hartmanis' conjecture for NL-hard sparse sets. *Theor. Comput. Sci.*, 240(2):257–269, 2000.
7. Julia Chuzhoy, David H. K. Kim, and Shi Li. Improved Approximation for Node-disjoint Paths in Planar Graphs. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing*, STOC 2016, pages 556–569. ACM, 2016.
8. Julia Chuzhoy, David H. K. Kim, and Rachit Nimavat. New Hardness Results for Routing on Disjoint Paths. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 86–99. ACM, 2017.
9. Julia Chuzhoy, David H. K. Kim, and Rachit Nimavat. Improved Approximation for Node-Disjoint Paths in Grids with Sources on the Boundary. In *45th International Colloquium on Automata, Languages, and Programming, ICALP*, pages 38:1–38:14, 2018.
10. L. Csanky. Fast Parallel Matrix Inversion Algorithms. In *Proceedings of the 16th Annual Symposium on Foundations of Computer Science*, SFCS '75, pages 11–12, 1975.
11. C. Damm. DET=$L^{(\#L)}$. Technical Report Informatik–Preprint 8, Fachbereich Informatik der Humboldt–Universität zu Berlin, 1991.
12. Samir Datta and Gautam Prakriya. Planarity Testing Revisited. In *Theory and Applications of Models of Computation - 8th Annual Conference, TAMC 2011, Tokyo, Japan, May 23-25, 2011. Proceedings*, pages 540–551, 2011.
13. Éric Colin de Verdière and Alexander Schrijver. Shortest vertex-disjoint two-face paths in planar graphs. *ACM Transactions on Algorithms*, 7(2):19, 2011.
14. Tali Eilam-Tzoreff. The disjoint shortest paths problem. *Discrete Applied Mathematics*, 85(2):113–138, 1998.
15. Jeff Erickson and Yipu Wang. Four Shortest Vertex-disjoint Paths in Planar Graphs. *preprint on webpage*, October 2017. (retrieved April, 2018). URL: `http://jeffe.cs.illinois.edu/pubs/pdf/disjoint.pdf`.
16. Steven Fortune, John E. Hopcroft, and James Wyllie. The Directed Subgraph Homeomorphism Problem. *Theor. Comput. Sci.*, pages 111–121, 1980. `doi:10.1016/0304-3975(80)90009-2`.
17. C. Hernando, F. Hurtado, and Marc Noy. Graphs of Non-Crossing Perfect Matchings. *Graphs and Combinatorics*, 18(3):517–532, 2002.
18. Richard M. Karp. On the computational complexity of combinatorial problems. *Networks*, 5:45–68, 1975.
19. Yusuke Kobayashi and Christian Sommer. On Shortest Disjoint Paths in Planar Graphs. *Discret. Optim.*, 7(4):234–245, November 2010.
20. James F. Lynch. The Equivalence of Theorem Proving and the Interconnection Problem. *SIGDA Newsl.*, 5(3):31–36, September 1975.

21 Meena Mahajan and V. Vinay. Determinant: Combinatorics, Algorithms, and Complexity. *Chicago J. Theor. Comput. Sci.*, 1997, 1997. URL: `http://cjtcs.cs.uchicago.edu/articles/1997/5/contents.html`.

22 Matthias Middendorf and Frank Pfeiffer. On the complexity of the disjoint paths problems. *Combinatorica*, 13(1):97–107, 1993.

23 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

24 Neil Robertson and Paul D. Seymour. Graph minors. VI. Disjoint paths across a disc. *J. Comb. Theory, Ser. B*, 41(1):115–138, 1986.

25 Neil Robertson and Paul D. Seymour. Graph minors. VII. Disjoint paths on a surface. *J. Comb. Theory, Ser. B*, 45(2):212–254, 1988.

26 Neil Robertson and Paul D. Seymour. Graph minors. XIII. The Disjoint Paths Problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.

27 Petra Scheffler. A practical linear time algorithm for disjoint paths in graphs with bounded tree-width. *Technical Report 396*, 1994.

28 Alexander Schrijver. Finding k Disjoint Paths in a Directed Planar Graph. *SIAM J. Comput.*, 23(4):780–788, 1994.

29 Werner Schwärzler. On the complexity of the planar edge-disjoint paths problem with terminals on the outer boundary. *Combinatorica*, 29(1):121–126, 2009.

30 Hitoshi Suzuki, Takehiro Akama, and Takao Nishizeki. Finding Steiner Forests in Planar Graphs. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '90, pages 444–453. SIAM, 1990.

31 Hitoshi Suzuki, Chiseko Yamanaka, and Takao Nishizeki. Parallel Algorithms for Finding Steiner Forests in Planar Graphs. In *Proceedings of the International Symposium on Algorithms*, SIGAL '90, pages 458–467. Springer-Verlag, 1990.

32 S. Toda. Counting Problems Computationally Equivalent to the Determinant. Technical Report CSIM 91-07, Dept of Comp Sc & Information Mathematics, Univ of Electro-Communications, Chofu-shi, Tokyo, 1991.

33 I. Tzameret. *Studies in Algebraic and Propositional Proof Complexity*. PhD thesis, Tel Aviv University, 2008.

34 H. van der Holst and J.C. de Pina. Length-bounded disjoint paths in planar graphs. *Discrete Applied Mathematics*, 120(1):251–261, 2002.

35 V. Vinay. Counting Auxiliary Pushdown Automata. In *Proceedings of the Sixth Annual Structure in Complexity Theory Conference*, pages 270–284, 1991.

36 Raphael Yuster. Matrix Sparsification for Rank and Determinant Computations via Nested Dissection. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 137–145, 2008.

A Appendix

A.1 Polynomial Interpolation

▶ **Fact 28** (Folklore [6, 33]). *Polynomial interpolation i.e. obtaining the coefficients of a univariate polynomial given its value at sufficiently many (i.e. degree plus one) points is in* $\mathsf{TC}^0 \subseteq \mathsf{NC}^1$. *It is also in* $O(n \log n)$ *time (where* n *is the degree of the polynomial) via Fast Fourier Transform.*

A.2 Weighted Graphs

▶ Remark. Our algorithms also work for weighted graphs where each edge e is assigned a weight $w(e)$ which is polynomially bounded in n. This can be done by putting odd (additive) weights $w'(e) = (|E| + 1)w(e) + 1$ on the edges i.e. replacing the entry corresponding to e

in the adjacency matrix by $x^{w'(e)}$ instead of just x. Notice that the length of a collection of edges has the same parity as the sum of its weights. So the calculation in Lemma 6 go through with small changes. This implies that we do not have to convert a weighted graph into unweighted one in order to run the counting algorithms and we get the sum of the weights of edges instead of counts as a result.

A.3 Proof of Lemma 20

Proof. Recall, the surface on which C_1, C_2 are drawn is given by

$$\sigma = \{(r, \theta) : 1 \leq r \leq 2, 0 \leq \theta \leq 2\pi\}$$

We quote from [24]. If P is a path drawn on σ with one end in C_1 and the other in C_2, let $f : [0, 1] \to \sigma$ be a continuous injection with image P and with $f(0) \in C_1$, $f(1) \in C_2$; then we define $\theta(P) = \theta(f)$. It is easy to see that this definition is independent of the choice of f. If P_1, P_2 are both paths drawn on σ from some $s \in C_1$, to some $t \in C_2$, then $\theta(P_1) - \theta(P_2)$ is an integer, and is zero if and only if P_1 is homotopic to P_2. Let $k > 0$ be some fixed integer, and let

$$M_i = \{(i, \frac{2j}{k}\pi) : 1 \leq j \leq k\}(i = 1, 2).$$

If $\mathcal{L}$ is a linkage then clearly $\theta(P)$ is constant for $P \in \mathcal{L}$, and we denote this common value by $\theta(\mathcal{L})$. Intuitively, this is because if any two simple paths wind around a face a different number of times then they both must intersect. ◀

A.4 Computing the univariate polynomial in the Two-Face case

In this section we show that we can extract the desired coefficients of the bivariate determinant polynomial (and thus the count) in GapL.

We firstly describe a procedure using which we can get rid of all the terms whose exponent in y is not equivalent $\mathsf{O_M}$ modulo k. For simplicity, let $\mathsf{O_M} = 0$. We first compute the determinant which is a bivariate polynomial in this case. Since we are looking for exponents of y to be modulo k, we evaluate this polynomial in y at all the k^{th} roots of unity. Upon taking their sum, all the monomials whose exponents are not equal to 0 (mod k) cancel out. We can divide the resulting polynomial by k to preserve the coefficients. If $\mathsf{O_M} \neq 0$, we can simply multiply the determinants by $y^{-\mathsf{O_M}}$ while performing the procedure described above. Note than in order to do this, we must shift to a model of computation which allows us to approximately evaluate polynomials at imaginary points. Since our determinant polynomial now does not have terms corresponding to unwanted cycle covers, we can evaluate it at n points and then interpolate like in the One-Face Case (Fact 28). This gives us the same complexity as in the One-face case, with an additional blow-up of k and can also be done in GapL modifying the algorithm in [21].

Another way of seeing that the computation is in GapLis as follows. The determinant of an integer matrix is complete for the class GapL [11, 32, 35] and Mahajan-Vinay [21] give a particularly elegant proof of this result by writing the determinant of an $n \times n$ matrix as the difference of two entries of a product of $n + 1$ matrices of size $2n^2 \times 2n^2$. By a simple modification of their proof we can obtain each coefficient of the determinant - which is a univariate polynomial (in fact for polynomials with constantly many variables) - in GapL. One way to do so is to evaluate the polynomial at several points and then interpolate.

Alternatively, we can also modify the division-free algorithm for determinant computation described in [21] as follows. We briefly review the algorithm described by Mahajan and Vinay [21] to compute the determinant. Instead of writing down the determinant as a sum of cycle

covers, they write it as a sum of clow sequences. A clow sequence which generalises from a cycle cover allows walks that may visit vertices many times as opposed to cycles where each vertex is visited exactly once (for more details see [21]). Even though the determinant is now written as a sum over more terms, they show an involution where any clow sequence which is not a cycle cover cancels out with a unique "mate" clow sequence which occurs with the opposite sign. In order to implement this determinant computation as an algorithm, each clow which can be realised as a closed walk in the graph is computed in a non-deterministic manner.

Our only modification to the algorithm is as follows: in each non-deterministic path, we maintain a $O(\log k)$-bit counter which counts the number of times edges from P_{axis} have been used in the clow sequence so far modulo k. In other words, every time the counts exceeds k, we shift the counter to 0. At the end of the computation, the number in this counter is exactly the exponent of y modulo k. It is easy to see that clow sequences which are not cycle covers, still cancel out because, in a clow sequence and its mate the set of directed edges traversed is the same. Consequently, at the end of the computation of each clow sequence, a clow and its made get the same exponent in y modulo k. This can be done in GapL as described in [21].

B Disjoint Paths on One-face: The parallel case

In this section, we consider directed planar graphs where all the terminal vertices lie on a single face in the parallel order. Here we exhibit a weight preserving bijection between the set of k-disjoint paths in the given graph and the set of cycle covers with exactly k cycles in a *modified graph* G'. This enables us to count all the shortest k-DPP solutions. Unlike the general case, here the bijection works even when the input graph is directed and we are also able to give efficient sequential and parallel algorithms when k is part of the input. We first modify the given graph as follows:

Notation and Modification. Let $G = (V, E)$ be the given directed planar graph with n vertices and m edges. Let $s_1, \ldots, s_k$ and $t_k, \ldots, t_1$ be the source and sink vertices respectively, all occurring on a face F in the order specified above. We apply the modification step described in Section 3. Let the modified graph be G' with n' vertices and m' edges where $n' = n + k$ and $m' = m + 2k$. G' remains planar. Let A' be the adjacency matrix of G'.

Recall that a cycle cover is a collection of directed vertex-disjoint cycles covering every vertex in the graph. A k-cycle cover is a cycle cover containing exactly k non-trivial cycles (i.e. cycles that are not self-loops). We show the following bijection:

▶ **Lemma 29** (Parallel Bijection). *There is a weight-preserving bijection between shortest k-disjoint paths and lightest k-cycle covers in the modified graph G'.*

Proof. Suppose the graph G contains a set of k disjoint paths. Consider a shortest set of k-disjoint paths of total length ℓ. There are k disjoint cycles in G' corresponding to the shortest k disjoint paths in G, using the new paths from t_i to s_i through r_i, inside the face, for each $i \in [k]$. The $n - \ell - k$ vertices which are not covered by these k cycles will use the self loops on them, yielding a k-cycle cover of G'. All these cycle covers have the same weight ℓ. For the other direction, consider a k-cycle cover in G'. If each non-trivial cycle includes exactly one pair s_i, t_i of terminals then we are done.

Suppose not, then there is a cycle in the cycle cover which contains s_i and t_j for some $1 \leq i \neq j \leq k$. We further assume, without loss of generality, that there are no terminals other than possibly s_j, t_i between s_i, t_j in the direction of traversal of this cycle, called, say,

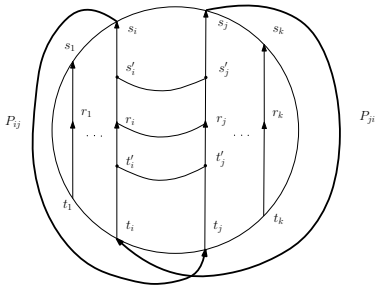


Figure 6 Parallel Configuration. The bipartite subgraph $\{s'_i, r_j, t'_i\} \cup \{s'_j, r_i, t'_j\}$ gives a $K_{3,3,}$.

C. Then C must go through the vertices r_j and s_j since the only incoming edge incident on r_j starts at t_j and the only outgoing edge leads to s_j. By the same logic t_i and r_i are on the cycle C. Also notice that the vertices t_i, r_i, s_i must occur consecutively in that order and so must t_j, r_j, s_j. Let the C be $t_i, r_i, s_i, P_{ij}, t_j, r_j, s_j, P_{ji}, t_i$ where P_{ij}, P_{ji} are paths. Let the face F be $s_i, F_{ij}, s_j, F_j, t_j, F_{ji}, t_i, F_i, s_i$ where F_{ij}, F_{ji}, F_i, F_j are paths made of vertices and edges from F. Since C is simple P_{ji} cannot intersect P_{ij}.

Thus the region inside F bounded by $t_i, r_i, s_i, F_{ij}, s_j, r_j, t_j, F_{ji}, t_i$ does not contain any vertex or edge from C. Thus we can subdivide $(t_i, r_i), (r_i, s_i), (t_j, r_j), (r_j, s_j)$ to introduce vertices t'_i, s'_i, t'_j, s'_j respectively and also the edges $(t'_i, t'_j), (r_i, r_j), (s'_i, s'_j)$ without affecting the planarity of $C \cup F$. But now observe that the complete bipartite graph with $\{s'_i, r_j, t'_i\}$ and $\{s'_j, r_i, t'_j\}$ as the two sets of branch vertices forms a minor of $C \cup F$ augmented with the above vertices and edges. This contradicts the planarity of G'.

As the newly added edges (including the self loops) have weight 1, the bijection is also weight preserving. ◀

▶ Remark. We also get an alternative shorter proof of the Parallel Bijection Lemma 29 from Lemma 12 in Section 4 by observing that the parallel pairing is the unique pairing with maximum length thus has no compatible pairing other than itself.

Hyper Partial Order Logic

Béatrice Bérard
Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6,
LIP6, F-75005 Paris, France
Beatrice.Berard@lip6.fr

Stefan Haar
INRIA and LSV, ENS Paris-Saclay and CNRS, Université Paris-Saclay, France
stefan.haar@inria.fr

Loic Hélouët
Université de Rennes, Inria, CNRS, IRISA, France
loic.helouet@inria.fr

Abstract

We define HyPOL, a local hyper logic for partial order models, expressing properties of sets of runs. These properties depict shapes of causal dependencies in sets of partially ordered executions, with similarity relations defined as isomorphisms of past observations. Unsurprisingly, since comparison of projections are included, satisfiability of this logic is undecidable. We then address model checking of HyPOL and show that, already for safe Petri nets, the problem is undecidable. Fortunately, sensible restrictions of observations and nets allow us to bring back model checking of HyPOL to a decidable problem, namely model checking of MSO on graphs of bounded treewidth.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Partial orders, logic, hyper-logic

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.20

Related Version An extended version of the paper is available at [5], `https://hal.inria.fr/hal-01884390`.

1 Introduction

Hyperproperties. A way to address information security in systems is to guarantee various information flow properties. Examples of such properties are non-interference [18] (an attacker of a system cannot obtain confidential information from its observation of the system), or opacity of secrets [2] (an attacker cannot decide whether the system is in some particular secret configuration). For a long time since the seminal work of [18] introducing non-interference, security properties have been characterized as equivalences between partially observed behaviors of systems. This idea was later formalized [23] as combinations of language closure properties, the so-called "basic security predicates". We refer to [29] for a survey on language based information flow properties. More recently, logics with path equivalences [1] encompassing indistinguishability among partially observed executions have been proposed as a generic framework to define security conditions. Security properties are now frequently called hyperproperties [11, 10], i.e., properties of sets of runs.

Most proposals address verification questions in an interleaved setting, ignoring concurrency aspects. For instance, non-interference properties were considered for Petri nets [8], but still with techniques relying on interleaved interpretation of behaviors. Recently, [7] showed how to characterize some non-interference properties that cannot be handled in an

interleaved model. This result is interesting, as it shows that even if complexity gains are not straightforward, considering causal dependences in systems leads to characterize types of attacks of a system that cannot be characterized in an interleaved setting.

Local logics. We focus here on *local logics* that account for causal dependencies and concurrency in behaviors of models. Several variants of local logic have been proposed: TLC^-, LD_0, PDL, $LPOC$, or even MSO. The first one, proposed by [27], is a logic tailored for Message Sequence Charts (MSCs). The logic features propositions, a next and an until operator and is interpreted over causal paths of MSCs. Model checking TLC^- is decidable for families of partial orders generated by High-level Message Sequence Charts (HMSCs). It is linear in the size of the considered HMSC, but exponential in the size of the formula.

The logic LD_0 [26] addresses properties of causal paths in partial orders. It resembles LTL in that its atomic propositions are attached to events, but it follows causal paths rather than linearizations, and is equipped with successor/predecessor relations.

An extension of TLC^- called Propositional Dynamic logic (PDL), which also subsumes LD_0, is given in [9] to express properties of Communicating Finite State Machines (CFSMs). This logic is divided into path formulas and local formulas. Path formulas make it possible to navigate forward or backward in partially ordered executions via two relations: One that indicates whether an event f is the next executed event after e on the same process, and one that indicates whether a pair (e, f) forms a message. At each event along a followed path, truth of a local formula can be checked. Local formulas are used to check whether some atomic proposition holds at a given event, or whether some path formula holds at an event together with another PDL subformula. In general, verification of PDL for CFSMs is undecidable, but checking whether some B-bounded execution of a CFSM (in which buffer contents can remain of size smaller than B) satisfies a PDL formula is PSPACE-complete. This result extends to HMSC specifications, whose executions are naturally bounded. Another approach to study properties of partial orders generated by system executions is to express them directly as MSO properties. As MSO verification can easily be undecidable for some families of graphs, decidability is proved for families of partial orders generated by Message Sequence Charts in [21]. The result is obtained thanks to the particular shape of orders generated by MSCs that are "layered". Similarly, [22] considers restrictions in executions of CFSMs that have to synchronize frequently.

LPOC [17] is a logic for partially ordered computations. It describes the shape of partial orders, and not only of their causal paths. In addition to standard local operators, the logic has the ability to require existence of a particular partial order pattern in the causal past of an event. It was used as a specification formalism for diagnosis purposes, but without restriction, satisfiability of an LPOC formula is undecidable.

Contributions. We propose a framework unifying path equivalence logics, hyperproperties and partial order approaches. The logic borrows ingredients from LPOC [17]: in particular, it expresses existence of a *pattern* in a partial order, rather than on a causal path. It also borrows the idea of comparing executions up to observation, as proposed in $CTL_{\equiv}$, one of the branching logics with path equivalence proposed in [1]. Events in a pair of executions are considered as equivalent if the (partial) observations of their causal pasts are isomorphic. One of the artifacts used by [1] to obtain decidability of $CTL_{\equiv}$ is to require equivalence to hold only among events located at the *same depth* in executions. We do not use such an interpretation of equivalence, and rather exhibit sufficient conditions on behaviors of systems that are almost a layeredness property [21], to obtain decidability.

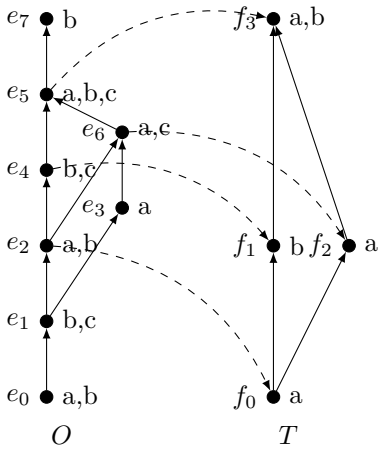


Figure 1 A partial order O over events $\{e_0, e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$, a template T with events $\{f_0, f_1, f_2, f_3\}$, and a mapping (dashed arrows) witnessing that O matches T.

We first define a partial order logic called Hyper Partial Order Logic (HyPOL for short). While we show undecidability for the satisfiability of this logic, we address model checking on a true concurrency model, and start with Labeled Safe Petri Nets (LSPNs). The universe of all behaviors of an LSPN can be defined as the set of processes of its complete unfolding [25]. Unsurprisingly, model checking HyPOL on runs of LSPNs is again undecidable. We then consider sensible assumptions on nets and projections saying that behaviors of a Petri net cannot remain unobserved for an arbitrary long time, and that equivalences necessarily link events whose common past is "not too old". We consider the unfolding of an LSPN as a graph connecting events and conditions via a successor relation. Isomorphism of causal pasts of events can be encoded as an additional relation on this unfolding graph. With these restrictions on nets and observations, model checking HyPOL can be brought back to verification of MSO on a graph generated by an hyperedge replacement grammar [19]. As MSO is decidable for such graphs [13], this yields decidability of HyPOL model checking for this subclass of nets and observations.

Outline. We introduce basic notations in Section 2. In Section 3, we define the logic HyPOL and prove undecidability of satisfiability. In Section 4, we show undecidability of HyPOL model checking on sets of processes of safe Petri nets, while decidability is proved in Section 5 for a subclass. Due to lack of space, proofs are omitted or only sketched, but can be found in an extended version available at [5].

2 Preliminaries

▶ **Definition 1.** A *labeled partial order (LPO)* over alphabet Σ is a triple $O = (E, \leq, \lambda)$ where E is a set of events, $\leq \subseteq E \times E$ is a partial ordering, i.e., a reflexive, transitive, antisymmetric relation, and $\lambda : E \to 2^{\Sigma}$ is a function associating with each event a set of labels from Σ.

We denote by $\mathcal{LPO}(\Sigma)$ the set of labeled partial orders over Σ. For $O = (E, \leq, \lambda)$, we denote by $\max(O) = \{e \in E \mid \nexists f \neq e, e \leq f\}$ the set of its *maximal events*, and by $\min(O) = \{e \in E \mid \nexists f \neq e, f \leq e\}$ the set of its *minimal events*. The *covering* relation of O is a relation $< \subseteq E \times E$ such that $e < f$ iff $e \leq f$, $e \neq f$ and $\forall e' : (e \leq e' \leq f) \Rightarrow (e' \in \{e, f\})$. A *causal path* of O is a sequence of events $e_1.e_2 \dots e_n$ such that $e_i < e_{i+1}$. If $e \in E$, the *ideal*

of e is the set $\downarrow e = \{f \mid f \leq e\}$ and its *ending section* is the set $\uparrow e = \{f \mid e \leq f\}$. The arrows and relations may be indexed by the order in case of ambiguity. A set $H \subseteq E$ of events is *downward closed* iff $H = \bigcup_{e \in H} \downarrow e$, and *upward closed* iff $H = \bigcup_{e \in H} \uparrow e$.

▶ **Definition 2.** The *restriction* of $O = (E, \leq, \lambda)$ to a subset $H \subseteq E$ is the LPO $O_{|H} = (H, \leq_{|H}, \lambda_{|H})$ where $\leq_{|H} = \leq \cap (H \times H)$ and $\lambda_{|H}$ is the restriction of λ to H. The *projection* of O on a subset of labels $\Sigma' \subseteq \Sigma$ is the restriction of O to events that carry labels in Σ'.

▶ **Definition 3.** Two partial orders $O = (E, \leq, \lambda)$ and $O' = (E', \leq', \lambda')$ over Σ are *isomorphic* (written $O \equiv O'$) iff there exists a bijective function $h : E \to E'$ such that $e \leq e' \iff h(e) \leq' h(e')$ and $\lambda(e) = \lambda'(h(e))$.

Note that two *discrete* LPOs O and O' are isomorphic iff their coverings are isomorphic.

▶ **Definition 4.** Let $O = (E, \leq, \lambda)$ and $T = (E_T, \leq_T, \lambda_T)$ be partial orders over Σ. Then O *matches* T iff there exists $H \subseteq E$ and a bijective mapping $h : H \to E_T$ such that $\lambda_T(h(e)) \subseteq \lambda(e)$, and $e <_T e'$ implies $h^{-1}(e) < h^{-1}(e')$. The partial order T is called a *template* and we say that h is *witnessing* the matching.

In the sequel, we constrain the mapping witnessing a matching, using the notion of *anchored matching*. We say that there exists an anchored matching of template T at event e in O and f in T iff O matches T, and there exists a mapping $h_{e,f}$ witnessing this matching such that $h_{e,f}(e) = f$. In the example shown in Figure 1, the order O matches template T: the mapping h (depicted by dashed arrows) is defined by $h(e_2) = f_0$, $h(e_4) = f_1$, $h(e_6) = f_2$, $h(e_5) = f_3$. It satisfies: $\lambda_T(f_0) \subseteq \lambda(e_2)$, $\lambda_T(f_1) \subseteq \lambda(e_4)$, $\lambda_T(f_2) \subseteq \lambda(e_6)$, $\lambda_T(f_3) \subseteq \lambda(e_5)$.

An *observation function* is a mapping $\mathcal{O} : \mathcal{LPO}(\Sigma) \to \mathcal{LPO}(\Sigma')$, representing the visible part of the system. One can notice that an observation maps an LPO on an alphabet Σ to another LPO on another alphabet Σ'. To illustrate this notion, consider the following example: A system is composed of 4 sites, A, B, C, D, that communicate asynchronously. An agent X logs communication events that have occurred on sites A and B, their ordering, but cannot distinguish between messages that are sent to sites C and D. Executions in this system can be represented by labeled partial orders. Events are labeled by the identity of the site $s \in \{A, B, C, D\}$ on which they occurred. They also carry indication on messages sent and received: a message sending from a site s to s' carries label $s!s'$, and a reception on s of a message sent by s' carries label $s?s'$. Executions of this system are hence LPOs over $\Sigma = \{A, B, C, B\} \cup \{s!s' \mid s, s' \in \{A, B, C, D\}\} \cup \{s?s' \mid s, s' \in \{A, B, C, D\}\}$. Let γ denote a new label attached to message sendings to C or D. The information that X can obtain from an execution $O = (E, \leq, \lambda)$ of the system can be modeled as an observation $\mathcal{O}_{A,B}$ such that $\mathcal{O}_{A,B}(O) = (F, \leq', \lambda')$, with $F = \{f \in E \mid A \in \lambda(e) \vee B \in \lambda(e)\}$, $\leq' = \leq \cap F \times F$ and $\lambda'(f) = (\lambda(f) \cap \{A, B\}) \cup \gamma$ if $\exists s!s' \in \lambda(f)$ with $s' \in \{C, D\}$, and $\lambda(f)$ otherwise. $\mathcal{O}_{A,B}(O)$ is hence an LPO over $\Sigma' = \{A, B\} \cup \{s?s' \mid s \in \{A, B\} \wedge s' \in \{A, B, C, D\}\} \cup \{s!s' \mid s, s' \in A, B\} \cup \{\gamma\}$. An example of LPO O and its observation $\mathcal{O}_{A,B}(O)$ is shown in Figure 2.

In what follows, we focus on observation functions that are the identity function *id* (i.e., the function such that $id(O) = O$), relabellings, and various restrictions of orders, for instance associating with $O = (E, \leq, \lambda)$ the order $O_{|F}$ for some $F \subseteq E$.

With a slight abuse, if $O = (E, \leq, \lambda)$ and $F \subseteq E$, we write $\mathcal{O}(F)$ for the corresponding subset of events of $\mathcal{O}(O)$. With observation functions like those described above, either an event is kept by observation (but it can be relabeled) or deleted. When event $e \in E$ has an image in $\mathcal{O}(E)$, we denote this image by $\mathcal{O}(e)$.

Consider the example of Figure 3. The partial order O contains events labeled by atomic propositions a, b, c. Let observation $\mathcal{O}_1$ be the projection of orders on events carrying a proposition in $\{a, b\}$. Such a projection can be used to indicate which actions are observed by

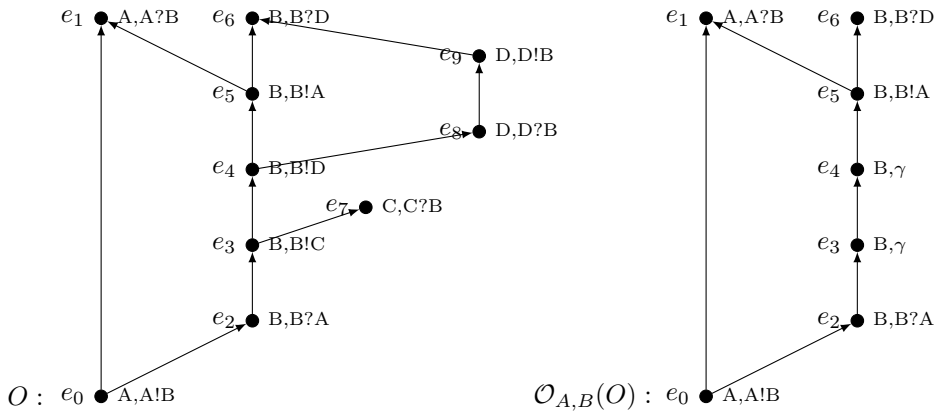


Figure 2 An LPO O and its observation $\mathcal{O}_{A,B}(O)$.

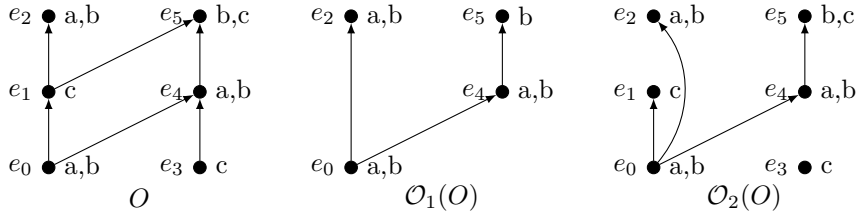


Figure 3 A partial order O, its projection $\mathcal{O}_1(O)$ on events that carry label a or b, and its restriction $\mathcal{O}_2(O)$ to causal dependencies from any event carrying label a to other events.

a particular user. Now, consider observation $\mathcal{O}_2$ that restricts an order to causal dependencies in $\leq \cap \{(e,f) \mid a \in \lambda(e)\}$. This kind of observation can encode the fact that a particular user observing the execution of a system is not able to know if some events are causally related or not. Last, we can combine projections and order restriction: the observation defined by $\mathcal{O}_3(O) = \mathcal{O}_1(\mathcal{O}_2(O))$ describes what would be visible to a user of the system that logs events tagged with propositions a and b, and can only know dependencies from events tagged by a. For the order O in Figure 3, $\mathcal{O}_3(O) = \mathcal{O}_1(O)$.

3 Hyper Partial Order Logic

We are now ready to define HyPOL, a hyperproperty partial order logic. HyPOL is designed to express properties of partially observed sets of executions described by LPOs in $\mathcal{LPO}(\Sigma)$.

3.1 Syntax and semantics

We consider a set A of atomic propositions, a finite set $\mathcal{T}$ of templates labeled over A, and a finite set $\mathcal{O}bs$ of observation functions producing LPOs over A. We assume that $\Sigma \subseteq A$ but, since an event labeling can be modified by observations, it is not always the case that $A = \Sigma$. The syntax of HyPOL is given by:

$$\phi ::= true \mid match(\mathcal{O}, T, f) \mid EX_{D,\mathcal{O}}\, \phi \mid EX_{\equiv,\mathcal{O}}\, \phi \mid \phi_1\, EU_{D,\mathcal{O}}\, \phi_2 \mid EG_{D,\mathcal{O}}\, \phi \mid \neg\, \phi \mid \phi_1 \vee \phi_2$$

where $D \subseteq A$, $T \in \mathcal{T}$, f is an event of T, and $\mathcal{O} \in \mathcal{O}bs$ an observation function.

A formula is *equivalence-free* iff it does not use the $EX_{\equiv,\mathcal{O}}$ operator. To reduce the number of primitives in our logic, we address labeling of events via templates. For $D \subseteq A$, we define a template T_D composed of a single event f_D labeled by all propositions in D.

In particular, when $D = \{a\}$ for some proposition $a \in A$, we write T_a instead of $T_{\{a\}}$ and f_a instead of $f_{\{a\}}$. When template T_a is matched at some event e in an order O under observation $\mathcal{O}$, this means that the image of e by $\mathcal{O}$ carries proposition a.

We define derived operators (with $D \subseteq A$):

$$\begin{array}{lcl} \lambda_{\not\in D} & ::= & \bigwedge_{a\in D} \neg match(id, T_a, f_a) \\ \lambda_{=D} & ::= & match(id, T_D, f_D) \wedge \lambda_{\not\in A\setminus D} \\ EF_{D,\mathcal{O}}\,\phi & ::= & true\; EU_{D,\mathcal{O}}\phi \end{array} \qquad \begin{array}{lcl} AG_{D,\mathcal{O}}\,\phi & ::= & \neg EF_{D,\mathcal{O}}\neg\phi \\ AX_{D,\mathcal{O}} & ::= & \neg EX_{D,\mathcal{O}}\neg\phi \\ AX_{\equiv,\mathcal{O}} & ::= & \neg EX_{\equiv,\mathcal{O}}\neg\phi \end{array}$$

The semantics of HyPOL formulas is defined over a set $\mathcal{W} \subseteq \mathcal{LPO}(\Sigma)$ of orders, for $O = (E, \leq, \lambda) \in \mathcal{W}$ and $e \in E$. Letting $\lambda_{\mathcal{O}}$ be the labeling of $\mathcal{O}(O)$ and $<_{\mathcal{O}}$ its covering, we say that $O \in \mathcal{W}$ *satisfies* ϕ at event e (denoted by $O, e \models \phi$) if formula ϕ is satisfied when starting its evaluation from event e in order O:

- $O, e \models true$ for every event $e \in E$;
- $O, e \models \neg\phi$ iff $O, e \not\models \phi$ and $O, e \models \phi_1 \vee \phi_2$ iff $O, e \models \phi_1$ or $O, e \models \phi_2$;
- $O, e \models match(\mathcal{O}, T, f)$ if and only if f is an event of T, e has image e' in $\mathcal{O}(\downarrow e)$, and $\mathcal{O}(\downarrow e)$ matches T with at least a witness mapping $h_{e',f}$ associating f with e';
- $O, e \models EX_{D,\mathcal{O}}\,\phi$ iff $\exists f \in E$, e has image $e' \in \mathcal{O}(\uparrow e)$, f has image $f' \in \mathcal{O}(\uparrow f)$, $e' <_{\mathcal{O}} f'$, such that $\lambda_{\mathcal{O}}(e') \cap D \neq \emptyset$ and $O, f \models \phi$;
- $O, e \models EX_{\equiv,\mathcal{O}}\phi$ iff there exists $O' \in \mathcal{W}$ and $e' \neq e \in O'$ such that $\mathcal{O}(\downarrow_O e) \equiv \mathcal{O}(\downarrow_{O'} e')$ and $O', e' \models \phi$;
- $O, e \models \phi_1\, EU_{D,\mathcal{O}}\, \phi_2$ iff there exists an event $f \in E$ such that $O, f \models \phi_2$, and a finite set of events $e'_1, e'_2, \ldots e'_k \in \mathcal{O}(O)$ such that
  - $e'_1 <_{\mathcal{O}} e'_2 <_{\mathcal{O}} \cdots <_{\mathcal{O}} e'_k$, $e'_1 = \mathcal{O}(e)$ and $e'_k = \mathcal{O}(f)$,
  - $\forall i \in 2..k-1$, e'_i is the image of some event $e_i \in E$ by $\mathcal{O}$, $\lambda_{\mathcal{O}}(e'_i) \cap D \neq \emptyset$ and $O, e_i \models \phi_1$;
- $O, e \models EG_{D,\mathcal{O}}\phi$ iff
  - either there exists an infinite sequence of events $(e_i)_{i\geq 1}$ in E such that $e = e_1$, every e_i has an image e'_i in $\mathcal{O}(O)$, and $\forall i \geq 1$, $e'_i <_{\mathcal{O}} e'_{i+1}$, $\lambda(e'_i) \cap D \neq \emptyset$ and $O, e_i \models \phi$, or
  - there exists a finite set of events $e_1, \ldots e_k \in E$ such that $e = e_1$, for every $i \in 1..k$, e_i has an image e'_i by $\mathcal{O}$ with $e'_1 <_{\mathcal{O}} e'_2 <_{\mathcal{O}} \cdots <_{\mathcal{O}} e'_k$, $\lambda_{\mathcal{O}}(e'_i) \cap D \neq \emptyset$, $O, e_i \models \phi$, and $e'_k \in \max(\mathcal{O}(O))$.

In particular, $O, e \models match(id, T_a, f_a)$ iff e carries label a in order O, i.e., $a \in \lambda(e)$. Intuitively, formulas of the form $O, e \models EG_{D,\mathcal{O}}\,\phi$, $O, e \models \phi_1\, EU_{D,\mathcal{O}}\, \phi_2$, and $O, e \models EX_{D,\mathcal{O}}\,\phi$ describe properties of causal paths in orders, and have the standard interpretation seen for instance in LTL for words. Observation $\mathcal{O}$ is used to select successive events along a path, and set D performs an additional filtering among possible next events, by requiring the next considered event in a path to carry a label in D. The definition $O, e \models EX_{\equiv,\mathcal{O}}\,\phi$ requires existence of another order $O' \in \mathcal{W}$ and of an event $e' \in E_{O'}$ such that $e' \neq e$, but nothing forces O' and O to be different orders. Hence, e and e' can be distinct events from the same order that cannot be distinguished by observing their causal past.

An order O *satisfies* ϕ, denoted by $O \models \phi$, iff there exists $e \in \min(O)$ such that $O, e \models \phi$. The set of orders $\mathcal{W}$ satisfies ϕ iff every LPO $O \in \mathcal{W}$ satisfies ϕ. Last, ϕ is *satisfiable* iff there exists a set of LPOs $\mathcal{W}$ such that $\mathcal{W} \models \phi$. Unsurprisingly, HyPOL is very powerful and satisfiability is undecidable on LPOs:

▶ **Theorem 5.** *Satisfiability of a HyPOL formula is undecidable.*

Proof Sketch. The proof is a reduction of Post's Correspondence Problem (PCP): given an instance I of PCP, we build a HyPOL formula ϕ_I such that I has a solution iff ϕ_I is satisfiable (See Appendix A for details). ◀

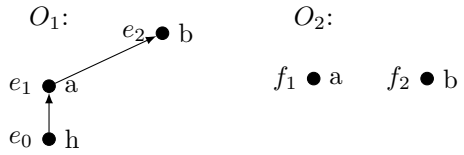


Figure 4 Two orders where observing linearizations is not enough to leak information.

3.2 An example: Causal Non-Interference

The example of Figure 4 shows that, in the context of concurrent models, languages are not discriminative enough to characterize some security leaks. Let $\mathcal{W} = \{O_1, O_2\}$ represent behaviors of a concurrent system, where h labels a non observable secret action, while events with labels a and b can be observed by an attacker. In a language-based setting, an attacker only observes the linearizations $a.b$ and $b.a$ of these orders. Hence it is not possible to deduce whether h has occurred or not. On the other hand, if causal dependencies are considered, observing that a precedes b reveals the occurrence of h, thus leaking the information that h occurred. Observation of causal dependencies during the execution of a system is not a purely hypothetic capacity of users. Indeed, systems equipped with mechanisms such as vectorial clocks [24] can be used to record faithfully dependencies among observed events. From a more practical point of view, messages exchange during web browsing sometimes allow to trace the last visits of users, and consequently some causal ordering among logged communications. Observation functions hence formalize which causal dependencies are captured by attackers. However, if an observation function erases some dependencies, and an attacker observes two apparently concurrent events, it might still be the case that these events are causally related in the execution that is observed. This information is simply lost during observation.

Non-interference is a more general example showing the discriminating power of HyPOL. In the setting proposed by [18], a system is non-interferent if users cannot infer that classified actions have occurred only from observation of the system, i.e., execution of a classified event does not affect what a user can see or do. Such situations occur in a distributed system which can be accessed by two kinds of users: those with a high accreditation level and low-level users that have limited access to operations and observations of the system. We suppose that high-level users can perform classified actions, the occurrences of which shall not be detected by low-level users. In a standard setting for non-interference properties, this situation is modeled by associating with each event occurring in the system a particular operation name. Let Σ be the set of all these names, with Σ_{high} the subset of confidential ones and $\Sigma_{low} = \Sigma \setminus \Sigma_{high}$ containing those which can be observed by low-level users. Observation $\mathcal{O}_{low}$ projects orders on events that carry at least one label in Σ_{low}. We can define a causal non-interference property with HyPOL as follows:

$$\phi_{CNI} ::= AG_{\Sigma,id}\left(\lambda_{\in\Sigma_{high}} \vee Pred_h \implies EX_{\equiv,\mathcal{O}_{low}}(\lambda_{\notin\Sigma_{high}} \wedge \neg Pred_h)\right)$$

where $\lambda_{\in\Sigma_{high}}$ stands for $\neg\lambda_{\notin\Sigma_{high}}$, $Pred_h ::= \bigvee_{a\in\Sigma} match(\mathcal{O}_{\mathrm{h},a}, T_{\mathrm{h}\leq a}, f)$, and $T_{\mathrm{h}\leq a}$ is the template containing a pair of events f_{h}, f such that $f_{\mathrm{h}} \leq f$, f_{h} carries proposition h, f carries proposition a and $\mathcal{O}_{\mathrm{h},a}$ is the observation that projects orders on $\Sigma_{high} \cup \{a\}$ and relabels events representing confidential operations with h.

Intuitively, satisfying $Pred_h$ means that a confidential operation occurred in the causal past of an event. Hence, an order O satisfies ϕ_{CNI} if, for every high-level event e in O, there exists an order O' and an event $e' \in O'$ such that $e \neq e'$, no high-level operation has occurred in the causal past of e', and a low level user cannot distinguish e from e' (i.e.,

$\mathcal{O}_{low}(\downarrow e) \equiv \mathcal{O}_{low}(\downarrow e'))$. A system is (causally) non-interferent iff every order generated by this system satisfies ϕ_{CNI}, i.e., every order that contains a confidential operation cannot be distinguished from other orders that do not contain confidential operations. Note that $\mathcal{O}_{low}(O)$ is a partial order, hence ϕ_{CNI} uses the discriminating power of causal dependencies. Notice also that local logics such as TLC^- or LD_0 cannot characterize (causal) non-interference, as they address properties of a single run, and cannot express the fact that a run must be observationally equivalent to another execution of the system, which is essential in ϕ_{CNI}.

4 Model-checking HyPOL

We address the question of model checking HyPOL formulas for a model for which at least reachability is decidable. As a starting point, we choose *Labeled Safe Petri Nets (LSPNs)*.

▶ **Definition 6.** A *Petri net* is a tuple $\mathcal{N} = (P, T, F, M_0)$ where P is a set of places, T is a set of transitions with $P \cap T = \emptyset$, $F \subseteq P \times T \cup T \times P$ is the flow relation, and $M_0 \in \mathbb{N}^P$ is the initial marking.

A net is *labeled* if it is equipped with a (not necessarily injective) mapping $\lambda : T \to \Sigma$ labeling the transitions. A *marking* is a multiset $M \in \mathbb{N}^P$. For $x \in P \cup T$, we define its *preset* by ${}^\bullet x = \{y \mid (y, x) \in F\}$ and its *postset* by $x^\bullet = \{y \mid (x, y) \in F\}$. The interleaved semantics of Petri nets can be defined as a (possibly infinite) transition system $LTS(\mathcal{N})$ where states are markings, the initial state is M_0, and the transition relation is defined by: $M \xrightarrow{t} M'$, iff (i) $M(p) \geq 1$ for all $p \in {}^\bullet t$, in which case transition t is said *firable* from M and (ii) $M' = (M \setminus {}^\bullet t) \uplus t^\bullet$ is the new marking reached by firing t. We write $M_0 \xrightarrow{*} M$ iff there exists a sequence of transition firings reaching M from M_0. The set of reachable markings is denoted by $Reach(\mathcal{N}) = \{M \mid M_0 \xrightarrow{*} M\}$.

We henceforth consider only safe Petri nets, where $Reach(\mathcal{N})$ is a subset of $\{0, 1\}^P$; we also assume that all transitions have at least one pre- and one post-place, i.e., $\forall\, t \in T :\ 1 \leq |{}^\bullet t| \geq 1 \leq |t^\bullet|$. Let us recall standard vocabulary and notations for nets (we borrow definitions from [16]). Two nodes $x, y \in P \cup T$ are in *causal relation* iff xF^*y. Transitions t and t' are in *immediate (structural) conflict* iff $t \neq t'$ and ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$. Nodes $x, x' \in T \cup P$ are in *conflict*, written $x \# x'$, iff there exist $t, t' \in T$ in immediate conflict such that tF^*x and $t'F^*x'$. A subset C of $T \cup P$ is *conflict free* if for all $x, x' \in C$, $\neg(x \# x')$.

▶ **Definition 7.** An *occurrence net* is a Petri net $ON = (B, E, F, Cut_0)$ where the elements of B are called *conditions* and those of E *events*, and $Cut_0 \subseteq B$ such that:

- ON is acyclic, and hence $< \stackrel{\text{def}}{=} F^+$ and $\leq \stackrel{\text{def}}{=} F^*$ are strict and weak partial orders;
- $\forall e \in E :\ \neg(e \# e)$ (no event is in conflict with itself);
- $\forall b \in B, |{}^\bullet b| \leq 1$ (every condition has a unique predecessor);
- ON is finitary: for all $x \in E \cup B$, the set $Past(x) \stackrel{\text{def}}{=} \{y \mid y \leq x\}$ is finite; and
- Cut_0 contains exactly the $<$-minimal nodes of ON.

Nodes x and y are in *concurrency relation*, denoted $x \mid\mid y$, if neither $x < y, x > y$ nor $x \# y$ holds. Note that *every* occurrence net is safe, and that occurrence net ON is *conflict free* iff for every $b \in B$, one has $|b^\bullet| \leq 1$.

▶ **Definition 8.** A *prefix* of an occurrence net $ON = (B, E, F, Cut_0)$ is an event set $R \subseteq E$ that is *downward closed*, i.e., such that $e \in R$ and $e' < e$ together imply $e' \in R$. A prefix $C \subseteq E$ is a *configuration* iff it is *conflict free*.

▶ **Definition 9.** Given a net $\mathcal{N} = (P, T, F, M_0)$, and an occurrence net $ON = (B, E, \hat{F}, Cut_0)$, a *homomorphism* is a map $\mu : E \cup B \to T \cup P$ such that:

- $\mu(B) \subseteq P$ and $\mu(E) \subseteq T$,
- for all $e \in E$, the restriction of μ to ${}^\bullet e$ is a bijection from ${}^\bullet e$ to ${}^\bullet\mu(e)$, and the restriction of μ to $e^\bullet$ is a bijection from $e^\bullet$ to $\mu(e)^\bullet$, and
- $\mu(Cut_0) = \{p \in P \mid M_0(p) = 1\}$

The "unfolding" semantics of a labeled safe Petri net yields a labeled occurrence net.

▶ **Definition 10** (Unfolding). A *branching process* of a labeled Petri net $\mathcal{N} = (P, T, F, M_0, \lambda)$ is a triple $BR = (ON, \mu, \lambda')$ where $ON = (B, E, \hat{F}, Cut_0)$ is an occurrence net, μ is a homomorphism and $\forall e \in E, \lambda'(e) = \lambda(\mu(e))$. A *process* of a net $\mathcal{N}$ is a branching process of $\mathcal{N}$ such that for every condition $b \in B$, $|b^\bullet| \leq 1$, or equivalently, such that E is a configuration. If $BR_1 = (B_1, E_1, \hat{F}_1, Cut_0, \mu_1, \lambda'_1)$ and $BR_2 = (B_2, E_2, \hat{F}_2, Cut_0, \mu_2, \lambda'_2)$ are two branching processes of $\mathcal{N}$, BR_1 is a *prefix* of BR_2 iff $E_1 \subseteq E_2$, and $\hat{F}_1, \mu_1, \lambda'_1$ are the respective restrictions of $\hat{F}_2, \mu_2, \lambda'_2$ to B_1 and E_1. The *unfolding* of $\mathcal{N}$, denoted by $\mathcal{U}(\mathcal{N})$, is the maximal branching process w.r.t. the prefix relation.

Although the construction is rather standard since [15], we give here, for the sake of completeness, a procedure to build an unfolding $\mathcal{U}(\mathcal{N})$ of an LSPN $\mathcal{N}$. We first define the notion of co-set and cut. A *co-set* of a branching process $BR = (ON, \mu, \lambda)$ with $ON = (B, E, \hat{F}, Cut_0)$ is a set of conditions that are pairwise concurrent. A maximal co-set (w.r.t. set inclusion) is called a *cut*. Finite configurations, cuts and markings are related as follows. If C is a configuration of a branching process $BR = (ON, \mu, \lambda')$, then we can define the co-set $Cut(C) = (Min(ON) \cup C^\bullet) \setminus {}^\bullet C$. The set of places in $Cut(C)$ represents the marking reached after firing transitions in $\mu(C)$ in an order compatible with the ordering prescribed by ON.

The construction of an unfolding of a net $\mathcal{N} = (P, T, F, M_0)$ consists in iteratively extending an initial branching process of $\mathcal{N}$. For convenience, we assume a dummy event $\perp$, whose postset fills all places of M_0. A *condition* of a branching process built by unfolding $\mathcal{N}$ is of the form $b = (e, p)$ where $p \in P$ is such that $\mu(b) = p$ and e is the (unique) input event of the condition b. Similarly, events are of the form $e = (X, t)$ where X is a set of conditions (and more precisely a co-set) and t the transition such that $\mu(e) = t$. One can notice that with these definitions of events and conditions, the flow relation in an unfolding is implicit : for an event $e = (X, t)$ and a condition $b = (e', p)$, $b \in {}^\bullet e$ iff $b \in X$, and $e \in {}^\bullet b$ iff $e' = e$. A *possible extension* of a branching process BR is an event (X, t), where $t \in T$ and X is a co-set such that $\mu(X) = {}^\bullet t$ and which does not belong to BR.

The initial branching process of the unfolding algorithm is $BR_0 = (ON_0, \mu_0, \lambda_0)$, where $ON_0 = (B_0, E_0, F_0)$, $B_0 = \{(\perp, p) \mid M_0(p) = 1\}$, $E_0 = \emptyset$, $F_0 = \{(\perp, b) \mid b \in B_0\}$, $\mu_0((\perp, p)) = p$. The following steps are then iterated to produce $BR_{i+1} = (B_{i+1}, E_{i+1}, F_{i+1}, \mu_{i+1}, \lambda_{i+1})$ from $BR_i = (B_i, E_i, F_i, \mu_i, \lambda_i)$:

1) find the set PE of possible extensions of BR_i;
2) if PE is not empty, choose a particular event $e = (X, t)$;
3) $E_{i+1} = E_i \cup \{e\}$
$B_{i+1} = B_i \cup X'$ with $X' = \{(e, p) \mid p \in t^\bullet\}$
$F_{i+1} = F_i \cup (X \times \{e\}) \cup (\{e\} \times X')$
μ_{i+1} extends μ_i by $\mu_{i+1}(e) = t$ and for any $b = (e, p) \in X'$, $\mu_{i+1}(b) = p$
λ_{i+1} extends λ_i by $\lambda_{i+1}(e) = \lambda(t)$.

With every process $BR = (ON, \mu, \lambda)$ contained in $\mathcal{U}(\mathcal{N})$, with $ON = (B, E, F, Cut_0)$, is associated an LPO $\mathcal{O}rd(BR) = (E, \leq, \lambda)$. Note that events in such LPOs are labeled by a singleton (transition label), which is a sub-case of the LPOs defined in Section 2. We define

$PR(\mathcal{N})$, the set of processes - up to isomorphism - that can be built from $\mathcal{N}$. Given a HyPOL formula ϕ, we say that $\mathcal{N}$ satisfies ϕ iff $\mathcal{O}rd(PR(\mathcal{N})) \models \phi$.

▶ **Theorem 11.** *The HyPOL model checking problem for safe Petri nets is undecidable.*

Proof (Sketch). We reuse the encoding of PCP from the proof of Theorem 5, and build a safe Petri net whose behaviors (processes) are exactly concatenations of the templates used in the HyPOL formula ϕ_I associated with an instance I of PCP. ◀

5 Decidability

The reason for the undecidability results above is that projections give a huge expressive power to HyPOL. Indeed, the difference in depth of equivalent events can be arbitrary large, and labeling allows for the design of a pair of growing sequences of letters w_1, w_2 where w_1 is always a prefix of w_2, yielding a non-terminating instance of PCP. We show in this section that one can recover decidability when restricting to Petri nets in which the difference in the depth of equivalent events is bounded.

Since the set of processes of a safe Petri net can be depicted in a compact way by its unfolding (as recalled in Section 4), a natural question is whether validity of a HyPOL formula expressing hyperproperties of the processes of a safe Petri net $\mathcal{N}$ can be rewritten as a property of its unfolding $\mathcal{U}(\mathcal{N})$. We first prove that this unfolding can be seen as a graph and defined as the production of a Hyperedge Replacement Grammar (HRG) [19].

▶ **Proposition 12.** *Let $\mathcal{N}$ be a safe labeled Petri net. Then, there exists a hyperedge replacement grammar $\mathcal{G}_{\mathcal{N}}$ that generates $\mathcal{U}(\mathcal{N})$.*

Proof (Sketch). We briefly give the principle for the construction of $\mathcal{G}_{\mathcal{N}}$. The unfolding algorithm in section 4 builds inductively an unfolding $\mathcal{U}(\mathcal{N})$ of $\mathcal{N}$. This unfolding can be infinite, but exhibits a regular structure. All markings and possible causal dependencies of $\mathcal{U}(\mathcal{N})$ are captured by a finite prefix of $\mathcal{U}(\mathcal{N})$ called a *complete finite prefix* [25]. Complete finite prefixes are built inductively as unfoldings, but with an additional constraint on the choice of events to add. Given a branching process BR_i and a possible extension e, e is called a *cut-off event* if the marking obtained after execution of $\downarrow e$ already appears in some execution of a process of BR_i. Construction of a complete finite prefix follows the same line as construction of unfoldings, but limits the choice of extensions of a branching process BR_i to events that *are not* cut-off events. The construction terminates for bounded nets [25].

Once a complete finite prefix of $\mathcal{N}$ is built, the principle for the construction of $\mathcal{G}_{\mathcal{N}}$ is to find the markings that can be reached when appending cut-off events to maximal configurations of the prefix. We then use these markings as hyperarcs, and the part of the prefix occurring after these markings as the right part of a grammar rule. We refer interested readers to Appendix B for a complete description of the construction of $\mathcal{G}_{\mathcal{N}}$. ◀

Note that $\mathcal{G}_{\mathcal{N}}$ does not define a semantics of $\mathcal{N}$ via application of one rewriting rule per transition firing, as proposed in [3, 4], but rather *builds* the unfolding. The grammar $\mathcal{G}_{\mathcal{N}}$ starts from an axiom Ax. Denoting by $\mathcal{G}^{\omega}_{\mathcal{N}}(Ax)$ the (unique) graph generated from Ax, we have $\mathcal{G}^{\omega}_{\mathcal{N}}(Ax) = \mathcal{U}(\mathcal{N})$. The grammar $\mathcal{G}_{\mathcal{N}}$ exhibits a certain form of regularity, but this is not yet sufficient to check HyPOL formulas, nor to express HyPOL properties in terms of properties of $\mathcal{G}_{\mathcal{N}}$. Indeed, the graphical representation of $\mathcal{U}(\mathcal{N})$ does not address equivalences. We adapt the idea of [1], and represent isomorphism of causal pasts of events w.r.t. an observation function as a new relation connecting events. In other words, we augment $\mathcal{U}(\mathcal{N})$ with additional edges connecting equivalent events.

▶ **Definition 13** (Execution Graph)**.** Given a set of observation functions $\mathcal{O}_1, \ldots, \mathcal{O}_k$, the *execution graph* of $\mathcal{N}$ is the graph $G_{\mathcal{U}(\mathcal{N})} = (E \cup B, \longrightarrow, \lambda)$, where E and B are the sets of events and conditions in $\mathcal{U}(\mathcal{N})$, and $\longrightarrow \subseteq (E \times \{0\} \times B) \cup (B \times \{0\} \times E) \cup (E \times \{1, \ldots k\} \times E)$ is the relation defined by: $(e, 0, b) \in \longrightarrow$ iff $e \in {}^\bullet b$ in $\mathcal{U}(\mathcal{N})$, $(b, 0, e) \in \longrightarrow$ iff $b \in {}^\bullet e$ in $\mathcal{U}(\mathcal{N})$, and $(e, i, e') \in \longrightarrow$ for $1 \leq i \leq k$ iff $e \neq e'$ and $\mathcal{O}_i(\downarrow e) \equiv \mathcal{O}_i(\downarrow e')$.

We write $e \xrightarrow{i} e'$ for $(e, i, e') \in \longrightarrow$. So far, we have simply recast ordering and equivalence of events into a graph setting, but this translation does not change decidability of hyperproperties. Even if the unfolding $\mathcal{U}(\mathcal{N})$ can be generated by an HRG, this is not the case for $G_{\mathcal{U}(\mathcal{N})}$. Indeed, to produce edges, hyperarcs of an HRG need to memorize nodes that will be at the origin or destination of an edge in future productions of the grammar. In particular, for $G_{\mathcal{U}(\mathcal{N})}$, this means that hyperarcs of any HRG producing this graph have to memorize a list of events that will be declared as equivalent to some event (w.r.t. a particular observation $\mathcal{O}_i$) generated in future rewritings.

▶ **Proposition 14.** *There exist labeled safe Petri nets and observation functions whose execution graphs are not of bounded treewidth, and cannot be represented by an hyperedge replacement grammar.*

Proof (Sketch). We exhibit a net, and an observation function whose execution graph contains grid minors of arbitrary sizes. It is well known [28] that a family of graphs FG has bounded treewidth iff there exists a constant m such that no graph $G \in FG$ has a minor isomorphic to the $m \times m$ grid and that HRGs can only generate graphs of bounded treewidth (see for instance [12]). See extended version for a complete proof. ◀

▶ **Definition 15.** Let $ON = (B, E, F, Cut_0)$ be an occurrence net. The *height* of en event e or condition b in ON is the function $\mathcal{H} : B \cup E \to \mathbb{N}$ be defined recursively by

$$\begin{aligned} \forall\, b \in Cut_0 :\ \mathcal{H}(b) &\stackrel{\text{def}}{=} 1 \\ \forall\, x \in B \cup E :\ \mathcal{H}(x) &\stackrel{\text{def}}{=} 1 + \max\{\mathcal{H}(y) \mid y \in {}^\bullet x\}\,. \end{aligned}$$

By extension, the height $\mathcal{H}(A)$ for a set $A \subseteq (B \cup E)$ is given by $\mathcal{H}(\emptyset) = 0$ and $\mathcal{H}(A) \stackrel{\text{def}}{=} \sup_{x \in A} \mathcal{H}(x)$. Now, define the *distance* $\mathsf{dist} : (B \cup E) \times (B \cup E) \to \mathbb{N}$ by

$$\begin{aligned} \mathcal{H}_\sqcap(e, e') &\stackrel{\text{def}}{=} \mathcal{H}\left(\downarrow e \cap \downarrow e'\right) \\ \mathsf{dist}(e, e') &\stackrel{\text{def}}{=} \max\left(\mathcal{H}\left(e\right), \mathcal{H}\left(e'\right)\right) - \mathcal{H}_\sqcap(e, e'). \end{aligned}$$

Intuitively, $dist(e, e')$ measures the maximal number of edges between e, e' and their common past. This distance dist defines a pseudometric. Using this notion of distance, we can define the *K-Ball* of an event e in the unfolding $\mathcal{U}(\mathcal{N})$ as the set of nodes in $\mathcal{U}(\mathcal{N})$ that are at distance at most K from e. Formally, $Ball_K(e) = \{n \in \mathcal{U}(\mathcal{N}) \mid \mathsf{dist}(n, e) \leq K\}$. In the rest of the paper, we consider classes of unfoldings where two events can only be equivalent w.r.t. any observation $\mathcal{O}_i$ if they are in the *K-Ball* of one another.

An important remark is that even for a safe Petri net $\mathcal{N}$, given an integer $K \in \mathbb{N}$, the *K-Ball* of an event e may not be finite. Furthermore, the graph $(E \cup B, \xrightarrow{0})$ depicting the unfolding $\mathcal{U}(\mathcal{N})$ without equivalence edges is always a graph of finite incoming degree, but this is not necessarily the case for $G_{\mathcal{U}(\mathcal{N})}$. In the rest of the paper, we will see that HyPOL formulas can be encoded as MSO properties of $G_{\mathcal{U}(\mathcal{N})}$. The reason for undecidability of HyPOL is hence the nature of execution graphs that cannot be generated in general by context free graph grammars, are not of bounded treewidth,... nor enjoy any of the properties that usually make MSO decidable. We can recover decidability with some restrictions. Let $\downarrow_K e = \downarrow e \cap Ball_K(e)$ denote the $K-$bounded past of e.

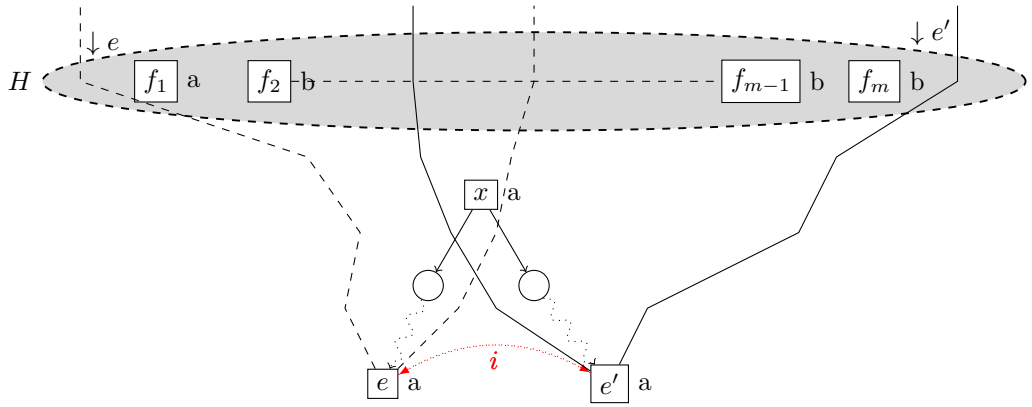


Figure 5 Equivalence w.r.t. $\mathcal{O}_i$ in the unfolding of a $K-$layered Petri net.

▶ **Definition 16.** Let $\mathcal{N}$ be a safe Petri net, and $\mathcal{O}_i$ be an observation function. $\mathcal{N}$ is $K-$layered w.r.t. $\mathcal{O}_i$ iff $\forall e, e' \in \mathcal{U}(\mathcal{N})$:

- there is a bound $S_K \in \mathbb{N}$ such that $|Ball_K(e)| \leq S_K$;
- $\mathsf{dist}(e, e') > K$ implies $e \not\equiv e'$;
- $\mathsf{dist}(e, e') \leq K$ implies that one can compute $H = \{f_1, \dots f_m\} \subseteq \downarrow_K e \cup \downarrow_K e'$ such that, letting $F_{e,e'} = \bigcup_{i \in 1..m} \downarrow f_i$ and $\hat{F}_{e,e'} = F_{e,e'} \setminus H$,

$$e \equiv_i e' \text{ iff } \mathcal{O}_i(\downarrow e \setminus \hat{F}_{e,e'}) \equiv_i \mathcal{O}_i(\downarrow e' \setminus \hat{F}_{e,e'}).$$

In the sequel, we assume that observation functions $\mathcal{O}_1, \dots \mathcal{O}_k$ are given, and we say that a safe Petri net $\mathcal{N}$ is $K-$layered iff it is $K-$layered for every $\mathcal{O}_i$. Intuitively, a Petri net is $K-$layered w.r.t. observation $\mathcal{O}_i$ iff one can decide equivalence of a pair of events e, e' w.r.t. $\mathcal{O}_i$ from their $K-$bounded past.

▶ **Proposition 17.** *Let $\mathcal{N}$ be a $K-$layered safe Petri net. Then, one can effectively compute a hyperedge replacement grammar $\mathcal{G}_{K,\mathcal{N}}$ that recognizes the execution graph $G_{\mathcal{U}(\mathcal{N})}$.*

Proof (Sketch). First, one can notice that in the unfolding of a $K-$layered safe Petri net, for every observation $\mathcal{O}_i$, every event e has a bounded number of events connected to it via relation $\xrightarrow{i}$. This is due to the fact that this set is contained in its finite *K-Ball*. The hyperedge replacement grammar $\mathcal{G}_{K,\mathcal{N}}$ starts from an axiom representing a complete finite prefix of the unfolding of $\mathcal{N}$ with hyperarcs. Its hyperarcs represent possible extensions of this prefix from its maximal markings. Rules of $\mathcal{G}_{K,\mathcal{N}}$ are of the form $r = (h_{t,lab}, HG_{t,lab})$ where $h_{t,lab}$ contains all conditions and events appearing in the *K-Balls* of the next occurrence of a transition t that can be appended after a maximal marking, and *lab* is a labeling providing sufficient information to know the ordering among events and a part of their common past. $HG_{t,lab}$ is an hypergraph containing the newly generated occurrences of events and conditions in the execution graph, the flow relation among them, and connects equivalent events (contained in the events of $h_{t,lab}$ and $HG_{t,lab}$) and creating one hyperarc per new maximal marking. A complete construction of this grammar is detailed in the extended version. ◀

We now show that model checking HyPOL on $K-$layered execution graphs can be brought back to verification of an equivalent MSO property. But the first question to address is decidability of MSO on execution graphs. An MSO formula uses the following syntax:

$$\phi \quad ::= \quad lab_a(x) \mid edge(x,y) \mid edge_i(x,y) \mid x = y \mid x \in X \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists x, \phi \mid \exists X, \phi$$

where $x, y, ...$ are first order variables representing vertices in a graph, and $X, Y, ...$ are second order variables representing sets of vertices in a graph. In execution graphs, first order variables will represent events or conditions, and an edge the flow relation or isomorphism.

An interpretation $\mathcal{I}$ of an MSO formula ϕ over a graph G is an assignment of nodes of G to first order variables used in ϕ and of subsets of nodes of G to second order variables. An MSO formula ϕ holds for G under interpretation $\mathcal{I}$ iff replacing variables in ϕ by their interpretation yields a tautology. A graph satisfies formula ϕ iff there exists an interpretation $\mathcal{I}$ such that ϕ holds for G under $\mathcal{I}$. Classes of graphs with decidable MSO theory have been considered for a long time (see for instance [12] for a complete monograph on this topic). As MSO is decidable for context free graphs such as the graphs generated by HRGs ([13], Corollary 4.10), we immediately have the following property:

▶ **Corollary 18.** *MSO is decidable on execution graphs of K–layered labeled safe Petri nets.*

Note that the decidability highlighted in corollary 18 does not necessarily hold outside the class of $K-$layered nets. As shown in Proposition 14, execution graphs of safe Petri nets may contain grids minors of arbitrary sizes and hence in general do not have a bounded treewidth [28]. MSO is also undecidable in general for execution graphs: one can use a safe Petri net whose unfolding is a binary tree and an observation that implements the "same level" relation on this tree. It is well known that MSO is undecidable on this graph [30]. We will use MSO to address decidability of HyPOL, by converting formulas to MSO, and in particular equivalences into $\xrightarrow{i}$ relations among events.

▶ **Proposition 19.** *Let ϕ be a HyPOL formula. Then there exists an MSO formula ψ such that $\mathcal{N} \models \phi$ iff $G_{\mathcal{U}(\mathcal{N})} \models \psi$.*

Proof (Sketch). We first encode in MSO a $succ(e, e')$ relation that relates pairs of events such that $e^\bullet \cap {}^\bullet e \neq \emptyset$. Then, causal precedence $\leq$ in an order can be encoded with MSO. A property of the form $x \models EX_{\equiv,\mathcal{O}_i}\ \phi$ asks existence of an edge $x \xrightarrow{i} y$ where y satisfies the MSO translation of ϕ. Until operations are described as properties of chains of events that can again be encoded with MSO, and pattern embedding are MSO properties checking existence of some subgraph. A complete translation is given in Appendix C. ◀

Proposition 19 holds for any net $\mathcal{N}$ and its execution graph $G_{\mathcal{U}(\mathcal{N})}$. However, in general, $G_{\mathcal{U}(\mathcal{N})}$ is not of bounded treewidth. One can always choose an integer K, and build a context free graph grammar $\mathcal{G}_{K,\mathcal{N}}$ as proposed in Proposition 17, but in general, the graph generated by $\mathcal{G}_{K,\mathcal{N}}$ is only a subgraph of $G_{\mathcal{U}(\mathcal{N})}$, where some $\xrightarrow{i}$ edges are missing. This is not surprising: in non-layered nets, the sizes of equivalence classes in $G_{\mathcal{U}(\mathcal{N})}$ need not be finite. If $\mathcal{N}$ is $K-$layered, the graph generated by $\mathcal{G}_{K,\mathcal{N}}$ and $G_{\mathcal{U}(\mathcal{N})}$ are equivalent. Further, isomorphism is one of the building blocks of HyPOL, but in general cannot be expressed in MSO. The translation from HyPOL to MSO applies to any HyPOL formula for any type of net and observation. Further, MSO is decidable for HRGs [13, 20]. So, in general, $G_{\mathcal{U}(\mathcal{N})}$ is not the production of an HRG. Altogether, these remarks give the following corollaries:

▶ **Corollary 20.** *It is undecidable whether the execution graph of a net $\mathcal{N}$ satisfies an MSO formula.*

▶ **Corollary 21.** *Model checking equivalence-free HyPOL properties on labeled safe Petri nets is decidable.*

▶ **Corollary 22.** *HyPOL model checking is decidable for $K-$layered safe Petri nets.*

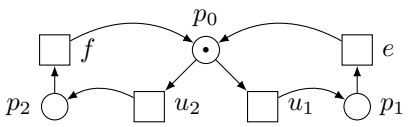


Figure 6 A net $\mathcal{N}_1$. Observation $\mathcal{O}_a$ projects LPOs on events labeled a. $\mathcal{N}_1$ is not observable: $\mathcal{O}_a$ cannot distinguish behaviors in $u_1.e.(u_1.e + u_2.f)^k$ from those in $u_2.f.(u_1.e + u_2.f)^k$.

K–layeredness is a semantic property that should hold on the possibly infinite unfolding of a net. However, some syntactic classes of nets meet the conditions needed to layer equivalences. In the following, we only consider observations that are projections. Slightly abusing our notations, for a transition t we will denote by $\mathcal{O}_i(t)$ the LPO obtained by applying observation $\mathcal{O}_i$ to the LPO O_t that contains a single event e with $\lambda(e)=\lambda(t)$.

▶ **Definition 23.** Let $\mathcal{N}$ be a safe Petri net. Two transitions t, t' are *independent* iff there is no link from t to t' in the flow relation of $\mathcal{N}$. We will say that $\mathcal{N}$ is *observable* iff,

- **i)** for every observation $\mathcal{O}_i$, and every cyclic behavior $t_1 \dots t_n$ of $LTS(\mathcal{N})$, $\mathcal{O}_i(t_1 \dots t_n) \neq \emptyset$,
- **ii)** For every reachable marking M of $\mathcal{N}$, every observation $\mathcal{O}_i$ and every pair of conflicting transitions t_1, t_2 enabled in M, there exists a bound k_c such that for every pair of paths $\rho = t_1.t_{1,1} \dots t_{1,p}$ and $\rho_2 = t_2.t_{2,1} \dots t_{2,q}$, if $p > k_c$ or $q > k_c$ then $\mathcal{O}_i(O_{\rho_1}) \neq \mathcal{O}_i(O_{\rho_2})$, where O_{ρ_1} (resp O_{ρ_2}) is the process of $\mathcal{N}$ obtained by successively appending $t_1, t_{1,1}, \dots$ (resp. $t_2, t_{2,1}, \dots$) to M_0.
- **iii)** for every observation $\mathcal{O}_i$ and every cyclic behavior $M \xrightarrow{\rho} M$ of $LTS(\mathcal{N})$ with $\rho = t_1 \dots t_n$ and such that $t_1 \dots t_n$ can be partitioned into sets $T_1, T_2, \dots T_k$ of independent transitions $\forall j, j' \in 1..k$, there exists $t_j \in T_j$ and $t_{j'} \in T_{j'}$ such that $\mathcal{O}_i(t_j) \neq \mathcal{O}_i(t_{j'})$.

Condition *i*) forbids cyclic behaviors that cannot be observed. This is a sensible restriction often required for diagnosis (where it is called *convergence*, as in [6]). It guarantees that an event cannot be equivalent to an arbitrary number of predecessors. Condition *ii*) indicates that each branch of a choice in the net is eventually visible by each observation after a bounded duration. Condition *iii*) says that parallel sequences of transitions cannot grow up to an arbitrary size without becoming distinguishable by all observations.

▶ **Proposition 24.** *Let $\mathcal{N} = (P, T, F, M_0, \lambda)$ be a safe labeled observable Petri net for observations $\mathcal{O}_1, \dots, \mathcal{O}_k$. Then $\mathcal{N}$ is $K-$layered, for some $K \leq \max(2 \cdot k_c, 3 \cdot |T|)$*

▶ **Corollary 25.** *HyPOL model-checking is decidable for observable safe Petri nets.*

6 Conclusion

HyPOL is a local logic for hyperproperties of partially observed set of labeled partial orders. It is powerful enough to express properties such as non-interference in distributed systems. This logic follows the same line as local logics such as TLC^- or LD_0, as it depicts shapes of causal chains in partially ordered computations. In addition, it is possible to check whether some finite behavior has occurred in the past, and a new modal operator is introduced to move from an event in an LPO to another equivalent event in another LPO. Unsurprisingly, such a powerful logic is undecidable, even for simple models such as safe labeled Petri nets. However, upon some restrictions, one can bring back verification of HyPOL formulas to verification of MSO properties on unfoldings of nets decorated with additional edges that simulate equivalences. The restrictions forbid nets with infinite unobservable runs, and

assume bounds on the depth of indistinguishable suffixes. In this context, equivalence of runs only depends on a bounded future and past of each event, and decorated unfoldings have bounded treewidth. So far, we do not know whether K–layeredness is decidable for a fixed K. Another interesting question is existence of a bound K such that a net $\mathcal{N}$ is K–layered. We strongly believe that some restrictions used in observable nets can be relaxed, or adapted to consider larger classes of nets for which decorated unfoldings are of bounded treewidth or split-width [14]. A natural question that follows is whether these classes of nets have sensible and decidable syntactic characterizations.

References

1 R. Alur, P. Cerný, and S. Chaudhuri. Model Checking on Trees with Path Equivalences. In *TACAS 2007*, volume 4424 of *LNCS*, pages 664–678. Springer, 2007.

2 E. Badouel, M.A. Bednarczyk, A.M. Borzyszkowski, B. Caillaud, and P. Darondeau. Concurrent Secrets. *Discrete Event Dynamic Systems*, 17(4):425–446, 2007.

3 P. Baldan, T. Chatain, S. Haar, and B. König. Unfolding-based Diagnosis of Systems with an Evolving Topology. In *19th International Conference on Concurrency Theory (CONCUR'08)*, volume 5201 of *LNCS*, pages 203–217. Springer, 2008. URL: `http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/BCHK-concur08.pdf`.

4 P. Baldan, T. Chatain, S. Haar, and B. König. Unfolding-based Diagnosis of Systems with an Evolving Topology. *Information and Computation*, 208(10):1169–1192, October 2010. `doi:10.1016/j.ic.2009.11.009`.

5 B. Bérard, S. Haar, and L. Hélouët. Hyper Partial Order Logic. *HAL-Inria*, 2018. URL: `https://hal.inria.fr/hal-01884390`.

6 B. Bérard, S. Haar, S. Schmitz, and S. Schwoon. The Complexity of Diagnosability and Opacity Verification for Petri Nets. In *PETRI NETS'17*, volume 10258 of *LNCS*, pages 200–220. Springer, 2017.

7 B. Bérard, L. Hélouët, and J. Mullins. Non-interference in Partial Order Models. *ACM Trans. Embedded Comput. Syst.*, 16(2):44:1–44:34, 2017.

8 E. Best, P. Darondeau, and R. Gorrieri. On the Decidability of Non Interference over Unbounded Petri Nets. In *Proc. of SecCo*, volume 51 of *EPTCS*, pages 16–33, 2010.

9 B. Bollig, D. Kuske, and I. Meinecke. Propositional Dynamic Logic for Message-Passing Systems. *Logical Methods in Computer Science*, 6(3), 2010.

10 M.R. Clarkson, B. Finkbeiner, M. Koleini, K.K. Micinski, M.N. Rabe, and C. Sánchez. Temporal Logics for Hyperproperties. In *POST*, pages 265–284, 2014.

11 M.R. Clarkson and F.B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.

12 B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-Order Logic, a language theoretic approach*. Cambridge Univ. Press, 2012.

13 Bruno Courcelle. The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs. *Inf. Comput.*, 85(1):12–75, 1990.

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

15 J. Engelfriet. Branching Processes of Petri Nets. *Acta Inf.*, 28(6):575–591, 1991.

16 J. Esparza, S. Römer, and W. Vogler. An Improvement of McMillan's Unfolding Algorithm. *Formal Methods in System Design*, 20(3):285–310, 2002.

17 T. Gazagnaire, L. Hélouët, and S. S. Yang. Logic-based diagnosis for distributed systems. *Perspectives in Concurrency, a festschrift for P.S. Thiagarajan*, pages 482–505, 2009.

18 J.A. Goguen and J. Meseguer. Security policies and security Models. In *Proc. of IEEE Symposium on Security and Privacy*, pages 11–20, 1982.

19 A. Habel. *Hyperedge Replacement: Grammars and Languages*, volume 643 of *LNCS*. Springer, 1992.

20 C. Lautemann. Tree Automata, Tree Decomposition and Hyperedge Replacement. In *Graph-Grammars and Their Application to Computer Science, 4th International Workshop*, volume 532 of *LNCS*, pages 520–537. Springer, 1990.

21 P. Madhusudan and B. Meenakshi. Beyond Message Sequence Graphs. In *FST TCS'01: Foundations of Software Technology and Theoretical Computer Science*, volume 2245 of *LNCS*, pages 256–267. Springer, 2001.

22 P. Madhusudan, P.S. Thiagarajan, and S. Yang. The MSO Theory of Connectedly Communicating Processes. In *FSTTCS'05*, volume 3821 of *LNCS*, pages 201–212. Springer, 2005.

23 H. Mantel. Possibilistic Definitions of Security - An Assembly Kit. In *Proc. of the 13th IEEE Computer Security Foundations Workshop, (CSFW'00)*, pages 185–199, 2000.

24 F. Mattern. Time and global states of distributed systems. *Proc. Int. Workshop on Parallel and Distributed Algorithms*, pages 215–226, 1988.

25 K.L. McMillan. A Technique of State Space Search Based on Unfolding. *Formal Methods in System Design*, 6(1):45–65, 1995.

26 B. Meenakshi and R. Ramanujam. Reasoning about layered message passing systems. *Computer Languages, Systems & Structures*, 30(3-4):171–206, 2004.

27 D.A. Peled. Specification and Verification of Message Sequence Charts. In *FORTE/PSTV'00*, volume 183 of *IFIP Conference Proceedings*, pages 139–154. Kluwer, 2000.

28 N. Robertson and P.D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *J. Comb. Theory, Ser. B*, 52(2):153–190, 1991.

29 A. Sabelfeld and A.C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.

30 A. Spelten, W. Thomas, and S. Winter. Trees over Infinite Structures and Path Logics with Synchronization. In *13th International Workshop on Verification of Infinite-State Systems, INFINITY 2011*, volume 73 of *EPTCS*, pages 20–34, 2011.

A Proof of Theorem 5

▶ **Theorem 5.** *Satisfiability of a HyPOL formula is undecidable.*

Proof. The proof consists of a reduction of the Post Correspondence Problem (PCP). Recall that an instance I of PCP is a sequence $(x_1, y_1), \ldots, (x_n, y_n)$ of n pairs of words over some alphabet. A (non trivial) solution of size k is a (non empty) sequence of indices $\sigma = i_1 \ldots i_k$ such that $x_{i_1} \ldots x_{i_k} = y_{i_1} \ldots y_{i_k}$. If the alphabet contains at least two letters, PCP is undecidable for $n \geq 7$. Moreover, we can assume that for all $1 \leq i \leq n$, $x_i \neq y_i$ (otherwise the problem can be trivially decided with a solution of size $k = 1$).

Given an instance I, we build a formula ϕ_I of HyPOL such that ϕ_I is satisfiable if and only if I has a (non trivial) solution.

Let I be the sequence $(x_1, y_1), \ldots, (x_n, y_n)$ of words over alphabet A. We write $z = z(1) \ldots z(\ell)$ where $\ell = |z|$ is the length of word z, with $\ell_i = |x_i|$ and $h_i = |y_i|$, $1 \leq i \leq n$ and we consider the family of templates T_i, $1 \leq i \leq n$, as depicted in Figure 7. The set of events of T_i is $E_i = \{x_i, y_i, s_i, e_i\} \cup \{x_{i,j} \mid 1 \leq j \leq \ell_i\} \cup \{y_{i,j} \mid 1 \leq j \leq h_i\}$ and labels are in $P_i = A \cup \{\sharp, \mathrm{start}_i, \mathrm{end}_i\}$. We set $Ind = \{start_i, end_i, 1 \leq i \leq n\}$, $S = \{start_i, 1 \leq i \leq n\}$ and the global set of labels is $P = \cup_{i=1}^{n} P_i$. Intuitively, a solution $\sigma = i_1 \ldots i_k$ will be described by the sequence of templates $T_{i_1} \ldots T_{i_k}$.

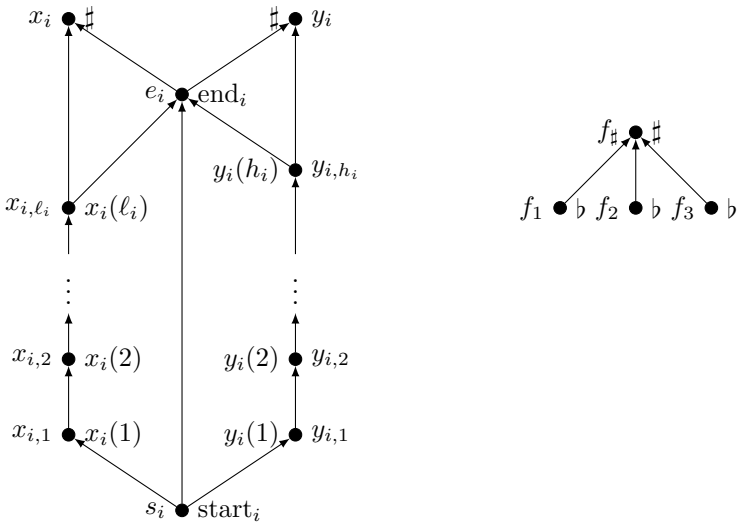


Figure 7 Templates T_i and $T_\sharp$.

To detect that a solution ends with an event labeled by ♯, we define the formula $stop ::= \lambda_{=\{\sharp\}} \wedge \neg EX_{P,id} true$. We can express that any event with label ♯ has at most two predecessors:

$$two\text{-}pred\sharp ::= AG_{P,id}(\lambda_{=\{\sharp\}} \implies \neg match(\mathcal{O}_\sharp, T_\sharp, f_\sharp))$$

where $T_\sharp$ is the pattern depicted onFigure 7 right and $\mathcal{O}_\sharp$ keeps any event with label ♯ unchanged and relabels all other events with ♭. Now, if $\mathcal{O}_S$ denotes the projection on S, keeping only events with labels in S, a solution is described by:

$$IsSeqIndex ::= EG_{S,\mathcal{O}_S}(\vee_{i=1}^{n} HoldsT_i) \wedge EF_{P,id} stop$$

where $HoldsT_i ::= match(id, T_i, s_i)$. Finally, we consider the subset $\mathcal{W}$ of orders of $\mathcal{LPO}(P)$ where all labels are singletons. Note that this condition can be ensured by the formula $Sing ::= AG_{P,id}(\vee_{p\in P} \lambda_{=\{p\}})$. For an order $O = (E, \leq, \lambda) \in \mathcal{W}$, we write $E = E_A \cup E_\sharp \cup E_{ind}$ as a disjoint union with $E_A = E \cap \lambda^{-1}(A)$, $E_\sharp = E \cap \lambda^{-1}(\{\sharp\})$ and $E_{ind} = E \cap \lambda^{-1}(Ind)$. We define the observation function $\mathcal{O}_{sol}$ over $\mathcal{W}$ by keeping all events and restricting $\leq$ to $(E \times E) \setminus ((E_A \times E_{ind}) \cup (E_{ind} \times E_A))$, thus removing the order between letters and indices. The formula ϕ_I is then defined by :

$$\phi_I ::= two\text{-}pred\sharp \wedge IsSeqIndex \wedge (stop \implies EX_{\equiv,\mathcal{O}_{sol}} true),$$

where the last sub-formula means that from some final ♯, it will not be possible to distinguish between paths with labels from the x_i's and those with labels from the y_i's.

Then, there is an order O in $\mathcal{W}$ satisfying ϕ_I if and only if I has a non trivial solution. ◀

B Construction of a hyperarc replacement grammar for $\mathcal{U}(\mathcal{N})$

▶ **Proposition 12.** *Let $\mathcal{N}$ be a safe labeled Petri net. Then, there exists a hyperedge replacement grammar $\mathcal{G}_\mathcal{N}$ that generates $\mathcal{U}(\mathcal{N})$.*

▶ **Definition 26.** A *hyperarc* is a pair (l, V), where l is a label, and $V \subseteq \mathbb{N}$ is an ordered set of vertices. A *hypergraph* is a triple (V, E, H) where V is a set of vertices, E a set of edges, and H a set of hyperarcs. A *hyperedge replacement grammar* (HRG) is defined as a pair $\mathcal{G} = (Ax, \mathcal{R})$, where Ax is a hypergraph called the axiom of the grammar and $\mathcal{R}$ is a set of rules. A grammar *rule* is a pair (L, R) where L, the left part of the rule is a hyperarc, and R, the right part of the rule is a hypergraph that contains all vertices of L.

Let $G = (V, E, H)$ be a hypergraph and $h = (l_h, V_h) \in H$ a hyperarc. Let $r = (L, R)$ be a rule where $L = (l, X)$ is a hyperarc with label $l = l_h$ and the same number of vertices as V_h, and $R = (V_R, E_R, H_R)$. The application of rule r to G simply replaces hyperarc h in G by the right part R. More formally, application of r produces a hypergraph $G' = (V', E', H')$ with $V' = V \uplus (\alpha(V_R) \setminus X)$, $E' = E \uplus \alpha(E_R)$ and $H' = H \setminus \{h\} \uplus \alpha(H_R)$, where $\alpha : \mathbb{N} \to \mathbb{N}$ is a map that associates with the j^{th} vertex of X the identity of the j^{th} vertex in V_h, and associates with vertices of $V_R \setminus X_R$ a fresh identity that does not appear in V. We denote by $G \xrightarrow{r} G'$ this rewriting step, and by $\mathcal{G}^\omega(G)$ the (possibly infinite) limit graph obtained by application of rules of grammar $\mathcal{G}$ on G.

Let $\mathcal{N} = (P, T, F, M_0, \lambda)$ be a safe labeled Petri net. We fix an arbitrary order $<_P$ on places. Given a marking M, and a set of integers $1 \dots |M|$, we denote by $index(p, M) \in 1 \dots |M|$ the rank of place p in the sequence of integers representing marked places in M. Similarly, given a marking M and a list of integers representing this marking, we denote by $place(i)$ the place represented by index i.

We have seen in section 4 an algorithm to build inductively an unfolding of a safe Petri net $\mathcal{N}$. This unfolding can be infinite, but exhibits a regular structure. Furthermore, many verification algorithms addressing reachability of coverability questions work on a structure called a *complete finite prefix*. A complete finite prefix is built inductively as an unfolding, but stops within a finite number of steps, according to some criterion that forbids the addition of events fulfilling some properties. A stopping criterion frequently met is the reachability criterion: it forbids a possible extension if adding the considered event produces a configuration that ends in a marking that was already visited in the branching process [25]. These events are called *cut-off events*. The principle of the HRG construction described hereafter is to build a complete finite prefix of net $\mathcal{N}$, to find the markings that can be reached when appending cut-off events to maximal configurations of the prefix. We then use these markings as hyperarcs, and the part of the prefix occurring after the marking as the right part of a grammar rule.

Let us first recall some definitions borrowed from [25]. Let $ON = (B, E, F)$ be an occurrence net and let S be a configuration of ON. We denote by $S^\bullet$ the set of all places that are maximal w.r.t. to this configuration, i.e., the set X of all places such that $\forall p \in X, \forall e \in S, p \notin {}^\bullet e$ and $\forall p \in X, \forall e \in E \setminus S, p \notin e^\bullet$. Let μ be a homomorphism from ON to $\mathcal{N}$. The final state of a configuration $\mathcal{F}(S)$ is the marking $\mu(S^\bullet)$. The local configuration of an event e is the set $\downarrow e$.

Let BR be a branching process. A possible extension e is a *cut-off event* (w.r.t. the reachability criterion) iff there exists another event e' such that $\mathcal{F}(\downarrow e^\bullet) = \mathcal{F}(\downarrow e'^\bullet)$, and $|\downarrow e'^\bullet| < |\downarrow e^\bullet|$. Now, the algorithm to compute a complete finite prefix is the following:

0) Start from the initial branching process BR_0
1) find the set PE of possible extensions of BR_i, i.e., the fresh pairs (X, t) such that X is a co-set of BR and $\mu_i(X) = {}^\bullet t$;
2) Compute $NE = \{pe \in PE \mid pe \text{ is not a cut-off event}\}$
3) while NE is not empty,

4) choose a particular event $e = (X, t)$ in NE
5) $E_{i+1} = E_i \cup \{e\}$
$B_{i+1} = B_i \cup X'$ with $X' = \{(e, p) \mid p \in t^\bullet\}$
$F_{i+1} = F_i \cup (X \times \{e\}) \cup (\{e\} \times X')$
μ_{i+1} extends μ_i by $\mu_{i+1}(e) = t$ and for any $b = (e, p) \in X'$, $\mu_{i+1}(b) = p$.
λ_{i+1} extends λ_i by $\lambda_{i+1}(e) = \lambda(t)$.
6) compute the set PE of possible extensions of BR_{i+1};
7) Compute $NE = \{pe \in PE \mid pe$ is not a cut-off event$\}$
8) endwhile

It is well known (see for instance [25]) that:

- the construction of a complete finite prefix w.r.t. the reachability criterion terminates,
- all cuts of the prefix (and in fact even all those of the unfolding) correspond via μ to a reachable marking, and
- conversely, all reachable markings of an unfolded net are represented by at least one cut in the prefix.

Let us call $CFP(\mathcal{N})$ the complete finite prefix thus built; then for every reachable marking M of $\mathcal{N}$, there exists a configuration S of $CFP(\mathcal{N})$ such that $\mathcal{F}(S^\bullet) = M$.

We can now detail the construction of a HRG that generates the unfolding of $\mathcal{N}$. We first build $CFP(\mathcal{N})$ using the algorithm above. Then, we compute the set PE of possible extensions in $CFP(\mathcal{N})$, and add these possible extensions to $CFP(\mathcal{N})$. Let $BR_{CFP,PE}$ be the branching process obtained by adding these events, and let $S_1, \dots S_k$ be the maximal configurations of $BR_{CFP,PE}$. For every S_i there exists at least one configuration S'_i of $CFP(\mathcal{N})$ such that $\mathcal{F}(S_i{}^\bullet) = \mathcal{F}(S'_i{}^\bullet)$. Note that for the reachability cut-off criterion, there can be more than one configuration of this form. We can choose arbitrarily one of them, for instance the configuration with the minimal number of events. For such a configuration S'_i we denote by $\uparrow_{BR_{CFP,PE}} S'_i$ the restriction of $BR_{CFP,PE}$ to events and conditions that are descendants of $S'_i{}^\bullet$.

We build the grammar $\mathcal{G}_\mathcal{N} = (Ax, \mathcal{R})$ as follows. We set $Ax = (N_0, H_0)$ where $N_0 = BR_{CFP,PE}$ and $H_0 = \{(l_i, X_i) \mid S_i$ is a maximal configuration of $BR_{CFP,PE}\}$ where each X_i is an ordered set of vertices containing all conditions in $S_i{}^\bullet$ (we can order vertices according to $<_P$ and according to the place $\mu(b)$ represented by each condition b in X_i).

Then, for every maximal configuration S_i in $BR_{CFP,PE}$, we create a rule $r_i = (L_i, R_i)$ where L_i is a hyperarc $L_i = (l_i, 1 \dots |S_i{}^\bullet|)$, and $R_i = (V_i, E_i, H_i)$, where (V_i, E_i) is a copy of $\uparrow_{BR_{CFP,PE}} S'_i$, in which conditions in S'_i are numbered $1 \dots |S'_i{}^\bullet|$. Last, H_i is the set of hyperarcs of the form $h = (l_i, X_i)$, where X_i is a set of conditions contained in $E_i \cap BR_{CFP,PE}$.

One can notice that $\mathcal{G}_\mathcal{N}$ may have up to $2^{|P|}$ rules. We can show that $\mathcal{G}^\omega_\mathcal{N}(Ax) = \mathcal{U}(\mathcal{N})$.

C Proof of Proposition 19

▶ **Proposition 19.** *Let ϕ be a HyPOL formula. Then there exists an MSO formula ψ such that $\mathcal{N} \models \phi$ iff $G_{\mathcal{U}(\mathcal{N})} \models \psi$.*

Proof. Without leaving MSO, we can define a particular labeling to differentiate events and conditions in $G_{\mathcal{U}(\mathcal{N})}$: We write $Cond(x)$ for the predicate that holds for every condition and $Event(x)$ for the predicate that holds on all events.

We first define some basic formulas, holding at some node of $G_{\mathcal{U}(\mathcal{N})}$:

- $true$ holds for every element of $G_{\mathcal{U}(\mathcal{N})}$;
- $Lab(x) \cap D \neq \emptyset$ is equivalent to the formula $\bigvee_{d \in D} lab_d(x)$;

- $Event(x)$ holds under any interpretation that assigns an event of $G_{\mathcal{U}(\mathcal{N})}$ to x;
- $Cond(x)$ holds under any interpretation that assigns a condition of $G_{\mathcal{U}(\mathcal{N})}$ to x;
- $edge(x, y)$ holds under an interpretation that assigns a condition b to x and an event e to y, and such that $b \in {}^\bullet e$, or an event e to x and a condition b to y such that $b \in e^\bullet$;
- $edge_i(x, y)$ holds under any interpretation $\mathcal{I}$ that assigns events $\mathcal{I}(x)$ and $\mathcal{I}(y)$ of $G_{\mathcal{U}(\mathcal{N})}$ to x and y and such that $\mathcal{I}(x) \xrightarrow{i} \mathcal{I}(y)$.

From these building blocks, we can define more advanced expressions.

- $succ(x, y)$ is a formula that holds under an interpretation $\mathcal{I}$ such that $e = \mathcal{I}(x)$ is an event, $f = \mathcal{I}(y)$ is an event, and the pair of events e, f is in immediate successor relation in $G_{\mathcal{U}(\mathcal{N})}$. Formally, this is written as:
  $succ(x, y) ::= \exists z, Event(x) \wedge Event(y) \wedge Cond(z) \wedge edge(x, z) \wedge edge(z, y)$.
- $isMinimal(x, X)$ is a formula that holds under an interpretation that maps variable x to an event, X to a set of nodes of $G_{\mathcal{U}(\mathcal{N})}$, and such that $\mathcal{I}(x)$ is minimal in X with respect to the causal ordering of $G_{\mathcal{U}(\mathcal{N})}$. Formally, we write:
  $isMinimal(x, X) ::= x \in X \wedge Event(x) \wedge \nexists y \in X, succ(y, x)$.
- $isMaximal(x, X)$ is similar to the previous formula, and requires $\mathcal{I}(x)$ to be maximal in X. It is defined as: $isMaximal(x, X) ::= x \in X \wedge Event(x) \wedge \nexists y \in X, succ(x, y)$
- $isAChain(x, X)$ is a formula that holds for any interpretation $\mathcal{I}$ in which X is a chain (a totally ordered sequence of events w.r.t. the successor relation) starting from x. It is formulated as follows:

$$\begin{array}{rl} & isMinimal(x, X) \wedge \forall y \in X, \\ isAChain(x, X) ::= & (isMinimal(y, X) \Longrightarrow x = y) \wedge \\ & (\exists z \in X, succ(y, z)) \Longrightarrow (\nexists z' \in X, z \neq z' \wedge succ(y, z')) \end{array}$$

- $x \leq y$ can be defined as the formula:

$$\begin{array}{rl} & Event(x) \wedge Event(y) \wedge \exists X, x \in X \wedge y \in X \\ x \leq y ::= & \wedge \forall z \in X, succ(z, z') \Longrightarrow z' \in X \\ & \wedge \forall u \in X, \nexists u', succ(u', u) \Longrightarrow u = x \end{array}$$

  More intuitively, this formula says that $\mathcal{I}(X)$ is the set of all successors of $\mathcal{I}(x)$ in $G_{\mathcal{U}(\mathcal{N})}$, and it contains $\mathcal{I}(y)$. This is a standard formula frequently used when addressing properties of partially ordered sets.
- $x < y$ (covering) is defined by $x < y ::= x \leq y \wedge \nexists z, z \neq x, z \neq y, x \leq z \wedge z \leq y$.
- Let $\mathcal{O}$ be a particular observation erasing events that do not carry a label from a particular subset D, and restrict covering of the obtained order to pairs of events carrying specific pairs of labels in $R \subseteq \Sigma \times \Sigma$. Then one can define $x <_{\mathcal{O}} y$ as the formula stating that the labels attached to x and y are contained in D, that $(lab(x), lab(y)) \in R$, that there exists a path from x to y such that every intermediate event visited between x and y carries a label that does not belong to D. This type of construction applies for all kind of labeling-based projection and order restriction. More formally :

$$\begin{array}{rl} & Event(x) \wedge Event(y) \wedge Lab(x) \cap D \neq \emptyset \wedge Lab(y) \cap D \neq \emptyset \\ & \wedge\, x \leq y \\ x <_{\mathcal{O}} y \;\; ::= & \wedge\, \forall z, x < z \wedge z < y \Longrightarrow Lab(z) \cap D = \emptyset \\ & \wedge \bigvee_{(a,b) \in R} lab_a(x) \wedge lab_b(y) \end{array}$$

We are now ready to transform HyPOL formulas into MSO formulas. For every hypol formula ϕ we will build inductively an MSO formula ψ. The inductive construction will use fresh first order variables $x, y, ...$ and second order variables $X, Y, ...$ at every induction step. Further, as HyPOL formulas should hold at a particular event, we will design ψ with a particular free variable x depicting the event at which ψ must hold. For every HyPOL formula ϕ, letting ψ be the MSO formula obtained by translation of ϕ into MSO, for every order O in $\mathcal{O}rd(PR(\mathcal{N}))$ and every event $e \in E_O$, $O, e \models \phi$ if and only if ψ holds in $G_{\mathcal{U}(\mathcal{N})}$ under an interpretation that assigns e to x. We hence define $\psi = MSO(\phi, x, C)$ where C is a context listing variable names already used, x is a free variable in ψ that appears in C, and ψ is an MSO formula over x and fresh variable names not used in C. For a given HyPOL formula ϕ, we build inductively $\psi = MSO(\phi, x, C)$ as follows:

- if $\phi = true$ then $MSO(\phi, x, C) = true$ for any variable x and context C;
- if $\phi = \neg\phi'$ then $MSO(\phi, x, C) = \neg(MSO(\phi', x, C))$;
- if $\phi = \phi_1 \wedge \phi_2$ then $MSO(\phi, x, C) = MSO(\phi_1, x, C) \wedge MSO(\phi_2, x, C)$;
- if $\phi = EX_{D,\mathcal{O}}\ \phi'$ then $MSO(\phi, x, C) = \exists y, x \leq_{\mathcal{O}} y \wedge MSO(\phi, y, C')$ where y is a fresh variable name (w.r.t. C and to the set $C_{x \leq_{\mathcal{O}} y}$ of variables used to encode subformula $x \leq_{\mathcal{O}} y$) and $C' = C \cup \{y\} \cup C_{x \leq_{\mathcal{O}} y}$;
- if $\phi = match(\mathcal{O}, T, f)$ where $T = (E, <_T, \lambda_T)$, with $E = \{f\} \cup \{e_1, e_{|E|-1}\}$ then
$$MSO(\phi, x, C) = \exists x_1, \dots x_{|E|-1}, \bigwedge_{(f,e_i)\in<_T} x <_{\mathcal{O}} x_i \wedge \bigwedge_{(e_i,e_j)\in<_T} x_j <_{\mathcal{O}} x_i \wedge \bigwedge_{i\in 1..|E|-1} Lab(x_i) \supseteq \lambda_T(x_i)$$
where $x_1, \dots x_{|E|-1}$ are fresh variable names (w.r.t. C);
- if $\phi = EX_{\equiv,\mathcal{O}_i}\ \phi'$ then
$MSO(\phi, X, C)) = \exists y, edge_i(x, y) \wedge MSO(\phi', y, C')$
where y is a fresh variable name (w.r.t. C) and $C' = C \cup \{y\}$;
- if $\phi = \phi_1\ EU_{D,\mathcal{O}}\ \phi_2$ then
$$MSO(\phi, x, C) = \exists X, \quad isAChain(x, X) \quad \wedge\forall y \in X, \exists y', y <_{\mathcal{O}} y' \Longrightarrow MSO(\phi_1, y, C') \quad \wedge\nexists y', y <_{\mathcal{O}} y' \Longrightarrow MSO(\phi_2, y, C')$$
where y, y', X are fresh variable names (w.r.t. C and to the sets $C_{y,y'}$ and C_{chain} of variables used to encode respectively formulas $y <_{\mathcal{O}} y'$ and $isAChain(x, X)$) and $C' = C \cup \{y\} \cup C_{y,y'} \cup C_{chain}$;
- if $\phi = EG_{D,\mathcal{O}}\ \phi'$ then $MSO(\phi, x, C) = \exists y, Event(y) \wedge x <_{\mathcal{O}} y \wedge MSO(\phi', y, C')$ where y is a fresh variable name (w.r.t. C) and $C' = C \cup \{y\}$.

We have assumed that the unfolding of $\mathcal{N}$ has a unique starting event denoted by $\perp$ and carrying label $\perp$. We can prove by induction on the length of HyPOL formulas that $\mathcal{N} \models \phi$ iff $G_{\mathcal{U}(\mathcal{N}} \models \exists s, x, lab_{\perp}(s) \wedge succ(s, x) \wedge MSO(\phi, x, \{s, x\})$. ◀

On the Way to Alternating Weak Automata

Udi Boker[1]
IDC Herzliya, Israel

Karoliina Lehtinen[2]
Kiel University, Kiel, Germany

Abstract

Different types of automata over words and trees offer different trade-offs between expressivity, conciseness, and the complexity of decision procedures. *Alternating weak automata* enjoy simple algorithms for emptiness and membership checks, which makes transformations into automata of this type particularly interesting. For instance, an algorithm for solving two-player infinite games can be viewed as a special case of such a transformation. However, our understanding of the worst-case size blow-up that these transformations can incur is rather poor. This paper establishes two new results, one on word automata and one on tree automata. We show that:

- Alternating parity word automata can be turned into alternating weak automata of quasi-polynomial (rather than exponential) size.
- Universal co-Büchi tree automata, a special case of alternating parity tree automata, can be exponentially more concise than alternating weak automata.

Along the way, we present a family of game languages, strict for the levels of the weak hierarchy of tree automata, which corresponds to a weak version of the canonical game languages known to be strict for the Mostowski–Rabin index hierarchy.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases Alternating automata, Parity games, Parity automata, Weak automata

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.21

1 Introduction

The interplay between games and automata has been proven fruitful for both game- and automata-theory. In particular, solving two-player infinite games of some winning condition, such as Büchi or parity, reduces to transforming an alternating word automaton with the same acceptance condition into an alternating weak automaton: i) solving a game over an arena A is the same as deciding whether A, seen as a one-letter alternating automaton, is empty; and ii) deciding the emptiness of one-letter alternating weak automata can be done in linear time. The simplicity of weak automata stems from their defining property, the lack of cycles with both accepting and rejecting states.

As a result, the time complexity of solving games is intimately connected to the size blow-up of translating alternating automata to alternating weak automata. Note, however, that the automata-translation question is more general, since automata need not be defined over a one-letter alphabet, and often a binary, or larger, alphabet adds substantial complexity.

Nevertheless, until recently, the best known algorithms for the two problems, with respect to the Büchi and parity conditions, were the same. For Büchi, they involved a quadratic time

[1] Research supported by the Israel Science Foundation grant 1373/16.
[2] Research supported by BMBF project no. 01IS160253 "ARAMiS II".

38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018).
Editors: Sumit Ganguly and Paritosh Pandya; Article No. 21; pp. 21:1–21:22

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

or size blow-up, and for parity an exponential one [13, 12]. Moreover, some competitive tools for solving parity games are based on the translation from parity to weak automata [23].

About a year ago the picture changed; After many years of incremental progress, two quasi-polynomial algorithms using different techniques for solving parity games were published [5, 8]. So far this breakthrough has not extended to automata-translation.

Recently, a third quasi-polynomial algorithm for solving parity games was published, employing yet another technique [14]. We extend this algorithm for addressing the more general problem, and provide a translation of alternating parity word automata into weak automata that involves only a quasi-polynomial size blow-up.

K. Lehtinen defines in [14] *register games*, which are a parameterised variant of parity games, and bases the new algorithm on them. A central result in the complexity analysis of this algorithm is that Eve wins a parity game with n positions if and only if she wins the corresponding k-register game for every $k > \log n$. The least such k is called the *register-index* of the game. We extend this result, showing that the register-index is also logarithmic in a more refined measure of game size: the maximal number of distinct strongly connected components in it. We call this measure the *ssc-size* of the game.

We link the automata setting to the game setting by defining for every alternating parity word automaton $\mathcal{A}$ and positive integer k, a parameterised alternating parity word automaton $\mathcal{A}_k$, such that $\mathcal{A}_k$ accepts an ultimately periodic word w if and only if Eve wins the k-register game on the arena of the model-checking game over $\mathcal{A}$ and w. When $\mathcal{A}$ has n states, $\mathcal{A}_k$ has $kn^{O(k)}$ states and $O(k)$ priorities. Using our logarithmic bound of k in the scc-size of games, we show that $\mathcal{A}$ and $\mathcal{A}_k$ are equivalent for $k > \log n$. Then, applying the standard $O(m^d)$ transformation into weak automata [12] (where m is the number of states in the automaton and d the number of its priorities) to $\mathcal{A}_{1+\log n}$ rather than to the original $\mathcal{A}$, we get a translation with a quasi-polynomial blow-up.

For tree automata, the picture is very different. While every alternating parity word automaton can be translated into a weak one, this is not the case with tree automata. In fact there is a strict expressiveness hierarchy of parity tree automata, defined by the automaton's *index*, that is, the number of its priorities [4]. It is known as the Mostowski–Rabin hierarchy in the automata-theoretic literature, and as the alternation hierarchy in the μ-calculus literature. The decidability of whether a given language is expressible in some level of the hierarchy, and specifically by an alternating weak automaton, is open. Here we show that even when a language is recognised by an alternating weak automaton, this automaton may be exponentially larger than an equivalent parity, or even a co-Büchi automaton.

Analogously to the parity hierarchy, there is a hierarchy of weak automata, defined by the number of alternations between accepting and rejecting states. Like the Mostowski–Rabin hierarchy, it also collapses in the word setting [16, 9] and is infinite in the tree setting [19, 21]. So far, its strictness has only been shown for ranked (directed/ordered) trees.

In ranked trees, each child of a node is distinguished by its unique direction – `left` and `right` for binary trees. In unranked (undirected/unordered) trees, which are more common when talking of Kripke structures, and modal or temporal logics, this is not the case. An unranked tree automaton (also known as a symmetric tree automaton) can only require that there exists a child ($\Diamond$) with some property and that all children ($\Box$) have some property. It thus cannot recognise properties such as "there are two distinct children that satisfy p".

We extend the proof of the strictness of the weak hierarchy to alternating automata that run on unranked trees. Our proof combines the technique used for ranked trees in [21] and a weak version of the languages known to be strict for the parity hierarchy [7, 3]. We show

that the language of unranked trees that represent co-Büchi games in which Eve wins with a strategy that sees at most n alternations between accepting and rejecting positions is only recognised by alternating weak automata of level at least n in the hierarchy.

We use the strictness of the weak hierarchy as an intermediate step in showing that there is at least an exponential size blow-up in translating a universal co-Büchi tree automaton into an alternating weak one. We define a language of unranked trees recognised by an automaton of the former type of size in $O(n)$, and show that it is strict for the $(n+1)2^n$ level of the weak hierarchy. This language combines the aforementioned language of trees representing games in which Eve wins in a restricted way, and explicit counting with n binary bits.

The lower bound we present on unranked tree automata also holds for ranked tree automata: the tree languages we work with can easily be translated into languages of ranked binary trees, and the automata that we construct, or argue that do not exist, can be adapted accordingly. Indeed, the trees we build in the proofs are already binary; and the automata that operate on them can be turned into automata on ranked binary trees by transforming $\Box q$ in transition conditions into $(\texttt{left}, q) \wedge (\texttt{right}, q)$ and $\Diamond q$ into $(\texttt{left}, q) \vee (\texttt{right}, q)$.

Related Work. Over words, the best known upper bound for the size blow-up involved in translating alternating parity to alternating weak automata is exponential [12]. The known lower bound, $\Omega(n \log n)$, is the lower bound for translating alternating Büchi into weak automata [13]. It is directly linked to the $2^{\Omega(n \log n)}$ lower bound in determinizing nondeterministic Büchi automata [18]. How to use the power of the parity condition and the limitations of alternation (as opposed to concurrency) to get a better lower bound is open.

Over trees, little is known about the decidability of the weak definability of languages recognised by alternating parity automata. It is known that the intersection of Büchi and co-Büchi definable languages is weakly definable [11]. Weak definability is decidable for tree languages recognised by alternating Büchi automata, as was first shown for ranked trees by Colcombet et al [6] via reduction to cost automata. Skrzypczak and Walukiewicz [22] provide a topological characterisation and exponential decision procedure, which was later extended to unranked trees [15]. It seems that a singly-exponential translation of universal co-Büchi to alternating weak tree automata can be extracted from these procedures to match our lower bound; however it is a non trivial procedure that is yet to be verified.

Due to space constraints, some of the proofs are omitted and appear in the appendix.

2 Preliminaries

Alphabets, words, and trees. An alphabet is an arbitrary finite and nonempty set, usually denoted by Σ. We also use two specific alphabets: $\Sigma_G = \{E_0, A_0, E_1, A_1\}$, to which we refer as the *game alphabet* and $\Sigma_B = \{\texttt{0}, \texttt{1}, \$\}$, to which we refer as the *binary-counting alphabet*.

A *word* over Σ is a (possibly infinite) sequence $w = w_0 \cdot w_1 \cdots$ of letters in Σ. We write $\mathsf{suffixes}(w)$ for the set of suffixes of w (which includes w itself). We consider a *tree* to be an unranked infinite rooted tree in the graph-theoretic sense. A Σ-*tree* is a tree together with a mapping of each of its nodes to a letter in Σ.

For a word or tree language L over an alphabet Σ, we denote by $\overline{L}$ its complement, namely the set of words over Σ or Σ-labeled trees that are not in L.

For natural numbers i and j, we write $[i..j]$ for the set of natural numbers between them, including i and j. For a set Q, we denote by $\mathsf{B}^+(Q)$ the set of positive boolean formulas over the atomic propositions $Q \cup \{\texttt{true}, \texttt{false}\}$.

Automata. Several definitions of alternating tree automata exist. The differences relate to how flexible the transition condition is with respect to ϵ-transitions and the combination of path quantifiers $(\Diamond, \Box)$ and boolean connectives $(\vee, \wedge)$. Usually, all of these definitions give the same expressiveness ([24, Proposition 1] and [10, Remark 9.4]), except for the case of very restricted automata, in which they do not [2]. In our definition of alternating automata, there are no epsilon transitions and path quantifiers are applied directly on states.

An *alternating tree automaton* is a tuple $\langle \Sigma, Q, \iota, \delta, \Omega \rangle$ where Σ is a finite alphabet, Q is a finite set of states, $\iota \in Q$ is the initial state, $\delta : Q \times \Sigma \to \mathsf{B}^+(\{\Diamond, \Box\} \times Q)$ is the transition function, and Ω is the acceptance condition, on which we further elaborate below.

Intuitively, given a state $q \in Q$ and a letter $\sigma \in \Sigma$, the transition function returns a positive boolean formula that defines which states the automaton should transition to, and whether to consider the next state at one non-deterministically chosen child $(\Diamond)$, or at all of the children $(\Box)$. Positive boolean formulas over $\{\Diamond, \Box\} \times Q$ are called *transition conditions*.

Formally, a *run* of an automaton with states Q over a Σ-tree t is a $(Q \times \Sigma)$-tree r that assigns states to nodes of t along the transition function of $\mathcal{A}$. That is, there is a binary relation ρ that relates nodes of t and nodes of r and satisfies the following constraints.

- For every pair $(n, n') \in \rho$, n is a node of t, n' is a node of r, and if n is labeled σ then n' is labeled $(\cdot, \sigma)$. (The $\cdot$ stands for an arbitrary value.) For every node n' of r, there is exactly one node n of t, such that $(n, n') \in \rho$.
- The roots of t and of r appear in exactly one pair, together, and r's root is labeled $(\iota, \cdot)$.
- For a node n of t with parent p, and a node n' of r with parent p', if $(n, n') \in \rho$ then $(p, p') \in \rho$.
- Consider a node n of t, and a node n' of r labeled (q, σ), such that $(n, n') \in \rho$. Let $\Phi = \delta(q, \sigma)$ be the transition condition of the state q over the letter σ. Then Φ should be satisfied by ρ as inductively defined below.
  - If $\Phi = \Diamond h$ then there exists a child c of n and a child c' of n', such that $(c, c') \in \rho$ and c' is labeled $(h, \cdot)$.
  - If $\Phi = \Box h$ then for every child c of n, there is a child c' of n', such that $(c, c') \in \rho$ and c' is labeled $(h, \cdot)$.
  - If $\Phi = b_1 \vee b_2$ (resp. $\Phi = b_1 \wedge b_2$), for transition conditions b_1 and b_2, then b_1 or b_2 (resp. b_1 and b_2) should be satisfied.

A run is *accepting* if each of its paths satisfies the acceptance condition Ω or ends with `true`. There exist various acceptance conditions in the literature; We use the following.

- *Büchi* (resp. *co-Büchi*), where $\Omega \subseteq Q$ is the set of *accepting states*, and a path is accepting if some state (resp. all states) that it visits infinitely often are in Ω.
- A Büchi (and a co-Büchi) automaton is *weak* if every strongly connected component in the transition graph consists of either only accepting states or only rejecting states.
- *Parity*, where $\Omega : Q \to I$ is a *priority function* that assigns to each state a priority from a set $I = [0..i]$ or $I = [1..i]$, for some $i \in \mathbb{N}$. A path is accepting if the maximal priority seen infinitely often in it is even.

Note that the weak condition is a special case of both the Büchi and co-Büchi conditions, which are dual and are both special cases of the parity condition.

An automaton $\mathcal{A}$ *accepts* a tree if it has an accepting run on it; the language that it *recognises*, denoted by $L(\mathcal{A})$, is the set of trees that it accepts. Two automata that recognise the same language are *equivalent*.

The *size* of an automaton is the maximum of the alphabet length, the number of states, the number of subformulas in the transition function, and the acceptance condition's *index*, which is 1 for Büchi and co-Büchi, and $|I|$ for parity. Observe that in alternating automata,

the difference between the size of an automaton and the number of states in it can stem from a transition function that has exponentially many subformulas. (In stronger acceptance conditions, not considered here, the index might also be exponential in the number of states.)

Nondeterminism in tree automata also varies in the literature. In general, it only concerns the boolean connectives of the transition condition and not the path quantifiers (or directions, in ranked trees). We consider an alternating automaton to be *nondeterministic* (resp. *universal*) if its transition conditions only use the $\vee$ (resp. $\wedge$) connective, in addition to the path quantifiers $\Diamond$ and $\Box$.

Word automata are defined exactly as tree automata, without the path quantifiers $\Diamond$ and $\Box$. Accordingly, a run of a word automaton is a sequence of states.

The *class* of an automaton characterises its transition mode (deterministic, nondeterministic, or alternating), its acceptance condition, and whether it runs on words or trees. We often abbreviate automata classes by acronyms in $\{\text{D, N, A}\} \times \{\text{W, B, C, P}\} \times \{\text{W, T}\}$. The first letter stands for the transition mode; the second for the acceptance-condition (weak, Büchi, co-Büchi, and parity); and the third indicates whether the automaton runs on Words or on Trees. For example, AWW stands for an alternating weak automaton on words.

It is known that AWWs recognise all ω-regular word languages [16], while AWTs do *not*: they have the same expressiveness as alternation-free μ-calculus (AFMC) [20].

Games. A *parity game* is an infinite-duration path-forming game, played between Eve and her opponent Adam on a game graph $G = \langle V, V_e, V_a, E, \Omega\rangle$ called the *arena*. The positions V of the arena are partitioned into those belonging to Eve, V_e, and those belonging to Adam, V_a. We assume that every position has at least one successor. The *priority assignment* $\Omega \to I$ maps every position to a *priority* in I, a finite prefix of the non-negative integers. Starting at some position of G, a *play* proceeds with the owner of the current position choosing the next position among its successors in the directed edge relation $E \subseteq V \times V$. The players collaboratively form a play, consisting of an infinite path along the edges of the game graph. Eve wins if the highest priority visited infinitely often is even, and Adam wins if it is odd.

A *co-Büchi game* is a parity game, in which the set of priorities is $I = \{0, 1\}$. (The positions with priority 0 are *accepting* and those with priority 1 are *rejecting*.)

We shall view a Σ_G-tree t also as a co-Büchi game, where nodes labelled E_i and A_i, for $i \in \{0, 1\}$, are interpreted as Eve's and Adam's positions respectively, and have priority i. If not stated differently, we assume that the game starts at the root of t.

A (positional) *strategy* σ for a player $P \in \{\text{Adam,Eve}\}$ maps every position v belonging to P to one of its successors $\sigma(v)$. A play $\pi = v_0 v_1 \dots$ is said to agree with σ when for all i, if v_i belongs to P, then v_{i+1} is $\sigma(v_i)$. A strategy σ for player P is said to be winning for P from a region $W \subseteq V$ if all plays starting within W that agree with σ are winning for P.

▶ **Proposition 1** (Positional determinacy [7]). *Parity games are positionally determined: at each position, either Adam or Eve has a positional winning strategy.*

▶ **Definition 2** (Model-checking game). Given a word $w \in \Sigma^\omega$ and an APW $\mathcal{A} = \langle \Sigma, Q, \iota, \delta, \Omega\rangle$, the model-checking game $\mathcal{G}(w, \mathcal{A})$ is the following parity game:

- Positions are $\mathsf{B}^+(Q) \times \mathsf{suffixes}(w)$.
- For $a \in \Sigma$, $u \in \Sigma^\omega$ and transition conditions b and b', there is an edge from:
  - $(b \wedge b', u)$ to (b, u) and (b', u)
  - (q, au) to $(\delta(q, a), u)$
  - $(\texttt{true}, au)$ to $(\texttt{true}, u)$
  - $(b \vee b', u)$ to (b, u) and (b', u)
  - $(\texttt{false}, au)$ to $(\texttt{false}, u)$.

- Positions $(b \wedge b', u)$ belong to Adam, other positions belong to Eve.
- A position (b, u) is of priority $\Omega(b)$ if b is a state, 1 if $b = \texttt{false}$, and 0 otherwise.

Note that for ultimately periodic words w, the game $\mathcal{G}(w, A)$ has finitely many positions.

▶ **Proposition 3.** *An APW $\mathcal{A}$ with initial state ι accepts a word w if and only if Eve has a winning strategy in the model-checking game $\mathcal{G}(w, \mathcal{A})$ from (ι, w).*

3 The Weak Hierarchy

A weak automaton can be categorised by the maximal number of alternations[3] between accepting and rejecting states in it [19]. This is a strict hierarchy over ranked trees [19, 21], whereas over words it collapses [16, 9]. We clarify the level to which it collapses over words, and extend the strictness result to the setting of unranked trees. We will use this hierarchy to show the lower bound for translating alternating parity to alternating weak tree automata.

Formally, to define its level, we consider a weak automaton as a parity automaton in which whenever a state q is reachable from a state q', $\Omega(q) \leq \Omega(q')$, *i.e.* the parities seen on any path of the automaton are non-increasing. In such parity automata, there are no cycles with both an even and an odd priority, and therefore all even priorities can be replaced with 0 and odd priorities with 1. This definition therefore coincides with the definition of a weak automaton as a special case of a Büchi or co-Büchi automaton, as given in Section 2, by interpreting states of even priority as accepting and odd priority as rejecting. Then, for every $n \in \mathbb{N}$, a $(0, n)$-*AWT* (resp. $(1, n+1)$-*AWT*) is a weak automaton with priorities (ranks) within $[0..n]$ (resp. $[1..n+1]$). Notice that every AWT can also have transitions to `true` and `false`, which "do not count" in the ranking

The classes of $(0, n)$- and $(1, n+1)$-AWTs form the *weak hierarchy*. Notice that for every $n \in \mathbb{N}$, an automaton is a $(0, n)$-AWT iff its dual is a $(1, n+1)$-AWT, and the class of $(1, n+1)$-AWTs is contained in the class of $(0, n+1)$-AWTs.

The weak hierarchy over words is defined analogously for AWWs rather than AWTs.

3.1 Word Automata: Collapse of the Hierarchy

AWWs recognise all ω-regular word languages: there is a translation of deterministic Muller word automata into AWWs $\mathcal{A}$ [16]. A close look at the construction reveals that $\mathcal{A}$ is in the $(1, 3)$ class. We now show that this is the first class that recognises all ω-regular languages.

▶ **Theorem 4.** *Every ω-regular word language is recognised by some $(0, 2)$-AWW, and there are ω-regular word languages not recognised by any $(0, 1)$-AWW.*

One direction follows from the translation in [16]; for the other direction we show that the Büchi condition, i.e. the language of words with infinitely many repetitions of a fixed letter, is not recognised by any $(0, 1)$-AWW.

3.2 Tree Automata: Strict Hierarchy for Ranked and Unranked Trees

We extend the result on the strictness of the weak hierarchy to the setting of unranked trees. Our proof combines the technique used for ranked trees in [21] and a weak version of the cannonical example of a family of languages known to exhaust the parity hierarchy [7, 3].

[3] This sort of "alternation" has nothing to do with the "alternation" of alternating automata, which refers to having both nondeterminism and universality in the transition condition.

For $n \in \mathbb{N}$, let Win_n be the language of trees over the game alphabet Σ_G in which, when viewed as co-Büchi games, Eve wins with up to n alternations between 0 and 1 nodes. That is, Eve has a winning strategy σ, such that every path of the tree that agrees with σ has up to n transitions from a node labeled E_x/A_x to a node labeled E_y/A_y, for $x \neq y \in \{0, 1\}$.

We start with the positive direction, showing that Win_n is recognised by a $(0, n)$-AWT. The $(0, n)$-AWT is simply the standard automaton to recognise a winning strategy for Eve, duplicated to count alternations.

▶ **Lemma 5.** *For every $n \in \mathbb{N}$, Win_n is recognised by a $(0, n)$-AWT.*

We continue with the negative direction, showing that the complement of Win_n is not recognised by a $(0, n)$-AWT.

▶ **Lemma 6.** *For every $n \in \mathbb{N}$, $\overline{\mathsf{Win}_n}$ is not recognised by a $(0, n)$-AWT.*

The proof consists of an induction on n. For the inductive case we assume, towards a contradiction, that a $(0, n + 1)$-AWT $\mathcal{A}$ recognises $\overline{\mathsf{Win}_{n+1}}$. The key to this proof is that when $n + 1$ is odd, there must be some depth after which an accepting run does not visit states of priority n. We can then use states of lower priority to build a $(0, n)$-AWT that recognises $\overline{\mathsf{Win}_n}$, reaching a contradiction. For odd n, we look at the dual $(1, n + 2)$-AWT in which $n + 2$ is odd and follow the analogous reasoning.

▶ **Theorem 7.** *The weak hierarchy is strict for alternating weak tree automata over unranked trees. That is, for every integer n, there is a language of unranked trees that is recognised by a $(0, n + 1)$-AWT and not by any $(0, n)$-AWT.*

Proof. The family of languages $\overline{\mathsf{Win}_n}$ fits the bill. By Lemma 5, for every $n \in \mathbb{N}$, Win_n is recognised by a $(0, n)$-AWT, implying that $\overline{\mathsf{Win}_n}$ is recognised by a $(1, n + 1)$-AWT, which is in particular a $(0, n + 1)$-AWT. By Lemma 6, $\overline{\mathsf{Win}_n}$ is not recognised by any $(0, n)$-AWT. ◀

4 From Alternating Parity to Alternating Weak Word Automata

We present a quasi-polynomial transformation from APW to AWW, based on the idea of register games [14]. These were developed as an automata-theoretic method for solving parity games in quasi-polynomial time; here we show that this approach is more general and can be used to turn APW into AWW with a quasi-polynomial state and size blow-up.

Register games are a parameterised variant of parity games, also played on a parity game arena. We define for any APW $\mathcal{A}$ an APW $\mathcal{A}_k$ that accepts an ultimately periodic word w if and only if Eve wins the k-register game on the arena of the model-checking game $\mathcal{G}(w, \mathcal{A})$. When $\mathcal{A}$ has n states, $\mathcal{A}_k$ has $O(kn^{k+1})$ states and $2k + 1$ priorities. We then show that $\mathcal{A}$ and $\mathcal{A}_k$ are equivalent for $k = 1 + \log n$. Then, applying the standard $O(m^d)$ transformation into weak automata [12], where m is the number of states in the automaton and d the number of priorities in it, however to $\mathcal{A}_{1+\log n}$ rather than to $\mathcal{A}$, we get a translation with a quasi-polynomial blow-up.

A similar line of reasoning does not work on trees because the register-index of the model-checking game between a tree and an APT depends on both the tree and the automaton.

▶ **Definition 8** (Register game [14])**.** For a strictly positive integer k, a *k-register game* consists of a normal parity game, augmented with k registers. Each register records the highest priority that has occurred in the parity game since it was last reset. The registers are ranked according to how long it has been since their last reset, with a newly reset register having rank 1. Eve is given control of the registers: Before every move in the parity game,

Eve can choose to reset a register of any rank r. If the register contains the priority p, this produces output $2r$ if p is even and $2r+1$ otherwise. As long as Eve resets registers infinitely often, this produces an infinite sequence in $[1..2k+1]^\omega$. To win, Eve has to produce an infinite sequence of outputs such that the maximal value output infinitely often is even. A play in a register game can begin at any position and register configuration (even having registers with an integer bigger than the maximal priority of the parity game); Note that the initial register configuration does not affect the number of registers needed. Unless stated otherwise, we assume all the registers to be initialised to 0.

We call a k-tuple $\bar{x}$ of priorities a *register configuration*, where x_i is the content of the register of rank i. The *top register* is the register of rank k.

▶ **Definition 9** (Register-index [14])**.** The *register-index* of a parity game G from a position v is the least k such that Eve wins either both the k-register game and the parity game on G from v or both the k-register game and the parity game on the dual of G from v (i.e. priorities are shifted by 1 and node ownership is inverted). The register-index always exists [14]. The register-index of a region W winning for Eve is the least k such that Eve wins the k-register game from *any* position in W. The register-index of G is the maximal register-index over all its positions.

We now define for every APW $\mathcal{A}$ its parameterised version $\mathcal{A}_k$, which is an APW that will be shown to accept a word w if and only if Eve wins the k-register game on $\mathcal{G}(w,\mathcal{A})$ starting from (ι, w). The idea is to emulate the k-register game by keeping track of register configurations with a tuple $\bar{x} \in I^k$ that is updated according to which priorities are seen and Eve's resetting choices, which are represented as nondeterministic choices in $\mathcal{A}_k$. The outputs from resets are captured by the priorities of the states of $\mathcal{A}_k$. Here we note a slight subtlety: In the k-register game on $\mathcal{G}(w,\mathcal{A})$, Eve can reset not only at positions (q,u) where q is a state of $\mathcal{A}$, but also at positions (b,u) where b is a boolean formula. In $\mathcal{A}_k$ we aggregate the outputs from all resets between two states (by taking the largest among them) into the priority of the next state – this is the third element $p \in [1..2k+1]$ of the states of $\mathcal{A}_k$. When Eve does not reset, we use the priority 1 so that if Eve does not reset infinitely often, she loses the game, as required.

▶ **Definition 10.** Given an APW $\mathcal{A} = \langle \Sigma, Q, \iota, \delta, \Omega\rangle$ with $\Omega : Q \to I$ and a strictly positive integer k, we define an APW $\mathcal{A}_k = \langle \Sigma, Q', \iota', \delta', \Omega'\rangle$ as follows:

- $Q' = Q \times I^k \times [1..2k+1]$
- $\iota' = (\iota, (0,..,0), 1)$
- Ω': For every $q \in Q, \bar{x} \in I^k$, and $p \in [1..2k+1]$, we have $\Omega'(q,\bar{x},p) = p$.
- δ': For every $q \in Q, \bar{y} \in I^k, p \in [1..2k+1]$, and $a \in \Sigma$, we have $\delta'((q,\bar{y},p),a) = \mathsf{move}(\delta(q,a),\bar{y},1) \vee \mathsf{reset}(\delta(q,a),\bar{y},1)$, where for every $q' \in Q, \bar{x} \in I^k, p' \in [1..2k+1]$, and transition conditions b and b':
  - $\mathsf{move}(q',\bar{x},p') = (q',\bar{x}',p')$, with $x'_i = \max(x_i, \Omega(q'))$.
  - $\mathsf{move}(b \wedge b',\bar{x},p') = (\mathsf{move}(b,\bar{x},p') \vee \mathsf{reset}(b,\bar{x},p')) \wedge (\mathsf{move}(b',\bar{x},p') \vee \mathsf{reset}(b',\bar{x},p'))$
  - $\mathsf{move}(b \vee b',\bar{x},p') = (\mathsf{move}(b,\bar{x},p') \vee \mathsf{reset}(b,\bar{x},p')) \vee (\mathsf{move}(b',\bar{x},p') \vee \mathsf{reset}(b',\bar{x},p'))$
  - $\mathsf{reset}(b,\bar{x},p') = \bigvee_{i\in[1..k]} \mathsf{reset}_i(b,\bar{x},p')$, where $\mathsf{reset}_i(b,\bar{x},p') = \mathsf{move}(b,\bar{x}',p'')$ with
    - $x'_j = x_j$ for $j > i$; $x'_j = x_{j-1}$ for $j \le i, j > 1$; $x'_1 = 0$
    - p'' is $\max(2i,p')$ if x_i is even; and $\max(2i+1,p')$ otherwise.

▶ **Lemma 11.** *Given an APW $\mathcal{A}$, the APW $\mathcal{A}_k$ accepts a word w if and only if Eve wins the k-register game on $\mathcal{G}(w,\mathcal{A})$ from (ι, w).*

The register-index of a parity game G is logarithmically bounded in the number of positions in G [14]. However, the number of positions of $\mathcal{G}(w, \mathcal{A})$ depend on both w and $\mathcal{A}$. We introduce a new measure of game size that we use to logarithmically bound the register-index, but which in $\mathcal{G}(w, \mathcal{A})$ only depends on $\mathcal{A}$.

▶ **Definition 12.** For a game G, let S_G be a maximal set of distinct (not necessarily maximal) strongly connected components in G. Call the size of S_G the *scc-size* of G.

The proof that the register-index of a game is logarithmic in its scc-size is similar to the proof that it is logarithmic in the number of positions of the game [14], while strengthening it by showing that: i) every game of scc-size 1 has register-index 1, and ii) as the register-index increases, the scc-size, rather than just the number of positions, has to double.

We use slightly stronger strategies for Eve: instead of just winning, we require her not to reset the top register when it contains an odd priority. This allows us to compose strategies.

▶ **Definition 13** (Defensive register-index [14])**.** A winning strategy for Eve in a register game on G is *defensive* if, whenever the strategy is played from a position in the winning region of G and a register-configuration in which the top register contains an even priority no smaller than the largest priority in G, the play never outputs $2k + 1$.

The *defensive register-index* of a winning region for Eve of a parity game is the lowest integer k such that Eve has a defensive winning strategy. Observe that the defensive register-index is trivially at most one larger than the register-index.

The proof that the defensive register-index is logarithmic in the scc-size is different for arbitrary scc-size and scc-size 1; we consider the two cases separately.

▶ **Lemma 14.** *A parity game with scc-size* 1 *has defensive register-index* 1*.*

Proof sketch. We prove by induction on the number of positions in an arena G that Eve has a defensive strategy in the 1-register game on G, such that if the initial value of the register is 0 then a play that agrees with the strategy outputs only 2's. The base case is trivial.

In the induction step, we consider the maximal strongly connected components consisting of vertices with priority smaller than the maximal priority p. Observe that there is up to one such component, denoted by G_s. If it does not exist, Eve's strategy is to reset whenever p is seen. If G_s does exist, Eve's strategy is to reset the register whenever entering G_s, and within G_s to follow the strategy that is guaranteed by the induction assumption. The correctness builds on the fact that in an arena with scc-size 1, all cycles intersect. ◀

▶ **Lemma 15.** *The register-index k of a parity game of scc-size z is at most $1 + \log z$.*

Proof. From the definition of register-index, it suffices to consider the single-player parity games G induced by any winning strategy for Eve in her winning region. Observe that in the register games on G, Eve's strategy consists of just choosing when to reset registers.

We proceed by induction on the number of positions n in G. The base case, $n = 1$, is trivial. For the inductive step, let G' be the game induced by positions of G of priority up to $p - 2$, where p is the maximal even priority that appears in some cycle of G. Then, let $G_1, \ldots, G_j$ be the maximal strongly connected subgames of G'. Let $k_1, \ldots, k_j$ be their respective defensive register-indices, and k_m the maximal among these. If there are no such subgames, then all cycles in G contain p in which case Eve wins defensively the 1-register game on G by resetting whenever p is seen.

Case of a unique i for which $k_i = k_m$, and $k_m > 1$: We show that the register-index of G is no more than k_m. Since the scc-size of G is at least that of G_i, and using the induction hypothesis, this suffices.

Eve's strategy in the k_m-register game on G is as follows: she first resets any odd priorities in the registers (this may take several turns); then, after this clean-up phase, she resets k_m whenever p occurs; within a subgame G_j she uses the bottom ranking k_j registers to simulate her defensive winning strategy in the k_j-register game on G_j.

Assume that this strategy is played from a register-configuration where the top register contains an even priority no smaller than p. First observe that since $k_m > 1$, after seeing p and resetting k_m, the top register still contains an even value no smaller than p. Furthermore, since this strategy encounters p between any two entrances into G_i, it always enters G_i with an even value no smaller than p in the top register. Thus, it never outputs $2k_m + 1$. It leaves Adam the choice of losing within a subgame G_j (where Eve follows a winning strategy) or changing subgames infinitely often. In the latter case, p is seen infinitely often, thus producing $2k_m$ as output infinitely often. It is therefore winning and defensive for Eve.

If it is played from a register-configuration in which the top register is not an even priority no smaller than p, then it might output $2k_m + 1$, but the values output infinitely often will still be the same as above, so the strategy is still winning and defensive.

Case of i, j where $i \neq j$ and $k_i = k_j = k_m$: We show that the register-index of G is no more than $k_m + 1$. This suffices, since by the induction hypothesis, each of G_i and G_j has scc-size at least $2^{k_m - 1}$; then G has scc-size at least 2^{k_m}.

Eve's strategy in the $k_m + 1$-register game on G is as follows: reset the register of rank $k_m + 1$ whenever p is seen; in a subgame G_i, use the bottom ranking k_i registers to simulate a winning strategy in the k_i-register game on G. As above, this strategy is winning, and since it only resets the register of rank $k_m + 1$ after seeing p, it is defensive.

Case of $k_m = 1$: If the scc-size of G is 1 then the result follows from Lemma 14. Otherwise, Eve can win the 2-register game on G, using a strategy as above: within a subgame, she uses her defensive 1-register strategy, and resets the top register whenever p is seen. This strategy is winning since a play either remains in a subgame and follows a winning strategy, or sees p infinitely often and therefore outputs 4 infinitely often, but does not output 5 infinitely often. It is therefore winning. If initially the top register contains an even priority no smaller than p, this strategy never outputs 5 and is therefore defensive. ◀

We now show that the scc-size of $\mathcal{G}(w, \mathcal{A})$, for an ultimately periodic word w, is independent of w and logarithmic in $\mathcal{A}$. It basically follows from Lemma 15 and the fact that w is represented by a Kripke structure with a single cycle.

▶ **Lemma 16.** *Given an ultimately periodic word $w = uc^\omega$ and an APW $\mathcal{A}$ with n states, the parity game $\mathcal{G}(w, \mathcal{A})$ has register-index at most $1 + \log n$.*

Then $\mathcal{A}$ is equivalent to its $1 + \log n$ parameterised version; the main result follows by applying the existing transformation – exponential in the number of priorities – to $\mathcal{A}_{1+\log n}$.

▶ **Lemma 17.** *Every APW A is equivalent to its parameterised version A_k, for $k = 1 + \log n$.*

▶ **Theorem 18.** *The size blow-up and state blow-up involved in translating alternating parity word automata to alternating weak word automata is at most quasi-polynomial. In particular, every APW $\mathcal{A}$ of size (resp. number of states) n is equivalent to an AWW of size (resp. number of states) $2^{O((\log n)^3)}$.*

5 From Universal Co-Büchi to Alternating Weak Tree Automata

As opposed to the word setting, alternating weak tree automata do not recognise all ω-regular tree languages. Furthermore, we show below that in cases that a translation from an alternating parity automaton, or even a universal co-Büchi automaton, is possible, there might be an exponential size blow-up.

We provide a family $\{L_n\}_{n\geq 1}$ of tree languages, such that L_n is only recognised by an AWT with at least 2^n states, while recognised by a UCT of size in $O(n)$.

The languages L_n are defined over the game alphabet combined with the binary-counting alphabet. Intuitively, a tree t belongs to L_n if Eve wins the co-Büchi game with respect to the game alphabet, and every path of t provides a correct prefix of an n-bit binary counter, with respect to the binary-counting alphabet, when only considering nodes that come right after an alternation between E_0/A_0 and E_1/A_1 labelling. Notice that the latter requirement guarantees that there are up to $(n+1)2^n$ such alternations in every path of t. (There are up to 2^n numbers in an n-bit counter, and with the separator \$ every number takes $n+1$ bits.)

Formally, L_n is the language of $(\Sigma_G \times \Sigma_B)$-labeled trees, such that a tree t is in L_n iff it satisfies the following properties:

I. Considering only the Σ_G labelling of t and viewing the tree as a co-Büchi game, Eve wins.

II. The Σ_G-labelling of t's root is E_0 or A_0.

III. For a string π over Σ, a node of π is in the derived string π' if it is labeled by E_x/A_x and its predecessor is labeled by E_y/A_y for $x \neq y$. Then, for every path π of t, the derived string π', when considering only its Σ_B labelling, should be a correct prefix of an n-bit binary counter, where each n bits of `0/1` are separated by \$. For example, `000$001$01` is a correct prefix, while `000$100` and `000$`$\ldots$`$111$000` are not.

Intuitively, i) L_n is recognised by a small UCT that combines three succinct components, each checking one of the three requirements in the definition of L_n; ii) L_n is AWT-recognizable, since the third requirement guarantees boundedly many alternations between accepting and rejecting states; and iii) L_n is only recognised by an AWT with at least $(n+1)2^n$ states, since L_n will be shown to be in that level of the weak alternation hierarchy.

▶ **Lemma 19.** *For every $n \in \mathbb{N}$, L_n is recognised by a UCT of size in $O(n)$.*

Proof. A UCT for L_n can be defined as the conjunction of three UCTs, $\mathcal{A}_I$, $\mathcal{A}_{II}$, and $\mathcal{A}_{III}$, each checking the corresponding requirement in the definition of L_n.

$\mathcal{A}_I$. The UCT $\mathcal{A}_I$ can be defined by straightforwardly relating E_0 and E_1 to accepting and rejecting nondeterministic states, respectively, and A_0 and A_1 to accepting and rejecting universal states, respectively. Formally, $\mathcal{A}_I = \langle \Sigma, Q_I, \iota_I, \delta_I, \alpha_I \rangle$, where

- $Q_I = \{q_0, q_1\}$.
- $\iota_I = q_0$.
- δ_I. For every $q \in Q$ and $b \in \Sigma_B$, we define: $\delta_I(q, (E_0, b)) = \Diamond q_0$; $\delta_I(q, (E_1, b)) = \Diamond q_1$; $\delta_I(q, (A_0, b)) = \Box q_0$; and $\delta_I(q, (A_1, b)) = \Box q_1$.
- $\alpha_I = q_0$.

A winning strategy of Eve on a game-tree t suggests how to resolve the nondeterminism in $\mathcal{A}_I$, while an accepting run of $\mathcal{A}_I$ on t can be translated to a winning strategy for Eve.

$\mathcal{A}_{II}$. Trivially checking the tree root's labeling.

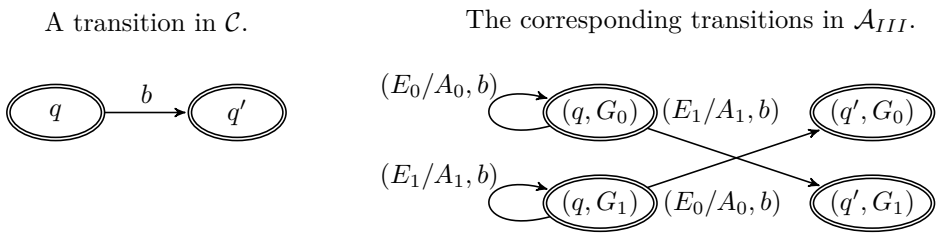


Figure 1 From the universal finite-tree automaton $\mathcal{C}$ to the UCT $\mathcal{A}_{III}$ in the proof of Lemma 19.

$\mathcal{A}_{III}$. The core idea behind the definition of the UCT $\mathcal{A}_{III}$ is that *improper* n-bit binary counting can be recognised by a *nondeterministic* finite-word automaton $\mathcal{A}$ of size in $O(n)$. Furthermore, all states of $\mathcal{A}$ are rejecting, except for a single "forever accepting" state (`true`). See, for example, [1, Figure 2] for a concrete construction of such an automaton.

Accordingly, *proper* n-bit binary counting can be recognised by the dual of $\mathcal{A}$, which is a *universal* finite-word automaton $\mathcal{B}$ of size in $O(n)$. Now, a universal word automaton $\mathcal{A}_w$ for a word language L can be adapted to a universal tree automaton $\mathcal{A}_t$ for the language of trees all of whose paths are in L, by simply adding $\Box$ to every transition in $\mathcal{A}_w$. (Notice that this does not hold for a nondeterministic word automaton.) Hence, we can adapt $\mathcal{B}$ to a universal finite-tree automaton $\mathcal{C}$ that recognises the language of finite trees all of whose paths are a proper prefix of n-bit counter. Notice that all states of $\mathcal{C}$ are accepting, except for a single "forever rejecting" state (`false`).

Now, all that is left for getting $\mathcal{A}_{III}$ is to adapt $\mathcal{C}$ to operate over the extended alphabet and to only consider the binary-counting alphabet when there is an alternation between E_0/A_0 and E_1/A_1 labeling. We do this by having two copies of $\mathcal{C}$, each "remembering" whether the last game-labeling was 0 or 1. This adaptation is illustrated in Figure 1.

Formally, let $\mathcal{C} = \langle \Sigma_B, Q, \iota, \delta, F \rangle$. We define $\mathcal{A}_{III} = \langle \Sigma, Q_{III}, \iota_{III}, \delta_{III}, \alpha_{III} \rangle$, where

- $Q_{III} = q_0 \cup Q \times \{G_0, G_1\}$.
- $\iota_{III} = q_0$.
- δ_{III}.
  - For every $b \in \Sigma_B$, $\delta_{III}(q_0, (E_0/A_0, b)) = \Box(\iota, G_0)$ and $\delta_{III}(q_0, (E_1/A_1, b)) = \Box(\iota, G_1)$.
  - For every $q \in Q$ and $b \in \Sigma_B$, we define:
    - $\delta_{III}((q, G_0), (E_0/A_0, b)) = \Box(q, G_0)$
    - $\delta_{III}((q, G_0), (E_1/A_1, b)) = G_1(\delta(q, b))$, where $G_1(TC)$, for a transition condition TC of $\mathcal{C}$, changes every instance of a state q in TC to (q, G_1). For example, $G_1(\Box q \wedge \Box q') = \Box(q, G_1) \wedge \Box(q', G_1)$.
    - $\delta_{III}((q, G_1), (E_0/A_0, b)) = G_0(\delta(q, b))$, where $G_0(TC)$, for a transition condition TC of $\mathcal{C}$, changes every instance of a state q in TC to (q, G_0).
    - $\delta_{III}((q, G_1), (E_1/A_1, b)) = \Box(q, G_1)$
- $\alpha_{III} = F \times \{G_0, G_1\}$.

Notice that by the special structure of $\mathcal{A}_{III}$, it is not only a UCT and even a $(0, 0)$-UWT. ◀

Since condition III guarantees a bound on the number of alternations between E_0/A_0 and E_1/A_1 labels, the languages $\{L_n\}$ can also be recognised by alternating weak automata.

▶ **Lemma 20.** *For every $n \in \mathbb{N}$, L_n is AWT-recognizable.*

Proof. Analogously to the construction of a UCT for L_n in the proof of Lemma 19, an AWT for L_n can be defined as the conjunction of three AWTs, $\mathcal{B}_I$, $\mathcal{B}_{II}$, and $\mathcal{B}_{III}$, each checking the corresponding requirement in the definition of L_n.

The automata $\mathcal{B}_{II}$ and $\mathcal{B}_{III}$ can be identical to the automata $\mathcal{A}_{II}$ and $\mathcal{A}_{III}$, respectively, from the proof of Lemma 19, as they are already AWTs (and even $(0,0)$-UWTs).

As for $\mathcal{B}_I$, it obviously cannot be identical to $\mathcal{A}_I$ from the proof of Lemma 19, as the latter heavily builds on the co-Büchi acceptance condition. Furthermore, the first requirement in the definition of L_n is not AWT-definable. Yet, due to the third requirement in the definition of L_n, in every tree t that belongs to L_n, all paths have up to $(n+1)2^n$ alternations between E_0/A_0 and E_1/A_1 labelling. Hence, the first requirement in the definition of L_n can be reformulated to "Eve wins t with up to $(n+1)2^n$ alternations between E_0/A_0 and E_1/A_1 nodes", namely to "t belongs to $\mathsf{Win}_{(n+1)2^n}$", without changing L_n.

By Lemma 5, there is an AWT recognizing $\mathsf{Win}_{(n+1)2^n}$, providing the desired AWT $\mathcal{B}_I$. ◀

We establish in two steps the lower bound on the size of an AWT recognizing L_n: i) We prove a claim on the weak-hierarchy levels of a family $\{H_m\}_{m\geq 1}$ of languages that are quite similar to $\{L_n\}$, but lack the explicit counting; and ii) We prove that an AWT for L_n must be in the same level of the weak hierarchy as an AWT for $H_{(n+1)2^n}$.

Formally, H_m is the language of Σ_G-labeled trees, such that a tree t is in H_m iff i) Eve wins t, when viewing t as a co-Büchi game, ii) the root of t is labeled E_0 or A_0 if m is even, and E_1 or A_1 if m is odd, and iii) In every path of t, there are up to m transitions from a node labeled E_x/A_x to a node labeled E_y/A_y, for $x \neq y \in \{0,1\}$.

The proof that H_m is strict for the m^{th} level of the weak hierarchy follows the inductive reasoning of showing that Win_m is in the m^{th} level of the weak hierarchy (Lemma 6), but requires some additional manoeuvres, due to the asymmetry between Adam and Eve in H_m. Specifically, in the induction step, when assuming toward contradiction a $(0,m+1)$-AWT that recognises $\overline{H_{m+1}}$ or a $(1,m+2)$-AWT that recognises H_{m+1}, we not only manipulate subtrees in H_m and $\overline{H_m}$, but also subtrees that are in the conjunction or disjunction of H_m with languages that correspond to the second and third requirements in the definition of H_m.

▶ **Lemma 21.** *For every $m \in \mathbb{N}$, there is no $(0,m)$-AWT recognizing $\overline{H_m}$.*

Proof. We use the following simple AWTs: Let $\mathcal{R}oot0$ (resp. $\mathcal{R}oot1$) be a $(0,0)$-AWT that accepts a tree t iff the root of t is labeled E_0/A_0 (resp. E_1/A_1). For every $m \in \mathbb{N}$, let $\mathcal{A}lt_m$ be a $(0,0)$-AWT that accepts a tree t iff in every path of t, there are up to m transitions from a node labeled E_x/A_x to a node labeled E_y/A_y, for $x \neq y \in \{0,1\}$. Observe that for an even m, a tree $t \in H_m$ iff $t \in L(\mathit{Eve\ wins}) \cap L(\mathcal{R}oot0) \cap L(\mathcal{A}lt_m)$, while t is in $\overline{H_m}$ iff $t \in L(\mathit{Adam\ wins}) \cup L(\mathcal{R}oot1) \cup \overline{L(\mathcal{A}lt_m)}$. We prove the claim by induction on m.

Base case. Recall that there is a $(0,0)$-AWT recognizing $\overline{H_0}$ iff there is a $(1,1)$-AWT recognizing H_0, which is the language of trees in which Eve wins and all paths are labeled A_0/E_0. Let t be the single-path tree labelled A_0 throughout. If a $(1,1)$-AWT $\mathcal{A}$ recognises H_0, it must accept t. Since every loop in $\mathcal{A}$ is rejecting, an accepting run r of $\mathcal{A}$ on t only has finite paths, ending with `true`. Let k be the length of the longest one. Let t' be the tree identical to t up to depth k and labelled A_1 from there on. Then $\mathcal{A}$ accepts t', but $t' \notin H_0$.

Induction step. The induction hypothesis is that $\overline{H_m}$ is not recognised by a $(0,m)$-AWT (and its dual, that H_m is not recognised by a $(1,m+1)$-AWT). There are two cases to consider, an even m and an odd m.

Even m. Assume, towards a contradiction, a $(0, m+1)$-AWT $\mathcal{A}$ that recognises $\overline{H_{m+1}}$. We will build a $(0, m)$-AWT that recognises $\overline{H_m}$, reaching a contradiction.

Consider an arbitrary tree $t' \in L(Adam\ wins) \cap L(\mathcal{R}oot0) \cap L(\mathcal{A}lt_m)$. Let t be the tree consisting of one branch along which all nodes are labelled E_1, and they all have a child labelled E_0 that in turn has a copy of t' as its unique child.

First, we claim that $t \in \overline{H_{m+1}}$: if Eve does not play into a copy of t', the play sees only labels E_1 and is winning for Adam; if Eve plays into a copy of t', then from there she loses.

Now let r be an accepting run of $\mathcal{A}$ on t. As $m+1$ is odd, there is a depth k of t, starting from which r only assigns states of rank at most m. Let $S_{t'}$ be the set of states that r assigns to the root of t' at depth k, and let $\mathcal{A}_{t'}$ be the automaton that is derived from $\mathcal{A}$ by setting $S_{t'}$ to be the initial set, namely having the initial formula $\bigwedge_{q \in S_{t'}} q$ (which can be translated to an initial state). $\mathcal{A}_{t'}$ is a $(0, m)$-AWT, since it lacks the $m+1$-ranked states of $\mathcal{A}$.

Clearly, $\mathcal{A}_{t'}$ accepts t'. Furthermore, $\mathcal{A}_{t'}$ does not accept any tree in H_m: If it accepted some tree $t'' \in H_m$ then $\mathcal{A}$ would also accept the tree $\hat{t}$ derived from t by replacing the occurrence of t' at depth k with t''. However, $\hat{t}$ is not in $\overline{H_{m+1}}$, since Eve wins it (by going to t''), its root is labeled E_0/A_0, and every path of it has up to $m+1$ alternations.

Let $\mathcal{B}$ be the automaton that is the disjunction of all of these $\mathcal{A}_{t'}$ automata (there are finitely many such automata, as each of them corresponds to a subset of $\mathcal{A}$'s states) for $t' \in L(Adam\ wins) \cap L(\mathcal{R}oot0) \cap L(\mathcal{A}lt_m)$, and let $\mathcal{C}$ be the disjunction of $\mathcal{B}$, $\mathcal{R}oot1$, and $\overline{\mathcal{A}lt_m}$. Observe that $\mathcal{C}$ is a $(0, m)$-AWT.

We claim that $\mathcal{C}$ recognises $\overline{H_m}$, leading to the claimed contradiction.

If $t \in \overline{H_m}$, there are two cases: either $t \in L(\mathcal{R}oot1) \cup \overline{L(\mathcal{A}lt_m)}$ or $t \in L(Adam\ wins) \cap L(\mathcal{R}oot0) \cap L(\mathcal{A}lt_m)$. If $t \in L(\mathcal{R}oot1) \cup \overline{L(\mathcal{A}lt_m)}$ then $\mathcal{C}$ clearly accepts t. Moreover, if $t \in L(Adam\ wins) \cap L(\mathcal{R}oot0) \cap L(\mathcal{A}lt_m)$ then $\mathcal{B}$ and therefore $\mathcal{C}$ accepts t. Hence $\overline{H_m} \subseteq L(\mathcal{C})$.

If $t \in H_m$, then $\mathcal{B}$ does not accepts t since no $\mathcal{A}_{t'}$ accepts t. Furthermore, $t \in L(\mathcal{R}oot0) \cap L(\mathcal{A}lt_m)$ so neither $\mathcal{R}oot1$ nor $\overline{\mathcal{A}lt_m}$ accepts t. Hence $\mathcal{C}$ does not accepts t and $L(\mathcal{C}) = \overline{H_m}$.

Odd m. Observe that $\overline{H_{m+1}}$ is recognised by a $(0, m+1)$-AWT if and only if H_{m+1} is recognised by a $(1, m+2)$-AWT. Assume, towards a contradiction, that there is a $(1, m+2)$-AWT $\mathcal{A}$ that recognises H_{m+1}. We will build a $(1, m+1)$-AWT that recognises H_m, reaching a contradiction.

Consider a tree $t' \in H_m$. Let t be the tree consisting of one branch along which all nodes are labelled A_0, and they each have a child labelled A_1 that in turn has a copy of t' as its unique child.

First, we claim that $t \in H_{m+1}$: i) The label of t's root is in $\{E_0, A_0\}$; ii) Since m is odd, the root of t' is labeled E_1/A_1 and thus there are up to $m+1$ alternations in all paths of t; and iii) Eve wins t: if Adam does not play into a copy of t', the play sees only labels A_0 and is winning for Eve; if Adam plays into a copy of t', then from there Eve has a winning strategy.

Now let r be an accepting run of $\mathcal{A}$ on t. Since $m+2$ is odd, there is a depth k starting from which r only assigns states of rank at most $m+1$. Let $S_{t'}$ be the set of states that r assigns to the root of t' at depth k, and let $\mathcal{A}_{t'}$ be the automaton that is derived from $\mathcal{A}$ by setting $S_{t'}$ to be the initial set. Observe that $\mathcal{A}_{t'}$ is a $(1, m+1)$-AWT, since it lacks the $m+2$-ranked states of $\mathcal{A}$.

Obviously, $\mathcal{A}_{t'}$ accepts t'. Furthermore, $\mathcal{A}_{t'}$ does not accept any tree t'' not in H_m, whose root is labeled E_1/A_1: If it accepted such a tree t'' then $\mathcal{A}$ would also accept the tree $\hat{t}$ that is derived from t by replacing the occurrence of t' at depth k with t''. However, $\hat{t}$ is not in

H_{m+1}, since i) if Adam wins t'' then Adam wins $\hat{t}$; and ii) if there exists a path in t'' with more than m alternations then there exists a path in $\hat{t}$ with more than $m+1$ alternations.

Let $\mathcal{B}$ be the automaton that is the disjunction of all of these $\mathcal{A}_{t'}$ automata, and let $\mathcal{C}$ be the automaton that is the conjunction of $\mathcal{B}$ and $\mathcal{R}oot1$. Observe that $\mathcal{C}$ is a $(1, m+1)$-AWT

We claim that $\mathcal{C}$ recognises H_m, leading to the claimed contradiction. If $t \in H_m$, then $\mathcal{B}$ and therefore $\mathcal{C}$ accepts t. If $t \notin H_m$ and the root of t is labeled E_0/A_0 then $\mathcal{R}oot1$ and therefore $\mathcal{C}$ does not accept t. If $t \notin H_m$ and the root of t is labeled E_1/A_1 then no $A_{t'}$ accepts t, and therefore $\mathcal{B}$ and $\mathcal{C}$ do not accept t. Hence $L(\mathcal{C}) = H_m$. ◀

We now show that an AWT $\mathcal{A}$ for L_n cannot be in a lower level of the weak hierarchy than an AWT $\mathcal{A}'$ for $H_{(n+1)2^n}$, and must therefore have at least $(n+1)2^n$ states. The idea is to construct $\mathcal{A}'$ by having $(n+1)2^n$ adapted copies of $\mathcal{A}$, each properly adding the extra letters of the binary-counting alphabet. We get a much bigger automaton than $\mathcal{A}$, yet on the same level of the weak hierarchy, which provides the desired result.

▶ **Lemma 22.** *For every $n \geq 1$, an AWT recognizing L_n must have at least $(n+1)2^n$ states.*

Proof. Let $N = (n+1)2^n$. Consider an AWT $\mathcal{A}$ that recognises L_n, and let m be the minimal natural number such that $\mathcal{A}$ is in the $(0, m)$-AWT class. We will construct from $\mathcal{A}$ a $(0, m)$-AWT $\mathcal{A}'$ that recognises H_N. This will imply the desired result, as by Lemma 21, m must be at least N.

The idea in the construction of $\mathcal{A}'$ is to follow the transitions of $\mathcal{A}$, while properly adding the extra letters of the binary-counting alphabet. In order to add the letters in a way that matches a proper counting, $\mathcal{A}'$ has N copies of $\mathcal{A}$, each corresponding to a number between 0 and $N-1$. The constructed automaton will be much bigger than $\mathcal{A}$, yet on the same level of the weak hierarchy, which provides the desired result.

Formally, let $\mathcal{A} = \langle \Sigma, Q, \iota, \delta, \Omega \rangle$. (For considering the level of $\mathcal{A}$ in the weak hierarchy, we view it as a parity automaton that satisfies the weakness constraint.) For every $i \in [0..N-1]$, let $\mathsf{bit}(i)$ stand for the letter in Σ_B that appears in the ith position of the n-bit counter. For example, for $n = 3$ and $i = 7$, we have $\mathsf{bit}(7) = \$$, since $\$$ is the letter in the 7th position in the counter `000$001$`$\ldots$.

In addition to the proper letter of Σ_B, $\mathcal{A}'$ should also "remember" whether the last alternation was from E_0/A_0 to $E1/A1$ or vice versa. Yet, this can be derived from the counter position, as in even positions of the counter, the last alternation should be to E_0/A_0, and in odd positions, the other way round. We therefore define the function $\mathsf{NextIndex} : \Sigma_G \times [0..N-2] \to [0..N-1]$, by $\mathsf{NextIndex}(g, i) = i+1$ if (i is even and $g \in \{E_1, A_1\}$ or i is odd and $g \in \{E_0, A_0\}$), while in other cases $\mathsf{NextIndex}(g, i) = i$.

We can now define the AWT $\mathcal{A}' = \langle \Sigma_G, Q', \iota', \delta', \Omega' \rangle$, where

- $Q' = Q \times [0..N-1]$.
- $\iota' = (\iota, 0)$.
- δ': For every $q \in Q$, $i \in [0..N-1]$, and $g \in \Sigma_G$, we have $\delta'((q,i), g) = \mathsf{Next}(\delta(q, (g, \mathsf{bit}(i))))$, where $\mathsf{Next}(TC)$, for a transition condition TC of $\mathcal{A}$, changes every instance of a state q in TC to $(q, \mathsf{NextIndex}(g, i))$.
- Ω': For every $q \in Q$ and $i \in [0..N-1]$, $\Omega'(q, i) = \Omega(q)$.

Observe that $\mathcal{A}'$ is in the same level of the weak hierarchy as $\mathcal{A}$, since the priorities and transitions in $\mathcal{A}'$ are the same as in $\mathcal{A}$, except for having a component number, which does not affect the priority of a state.

It is left to show that $\mathcal{A}'$ recognises H_N. Consider a tree $t' \in H_N$. Let t be the Σ-tree that is derived from t' by adding the binary-counting labelling as follows: i) labelling the

root of t with 0; ii) keeping the binary-counting labelling of the parent node if there was no alternation between E_0/A_0 and E_1/A_1 labelling; and iii) labelling the node with the next letter of a proper binary-counter when there is an alternation between E_0/A_0 and E_1/A_1 labelling. Since t' belongs to H_N, t belongs to L_n and there is some accepting run-tree r of $\mathcal{A}$ on t. Recall that r has labels in $Q \times (\Sigma_G \times \Sigma_B)$. Let r' be the $(Q \times \Sigma_B) \times \Sigma_G$-tree that is derived from r by changing the labelling of every node from $(q, (\sigma_G, \sigma_B))$ to $((q, \sigma_B), \sigma_G)$. Observe that r' is an accepting run of $\mathcal{A}'$ on t'.

As for the other direction, consider a tree t' accepted by $\mathcal{A}'$ via some run-tree r'. Recall that r' has labels in $(Q \times \Sigma_B) \times \Sigma_G$. Let r be the $Q \times (\Sigma_G \times \Sigma_B)$-tree that is derived from r' by changing the labelling of every node from $((q, \sigma_B), \sigma_G)$ to $(q, (\sigma_G, \sigma_B))$. Observe that r is an accepting run of $\mathcal{A}$ on some Σ-tree t, such that t is identical to t' except for having additional Σ_B labelling to each node. Hence, $t \in L_n$ and $t' \in H_N$. ◀

Gathering the lemmas above, we get the blow-up involved in the translation.

▶ **Theorem 23.** *The size blow-up and state blow-up involved in translating UCTs to AWTs, when possible, is in* $2^{\Omega(n)}$.

6 Conclusions

Up until the recent quasi-polynomial parity game algorithms, the best known blow-up of turning APW into AWW roughly matched the complexity of known algorithms for solving parity games. Here we have again closed this gap by establishing a quasi-polynomial APW to AWW transformation. This immediately improves the worst-case complexity of parity game solving via reduction to the AWW emptiness problem [23] to quasi-polynomial. While a more effecient APW to AWW transformation will always yield a more efficient parity game solving algorithm, the extent to which the converse holds is an open question.

In stark contrast, our lower bound for transformations to weak tree automata shows that on trees the simplicity of the weak acceptance condition comes at a much higher cost.

References

1 U. Boker and O. Kupferman. Translating to Co-Büchi Made Tight, Unified, and Useful. *ACM Trans. Comput. Log.*, 13(4):29:1–29:26, 2012.

2 U. Boker and Y. Shaulian. Automaton-Based Criteria for Membership in CTL. In *Proceedings of LICS*, pages 155–164, 2018.

3 J. Bradfield. Simplifying the modal mu-calculus alternation hierarchy. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 39–49. Springer, 1998.

4 J.C. Bradfield. The modal mu-calculus alternation hierarchy is strict. *Theoretical Computer Science*, 195(2):133–153, 1998.

5 C. S. Calude, S. Jain, B. Khoussainov, L. Bakhadyr, W. Li, and F. Stephan. Deciding Parity Games in Quasipolynomial Time. In *Proceedings of STOC*, pages 252–263. ACM, 2017.

6 T. Colcombet, D. Kuperberg, C. Löding, and M. Vanden Boom. Deciding the weak definability of Büchi definable tree languages. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 23. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

7 E.A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *Proceeding of FoCS*, pages 368–377. IEEE, 1991.

8 M. Jurdzinski and R. Lazic. Succinct progress measures for solving parity games. In *Proceedings of (LICS)*, pages 1–9, 2017.

9 R. Kaivola. Axiomatising linear time mu-calculus. In *International Conference on Concurrency Theory*, pages 423–437. Springer, 1995.

10 D. Kirsten. *Alternating Tree Automata and Parity Games*, pages 153–167. Springer Berlin Heidelberg, 2002.

11 O. Kupferman and M. Y. Vardi. $\Pi_2 \bigcap \Sigma_2 \equiv$ AFMC. In *Proceedings of ICALP*, pages 697–713. Springer, 2003.

12 O. Kupferman and M.Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proceedings of STOC*, pages 224–233, 1998.

13 O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. *ACM Transactions on Computational Logic*, 2(2):408–429, 2001.

14 K. Lehtinen. A modal μ perspective on solving parity games in quasipolynomial time. In *Proceedings of LICS*, 2018.

15 K. Lehtinen and S. Quickert. Σ_2^μ is decidable for Π_2^μ. In *Conference on Computability in Europe*, pages 292–303. Springer, 2017.

16 P. A. Lindsay. On Alternating omega-Automata. *J. Comput. Syst. Sci.*, 36(1):16–24, 1988.

17 R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and control*, 9(5):521–530, 1966.

18 M. Michel. Complementation is more difficult with automata on infinite words. CNET, Paris, 1988.

19 A.W. Mostowski. Hierarchies of weak automata and weak monadic formulas. *Theoretical Computer Science*, 83(2):323–335, 1991.

20 D.E. Muller and P.E. Schupp. Simulating Alternating tree automata by nondeterministic automata: New results and new proofs of theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141:69–107, 1995.

21 D. Niwiński and I. Walukiewicz. A gap property of deterministic tree languages. *Theoretical Computer Science*, 303(1):215–231, 2003.

22 M. Skrzypczak and I. Walukiewicz. Deciding the topological complexity of Büchi languages. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 55. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

23 A. Di Stasio, A. Murano, G. Perelli, and M. Y. Vardi. Solving parity games using an automata-based algorithm. In *International Conference on Implementation and Application of Automata*, pages 64–76. Springer, 2016.

24 T. Wilke. CTL$^+$ is exponentially more succinct than CTL. In *Proc. of FoSSaCS*, volume 1738 of *LNCS*, pages 110–121. Springer, 1999.

A Appendix – Omitted Proofs

A.1 Proofs of Section 3

▶ **Theorem 4.** *Every ω-regular word language is recognised by some $(0,2)$-AWW, and there are ω-regular word languages not recognised by any $(0,1)$-AWW.*

Proof. In [16], there is a translation of an arbitrary deterministic Muller automaton to a $(1,3)$-AWW. Hence, given an ω-regular language L, one can represent its dual language $\overline{L}$ by a deterministic Muller automaton $\mathcal{A}$ and translate it to an equivalent $(1,3)$-AWW $\mathcal{B}$. The dual of $\mathcal{B}$ is a $(0,2)$-AWW recognizing L.

As for the negative part, consider the language L over the alphabet $\Sigma = \{a,b\}$, consisting of words with infinitely many a's. Observe that the word $w = ababbabbbab^4ab^5\ldots$ is in L.

Assume towards contradiction a $(0,1)$-AWW $\mathcal{A}$ recognizing L, and having a set of states Q. Then, $\mathcal{A}$ accepts w via some run r. Recall that r is a $Q \times \Sigma$-tree. For every $n \in \mathbb{N}$, let S_n be the set of states of $\mathcal{A}$ that appear in the labeling of nodes of r at the level n.

Since Q is finite and the sequences of subsequent b's in w are unbounded, there is a set $S \subseteq Q$ and two numbers $x < y \in \mathbb{N}$, such that there are only b's between the positions x and y of w, $S = S_x = S_y$, and $S = S_n$ for infinitely many $n \in \mathbb{N}$.

As $\mathcal{A}$ is weak and its top rank is rejecting, no state of S can be from the top rank. (Otherwise there is a path in r all of whose nodes are from the top rank, implying that the path is rejecting.) Hence, all states of S are from the 0 rank. Thus, from every node v in the x level of r, there is a run on the finite word b^{y-x}, such that every state that appears in its labeling is from the 0-level of $\mathcal{A}$, and all its leaves are labeled with states from S or `true`.

Let w' be the infinite word that is the same as w up to position x, and from there on has only b's. Then $\mathcal{A}$ has an accepting run r' on w' that is the same as r up to level y, and then for every $i \in \mathbb{N}$ and node v in a $y + i(y - x)$ level that is labeled with some state q, it makes the next $y - x$ steps from v as the run r from a node in its x level that is labeled with q. Yet, $w' \notin L$, leading to a contradiction. ◀

▶ **Lemma 5.** *For every $n \in \mathbb{N}$, Win_n is recognised by a $(0, n)$-AWT.*

Proof. We first define a $(0, n)$-AWT $\mathcal{W}_n$ and then show that it recognises Win_n.
$\mathcal{W}_n$:

- Alphabet: Σ_G (Namely, $\{E_0, E_1, A_0, A_1\}$.)
- States: $\{q_0, q_1, \ldots, q_n\}$
- Initial state: q_n
- Priorities (ranks): $\Omega(q_j) = j$ for all j
- Transitions:
  $\delta(q_0, E_0) = \Diamond q_0$; $\delta(q_0, A_0) = \Box q_0$; $\delta(q_0, E_1/A_1) =$ `false`
  For every $j \in [1..n]$:

$$\delta(q_j, E_0) = \begin{cases} \Diamond q_j & n \text{ is even} \\ \Diamond q_{j-1} & n \text{ is odd} \end{cases}$$

$$\delta(q_j, E_1) = \begin{cases} \Diamond q_j & n \text{ is odd} \\ \Diamond q_{j-1} & n \text{ is even} \end{cases}$$

$$\delta(q_j, A_0) = \begin{cases} \Box q_j & n \text{ is even} \\ \Box q_{j-1} & n \text{ is odd} \end{cases}$$

$$\delta(q_j, A_1) = \begin{cases} \Box q_j & n \text{ is odd} \\ \Box q_{j-1} & n \text{ is even} \end{cases}$$

Observe that $\mathcal{W}_n$ is indeed a $(0, n)$-AWT.

Next, we show that $\mathcal{W}_n$ recognises Win_n. The transition relation guarantees that in every infinite run of $\mathcal{W}_n$ on any tree t, whenever a node v is labeled E_0/A_0 (resp. $E1/A1$), every successor of v is seen at a state of $\mathcal{W}_n$ of even (resp. odd) rank.

Let σ be a winning strategy for Eve in t, such that any play that agrees with σ only sees up to n alternations. We translate σ into an accepting run r_σ of $\mathcal{W}_n$ on t by resolving the nondeterminism of $\mathcal{W}_n$ at a node $v \in t$ labelled E_i along $\sigma(v)$. That is, the choice of a successor node in a $\Diamond$-transition is done along the choice of Eve in σ. Since no play that agrees with σ sees more than n alternations, r_σ does not reach `false`. Furthermore, since every path of t that agrees with σ sees only finitely many nodes labeled E_1/A_1, the run r_σ is accepting.

Conversely, an accepting run r of $\mathcal{W}_n$ on a tree t translates into a strategy σ for Eve on the game t: first observe that r visits up to one state for every node of t, and up to one successor of every node labelled E_0/E_1; Then, at node v labelled E_0/E_1, the strategy σ chooses the unique successor that r visits. Since r is accepting, every path of r can have only finitely many nodes labeled E_1/A_1, implying that every play that agrees with σ sees E_1/A_1 only finitely often. ◀

▶ **Lemma 6.** *For every $n \in \mathbb{N}$, $\overline{\mathsf{Win}_n}$ is not recognised by a $(0,n)$-AWT.*

Proof. We prove the claim by induction on n.

Base case. Recall that there is a $(0,0)$-AWT recognizing $\overline{\mathsf{Win}_0}$ iff there is a $(1,1)$-AWT recognizing Win_0, which is the language of trees in which Eve wins without ever seeing A_1/E_1. Let t be the single-path tree labelled A_0 throughout. If a $(1,1)$-AWT $\mathcal{A}$ recognises Win_0, it must accept t. Since every loop in $\mathcal{A}$ is rejecting, an accepting run r of $\mathcal{A}$ on t only has finite paths, ending with `true`. Let k be the length of the longest one. Let t' be the tree identical to t up to depth k and labelled A_1 from there on. Then $\mathcal{A}$ accepts t', but $t' \notin \mathsf{Win}_0$.

Induction step. The induction hypothesis is that $\overline{\mathsf{Win}_n}$ is not recognised by a $(0,n)$-AWT (and its dual, that Win_n is not recognised by a $(1,n+1)$-AWT). There are two cases to consider, an even n and an odd n.

Even n. Assume, towards a contradiction, a $(0,n+1)$-AWT $\mathcal{A}$ that recognises $\overline{\mathsf{Win}_{n+1}}$. We will build a $(0,n)$-AWT that recognises $\overline{\mathsf{Win}_n}$, reaching a contradiction.

Consider an arbitrary tree $t' \in \overline{\mathsf{Win}_n}$. Let t be the tree consisting of one branch along which all nodes are labelled E_1, and they all have a child labelled E_0 that in turn has a copy of t' as its unique child.

First, we claim that $t \in \overline{\mathsf{Win}_{n+1}}$: if Eve does not play into a copy of t', the play sees only labels E_1 and is winning for Adam; if Eve plays into a copy of t', then from there she either loses or wins but sees over n alternations within t' and therefore over $n+1$ alternations overall.

Now let r be an accepting run of $\mathcal{A}$ on t. Since $n+1$ is odd, there is a depth k of t, starting from which r only assigns states of rank at most n. Let $S_{t'}$ be the set of states that r assigns to the root of t' at depth k, and let $\mathcal{A}_{t'}$ be the automaton that is derived from $\mathcal{A}$ by setting $S_{t'}$ to be the initial set, namely having the initial formula (which can be translated to an initial state) $\bigwedge_{q \in S_{t'}} q$. Observe that $\mathcal{A}_{t'}$ is a $(0,n)$-AWT, since it lacks the n+1-ranked states of $\mathcal{A}$.

Obviously, $\mathcal{A}_{t'}$ accepts t'. Furthermore, $\mathcal{A}_{t'}$ does not accept any tree out of $\overline{\mathsf{Win}_n}$, namely in Win_n: If it accepted some tree $t'' \in \mathsf{Win}_n$ then $\mathcal{A}$ would also accept the tree $\hat{t}$ that is derived from t by replacing the occurrence of t' at depth k with t''. However, $\hat{t}$ is not in $\overline{\mathsf{Win}_{n+1}}$ – Notice that since the last alternation on any play winning for Eve must be from E_1/A_1 to E_0/A_0, in t'' Eve either wins while seeing only up to $n-1$ alternations or the root of t'' is labelled A_0/E_0. Thus, $\hat{t}$ is not in $\overline{\mathsf{Win}_{n+1}}$, since if Eve plays to t'', she wins and sees either at most $n-1$ alternations within t'' and at most $n+1$ overall, or up to n alternations within t'' but only one more, i.e. $n+1$ alternations overall.

Since $\mathcal{A}$ is finite, the set $\{S_{t'} | t' \in \overline{\mathsf{Win}_n}\}$ is finite and therefore the disjunction $\bigvee_{t' \in \overline{\mathsf{Win}_n}} \mathcal{A}_{t'}$ of $(0,n)$-AWTs is a $(0,n)$-AWT that recognises $\overline{\mathsf{Win}_n}$ – a contradiction.

Odd n. Observe that $\overline{\mathsf{Win}_{n+1}}$ is recognised by a $(0,n+1)$-AWT if and only if Win_{n+1} is recognised by a $(1,n+2)$-AWT. Assume, towards a contradiction, that there is a $(1,n+2)$-AWT $\mathcal{A}$ that recognises Win_{n+1}. We will build a $(1,n+1)$-AWT that recognises Win_n,

reaching a contradiction. The construction is analogous to the above case of an even n, using the dual trees.

Consider a tree $t' \in \mathsf{Win}_n$. Let t be the tree consisting of one branch along which all nodes are labelled A_0, and they each have a child labelled A_1 that in turn has a copy of t' as its unique child.

First, we claim that $t \in \mathsf{Win}_{n+1}$: if Adam does not play into a copy of t', the play sees only labels A_0 and is winning for Eve; if Adam plays into a copy of t', then from there Eve has a winning strategy, such that every play that agrees with it has up to n alternations. Observe that such a play in t' either sees up to $n-1$ alternations, or begins with A_1/E_1, since the last alternation must be from E_1/A_1 to E_0/A_0. Then, in either cases, the overall play sees up to n alternation.

Now let r be an accepting run of $\mathcal{A}$ on t. Since $n+2$ is odd, there is a depth k starting from which r only assigns states of rank at most $n+1$. Let $S_{t'}$ be the set of states that r assigns to the root of t' at depth k, and let $\mathcal{A}_{t'}$ be the automaton that is derived from $\mathcal{A}$ by setting $S_{t'}$ to be the initial set. Observe that $\mathcal{A}_{t'}$ is a $(1, n+1)$-AWT, since it lacks the n+2-ranked states of $\mathcal{A}$.

Obviously, $\mathcal{A}_{t'}$ accepts t'. Furthermore, $\mathcal{A}_{t'}$ does not accept any tree not in Win_n: If it accepted some $t'' \notin \mathsf{Win}_n$ then $\mathcal{A}$ would also accept the tree $\hat{t}$ that is derived from t by replacing the occurrence of t' at depth k with t''. However, $\hat{t}$ is not in Win_{n+1}, since if Adam plays to t'', he either wins or forces to see more than n alternations within t'' and therefore more than $n+1$ alternations overall.

Since $\mathcal{A}$ is finite, the set $\{S_{t'} | t' \in \overline{\mathsf{Win}_n}\}$ is finite and therefore the disjunction $\bigvee_{t' \in \overline{\mathsf{Win}_n}} \mathcal{A}_{t'}$ of $(1, n+1)$-AWTs is a $(1, n+1)$-AWT that recognises Win_n – a contradiction. ◀

A.2 Proofs of Section 4

▶ **Lemma 11.** *Given an APW $\mathcal{A}$, the APW $\mathcal{A}_k$ accepts a word w if and only if Eve wins the k-register game on $\mathcal{G}(w, \mathcal{A})$ from (ι, w).*

Proof. Recall that by Proposition 3, $\mathcal{A}$ accepts a word w if and only if Eve wins the parity game $\mathcal{G}(w, \mathcal{A})$ from (ι, w). The intuition is that $\mathcal{G}(w, \mathcal{A}_k)$ encodes as a parity game the k-register game on $\mathcal{G}(w, \mathcal{A})$ by encoding the register-configuration in the state space, Eve's resetting choices as new disjunctions and the highest output from resets between two states as priorities.

Positions of $\mathcal{G}(w, \mathcal{A}_k)$ are in $\mathsf{B}^+(Q \times I^k \times [1..2k+1]) \times \Sigma^\omega$ while configurations of the k-register game on $\mathcal{G}(w, \mathcal{A})$ consist of a position $\mathsf{B}^+(Q) \times \Sigma^\omega$ and a tuple of register values.

$\mathcal{G}(w, \mathcal{A}_k)$ begins at $((\iota, (0, .., 0), 1), w)$ while the k-register game on $\mathcal{G}(w, \mathcal{A})$ begins at (ι, w) with register configuration $(0, .., 0)$.

Now, observe that at a position (q, au) in $\mathcal{G}(w, \mathcal{A})$ at register configuration $\bar{x}$, Eve has a choice to reset a register, or let the parity game proceed to $(\delta(q, a), u)$. Similarly in $\mathcal{G}(w, \mathcal{A}_k)$, from $((q, \bar{x}, p), au)$ Eve has a choice to either proceed with a move or reset a register.

Whenever Eve decides to proceed with a parity-game move, the decision falls in both games to Adam if $\delta(q, a)$ is a conjunction, and to Eve otherwise. Then Eve again has, in both games, the option between proceeding in the parity game or a reset. Observe that a move (that is, not a reset) only affects the register configuration in the register game on $\mathcal{G}(w, \mathcal{A})$ if it reaches a position (q, j) where q is a state (because intermediate positions have priority 0 in the parity game) and in $\mathcal{G}(w, \mathcal{A}_k)$ a non-reset move only affects $\bar{x}$ if it is $\mathsf{move}(q, \bar{x}, p)$ where q is a state. In both cases, if q is a state, the register configuration is updated to contain the maximum of the old value and the priority of q in A.

If Eve decides to reset a register i in the register game on $\mathcal{G}(w, \mathcal{A})$ at (b, u) with register configuration $\bar{x}$, then this updates the register configuration in the same way as Eve's choice of $\mathsf{reset}_i(b, \bar{x}, p)$ updates $\bar{x}$. Furthermore, the output in $\mathcal{G}(w, \mathcal{A})$ is $2i$ if x_i is even and $2i + 1$ if x_i is odd. Observe that the largest such output between two positions (q, au) and (q', u) is recorded in $\mathcal{G}(w, \mathcal{A}_k)$ as the priority p of $(q', \bar{x}, p)$. This guarantees that the largest priority output infinitely often on a play in the register-game on $\mathcal{G}(w, \mathcal{A})$ matches the highest priority seen infinitely often in the corresponding play in $\mathcal{G}(w, \mathcal{A}_k)$. On the other hand, in a play with finitely many resets, the maximal priority seen in $\mathcal{G}(w, \mathcal{A}_k)$ infinitely often is 1, causing Eve to lose, as required.

Then a winning strategy for Eve in one game translates into a winning strategy for Eve in the other game. ◀

▶ **Lemma 14.** *A parity game with scc-size* 1 *has defensive register-index* 1.

Proof. From the definition of register-index, instead of considering an arbitrary parity game and an arbitrary position of it, it suffices to consider the single-player parity games G induced by any winning strategy for Eve from that position. Observe that in the register games on G, Eve's strategy consists of just resetting the register.

We say that a defensive winning strategy in a 1-register game G is "very defensive" if whenever starting the game with the value 0 in the register, a play that agrees with the strategy does not output 3. (That is, it only outputs 2.)

We prove by induction on the number n of positions in a game G with scc-size 1 that Eve has a very-defensive winning strategy in the 1-register game on G.

The case of $n = 1$ is trivial.

For the inductive case, let G' be the game induced by the positions of G of priority up to $p - 2$, where p is the maximal even priority that appears in some cycle of G. Let G_s be the unique maximal strongly connected component in G'. If such a G_s does not exist, then Eve's strategy in the 1-register game on G is to reset whenever p is seen; since there are no cycles without p, this strategy is winning and very defensive.

If G_s does exist, then by the induction hypothesis, Eve has a very defensive winning strategy σ_s in the 1-register game on G_s. In addition, since G is of scc-size 1, there is no cycle in G that is disjoint to G_s.

Eve's very defensive winning strategy σ in the 1-register game on G is as follows: Reset whenever entering G_s, and follow σ_s within G_s.

Observe that a play that agrees with σ must see p between leaving and entering G_s (else G_s either isn't maximal, or there is a cycle with an odd maximal priority). Hence, when entering G_s, the play outputs 2, the register's content is 0, and the play continues along G_s, which is very defensive. Thus, σ is a very defensive winning strategy: there is no output of value 3, neither when entering G_s nor when remaining in it, and there are infinitely many outputs of value 2, either by the strategy σ_s or by leaving and entering G_s infinitely often. ◀

▶ **Lemma 16.** *Given an ultimately periodic word $w = uc^\omega$ and an APW $\mathcal{A}$ with n states, the parity game $\mathcal{G}(w, \mathcal{A})$ has register-index at most $1 + \log n$.*

Proof. We show that the scc-size of $\mathcal{G}(w, \mathcal{A})$ is at most n. Then, from Lemma 15, its register-index is at most $1 + \log n$.

First note that all strongly connected components of $\mathcal{G}(w, \mathcal{A})$ occur in the subgame $\mathcal{G}(c^\omega, \mathcal{A})$. We consider the graph H that is derived from $\mathcal{G}(c^\omega, \mathcal{A})$ by ignoring the intermediate positions (b, v), where b is a boolean formula between positions of the form (q, v), for a state

q. That is, let H be the graph consisting of just the vertices (q, v) of $\mathcal{G}(c^\omega, \mathcal{A})$, where q is a state. The edges of H connect positions (q, v) and (q', v') if (q', v') is reachable from (q, v) in $\mathcal{G}(c^\omega, \mathcal{A})$ directly, that is, with a path which does not visit yet another position (q'', v'') where q'' is a state.

$\mathcal{G}(c^\omega, A)$ has ssc-size no larger than the graph H: a set of distinct strongly connected components in $\mathcal{G}(c^\omega, A)$ induces a set of strongly connected components in H. If m is the size of the cycle c, then each strongly connected component of H must be of size at least m. H is of size at most mn, so the ssc-size of H, and therefore of $\mathcal{G}(w, \mathcal{A})$, is at most n. ◀

▶ **Lemma 17.** *Every APW A is equivalent to its parameterised version A_k, for $k = 1 + \log n$.*

Proof. Since two ω-regular languages are equivalent if they agree on the set of ultimately periodic words [17], it suffices to argue that A and A_k agree on ultimately periodic words.

From Lemma 11, A_k accepts an ultimately periodic word w if and only if Eve wins the k-register game on $\mathcal{G}(w, \mathcal{A})$. From Lemma 16, this is the case exactly when Eve wins the parity game on $\mathcal{G}(w, \mathcal{A})$, that is, when A accepts w. ◀

▶ **Theorem 18.** *The size blow-up and state blow-up involved in translating alternating parity word automata to alternating weak word automata is at most quasi-polynomial. In particular, every APW $\mathcal{A}$ of size (resp. number of states) n is equivalent to an AWW of size (resp. number of states) $2^{O((\log n)^3)}$.*

Proof. From Lemma 17, an APW $\mathcal{A}$ with n states and d priorities is equivalent to its parameterised APW $\mathcal{A}_k$ for $k = 1 + \log n$, having $n \cdot d^k \cdot (2k + 1)$ states and $2k + 1$ priorities. $\mathcal{A}_k$ can then be turned into a weak automaton using standard techniques [12] with a $O(m^{d'})$ blow-up, where m is the number of states and d' the number of priorities, which yields an AWW with $2^{O((\log n)^3)}$ many states, since m is here in $O(kn^{k+1}) \leq 2^{O((\log n)^2)}$ and d' is $2k + 1 \in O(\log n)$.

In case that the size of $\mathcal{A}$ is dominated by the size e of its transition function, namely when $e > n$, observe that the parameter k, the number of states in $\mathcal{A}_k$, and the number of priorities in $\mathcal{A}_k$ do not depend on e, while the size of $\mathcal{A}_k$'s transition function is in $O(k^2 e d^k) \leq 2^{O((\log e)^2)}$. Since the translation in [12] does not blow up the transition-function size more than it blows up the number of states, we end up with an AWW of size in $2^{O((\log e)^3)}$. ◀

Origin-Equivalence of Two-Way Word Transducers Is in PSPACE

Sougata Bose
LaBRI, University of Bordeaux, France

Anca Muscholl
LaBRI, University of Bordeaux, France

Vincent Penelle
LaBRI, University of Bordeaux, France

Gabriele Puppis
CNRS and LaBRI, University of Bordeaux, France

Abstract

We consider equivalence and containment problems for word transductions. These problems are known to be undecidable when the transductions are relations between words realized by non-deterministic transducers, and become decidable when restricting to functions from words to words. Here we prove that decidability can be equally recovered the *origin semantics*, that was introduced by Bojańczyk in 2014. We prove that the equivalence and containment problems for two-way word transducers in the origin semantics are PSPACE-complete. We also consider a variant of the containment problem where two-way transducers are compared under the origin semantics, but in a more relaxed way, by allowing distortions of the origins. The possible distortions are described by means of a *resynchronization* relation. We propose MSO-definable resynchronizers and show that they preserve the decidability of the containment problem under resynchronizations.

2012 ACM Subject Classification Theory of computation → Transducers

Keywords and phrases Transducers, origin semantics, equivalence

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.22

Related Version A full version of the paper is available at `https://arxiv.org/abs/1807.08053`.

1 Introduction

Finite-state transducers over words were studied in computer science very early, at the same time as finite-state automata, see e.g. [19, 1, 12, 4]. A transducer defines a binary relation between words by associating an output with each transition. It is called functional if this relation is a partial function. Whereas the class of functions defined by one-way transducers, i.e. transducers that process their input from left to right, has been extensively considered in the past, the study of two-way transducers is quite recent. Connections to logic, notably through the notion of graph transformations definable in monadic second-order logic (MSO) [7, 13], have shown that the functions realized by two-way word transducers can be equally defined in terms of MSO transductions [14]. This result is reminiscent of Büchi-Elgot characterizations that hold for many classes of objects (words, trees, traces, etc). For this reason, transductions described by functional two-way word transducers are called

regular functions. For a recent, nice survey on logical and algebraic properties of regular word transductions the reader is referred to [17].

Non-determinism is a very natural and desirable feature for most types of automata. However, for word transducers, non-determinism means less robustness. As an example, non-deterministic transducers are not equivalent anymore to NMSOT (the non-deterministic version of MSO transductions), however the latter is equivalent to non-deterministic streaming transducers [3]. A major problem is the undecidability of the equivalence of non-deterministic, one-way word transducers [18] (also called NGSM, and capturing the class of *rational relations*). In contrast, equivalence of functional, two-way word transducers is decidable [9], even in PSpace. This complexity is mainly based on the fact that it can be checked in PSpace if two non-deterministic, two-way automata are equivalent (see e.g. [21]).

The equivalence test is one of the most widely used operation on automata, so that it becomes a natural question to know what is needed to recover decidability of equivalence for rational relations, and even for *regular relations*, which are transductions defined by non-deterministic, two-way transducers. The main result of our paper is that equivalence of non-deterministic, two-way transducers is decidable if one adopts a semantics based on origin information. According to this *origin semantics* [5], each letter of the output is tagged with the input position that generated it. Thus, a relation in the origin semantics becomes a relation over $\Sigma^* \times (\Gamma \times \mathbb{N})^*$. Surprisingly, the complexity of the equivalence test turns out to be as low as it could be, namely in PSpace (thus PSpace-complete for obvious reasons).

As a second result, we introduce a class of MSO-definable resynchronizations for regular relations. A resynchronization [15] is a binary relation over $\Sigma^* \times (\Gamma \times \mathbb{N})^*$ that preserves the input and the output (i.e., the fist two components), but can change the origins (i.e., the third component). Resynchronizations allow to compare transducers under the origin semantics in a more relaxed way, by allowing distortions of origins. Formally, given two non-deterministic, two-way transducers $\mathcal{T}_1, \mathcal{T}_2$, and a resynchronization R, we want to compare $\mathcal{T}_1, \mathcal{T}_2$ under the origin semantics modulo R. Containment of $\mathcal{T}_1$ in $\mathcal{T}_2$ modulo R means that for each tagged input/output pair σ' generated by $\mathcal{T}_1$, there should be some tagged input/output pair σ generated by $\mathcal{T}_2$ such that $(\sigma, \sigma') \in R$. In other words, the resynchronization R describes possible distortions of origin, and we ask whether $\mathcal{T}_1$ is contained in $R(\mathcal{T}_2)$. The resynchronizations defined here correspond to MSO formulas that describe the change of origin by mainly considering the input (and to some small extent, the output). The containment problem under such resynchronizations turns to be undecidable, unless we enforce some restrictions. It is decidable for those resynchronizations R that use formulas satisfying a certain (decidable) "boundedness" property. In addition, if R is fixed, then the containment problem modulo R is solvable in PSpace, thus with the same complexity as the origin-equivalence problem. This is shown by providing a two-way transducer $\mathcal{T}_2'$ that is equivalent to $R(\mathcal{T}_2)$ (we say that $R(\mathcal{T}_2)$ is realizable by a transducer), and then we check containment of $\mathcal{T}_1$ in $\mathcal{T}_2'$ using our first algorithm. We conjecture that our class of resynchronizations captures the rational resynchronizations of [15], but we leave this for future work.

Related work and discussion. The origin semantics for transducers has been introduced in [5], and was shown to enjoy several nice properties, in particular a Myhill-Nerode characterization that can be used to decide the membership problem for subclasses of transductions, like first-order definable ones. The current state of the art counts quite a number of results related to the origin semantics of transducers. In [6] a characterization of the class of origin graphs generated by (functional) streaming transducers is given (the latter models were

studied in [2, 3]). Decision problems for tree transducers under the origin semantics have been considered in [16], where it is shown that origin-equivalence of top-down tree transducers is decidable. Note that top-down transducers correspond on words to one-way transducers, so the result of [16] is incomparable with ours.

As mentioned before, the idea of resynchronizing origins of word transducers has been introduced by Filiot et al. in [15] for the case of one-way transducers. Rational resynchronizers as defined in [15] are one-way transducers $\mathcal{R}$ that read sequences of the form $u_1v_1u_2v_2\cdots u_nv_n$, where $u_1\cdots u_n$ represents the input and $v_1\cdots v_n$ the output, with the origin of v_i being the last letter of u_i. It is required that any image of $u_1v_1u_2v_2\cdots u_nv_n$ through $\mathcal{R}$ leaves the input and the output part unchanged, thus only origins change. The definition of resynchronizer in the one-way case is natural. However, it cannot be extended to two-way transducers, since there is no word encoding of tagged input-output pairs realized by arbitrary two-way transducers. Our approach is logic-based: we define MSO resynchronizations that refer to origin graphs. More precisely, our MSO resynchronizations are formulas that talk about the input and, to a limited extent, about the tagged output.

Overview. After introducing the basic definitions and notations in Section 2, we present the main result about the equivalence problem in Section 3. Resynchronizations are considered in Section 4. A full version of the paper is available at `https://arxiv.org/abs/1807.08053`.

2 Preliminaries

Given a word $w = a_1 \dots a_n$, we denote by $\mathrm{dom}(w) = \{1, \dots, n\}$ its domain, and by $w(i)$ its i-th letter, for any $i \in \mathrm{dom}(w)$.

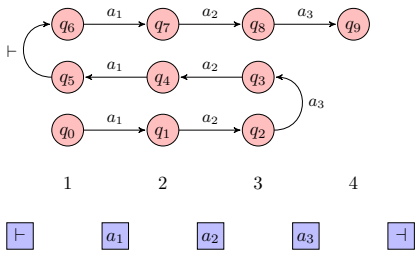
Automata. To define two-way automata, and later two-way transducers, it is convenient to adopt the convention that, for any given input $w \in \Sigma^*$, $w(0) = \vdash$ and $w(|w|+1) = \dashv$, where $\vdash, \dashv \notin \Sigma$ are special markers used as delimiters of the input. In this way an automaton can detect when an endpoint of the input has been reached and avoid moving the head outside.

A *two-way automaton* (*2NFA* for short) is a tuple $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$, where Σ is the input alphabet, $Q = Q_\prec \cup Q_\succ$ is the set of states, partitioned into a set $Q_\prec$ of left-reading states and a set $Q_\succ$ of right-reading states, $I \subseteq Q_\succ$ is the set of initial states, $F \subseteq Q$ is the set of final states, and $\Delta \subseteq Q \times (\Sigma \uplus \{\vdash, \dashv\}) \times Q \times \{\mathsf{left}, \mathsf{right}\}$ is the transition relation. The partitioning of the set of states is useful for specifying which letter is read from each state: left-reading states read the letter to the left, whereas right-reading states read the letter to the right. A transition $(q, a, q', d) \in \Delta$ is *leftward* (resp. *rightward*) if $d = \mathsf{left}$ (resp. $d = \mathsf{right}$). Of course, we assume that no leftward transition is possible when reading the left marker $\vdash$, and no rightward transition is possible when reading the right marker $\dashv$. We further restrict Δ by asking that $(q, a, q', \mathsf{left}) \in \Delta$ implies $q' \in Q_\prec$, and $(q, a, q', \mathsf{right}) \in \Delta$ implies $q' \in Q_\succ$.

To define runs of 2NFA we need to first introduce the notion of configuration. Given a 2NFA $\mathcal{A}$ and a word $w \in \Sigma^*$, a *configuration* of $\mathcal{A}$ on w is a pair (q, i), with $q \in Q$ and $i \in \{1, \dots, |w|+1\}$. Such a configuration represents the fact that the automaton is in state q and its head is *between* the $(i-1)$-th and the i-th letter of w (recall that we are assuming $w(0) = \vdash$ and $w(|w|+1) = \dashv$). The transitions that depart from a configuration (q, i) and read a are denoted $(q, i) \xrightarrow{a} (q', i')$, and must satisfy one of the following conditions:

- $q \in Q_\succ$, $a = w(i)$, $(q, a, q', \mathsf{right}) \in \Delta$, and $i' = i + 1$,
- $q \in Q_\succ$, $a = w(i)$, $(q, a, q', \mathsf{left}) \in \Delta$, and $i' = i$,
- $q \in Q_\prec$, $a = w(i-1)$, $(q, a, q', \mathsf{right}) \in \Delta$, and $i' = i$,
- $q \in Q_\prec$, $a = w(i-1)$, $(q, a, q', \mathsf{left}) \in \Delta$, and $i' = i - 1$.

A configuration (q, i) on w is *initial* (resp. *final*) if $q \in I$ and $i = 1$ (resp. $q \in F$ and $i = |w|+1$). A *run* of $\mathcal{A}$ on w is a sequence $\rho = (q_1, i_1) \xrightarrow{b_1} (q_2, i_2) \xrightarrow{b_2} \cdots \xrightarrow{b_m} (q_{m+1}, i_{m+1})$ of configurations connected by transitions. The figure below depicts an input $w = a_1a_2a_3$ (in blue) and a possible run on it (in red), where $q_0, q_1, q_2, q_6, q_7, q_8 \in Q_\succ$ and $q_3, q_4, q_5 \in Q_\prec$, and $1, 2, 3, 4$ are the positions associated with the various configurations.



A run is *successful* if it starts with an initial configuration and ends with a final configuration. The *language* of $\mathcal{A}$ is the set $[\![\mathcal{A}]\!] \subseteq \Sigma^*$ of all words on which $\mathcal{A}$ has a successful run.

When $\mathcal{A}$ has only right-reading states (i.e. $Q_\prec = \emptyset$) and rightward transitions, we say that $\mathcal{A}$ is a *one-way automaton* (*NFA* for short).

Transducers. Two-way transducers are defined similarly to two-way automata, by introducing an output alphabet Γ and associating an output from Γ^* with each transition rule. So a *two-way transducer* (*2NFT* for short) $\mathcal{T} = (Q, \Sigma, \Gamma, \Delta, I, F)$ is basically a 2NFA as above, but with a transition relation $\Delta \subseteq Q \times (\Sigma \uplus \{\vdash, \dashv\}) \times \Gamma^* \times Q \times \{\mathsf{left}, \mathsf{right}\}$. A transition is usually denoted by $(q, i) \xrightarrow{a|v} (q', i')$, and describes a move of the transducer from configuration (q, i) to configuration (q', i') that reads the input letter a and outputs the word v. The same restrictions and conventions for two-way automata apply to the transitions of two-way transducers, and configurations and runs are defined in a similar way.

The *output* associated with a successful run $\rho = (q_1, i_1) \xrightarrow{b_1|v_1} (q_2, i_2) \xrightarrow{b_2|v_2} \cdots \xrightarrow{b_m|v_m} (q_{m+1}, i_{m+1})$ is the word $v_1v_2 \cdots v_m \in \Gamma^*$. A two-way transducer $\mathcal{T}$ defines a relation $[\![\mathcal{T}]\!] \subseteq \Sigma^* \times \Gamma^*$ consisting of all the pairs (u, v) such that v is the output of some successful run ρ of $\mathcal{T}$ on u. Throughout the paper, a transducer is non-deterministic and two-way.

Origin semantics. In the origin semantics for transducers [5], the output is tagged with information about the position of the input where it was produced. If reading the i-th letter of the input we output v, then all letters of v are tagged with i, and we say they have *origin* i. We use the notation (v, i) to denote that all positions in v have origin i. The outputs associated with a successful run $\rho = (q_1, i_1) \xrightarrow{b_1|v_1} (q_2, i_2) \xrightarrow{b_2|v_2} (q_3, i_3) \cdots \xrightarrow{b_m|v_m} (q_{m+1}, i_{m+1})$ in the origin semantics are the words of the form $\nu = (v_1, j_1)(v_2, j_2) \cdots (v_m, j_m)$ over $\Gamma \times \mathbb{N}$, where, for all $1 \leq k \leq m$, $j_k = i_k$ if $q_k \in Q_\succ$, and $j_k = i_k - 1$ if $q_k \in Q_\prec$. Under the origin semantics, the relation defined by $\mathcal{T}$, denoted $[\![\mathcal{T}]\!]_o$, is the set of pairs $\sigma = (u, \nu)$ – called *synchronized pairs* – such that $u \in \Sigma^*$ and $\nu \in (\Gamma \times \mathbb{N})^*$ is the output of some successful run on u. Take as example the 2NFA run depicted in the previous figure, and assume that any transition on a letter $a \in \Sigma$ outputs a, while a transition on a marker $\vdash$ or $\dashv$ outputs the empty word ε. The output associated with that run in the origin semantics is $(a_1, 1)(a_2, 2)(a_3, 3)(a_2, 2)(a_1, 1)(a_1, 1)(a_2, 2)(a_3, 3)$.

Given two transducers $\mathcal{T}_1, \mathcal{T}_2$, we say they are *origin-equivalent* if $[\![\mathcal{T}_1]\!]_o = [\![\mathcal{T}_2]\!]_o$. Note that two transducers $\mathcal{T}_1, \mathcal{T}_2$ can be equivalent in the classical semantics, i.e. $[\![\mathcal{T}_1]\!] = [\![\mathcal{T}_2]\!]$, while they can have different origin semantics, so $[\![\mathcal{T}_1]\!]_o \neq [\![\mathcal{T}_2]\!]_o$.

Regular outputs. The transducers we defined just above consume input letters while outputting strings of bounded length. In order to perform some crucial constructions later – notably, to shortcut factors of runs with empty output – we need to slightly generalize the notion of output associated with a transition, so as to allow producing arbitrarily long words on reading a single letter. Formally, the transition relation of a transducer *with regular outputs* is allowed to be any subset Δ of $Q \times (\Sigma \uplus \{\vdash, \dashv\}) \times \mathbf{2}^{\Gamma^*} \times Q \times \{\mathsf{left}, \mathsf{right}\}$ such that, for all $q, q' \in Q$, $a \in \Sigma$, $d \in \{\mathsf{left}, \mathsf{right}\}$, there is at most one language $L \subseteq \Gamma^*$ such that $(q, a, L, q', d) \in \Delta$; moreover, this language L must be non-empty and regular. A transition $(q, i) \xrightarrow{a|L} (q', i')$ means that the transducer can move from configuration (q, i) to configuration (q', i') while reading a and outputting any word $v \in L$. Accordingly, the outputs that are associated with a successful run $\rho = (q_1, i_1) \xrightarrow{b_1|L_1} (q_2, i_2) \xrightarrow{b_2|L_2} (q_3, i_3) \cdots \xrightarrow{b_n|L_m} (q_{m+1}, i_{m+1})$ in the origin semantics are the words of the form $\nu = (v_1, j_1)(v_2, j_2) \cdots (v_m, j_m)$, where $v_k \in L_k$ and $j_k = i_k$ or $j_k = i_{k-1}$ depending on whether $q_k \in Q_\succ$ or $q_k \in Q_\prec$. We say that two runs ρ_1, ρ_2 are *origin-equivalent* if they have the same sets of associated outputs. Clearly, the extension with regular outputs is only syntactical, and it preserves the expressiveness of the class of transducers we consider. In the remaining of the paper, we will tacitly refer to above notion of transducer.

PSpace-constructibility. As usual, we call the *size* of an automaton or a transducer the number of its states, input symbols, transitions, plus, if present, the sizes of the NFA descriptions of the regular output languages associated with each transition rule.

In our complexity analysis, however, we will often need to work with online presentations of automata and transducers. For example, we may say that an automaton or a transducer can be computed using a polynomial amount of working space (at thus its size would be at most exponential) w.r.t. a given input. The terminology introduced below will be extensively used throughout the paper to describe the computational complexity of an automaton, a transducer, or a part of it, in terms of a specific parameter.

Given a parameter $n \in \mathbb{N}$, we say that an automaton or a transducer has PSpace-*constructible transitions w.r.t.* n if its transition relation can be enumerated by an algorithm that uses working space polynomial in n, and in addition, when the device is a transducer, every transition has at most polynomial size in n. In particular, if a transducer has PSpace-constructible transitions, then the size of the NFA representing every output language is polynomial. Similarly, we say that an automaton is PSpace-*constructible w.r.t.* n if all its components, – the alphabets, the state set, the transition relation, etc. – are enumerable by algorithms that use space polynomial in n.

3 Equivalence of transducers with origins

We focus on the equivalence problem for two-way transducers. In the classical semantics, this problem is known to be undecidable even if transducers are one-way [18] (called NGSM in the latter paper). We consider this problem in the origin semantics. We will show that, in this setting, equivalence becomes decidable, and can even be solved in PSpace – so with no more cost than equivalence of non-deterministic two-way automata:

▶ **Theorem 1.** *Containment and equivalence of two-way transducers under the origin semantics is* PSpace-*complete.*

The proof of this result is quite technical. As a preparation, we first show how to check origin-equivalence for transducers in which all transitions produce non-empty outputs, namely,

where the transition relation is of the form $\Delta \subseteq Q \times (\Sigma \uplus \{\vdash, \dashv\}) \times \mathbf{2}^{\Gamma^+} \times Q \times \{\mathsf{left}, \mathsf{right}\}$. We call *busy* any such transducer.

3.1 Origin-equivalence of busy transducers

An important feature of our definition of transducers is that, along any possible run, an input position is never read twice in a row. In other words, our transducers do not have "stay" transitions. For a busy transducer, this implies that the origins of outputs of consecutive transitions are always different. As a consequence, runs of two busy transducers can be only origin-equivalent if they visit the same sequences of positions of the input and have the same possible outputs transition-wise. To give the intuition, we note that origin-equivalence of busy classical transducers, with single output words associated with transitions, can be reduced to a version of equivalence of 2NFA, where we ask that runs have the same shape. Already the last condition does not allow to apply the PSPACE algorithm for equivalence of 2NFA of [21], and the naive algorithm would be of exponential time. Some more complications arise when we assume regular outputs, which will be required when we shall deal with non-busy transducers. We introduce now the key notions of transition shape and witness procedure.

Let $\mathcal{T}_1, \mathcal{T}_2$ be busy transducers over the same input alphabet, with $\mathcal{T}_i = (Q_i, \Sigma, \Gamma_i, \Delta_i, I_i, F_i)$ for $i = 1, 2$. We say that two transitions $t_1 \in \Delta_1$ and $t_2 \in \Delta_2$, with $t_i = (q_i, a_i, q_i', L_i, d_i)$, have the *same shape* if $a_1 = a_2$, $q_1 \in Q_{1,\prec} \Leftrightarrow q_2 \in Q_{2,\prec}$, and $q_1' \in Q_{1,\prec} \Leftrightarrow q_2' \in Q_{2,\prec}$ (and hence $d_1 = d_2$).

We assume that there is a non-deterministic procedure $\mathcal{W}$, called *witness procedure*, that does the following. Given a transition $t_1 = (q_1, a_1, q_1', L_1, d_1)$ of $\mathcal{T}_1$, $\mathcal{W}$ returns a set $X \subseteq \Delta_2$ of transitions of $\mathcal{T}_2$ satisfying the following property: *for some word* $v \in L_1$, we have

$$X \;=\; \big\{t_2 = (p_2, a_2, q_2, L_2, d_2) \in \Delta_2 \;:\; v \in L_2, \text{ and } t_2 \text{ has } \textit{same shape} \text{ as } t_1\big\}.$$

Note that $\mathcal{W}$ is non-deterministic: it can return several sets based on the choice of v. If $t_1 \in \Delta_1$ and X is a set that could be returned by $\mathcal{W}$ on t_1, we write $X \in \mathcal{W}(t_1)$. However, if $\mathcal{T}_1$ and $\mathcal{T}_2$ were classical transducers, specifying a single output word for each transition, then $\mathcal{W}$ could return only one set on $t_1 \in \Delta_1$, that is, the set of transitions of $\mathcal{T}_2$ with the same shape and the same output as t_1.

The intuition behind the procedure $\mathcal{W}$ is the following. Consider a successful run ρ_1 of $\mathcal{T}_1$ on u. Since $\mathcal{T}_1, \mathcal{T}_2$ are both busy, $[\![\mathcal{T}_1]\!]_o \subseteq [\![\mathcal{T}_2]\!]_o$ necessarily means that for all possible outputs produced by the transitions of ρ_1, there is some run ρ_2 of $\mathcal{T}_2$ on u that has the same shape as ρ_1 and the same outputs, transition-wise. Procedure $\mathcal{W}$ will precisely provide, for each transition t_1 of $\mathcal{T}_1$ with output language $L \subseteq \Gamma^+$, and for each choice of $v \in L$, the set of all transitions of $\mathcal{T}_2$ with the same shape as t and that could produce the same output v.

We introduce a last piece of terminology. Given a run $\rho_1 = t_1 \dots t_m$ of $\mathcal{T}_1$ of length m and a sequence $\xi = X_1, \dots, X_m$ of subsets of Δ_2 (*witness sequence*), we write $\xi \in \mathcal{W}(\rho_1)$ whenever $X_i \in \mathcal{W}(t_i)$ for all $1 \leq i \leq m$. We say that a run $\rho_2 = t_1' \dots t_m'$ of $\mathcal{T}_2$ is *ξ-compatible* if $t_i' \in X_i$ for all $1 \leq i \leq m$. The following result (proved in the appendix) is crucial:

▶ **Proposition 2.** *Let $\mathcal{T}_1, \mathcal{T}_2$ be two busy transducers over the same input alphabet, and $\mathcal{W}$ a witness procedure. Then $[\![\mathcal{T}_1]\!]_o \subseteq [\![\mathcal{T}_2]\!]_o$ if and only if for every successful run ρ_1 of $\mathcal{T}_1$, and for every witness sequence $\xi \in \mathcal{W}(\rho_1)$, there is a successful run ρ_2 of $\mathcal{T}_2$ which is ξ-compatible.*

Next, we reduce the problem $[\![\mathcal{T}_1]\!]_o \subseteq [\![\mathcal{T}_2]\!]_o$ to the emptiness problem of a one-way automaton (NFA) $\mathcal{B}$. In this reduction, the NFA $\mathcal{B}$ can be exponentially larger than $\mathcal{T}_1, \mathcal{T}_2$, but is PSPACE-constructible under suitable assumptions on $\mathcal{T}_1, \mathcal{T}_2$, and $\mathcal{W}$.

▶ **Lemma 3.** *Given two busy transducers $\mathcal{T}_1, \mathcal{T}_2$ with input alphabet Σ and a witness procedure $\mathcal{W}$, one can construct an NFA $\mathcal{B}$ that accepts precisely the words $u \in \Sigma^*$ for which there exist a successful run ρ_1 of $\mathcal{T}_1$ on u and a witness sequence $\xi \in \mathcal{W}(\rho_1)$ such that no ξ-compatible run ρ_2 of $\mathcal{T}_2$ is successful.*

Moreover, if $\mathcal{T}_1, \mathcal{T}_2$ have a total number n of states and PSPACE*-constructible transitions w.r.t. n, and $\mathcal{W}$ uses space polynomial in n, then $\mathcal{B}$ is* PSPACE*-constructible w.r.t. n.*

The proof of the lemma (in the appendix) is based on variants of the classical techniques of subset construction and crossing sequences [20, 21]. Below, we state an immediate consequence of the previous proposition and lemma:

▶ **Corollary 4.** *Given two busy transducers $\mathcal{T}_1, \mathcal{T}_2$ with a total number n of states, and given a witness procedure that uses space polynomial in n, the problem of deciding $[\![\mathcal{T}_1]\!]_o \subseteq [\![\mathcal{T}_2]\!]_o$ is in* PSPACE *w.r.t. n.*

3.2 Origin-equivalence of arbitrary transducers

We now consider transducers that are not necessarily busy. To show that origin-equivalence remains decidable in PSPACE, we will modify the transducers so as to make them busy, and reduce in this way the origin equivalence problem to the case treated in Section 3.1.

A naive idea would be to modify the transitions that output the empty word ε and make them output a special letter #. This however would not give a correct reduction towards origin-equivalence with busy transducers. Indeed, a transducer may produce non-empty outputs, say $v_1, v_2, \ldots$, with transitions that occur at the same position, say i, and that are interleaved by runs traversing other positions of the input but producing only ε. In that case, we would still need to compare where the words $v_1, v_2, \ldots$ were produced, and see that they may form a contiguous part of the output with origin i. The above idea is however useful if we first *normalize* our transducers in such a way that maximal subruns generating empty outputs are unidirectional. Paired with the fact that the same input position is never visited twice on two consecutive transitions, this will give the following characterization: two arbitrary transducers are origin-equivalent if and only if their normalized versions, with empty outputs replaced by #, are also origin-equivalent.

For simplicity, we fix a single transducer $\mathcal{T} = (Q, \Sigma, \Gamma, \Delta, I, F)$, which could be thought of as any of the two transducers $\mathcal{T}_1, \mathcal{T}_2$ that are tested for origin-equivalence. To normalize $\mathcal{T}$ we consider runs that start and end in the same position, and that produce empty output. Such runs are called lazy U-turns and are formally defined below. We will then abstract lazy U-turns by pairs of states, called U-pairs for short.

▶ **Definition 5.** Given an input word u, a *left* (resp. *right*) *lazy U-turn at position i of u* is any run of $\mathcal{T}$ on u of the form $(q_1, i_1) \xrightarrow{b_1|v_1} (q_2, i_2) \xrightarrow{b_2|v_2} \cdots \xrightarrow{b_m|v_m} (q_{m+1}, i_{m+1})$, with $i_1 = i_{m+1} = i$, $i_k < i$ (resp. $i_k > i$) for all $2 \leq k \leq m$, and $v_k = \varepsilon$ for all $1 \leq k \leq m$.

For brevity, we shall often refer to a *left/right lazy U-turn* without specifying the position i and the word u, assuming that these are clear from the context.

The pair (q_1, q_{m+1}) of states at the extremities of a left/right lazy U-turn is called a *left/right U-pair (at position i of u)*. We denote by $U_i^{\curvearrowright}$ (resp. $U_i^{\curvearrowleft}$) the set of all left (resp. right) U-pairs at position i (again, the input u is omitted from the notation as it usually understood from the context).

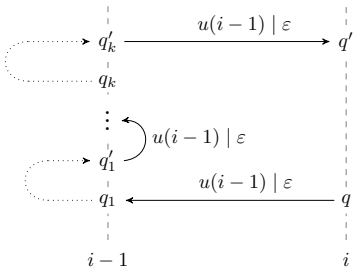
Note that we have $U_i^{\curvearrowright} \subseteq Q_\prec \times Q_\succ$ and $U_i^{\curvearrowleft} \subseteq Q_\succ \times Q_\prec$. Accordingly, we define the word $u^{\curvearrowright}$ over $2^{Q_\prec \times Q_\succ}$ that has the same length as u and labels every position i with the

set $U_i^{\curvearrowright}$ of left U-pairs. This $u^{\curvearrowright}$ is seen as an annotation of the original input u, and can be computed from $\mathcal{T} = (Q, \Sigma, \Gamma, \Delta, I, F)$ and u using the following recursive rule:

$(q, q') \in u^{\curvearrowright}(i)$ if and only if $q \in Q_{\prec} \wedge \big(q, u(i-1), \varepsilon, q', \mathsf{right}\big) \in \Delta$

or $q \in Q_{\prec} \wedge \exists\, (q_1, q_1'), \ldots, (q_k, q_k') \in u^{\curvearrowright}(i-1)$

$$\begin{cases} \big(q, u(i-1), \varepsilon, q_1, \mathsf{left}\big) \in \Delta \\ \big(q_j', u(i-1), \varepsilon, q_{j+1}, \mathsf{left}\big) \in \Delta \quad \forall 1 \le j \le k \\ \big(q_k', u(i-1), \varepsilon, q_k, \mathsf{right}\big) \in \Delta. \end{cases}$$



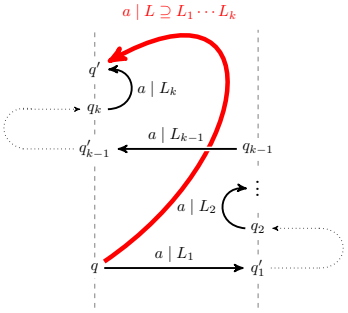
The annotation $u^{\curvearrowleft}$ with right U-pairs satisfies a symmetric recursive rule.

Of course, the annotations $u^{\curvearrowright}, u^{\curvearrowleft}$ are over alphabets of exponential size. This does not raise particular problems concerning the complexity, since we aim at deciding origin-equivalence in PSPACE, and towards this goal we could work with automata and transducers that have PSPACE-constructible transitions. In particular, we can use the recursive rules for $u^{\curvearrowright}$ and $u^{\curvearrowleft}$ to get the following straightforward lemma (proof omitted):

▶ **Lemma 6.** *Given a transducer $\mathcal{T}$, one can compute an NFA $\mathcal{U}$ that has the same number of states as $\mathcal{T}$ and such that $[\![\mathcal{U}]\!] = \{u \otimes u^{\curvearrowright} \otimes u^{\curvearrowleft} : u \in \Sigma^*\}$, where $\otimes$ denotes the convolution of words of the same length. The NFA $\mathcal{U}$ is* PSPACE*-constructible w.r.t. the size of $\mathcal{T}$.*

To normalize the transducer $\mathcal{T}$ it is convenient to assume that the sets of left and right U-pairs can be read directly from the input (we refer to this as *annotated input*). For this, we introduce the transducer $\mathcal{T}_U$, that is obtained from $\mathcal{T}$ by extending its input alphabet from Σ to $\Sigma \times 2^{Q_{\prec} \times Q_{\succ}} \times 2^{Q_{\succ} \times Q_{\prec}}$, and by modifying the transitions in the obvious way, that is, from (q, a, L, q', d) to $\big(q, (a, U^{\curvearrowright}, U^{\curvearrowleft}), L, q', d\big)$ for any $U^{\curvearrowright} \subseteq Q_{\prec} \times Q_{\succ}$ and $U^{\curvearrowleft} \subseteq Q_{\succ} \times Q_{\prec}$. Note that $\mathcal{T}_U$ does not check that the input is correctly annotated, i.e. of the form $u \otimes u^{\curvearrowright} \otimes u^{\curvearrowleft}$ (this is done by the NFA $\mathcal{U}$). Further note that $\mathcal{T}_U$ has exponential size w.r.t. $\mathcal{T}$, but its state space remains the same as $\mathcal{T}$, and its transitions can be enumerated in PSPACE.

We are now ready to describe the normalization of $\mathcal{T}_U$, which produces an origin-equivalent transducer $\mathsf{Norm}(\mathcal{T}_U)$ with no lazy U-turns. The transducer $\mathsf{Norm}(\mathcal{T}_U)$ is obtained in two steps. First, using the information provided by the annotation of the input, we shortcut all runs of $\mathcal{T}_U$ that consist of multiple transitions outputting at the same position and interleaved by lazy U-turns. The resulting transducer is denoted $\mathsf{Shortcut}(\mathcal{T}_U)$. After this step, we will eliminate the lazy U-turns, thus obtaining $\mathsf{Norm}(\mathcal{T}_U)$. Formally, $\mathsf{Shortcut}(\mathcal{T}_U)$ has for transitions the tuples of the form $\big(q, (a, U^{\curvearrowright}, U^{\curvearrowleft}), L, q', d\big)$, where L is the smallest language that contains every language of the form $L_1 \cdot L_2 \cdots L_k$ for which there are $q_1, q_1', \ldots, q_k, q_k'$ and $d_1, \ldots, d_k$, with $q = q_1$, $q_k' = q'$ (and hence $d_k = d$), $(q_i, a, L_i, q_i', d_i) \in \Delta$, and $(q_i', q_{i+1}) \in U^{\curvearrowright} \cup U^{\curvearrowleft}$ for all i (see the figure below).



Note that there is no transition $\big(q, (a, U^{\circlearrowright}, U^{\circlearrowleft}), L, q', d\big)$ when there are no languages $L_1, L_2, \ldots, L_k$ as above. The output languages associated with the transitions of $\mathsf{Shortcut}(\mathcal{T}_U)$ can be constructed using a classical saturation mechanism, which is omitted here, they are regular, and their NFA representations are polynomial-sized w.r.t. the size of the NFA representations of the output languages of $\mathcal{T}_U$. This implies that $\mathsf{Shortcut}(\mathcal{T}_U)$ has PSPACE-constructible transitions w.r.t. the number of its states, exactly like $\mathcal{T}_U$.

Recall that two runs are *origin-equivalent* if they produce the same synchronized pairs. The next lemma shows that, on correctly annotated inputs, $\mathsf{Shortcut}(\mathcal{T}_U)$ is origin-equivalent to $\mathcal{T}_U$, even when avoiding U-turns. The proof of the lemma is in the appendix.

▶ **Lemma 7.** *Let $w = u \otimes u^{\circlearrowright} \otimes u^{\circlearrowleft}$ be an arbitrary input annotated with left and right U-pairs. Every successful run of $\mathcal{T}_U$ on w is origin-equivalent to some successful run of $\mathsf{Shortcut}(\mathcal{T}_U)$ on w without lazy U-turns. Conversely, every successful run of $\mathsf{Shortcut}(\mathcal{T}_U)$ on w is origin-equivalent to some successful run of $\mathcal{T}_U$ on w.*

The next step of the normalization consists of restricting the runs of $\mathsf{Shortcut}(\mathcal{T}_U)$ so as to avoid any lazy U-turn. This is done by simply forbidding the shortest possible lazy U-turns, namely, the transitions that output ε and that remain on the same input position. On the one hand, since every lazy U-turn contains a transition of the previous form, forbidding this type of transitions results in forbidding arbitrary lazy U-turns. On the other hand, thanks to Lemma 7, this will not affect the semantics of $\mathsf{Shortcut}(\mathcal{T}_U)$. Formally, we construct from $\mathsf{Shortcut}(\mathcal{T}_U)$ a new transducer $\mathsf{Norm}(\mathcal{T}_U)$ by replacing every transition rule (q, a, L, q', d) with (q, a, L', q', d), where L' is either L or $L \setminus \{\varepsilon\}$, depending on whether $q \in Q_{\prec} \Leftrightarrow q' \in Q_{\prec}$ or not. We observe that $\mathsf{Norm}(\mathcal{T}_U)$ has the same set of states as the original transducer $\mathcal{T}$, and PSPACE-constructible transitions w.r.t. the size of $\mathcal{T}$. The proof of the following result is straightforward and thus omitted.

▶ **Lemma 8.** *$\mathcal{T}_U$ and $\mathsf{Norm}(\mathcal{T}_U)$ are origin-equivalent when restricted to correctly annotated inputs, i.e.: $[\![\mathcal{T}_U]\!]_o \cap R = [\![\mathsf{Norm}(\mathcal{T}_U)]\!]_o \cap R$, where $R = [\![\mathcal{U}]\!] \times (\Gamma \times \mathbb{N})^*$.*

We finally come to the last step of the reduction. This amounts to consider two normalized transducers $\mathsf{Norm}(\mathcal{T}_{1,U})$ and $\mathsf{Norm}(\mathcal{T}_{2,U})$ that read inputs annotated with the left/right U-pairs of both $\mathcal{T}_1$ and $\mathcal{T}_2$, and replacing, in their transitions, the empty output by a special letter $\# \notin \Gamma$. Formally, in the transition relation of $\mathsf{Norm}(\mathcal{T}_{i,U})$, for $i = 1, 2$, we replace every tuple (q, a, L, q', d), where q is left-reading iff q' is left-reading, with the tuple (q, a, L', q', d), where L' is either $(L \setminus \{\varepsilon\}) \cup \{\#\}$ or L, depending on whether $\varepsilon \in L$ or not. The transducer obtained in this way is *busy*, and is thus called $\mathsf{Busy}(\mathcal{T}_{i,U})$. Moreover, the states of $\mathsf{Busy}(\mathcal{T}_{i,U})$ are the same as those of $\mathcal{T}_i$, and its transitions are PSPACE-constructible. The proposition below follows immediately from the previous arguments, and reduces origin-containment between

$\mathcal{T}_1$ and $\mathcal{T}_2$ to an origin-containment between $\mathsf{Busy}(\mathcal{T}_{1,U})$ and $\mathsf{Busy}(\mathcal{T}_{2,U})$, but relativized to correctly annotated inputs.

▶ **Proposition 9.** *Given two transducers $\mathcal{T}_1$ and $\mathcal{T}_2$,*

$$[\![\mathcal{T}_1]\!]_o \subseteq [\![\mathcal{T}_2]\!]_o \quad \text{if and only if} \quad [\![\mathsf{Busy}(\mathcal{T}_{1,U})]\!]_o \cap R \subseteq [\![\mathsf{Busy}(\mathcal{T}_{2,U})]\!]_o \cap R.$$

where $R = [\![\mathcal{U}']\!] \times (\Gamma \times \mathbb{N})^*$ and $\mathcal{U}'$ is an NFA that recognizes inputs annotated with left/right U-pairs of both $\mathcal{T}_1$ and $\mathcal{T}_2$.

It remains to show that, given the transducers $\mathcal{T}_1, \mathcal{T}_2$, there is a PSPACE witness procedure for $\mathsf{Busy}(\mathcal{T}_{1,U}), \mathsf{Busy}(\mathcal{T}_{2,U})$:

▶ **Proposition 10.** *Let $\mathcal{T}_1, \mathcal{T}_2$ be transducers with a total number n of states, and $\mathsf{Busy}(\mathcal{T}_{i,U}) = (Q_i, \hat{\Sigma}, \Gamma, \Delta_i, I_i, F_i)$ for $i = 1, 2$ (the input alphabet $\hat{\Sigma}$ is the same as for $\mathcal{U}'$). There is a non-deterministic procedure $\mathcal{W}$ that works in polynomial space in n and returns on a given transition $t_1 = (q_1, a_1, q_1', L_1, d_1)$ of $\mathsf{Busy}(\mathcal{T}_{1,U})$ any set $X \subseteq \Delta_2$ of transitions of $\mathsf{Busy}(\mathcal{T}_{2,U})$ such that for some $v \in L_1$:*

$$X = \{t_2 = (p_2, a_2, q_2, L_2, d_2) \in \Delta_2 \,:\, v \in L_2, \text{ and } t_2 \text{ has same shape as } t_1\}.$$

We can now conclude with the proof of our main result, which we recall here:

▶ **Theorem 1.** *Containment and equivalence of two-way transducers under the origin semantics is* PSPACE*-complete.*

Proof. The lower bound follows from a straightforward reduction from the classical equivalence problem of NFA. For the upper bound, in view of Proposition 9, we can consider origin containment between the transducers $\mathsf{Busy}(\mathcal{T}_{1,U})$ and $\mathsf{Busy}(\mathcal{T}_{2,U})$, obtained from $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively. We recall that $\mathsf{Busy}(\mathcal{T}_{i,U})$ has at most $n_i = |\mathcal{T}_i|$ states and PSPACE-constructible transitions w.r.t. n_i, for $i = 1, 2$. We then apply Proposition 2 and Lemma 3 to reduce the containment problem to an emptiness problem for the intersection of two PSPACE-constructible NFA, i.e. $\mathcal{B}$ and $\mathcal{U}'$. We observe that $\mathcal{U}'$ is basically the product of two NFA obtained from Lemma 6 by letting $\mathcal{T} = \mathcal{T}_i$, once for $i = 1$ and once for $i = 2$. Finally, we use the same arguments as in the proof of Corollary 4 to conclude that the latter emptiness problem is decidable in PSPACE w.r.t. $n = n_1 + n_2$. ◀

4 Containment modulo resynchronization

In this section we aim at generalizing the equivalence and containment problems for transducers with origins. The goal is to compare the origin semantics of any two transducers up to "distortions", that is, differences in the origin tagging each position of the output.

Recall that $[\![\mathcal{T}]\!]_o$ is the set of synchronized pairs $\sigma = (u, \nu)$, where $u \in \Sigma^*$ is a possible input for the transducer $\mathcal{T}$ and $\nu \in (\Gamma \times \mathbb{N})^*$ is an output (tagged with origins) produced by a successful run of $\mathcal{T}$ on u. Given a pair $\sigma = (u, \nu) \in [\![\mathcal{T}]\!]_o$, we denote by $\mathsf{in}(\sigma)$, $\mathsf{out}(\sigma)$, and $\mathsf{orig}(\sigma)$ respectively the input word u, the output word obtained by projecting ν onto the finite alphabet Γ, and the sequence of input positions obtained by projecting ν onto $\mathbb{N}$. This notation is particularly convenient for describing resynchronizations, that is, relations between synchronized pairs. Following prior terminology from [15], we call *resynchronization* any relation R between synchronized pairs that preserves input and output words, but can modify the origins, namely, such that $(\sigma, \sigma') \in R$ implies $\mathsf{in}(\sigma) = \mathsf{in}(\sigma')$ and $\mathsf{out}(\sigma) = \mathsf{out}(\sigma')$.

The *containment problem modulo a resynchronization* R [15] is the problem of deciding, given two transducers $\mathcal{T}_1, \mathcal{T}_2$, if for every $\sigma' \in [\![\mathcal{T}_1]\!]_o$, there is $\sigma \in [\![\mathcal{T}_2]\!]_o$ such that $(\sigma, \sigma') \in R$, or in other words if every synchronized pair of $\mathcal{T}_1$ can be seen as a distortion of a synchronized pair of $\mathcal{T}_2$. In this case we write for short $\mathcal{T}_1 \subseteq_R \mathcal{T}_2$. We remark that, despite the name "containment" and the notation, the relation $\subseteq_R$ is not necessarily transitive, in the sense that it may happen that $\mathcal{T}_1 \subseteq_R \mathcal{T}_2$ and $\mathcal{T}_2 \subseteq_R \mathcal{T}_3$, but $\mathcal{T}_1 \not\subseteq_R \mathcal{T}_3$.

We propose a class of resynchronizations that can be described in monadic second-order logic (*MSO*). The spirit is that, as synchronized pairs are (special) graphs, graph transformations à la Courcelle and Engelfriet [8] are an adequate tool to define resynchronizers. However, we cannot directly use MSO logic over origin graphs, since this would result in an undecidable containment problem. Our MSO resynchronizers will be able to talk about (regular) properties of the input word and the output word, and say how origins are distorted. We will show that containment modulo MSO resynchronizations is decidable, assuming a (decidable) restriction on the change of origins.

▶ **Definition 11.** An *MSO resynchronizer* R is a tuple $(\alpha, \beta, \gamma, \delta)$, where

- $\alpha(\overline{I})$ is an MSO formula over the signature of the input word, and has some free monadic variables $\overline{I} = (I_1, \ldots, I_m)$, called *input parameters*,
- $\beta(\overline{O})$ is an MSO formula over the signature of the output word, and has some free monadic variables $\overline{O} = (O_1, \ldots, O_n)$, called *output parameters*,
- γ is a function that maps any element $\tau \in \Gamma \times \mathbb{B}^n$, with $\mathbb{B} = \{0, 1\}$, to an MSO formula $\gamma(\tau)(\overline{I}, y, z)$ over the input signature that has a tuple of free monadic variables $\overline{I}$ (input parameters) and two free first-order variables y, z (called *source* and *target*, respectively),
- δ is a function that maps any pair of elements $\tau, \tau' \in \Gamma \times \mathbb{B}^n$, with $\mathbb{B} = \{0, 1\}$, to an MSO formula $\delta(\tau, \tau')(\overline{I}, z, z')$ over the input signature that has a tuple of free monadic variables $\overline{I}$ (input parameters) and two free first-order variables z, z' (called *targets*).

The input and output parameters $\overline{I}$, $\overline{O}$ that appear in the formulas of an MSO resynchronizer play the same role as the parameters of an NMSO-transduction [14]. They allow to express regular properties of the input and output word, respectively, through the first two formulas, $\alpha(\overline{I})$ and $\beta(\overline{O})$. The other two formulas are used to describe how the origin function is transformed. They depend on the input and output parameters, in particular, on the "type" of the positions of the output word, as defined next.

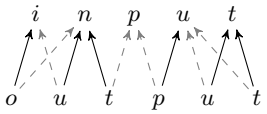
Given an output word $v = \mathsf{out}(\sigma)$ and an interpretation $\overline{O} = O_1, \ldots, O_n \subseteq \mathrm{dom}(v)$ for the output parameters, let us call *output-type* of a position $x \in \mathrm{dom}(v)$ the element $\tau = (v(x), b_1, \ldots, b_n) \in \Gamma \times \mathbb{B}^n$, where each b_i is either 1 or 0 depending on whether $x \in O_i$ or not. Based on the output-type τ of x, the formula $\gamma(\tau)(\overline{I}, y, z)$ describes how the origin of x is redirected from a source y to a target z in the input word. Similarly, $\delta(\tau, \tau')(\overline{I}, z, z')$ constraints the origins z, z' of two consecutive output positions $x, x+1$, based on their output-types τ and τ'. This is formalized below.

▶ **Definition 12.** An MSO resynchronizer $R = (\alpha, \beta, \gamma, \delta)$ induces the resynchronization $[\![R]\!]$ defined by $(\sigma, \sigma') \in [\![R]\!]$ if and only if $\mathsf{in}(\sigma) = \mathsf{in}(\sigma')$, $\mathsf{out}(\sigma) = \mathsf{out}(\sigma')$, and there are $\overline{I} = I_1, \ldots, I_m \subseteq \mathrm{dom}(\mathsf{in}(\sigma))$ and $\overline{O} = O_1, \ldots, O_n \subseteq \mathrm{dom}(\mathsf{out}(\sigma))$ such that:

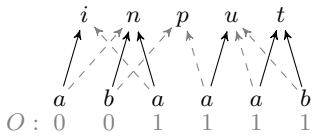
- $(\mathsf{in}(\sigma), \overline{I}) \vDash \alpha$ (or equally, $(\mathsf{in}(\sigma'), \overline{I}) \vDash \alpha$),
- $(\mathsf{out}(\sigma), \overline{O}) \vDash \beta$ (or equally, $(\mathsf{out}(\sigma'), \overline{O}) \vDash \beta$),
- for all $x \in \mathrm{dom}(\mathsf{out}(\sigma))$ with output-type τ, if $\mathsf{orig}(\sigma)(x) = y$ and $\mathsf{orig}(\sigma')(x) = z$, then $(\mathsf{in}(\sigma), \overline{I}, y, z) \vDash \gamma(\tau)$,
- for all pairs of consecutive positions x and $x+1$ in $\mathsf{out}(\sigma)$ with output-types τ and τ', respectively, if $\mathsf{orig}(\sigma')(x) = z$ and $\mathsf{orig}(\sigma')(x+1) = z'$, then $(\mathsf{in}(\sigma), \overline{I}, z, z') \vDash \delta(\tau, \tau')$.

▶ **Example 13.** We provide a few examples of MSO resynchronizers:

1. The most unconstrained resynchronization, called *universal resynchronization*, groups any two synchronized pairs with the same input and output strings. This is readily defined by an MSO resynchronizer $R_{univ} = (\alpha, \beta, \gamma, \delta)$ without parameters, where $\alpha = \beta = \gamma(\tau)(y, z) = \delta(\tau, \tau')(z, z') = \mathit{true}$ for all $\tau, \tau' \in \Gamma$.
2. Consider a resynchronization that displaces the origin in a synchronized pair by one position to the left or to the right. An instance of this resynchronization is shown below, where solid arrows represent origins in the initial synchronized pair, and dashed arrows represent origins in the modified synchronized pair. This transformation can be defined by a parameterless MSO resynchronizer $R_{\pm 1} = (\alpha, \beta, \gamma, \delta)$, where $\alpha = \beta = \mathit{true}$, $\gamma(\tau)(y, z) = (z = y - 1) \vee (z = y + 1)$, and $\delta(\tau, \tau')(z, z') = \mathit{true}$ for all $\tau, \tau' \in \Gamma$. Note that the reflexive and transitive closure of $[\![R_{\pm 1}]\!]$ gives the universal resynchronization $[\![R_{univ}]\!]$.

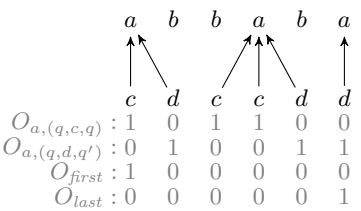


3. We give a variant of the previous resynchronizer $R_{\pm 1}$, where the direction of movement of the origin is controlled by a property of the output position, e.g. the parity of the number of b's that precede that position in the output. We see an instance of the resynchronization below (as usual, solid and dashed arrows represent initial and modified origins). The transformation can be defined by an MSO resynchronizer with a single output parameter O that encodes the parity condition (an interpretation of O is shown in the figure as an annotation of the output over $\mathbb{B} = \{0, 1\}$).



Formally, we let $R'_{\pm 1} = (\alpha, \beta, \gamma, \delta)$, where $\alpha = \mathit{true}$, $\beta(O) = \forall x \ \big(b(x) \to (x \in O \leftrightarrow x + 1 \notin O)\big) \wedge \big(\neg b(x) \to (x \in O \leftrightarrow x + 1 \in O)\big) \wedge (x = \mathit{first} \to x \in O))$, and γ and δ are defined on the basis of the output-types $\tau, \tau' \in \Gamma \times \mathbb{B}$, as follows: $\gamma(\tau)(y, z)$ enforces either $z = y + 1$ or $z = y - 1$ depending on whether $\tau \in \Gamma \times \{0\}$ or $\tau \in \Gamma \times \{1\}$, and $\delta(\tau, \tau')(z, z') = \mathit{true}$.

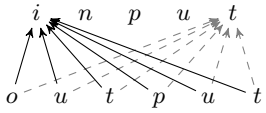
4. Let us now consider a resynchronization that does not modify the origins, but only constrain them so as to obtain a "regular" subset of synchronization pairs. Here the allowed synchronization pairs are those that correspond to the process of applying a rational substitution $f : \Sigma \to 2^{\Gamma^*}$ to an input over Σ. The figure below describes a possible synchronized pair for f defined by $f(a) = c^*d$ and $f(b) = \varepsilon$.



Below, we show how to define such a resynchronization for an arbitrary rational substitution f. We fix, for each letter $a \in \Sigma$, an NFA $\mathcal{A}_a$ that recognizes the regular language $f(a)$. We then define an MSO resynchronizer R_f that uses one output parameter $O_{a,t}$ for every letter $a \in \Sigma$ and every transition t of $\mathcal{A}_a$, plus two additional output parameters O_{first} and O_{last}. By a slight abuse of notation, given the output-type τ of a position x, we write $\tau[a,t] = 1$ whenever $x \in O_{a,t}$, and similarly for $\tau[\mathit{first}]$ and $\tau[\mathit{last}]$. The first formula α of the resynchronizer holds vacuously, as we have no restriction on the input. The second formula β requires that the parameters $O_{a,t}$ form a partition of the output domain in such a way that if a position x is labeled by a letter $c \in \Gamma$ and $x \in O_{a,t}$, then t is a c-labeled transition of $\mathcal{A}_a$. In addition, β requires that the parameters O_{first} and O_{last} are singletons consisting of the first and the last output position, respectively. The third component γ of the resynchronizer is defined by $\gamma(\tau)(y,z) = a(y) \wedge (y = z)$ whenever $\tau[a,t] = 1$ (the origin is not modified, and the transition annotating the output position must belong to the correct NFA). The last component δ restricts further the parameters and the origins for consecutive output positions, so as to simulate successful runs of the NFA. Formally, $\delta(\tau,\tau')(z,z')$ enforces the following constraints:

- if $\tau[a,t] = 1$, $\tau'[a,t'] = 1$, and $z = z'$, then the target state of t coincide with the source state of t', namely, tt' forms a factor of a run of $\mathcal{A}_a$,
- if $\tau[a,t] = 1$, $\tau'[a',t'] = 1$, and $z < z'$, then the target state of t must be final, the source state of t' must be initial, and every input letter strictly between z and z' is mapped via f to a language that contains ε,
- if $\tau[\mathit{first}] = 1$ and $\tau[a,t]$, then the source state of t must be initial, and every input letter strictly before z is mapped via f to a language that contains ε,
- if $\tau'[\mathit{last}] = 1$ and $\tau'[a,t]$, then the target state of t must be final, and every input letter strictly after z' is mapped via f to a language that contains ε.

5. We conclude the list of examples with a resynchronization that moves the origin of an arbitrarily long output over an arbitrarily long distance. This resynchronization contains the pairs (σ,σ'), where σ (resp. σ') maps every output position to the first (resp. last) input position, as shown in the figure.



This is defined by the MSO resynchronizer $R_{1st-to-last} = (\alpha,\beta,\gamma,\delta)$, where $\alpha = \beta = \delta(\tau,\tau')(z,z') = \mathit{true}$ and $\gamma(\tau)(y,z) = (y = \mathit{first}) \wedge (z = \mathit{last})$, for all $\tau,\tau' \in \Gamma$. Note that the resynchronization $[\![R_{1st-to-last}]\!]$ is also "one-way", in the sense that it contains only synchronized pairs that are admissible outcomes of runs of one-way transducers. However, $[\![R_{1st-to-last}]\!]$ is not captured by the formalism of one-way *rational* resynchronizers from [15].

Recall the Example 13.1 above, which defines the universal resynchronization $[\![R_{univ}]\!]$. The containment problem modulo $[\![R_{univ}]\!]$ boils down to testing classical containment between transducers *without origins*, which is known to be undecidable [18]. Based on this, it is clear that in order to compare effectively transducers modulo resynchronizations, we need to restrict further our notion of MSO resynchronizer. Intuitively, what makes the containment problem modulo resynchronization undecidable is the possibility of redirecting many sources

to the same target. This possibility is explicitly forbidden in the definition below. Also observe that, since $[\![R_{univ}]\!]$ is the reflexive and transitive closure of $[\![R_{\pm 1}]\!]$, we cannot take our resynchronizations to be equivalence relations.

▶ **Definition 14.** An MSO resynchronizer $(\alpha, \beta, \gamma, \delta)$ is *k-bounded* if for all inputs u, parameters $\overline{I} = I_1, \ldots, I_m \subseteq \mathrm{dom}(u)$, output-types $\tau \in \Gamma \times \mathbb{B}^n$, and targets $z \in \mathrm{dom}(u)$, there are at most k distinct sources $y_1, \ldots, y_k \in \mathrm{dom}(u)$ such that $(u, \overline{I}, y_i, z) \vDash \gamma(\tau)$ for all $i = 1, \ldots, k$. An MSO resynchronizer is *bounded* if it is k-bounded for some k.

Note that all resynchronizers from Example 13 but R_{univ} are bounded. For instance, $R_{1st-to-last}$ is 1-bounded and $R_{\pm 1}$ is 2-bounded.

It is also easy to decide the boundedness property of an MSO resynchronizer by reducing it to a problem of finite-ambiguity for finite automata [22]:

▶ **Proposition 15.** *t is decidable to know whether a given MSO resynchronizer is bounded.*

The goal is to reduce the problem of containment modulo a bounded MSO resynchronizer to a standard containment problem in the origin semantics. For this, the natural approach would be to show that transducers are effectively closed under bounded MSO resynchronizers. However, this closure property cannot be proven in full generality, because of the (input) parameters that occur in the definition of resynchronizers. More precisely, *two-way* transducers cannot guess parameters in a consistent way (different guesses could be made at different visits of the same input position). We could show the closure if we adopted a slightly more powerful notion of transducer, with so-called *common guess* [6]. Here we prefer to work with classical two-way transducers and explicitly deal with the parameters. Despite the different terminology, the principle is the same: parameters are guessed beforehand and accessed by the two-way transducer as explicit annotations of the input. Given a transducer $\mathcal{T}$ over an expanded input alphabet $\Sigma \times \Sigma'$ and an NFA $\mathcal{A}$ over $\Sigma \times \Sigma'$, we let

$$[\![\mathcal{T}]\!]_o \lfloor_\Sigma^{\mathcal{A}} \;=\; \big\{(u, \nu) \in \Sigma^* \times (\Gamma \times \mathbb{N})^* \;:\; \exists\, u' \in \Sigma'^{|u|} \;\; u \otimes u' \in [\![\mathcal{A}]\!],\; (u \otimes u', \nu) \in [\![\mathcal{T}]\!]_o\big\}.$$

In other words, $[\![\mathcal{T}]\!]_o \lfloor_\Sigma^{\mathcal{A}}$ is obtained from $[\![\mathcal{T}]\!]_o$ by restricting the inputs of $\mathcal{T}$ via $\mathcal{A}$, and then projecting them on Σ.

The following result is the key to reduce containment modulo bounded MSO resynchronizers to containment in the origin semantics.

▶ **Theorem 16.** *Given a bounded MSO resynchronizer R with m input parameters, a transducer $\mathcal{T}$ with input alphabet $\Sigma \times \Sigma'$ and an NFA $\mathcal{A}$ over $\Sigma \times \Sigma'$, one can construct a transducer $\mathcal{T}'$ with input alphabet $\Sigma \times \Sigma' \times \mathbb{B}^m$ and an NFA $\mathcal{A}'$ over $\Sigma \times \Sigma' \times \mathbb{B}^m$ such that*

$$[\![\mathcal{T}']\!]_o \lfloor_\Sigma^{\mathcal{A}'} \;=\; \big\{\sigma' \;:\; (\sigma, \sigma') \in [\![R]\!] \text{ for some } \sigma \in [\![\mathcal{T}]\!]_o \lfloor_\Sigma^{\mathcal{A}}\big\}.$$

Moreover, if R is fixed, $\mathcal{T}$ has n states and PSpace*-constructible transitions w.r.t. n, and $\mathcal{A}$ is* PSpace*-constructible w.r.t. n, then $\mathcal{T}'$ and $\mathcal{A}'$ have similar properties, namely, $\mathcal{T}'$ has a number of states polynomial in n and* PSpace*-constructible transitions w.r.t. n, and $\mathcal{A}'$ is* PSpace*-constructible w.r.t. n.*

Proof. To prove the claim, it is convenient to assume that R has no input nor output parameters. If this were not the case, we could modify $\mathcal{T}$ in such a way that it reads inputs over $\Sigma \times \Sigma' \times \mathbb{B}^m$, exposing a valuation of the parameters $\overline{I}$, and produces outputs over $\Gamma \times \mathbb{B}^n$, exposing a valuation $\overline{O}$. We could then apply the constructions that follow, and finally modify the resulting transducer $\mathcal{T}'$ by projecting away the input and output annotations.

We observe that the projection operation on the output is easier and can be implemented directly at the level of the transitions of $\mathcal{T}'$, while the projection of the input requires the use of the notation $[\![\mathcal{T}']\!]_o|_\Sigma^{\mathcal{A}'}$, for the reasons that we discussed earlier. In both cases the complexity bounds are preserved.

Let $R = (\alpha, \beta, \gamma, \delta)$ a bounded MSO resynchronizer with no input/output parameters. Since there are no existentially quantified parameters, $[\![R]\!]$ can be seen as the relational composition of four different resynchronizations, as induced by the formulas of R. Formally, we have $[\![R]\!] = [\![R_\alpha]\!] \circ [\![R_\beta]\!] \circ [\![R_\gamma]\!] \circ [\![R_\delta]\!]$, where $R_\alpha = (\alpha, \mathit{true}, \gamma_{id}, \delta_{\mathit{true}})$, $R_\beta = (\mathit{true}, \beta, \gamma_{id}, \delta_{\mathit{true}})$, $R_\gamma = (\mathit{true}, \mathit{true}, \gamma, \delta_{\mathit{true}})$, $R_\delta = (\mathit{true}, \mathit{true}, \gamma_{id}, \delta)$, $\gamma_{id}(\tau)(y, z) = (y = z)$ and $\delta_{\mathit{true}}(\tau, \tau')(z, z') = \mathit{true}$ for all output-types $\tau, \tau' \in \Gamma$. This means that, to prove the claim, it suffices to consider only one resynchronizer at a time among R_α, R_β, R_γ, R_δ. The case of R_α, R_β is quite straightforward, since it amounts to intersect the input and output with the regular language of α and β, respectively.

We now consider the most interesting case, that of $R_\gamma = (\mathit{true}, \mathit{true}, \gamma, \delta_{\mathit{true}})$, which modifies the origins. The case of δ is similar and can be found in the full version. The rough idea here is that $\mathcal{T}'$ has to simulate an arbitrary run of $\mathcal{T}$, by displacing the origins of any output letter with type τ from a source y to a target z, as indicated by the formula $\gamma(\tau)(x, y)$. Since a factor of an output of $\mathcal{T}$ that originates at the same input position can be broken up into multiple sub-factors with origins at different positions, here it is convenient to assume that $\mathcal{T}$ outputs at most a single letter at each transition. This assumption can be made without loss of generality, since we can reproduce any longer word v that is output by some transition $(q, i) \xrightarrow{a|L} (q', i')$ letter by letter, with several transitions that move back and forth around position i.

The idea of the construction is as follows. Whenever $\mathcal{T}$ outputs a letter b with origin in y, $\mathcal{T}'$ non-deterministically moves to some position z that, together with y, satisfies the formula $\gamma(b)$ (note that b is also the output-type of the produced letter). Then $\mathcal{T}'$ produces the same output b as $\mathcal{T}$, but at position z. Finally, it moves back to the original position y. For the latter step, we will exploit the fact that R is bounded.

Now, for the details, we construct from γ a finite monoid $(M, \cdot)$, a monoid morphism $h : (\Sigma \times \Sigma' \times \mathbb{B} \times \mathbb{B})^* \to M$, and some subsets F_τ of M_τ, for each output-type $\tau \in \Gamma$, such that $(u, x, y) \vDash \gamma(\tau)$ if and only if $h(u_{x,y}) \in F_\tau$, where $u_{x,y}$ is the encoding on u of the positions x and y, namely, $u_{x,y}(i) = \big((u(i), b_{i=x}, b_{i=y})\big)$ for all $1 \le i \le |u|$, and $b_{i=x}$ (resp. $b_{i=y}$) is either 1 or 0 depending on whether $i = x$ (resp. $i = y$) or not. Similarly, we denote by $u_{\emptyset,\emptyset}$ the encoding on u of two empty monadic predicates. For all $1 \le i \le j \le |u|$, we then define $\ell_i = h(u_{\emptyset,\emptyset}[1, i-1])$, $r_j = h(u_{\emptyset,\emptyset}[j+1, |u|])$, and $m_{i,j} = h(u_{i,j}[i, j])$. We observe that

$$(u, y, z) \vDash \gamma(\tau) \qquad \text{iff} \qquad \begin{cases} \ell_y \cdot m_{y,z} \cdot r_z \in F_\tau & \text{if } y \le z \\ \ell_z \cdot m_{z,y} \cdot r_y \in F_\tau & \text{if } y > z. \end{cases}$$

The elements ℓ_i and r_i associated with each position i of the input u are functionally determined by u. In particular the word $\ell_1 \dots \ell_{|u|}$ (resp. $r_1 \dots r_{|u|}$) can be seen as the run of a deterministic (resp. co-deterministic) automaton on u. Without loss of generality, we can assume that the values ℓ_i and r_i are readily available as annotations of the input at position i, and checked by means of a suitable refinement $\mathcal{A}'$ of the NFA $\mathcal{A}$.

We now describe how $\mathcal{T}'$ simulates a transition of $\mathcal{T}$, say $q \xrightarrow{a|b} q'$, that originates at a position y and produces the letter b (the simulation of a transition with empty output is straightforward). The transducer $\mathcal{T}'$ stores in its control state the transition rule to be simulated and the monoid element ℓ_y associated with the current position y (the source). It then guesses whether the displaced origin z (i.e. the target) is to the left or to the right of y.

We only consider the case where $z \geq y$ (the case $z < y$ is symmetric). In this case $\mathcal{T}'$ starts moving to the right, until it reaches some position $z \geq y$ such that $(u, y, z) \vDash \gamma(b)$ (as we explained earlier, this condition is equivalent to checking that $\ell_y \cdot m_{y,z} \cdot r_z \in F_b$). Once a target z is reached, $\mathcal{T}'$ produces the same output b as the original transition, and begins a new phase for backtracking to the source y. During this phase, the transducer will maintain the previous monoid elements ℓ_y, $m_{y,z}$, and, while moving leftward, compute $m_{z',z}$ for all $z' \leq z$. We claim that there is a unique z' such that $\ell_{z'} = \ell_y$ and $m_{z',z} = m_{y,z}$, and hence such z' must coincide with the source y. Indeed, if this were not the case, we could pump the factor of the input between the correct source y and some $z' \neq y$, showing that the MSO resynchronizer R_γ is not bounded. Based on this, the transducer $\mathcal{T}'$ can move back to the correct source y, from which it can then simulate the change of control state from q to q' and move to the appropriate next position. Any run of $\mathcal{T}'$ that simulates a run of $\mathcal{T}$ on input u, as described above, results in producing the same output v as $\mathcal{T}$, but with the origin mapping modified from $i \mapsto y_i$ to $i \mapsto z_i$, for all $1 \leq i \leq |v|$ and for some $1 \leq y_i, z_i \leq |u|$ such that $(u, y_i, z_i) \vDash \gamma(v(i))$. In other words, we have $[\![\mathcal{T}']\!]_o = \{\sigma' : (\sigma, \sigma') \in [\![R]\!], \sigma \in [\![\mathcal{T}]\!]_o\}$.

With the above constructions, if R is fixed and if $\mathcal{T}$ has n states and PSPACE-constructible transitions w.r.t. n, then similarly $\mathcal{T}'$ has a number of states polynomial in n and PSPACE-constructible transitions w.r.t. n. Finally, a PSPACE-constructible NFA $\mathcal{A}'$ can be obtained from a direct product of the NFA $\mathcal{A}, \mathcal{U}$, and a suitable NFA for checking inputs annotated with the monoids elements ℓ_z, r_z. ◀

The above result is used to reduce the containment problem modulo a bounded MSO resynchronizer R to a containment problem with origins (relativized to correctly annotated inputs). That is, if $\mathcal{T}_1, \mathcal{T}_2$ are transducers with input alphabet Σ, and $\mathcal{T}_2', \mathcal{A}'$ are over the input alphabet $\Sigma \times \Sigma'$ and constructed from $\mathcal{T}_2$ using Theorem 16, then

$$\mathcal{T}_1 \subseteq_R \mathcal{T}_2 \quad \text{iff} \quad [\![\mathcal{T}_1]\!]_o \subseteq [\![\mathcal{T}_2']\!]_o \downarrow_\Sigma^{\mathcal{A}'} \quad \text{iff} \quad [\![\mathcal{T}_1]\!]_o \uparrow^{\Sigma \times \Sigma'} \cap R' \subseteq [\![\mathcal{T}_2']\!]_o \cap R'$$

where $R' = [\![\mathcal{A}']\!] \times (\Gamma \times \mathbb{N})^*$ and $\uparrow^{\Sigma'}$ is the inverse of the input-projection operation, i.e. $[\![\mathcal{T}]\!]_o \uparrow^{\Sigma \times \Sigma'} = \{(u \otimes u', \nu) \in (\Sigma \times \Sigma')^* \times (\Gamma \times \mathbb{N})^* : u \otimes u' \in [\![\mathcal{A}]\!], (u, \nu) \in [\![\mathcal{T}]\!]_o\}$. We also recall from Section 3 that the latter containment reduces to emptiness of a PSPACE-constructible NFA, which can then be decided in PSPACE w.r.t. the sizes of $\mathcal{T}_1$ and $\mathcal{T}_2$. We thus conclude with the following result:

▶ **Corollary 17.** *The problem of deciding whether $\mathcal{T}_1 \subseteq_R \mathcal{T}_2$, for any pair of transducers $\mathcal{T}_1, \mathcal{T}_2$ and for a fixed bounded MSO resynchronizer R, is* PSPACE*-complete.*

5 Conclusions

We studied the equivalence and containment problems for non-deterministic, two-way word transducers in the origin semantics, and proved that the problems are decidable in PSPACE, which is the lowest complexity one could expect given that equivalence and containment of NFA are already PSPACE-hard. This result can be contrasted with the undecidability of equivalence of non-deterministic, one-way word transducers in the classical semantics.

We have also considered a variant of containment up to "distortions" of the origin, called containment modulo a resynchronization R, and denoted $\subseteq_R$. We identified a broad class of resynchronizations, definable in MSO, and established decidability of the induced containment problem. In fact, we obtained an optimal fixed-parameter complexity result: testing a containment $\mathcal{T}_1 \subseteq_R \mathcal{T}_2$ modulo a bounded MSO resynchronization R is PSPACE-complete in the size of the input transducers $\mathcal{T}_1, \mathcal{T}_2$, where the fixed parameter is the size of the formulas used to describe R.

Our logical definition of resynchronizations talks implicitly about origin graphs. Since we cannot encode the origin semantics of arbitrary two-way transducers by words, we have chosen to work directly with origin graphs defined by two-way transducers. A classical way to define resynchronizations of origin graphs would be to use logical formalisms for graph transformations. Unfortunately, the classical MSO approach [7, 13] does not work in our setting, since satisfiability of MSO over origin graphs of two-way transducers is already undecidable, which means that the definable resynchronizations would not be realizable by two-way transducers.

Another possibility would be to use a decidable logic over origin graphs, like the one introduced by Filiot et al. in [10]. Their logic is not suited either for our purposes, since it allows predicates of arbitrary arity defined using MSO over the input word. The reason is that single head devices would not be able to move between the tuples of positions related by those definable predicates, and thus, in particular, we would not be able to guarantee that the definable resynchronizations are realizable by two-way transducers.

Yet an alternative approach to the above problem consists in viewing tagged outputs as data words, and using transducers to transform data words. Durand-Gasselin and Habermehl introduced in [11] a framework for transformations of data words. However, this approach would be unsatisfactory, because the transformation does not take the input into account.

We conjecture that the bounded MSO resynchronizers defined here strictly capture the rational resynchronizers introduced in [15]. In particular, although MSO resynchronizers do not have the ability to talk about general origin graphs, they presumably can describe "regular" origin graphs of one-way transducers, i.e., graphs expressed by regular languages over sequences that alternate between the input and the output word. We also recall that the MSO resynchronizer $R_{1st-to-last}$ from Example 13.5 contains only origin graphs of one-way transducers, but cannot be defined by a rational resynchronizer.

Another natural question that we would like to answer concerns compositionality: are bounded MSO resynchronizers closed under relational composition? In other words, given two bounded MSO resynchronizers R, R', is it possible to find (possibly effectively) a bounded MSO resynchronizer R'' such that $[\![R'']\!] = [\![R]\!] \circ [\![R']\!]$?

References

1 A.V. Aho, J.E. Hopcroft, and J.D. Ullman. A general theory of translation. *Math. Syst. Theory*, 3(3):193–221, 1969.

2 Rajeev Alur and Pavel Cerný. Expressiveness of streaming string transducer. In *Proc. of FSTTCS'10*, volume 8 of *LIPIcs*, pages 1–12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010.

3 Rajeev Alur and Jyotirmoy Deshmukh. Nondeterministic streaming string transducers. In *Automata, Languages and Programming - 38th International Colloquium (ICALP'11)*, volume 6756 of *LNCS*. Springer, 2011.

4 Jean Berstel. *Transductions and context-free languages*. Teubner Studienbücher Stuttgart, 1979.

5 Mikolaj Bojańczyk. Transducers with origin information. In *ICALP'14*, LNCS, pages 26–37. Springer, 2014.

6 Mikolaj Bojańczyk, Laure Daviaud, Bruno Guillon, and Vincent Penelle. Which Classes of Origin Graphs Are Generated by Transducers? In *ICALP'17*, volume 80 of *LIPIcs*, pages 114:1–114:13, 2017.

7 Bruno Courcelle. The Expression of Graph Properties and Graph Transformations in Monadic Second-Order Logic. In G. Rozenberg, editor, *Handbook of Graph Transformations: Foundations*, volume 1, pages 165–254. World Scientific, 1997.

8 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic. A language-theoretic approach*. Encyclopedia of Mathematics and its applications, Vol. 138. Cambridge University Press, 2012.

9 Karel Culik II and Juhani Karhumäki. The Equivalence Problem for Single-Valued Two-Way Transducers (on NPDT0L Languages) is Decidable. *SIAM J. Comput*, 16(2):221–230, 1987.

10 Luc Dartois, Emmanuel Filiot, and Nathan Lhote. Logics for Word Transductions with Synthesis. In *ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 295–304. ACM, 2018.

11 Antoine Durand-Gasselin and Peter Habermehl. Regular transformations of data words through origin information. In *Proc. of FoSSaCS'15*, LNCS, pages 285–300. Springer, 2016.

12 Samuel Eilenberg. *Automata, languages, and machines*. Academic press, 1974.

13 Joost Engelfriet. Context-free graph grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 125–213. Springer, 1997.

14 Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Logic*, 2(2):216–254, 2001.

15 Emmanuel Filiot, Ismaël Jecker, Christof Löding, and Sarah Winter. On Equivalence and Uniformisation Problems for Finite Transducers. In *ICALP'16*, volume 55 of *LIPIcs*, pages 125:1–125:14, 2016.

16 Emmanuel Filiot, Sebastian Maneth, Pierre-Alain Reynier, and Jean-Marc Talbot. Decision problems of tree transducers with origin. *Inf. Comput.*, 261:311–335, 2018.

17 Emmanuel Filiot and Pierre-Alain Reynier. Transducers, logic and algebra for functions of finite words. *ACM SIGLOG News*, pages 4–19, 2016.

18 T. V. Griffiths. The unsolvability of the equivalence problem for lambda-free nondeterministic generalized machines. *J. ACM*, 15(3):409–413, 1968.

19 M.-P. Schützenberger. A Remark on Finite Transducers. *Information and Control*, 4(2-3):185–196, 1961.

20 J.C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM J. Res. Dev.*, 3(2):198–200, 1959.

21 Moshe Y. Vardi. A Note on the Reduction of Two-Way Automata to One-Way Automata. *Information Processing Letters*, 30(5):261–264, 1989.

22 Andreas Weber and Helmut Seidl. On the Degree of Ambiguity of Finite Automata. *Theor. Comput. Sci.*, 88(2):325–349, 1991.

Constant Factor Approximation Algorithm for Uniform Hard Capacitated Knapsack Median Problem

Sapna Grover
Department of Computer Science, University of Delhi, Delhi, India
sgrover@cs.du.ac.in

Neelima Gupta
Department of Computer Science, University of Delhi, Delhi, India
ngupta@cs.du.ac.in

Samir Khuller
Department of Computer Science, University of Maryland, College Park, MD 20742, USA
samir@cs.umd.edu

Aditya Pancholi
Department of Computer Science, University of Delhi, Delhi, India
apancholi@cs.du.ac.in

Abstract

In this paper, we give the first constant factor approximation algorithm for capacitated knapsack median problem (CKnM) for hard uniform capacities, violating the budget by a factor of $1+\epsilon$ and capacities by a $2+\epsilon$ factor. To the best of our knowledge, no constant factor approximation is known for the problem even with capacity/budget/both violations. Even for the uncapacitated variant of the problem, the natural LP is known to have an unbounded integrality gap even after adding the covering inequalities to strengthen the LP. Our techniques for CKnM provide two types of results for the capacitated k-facility location problem. We present an $O(1/\epsilon^2)$ factor approximation for the problem, violating capacities by $(2+\epsilon)$. Another result is an $O(1/\epsilon)$ factor approximation, violating the capacities by a factor of at most $(1+\epsilon)$ using at most $2k$ facilities for a fixed $\epsilon > 0$. As a by-product, a constant factor approximation algorithm for capacitated facility location problem with uniform capacities is presented, violating the capacities by $(1+\epsilon)$ factor. Though constant factor results are known for the problem without violating the capacities, the result is interesting as it is obtained by rounding the solution to the natural LP, which is known to have an unbounded integrality gap without violating the capacities. Thus, we achieve the best possible from the natural LP for the problem. The result shows that the natural LP is not too bad.

2012 ACM Subject Classification Theory of computation → Facility location and clustering, Theory of computation → Rounding techniques

Keywords and phrases Capacitated Knapsack Median, Capacitated k -Facility Location

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.23

1 Introduction

Facility location and k-median problems are well studied in the literature. In this paper, we study some of their generalizations. In particular, we study capacitated variants of the knapsack median problem (KnM) and the k facility location problem (kFLP). Knapsack median problem is a generalization of the k-median problem, in which we are given a set $\mathcal{C}$

of clients with demands, a set $\mathcal{F}$ of facility locations and a budget $\mathcal{B}$. Setting up a facility at location i incurs cost f_i (called the *facility opening cost* or simply the *facility cost*) and servicing a client j by a facility i incurs cost $c(i,\ j)$ (called the *service cost*). We assume that the costs are metric i.e., they satisfy the triangle inequality. The goal is to select the locations to install facilities, so that the total cost for setting up the facilities does not exceed $\mathcal{B}$ and the cost of servicing all the clients by the opened facilities is minimized. When $f_i = 1\ \forall i \in \mathcal{F}$ and $\mathcal{B} = k$, it reduces to the k-median problem. In the *capacitated* version of the problem, we are also given a bound u_i on the maximum number of clients that facility i can serve. Given a set of open facilities, an assignment problem is solved to determine the best way of servicing the clients. Thus any solution is completely determined by the set of open facilities. In this paper, we address the capacitated knapsack median (CKnM) problem with uniform capacities i.e., $u_i = u\ \forall i \in \mathcal{F}$ and clients with unit demands. In particular, we present the following result:

▶ **Theorem 1.** *There is a polynomial time algorithm that approximates hard uniform capacitated knapsack median problem within a constant factor violating the capacity by a factor of at most* $(2+\epsilon)$ *and budget by a factor of at most* $(1+\epsilon)$*, for every fixed* $\epsilon > 0$*.*

Our result is nearly the best achievable from rounding the natural LP: we cannot expect to get rid of the violation in the budget as it would imply a constant factor integrality gap for the uncapacitated case which is known to have an unbounded integrality gap. Even with budget violation, capacity violation cannot be reduced to below 2 as it would imply less than 2 factor capacity violation for k-median problem with $k+1$ facilities. The natural LP has an unbounded integrality gap for this scenario as well[1] [2].

The k-facility location problem (kFLP) is a common generalization of the facility location problem and the k-median problem. In kFLP, we are given a bound k on the maximum number of facilities that can be opened (instead of a budget on the total facility opening cost) and the objective is to minimize the total of facility opening cost and the cost of servicing the clients by the opened facilities. In particular we present the following two results:

▶ **Theorem 2.** *There is a polynomial time algorithm that approximates hard uniform capacitated k-facility location problem within a constant factor* $(O(1/\epsilon^2))$ *violating the capacities by a factor of at most* $(2+\epsilon)$ *for every fixed* $\epsilon > 0$*.*

▶ **Theorem 3.** *There is a polynomial time algorithm that approximates hard uniform capacitated k-facility location problem within a constant factor* $(O(1/\epsilon))$ *violating the capacity by a factor of at most* $(1+\epsilon)$ *using at most* $2k$ *facilities for every fixed* $\epsilon > 0$*.*

As a particular case of CkFLP, we obtain the following interesting result for the capacitated facility location problem (CFLP):

---

[1] Let M be a large integer, $u_i = M$ and $k = 2M-2$. There are M groups of locations; distance between locations within a group is 0 and distance between locations in two different groups is 1. Each group has $2M-2$ facilities and $2M-2$ clients, all co-located. In an optimal LP solution each facility is opened to an extent of $1/M$ thereby creating a capacity of $2M-2$ within each group. In an integer solution, if at most $k+1 = 2M-1$ facilities are allowed to be opened then there is at least one group with only one facility opened in it. Thus capacity in the group is M whereas the demand is $2M-2$. Thus the blowup in capacity is $(2M-2)/M$.

[2] We thank Moses Charikar for providing the above example where violation in one of the parameters is less than 2 factor and no violation in the other. The example was subsequently modified by us to allow $k+1$ facilities.

▶ **Corollary 4.** *There is a polynomial time algorithm that approximates hard uniform capacitated facility location problem within a constant factor* $(O(1/\epsilon))$ *violating the capacity by a factor of at most* $(1+\epsilon)$ *for every fixed* $\epsilon > 0$.

The standard LP is known to have an unbounded integrality gap for CFLP even with uniform capacities. Though constant factor results are known for the problem without violating the capacities [2, 4], our result is interesting as it is obtained by rounding the solution to the natural LP. Our result shows that the natural LP is not too bad.

1.1 Motivation and Challenges

The natural LP for KnM is known to have an unbounded integrality gap [10] even for the uncapacitated case. Obtaining a constant factor approximation for the (capacitated) k-median (CkM) problem is still open, let alone the CKnM problem. Existing solutions giving constant-factor approximation for CkM violate at least one of the two (*cardinality* and *capacity*) constraints. Natural LP is known to have an unbounded integrality gap when any one of the two constraints is allowed to be violated by a factor of less than 2 without violating the other.

Several results [9, 11, 6, 21, 16, 1] have been obtained for CkM that violate either the capacities or the cardinality by a factor of 2 or more. The techniques used for CkM cannot be used for CKnM as they work by transferring the opening from one facility to another (ensuring bounded service cost) facility thereby maintaining the cardinality within claimed bounds. This works well when there are no facility opening costs or the (facility opening) costs are uniform. For the general opening costs, this is a challenge as a facility, good for bounded service cost, may lead to budget violation. To the best of our knowledge, capacitated knapsack median problem has not been addressed earlier.

CkFLP is NP-hard even when there is only one client and there are no facility costs [1]. The hardness results for CkM hold for CkFLP as well. On the other hand, standard LP for capacitated facility location problem (CFLP) has an unbounded integrality gap, thereby implying that constant integrality ratio can not be obtained for CkFLP without violating the capacities even if $k = n$. Byrka *et al.* [6] gave an $O(1/\epsilon^2)$ algorithm for CkFLP when the capacities are uniform (UCkFLP) violating the capacities by a factor of $2+\epsilon$. They use randomized rounding to bound the expected cost. It can be shown that deterministic pipage rounding cannot be used here. The strength of our techniques is demonstrated in obtaining the first deterministic constant factor approximation with the same capacity violation. The primary source of inspiration for our result in Theorem 3 comes from its corollary.

1.2 Related Work

Capacitated k-median problem has been studied extensively in the literature. For the case of uniform capacities, several results [6, 9, 11, 21, 16] have been obtained that violate either the capacities or the cardinality by a factor of 2 or more. In case of non-uniform capacities, a $(7+\epsilon)$ algorithm was given by Aardal *et al.* [1] violating the cardinality constraint by a factor of 2 as a special case of Capacitated k-FLP when the facility costs are all zero. Byrka *et al.* [6] gave an $O(1/\epsilon)$ approximation result violating capacities by a factor of $(3+\epsilon)$.

Li [22] broke the barrier of 2 in cardinality and gave an $\exp(O(1/\epsilon^2))$ approximation using at most $(1+\epsilon)k$ facilities for uniform capacities. Li gave a sophisticated algorithm using a novel linear program which he calls the *rectangle LP*. The result was extended to non-uniform capacities by the same author using a new LP called *configuration LP* [23]. The

approximation ratio was also improved from $\exp(O(1/\epsilon^2))$ to $(O(1/\epsilon^2 \log(1/\epsilon)))$. Though the algorithm violates the cardinality only by $1+\epsilon$, it introduces a softness bounded by a factor of 2. The running time of the algorithm is $n^{O(1/\epsilon)}$.

Byrka *et al.* [8] broke the barrier of 2 in capacities and gave an $O(1/\epsilon^2)$ approximation violating capacities by a factor of $(1+\epsilon)$ factor for uniform capacities. The algorithm uses randomized rounding to round a fractional solution to the configuration LP. For non-uniform capacities, a similar result has been obtained by Demirci *et al.* [14]. The paper presents an $O(1/\epsilon^5)$ approximation algorithm with capacity violation by a factor of at most $(1+\epsilon)$. The running time of the algorithm is $n^{O(1/\epsilon)}$.

Another closely related problem to Capacitated k-median problem is the Capacitated k-center problem, where-in we have to minimize the maximum distance of a client to a facility. A 6 factor approximation algorithm was given by Khuller and Sussmann [15] for the case of uniform hard capacities (5 factor for soft capacitated case). For non-uniform hard capacities, Cygan *et al.* [13] gave the first constant approximation algorithm for the problem, which was further improved by An *et al.* in [3] to 9 factor.

Though the knapsack median problem (a.k.a. weighted W-median) is a well motivated problem and occurs naturally in practice, not much work has been done on the problem. Krishnaswamy *et al.* [17] showed that the integrality gap, for the uncapacitated case, holds even on adding the covering inequalities to strengthen the LP, and gave a 16 factor approximation that violates the budget constraint by a factor of $(1+\epsilon)$. Kumar [19] strengthened the natural LP by obtaining a bound on the maximum distance a client can travel and gave first constant factor approximation without violating the budget constraint. Charikar and Li [12] reduced the large constant obtained by Kumar to 34 which was further improved to 32 by Swamy [26]. Byrka *et al.* [7] extended the work of Swamy and applied sparsification as a pre-processing step to obtain a factor of 17.46. The result was further improed to $7.081(1+\epsilon)$ very recently by Krishnaswamy *et al.* [18] using iterative rounding technique, with a running time of $n^{O(1/\epsilon^2)}$.

For CkFLP, Aardal *et al.* [1] extended the FPTAS for knapsack problem to give an FPTAS for single client CkFLP. They also extend an $\alpha-$ approximation algorithm for (uncapacitated) k-median to give a $(2\alpha+1)-$ approximation for CkFLP with uniform opening costs using at most $2k$ for non-uniform and $2k-1$ for uniform capacities. Byrka *et al.* [6] gave an $O(1/\epsilon^2)$ factor approximation violating the capacities by a factor of $(2+\epsilon)$ using dependent rounding.

For CFLP, An, Singh and Svensson [4] gave the first LP-based constant factor approximation by strengthening the natural LP. Other LP-based algorithms known for the problem are due to Byrka *et al.* and Levi *et al.* ([6, 20]). The local search technique has been particularly useful to deal with capacities. The approach provides 3 factor for uniform capacities [2] and 5 factor for the non-uniform case [5].

1.3 Our techniques

We extend the work of Krishnaswamy *et al.* [17] to capacitated case. The major challenge is in writing the LP which opens sufficient number of facilities for us in bounded cost.

Filtering and clustering techniques [24, 11, 20, 25, 6, 17, 1] are used to partition the set of facilities and demands. Routing trees are used to bound the assignment costs. Main contribution of this work is a new LP and an iterative rounding algorithm to obtain a solution with at most two fractionally opened facilities.

High Level Ideas. We first use the filtering and clustering techniques to partition the set of facilities and demands. Each partition (*called cluster*) has sufficient opening ($\geq 1-1/\ell \geq 1/2$) for a fixed parameter $\ell \geq 2$ in it. An integrally open solution is obtained where-in some

clusters have at least 1 integrally opened facility and some do not have any facility opened in them. To assign the demand of the cluster that cannot be satisfied locally within the cluster, a (directed) rooted binary routing tree is constructed, on the cluster centers. If (s, t) is an edge in the routing tree then the cost of sending the unmet demand of the cluster centered at s to t is bounded. The edges of the tree have non-increasing costs as we go up the tree, with the root being at the top. Hence the cost of sending the unmet demand of the cluster centered at s to any node r up in the tree at a constant number of edges away from s is bounded.

In order to decide which facilities to open integrally, clusters are grouped into meta-clusters of size (the number of clusters in it) ℓ so as to have at least $\ell - 1$ opening in it. The routing tree is used to group the clusters into meta-clusters (MCs) in a top-down greedy manner, i.e., starting from the root, a meta-cluster grows by including the cluster (center) that connects to it by the cheapest edge. A MC grows until its size reaches ℓ. We then proceed to make a new MC from the tree with the remaining nodes in the same greedy manner. This imposes a natural directed (not necessarily binary) rooted tree structure on the meta-clusters with the property that the edge going out of a MC is cheaper than the edges inside the MC which are further cheaper than the edges coming into the MC. Out-degree of a MC is 1 whereas the in-degree is at most $q + 1$ where q is the number of clusters in a MC.

Next, we write a new LP to open sufficient number of facilities within each cluster and each MC. We also give an iterative rounding algorithm to solve the LP, removing the integral variables and updating the constraints accordingly in each iteration until either all the variables are fractional or all are integral. In case all the variables are fractional, we use the property of extreme point solutions to claim that the number of non-integral variables is at most two. Thus we obtain a solution to the LP with at most two fractional openings. Both the fractionally opened facilities are opened integrally at a loss of additive f_{max} in the budget where f_{max} is the maximum facility opening cost [3].

Finally a min-cost flow problem is solved with capacities scaled up by a factor of $(2+\epsilon)$ to obtain an integral assignment. A feasible solution to the min-cost flow problem of bounded cost is obtained as follows: consider a scenario in which the demand accumulated within each cluster is less than u (we call such clusters as *sparse*). For the sake of easy exposition of the ideas, let each MC be of size exactly ℓ. The LP solution opens at least $\ell - 1$ facilities integrally in each MC, with at least one facility in each cluster except for one cluster. If the cluster with unmet demand is at the root of the induced subgraph of the MC, then its demand cannot be met within the MC. We make sure that such a demand is served in the parent MC. Total demand to be served by the facilities in a MC is at most ℓu plus at most $(\ell + 1)u$ coming from the children of the MC. Thus $(\ell - 1)$ facilities have to serve at most $(2\ell + 1)u$ demand leading to a violation of $(2 + O(1/\ell))$ in capacity. Demands have to travel $O(\ell)$ edges upwards (at most ℓ within its own MC and at most ℓ in the parent MC), and hence the cost of serving them is bounded.

The situation becomes a little tricky when there are clusters with more than u demand (we call such clusters as *dense*). One way to deal with dense clusters is to open $\lfloor demand/u \rfloor$ facilities integrally within such a cluster and assign the residual demand to one of them at a capacity violation of 2. But if this cluster also has to serve u units of unmet demand of one of its children (we will see later that a dense cluster has at most one child), the capacity violation could blow upto 3 in case $\lfloor demand/u \rfloor = 1$. We deal with this scenario carefully.

[3] Let F' be the set of facilities i with $f_i > \epsilon \cdot \mathcal{B}$. Enumerate all possible subsets of F' of $size <= 1/\epsilon$. There are at most $n^{O(1/\epsilon)}$ such sets. For each such set S, solve the LP with $y_i = 1 \ \forall \ i \in S$ and $y_i = 0 \ \forall \ i \in F' \setminus S$. The additive f_{max} (which comes from the fractionally opened facilities) is $<= \epsilon \cdot \mathcal{B}$. Choose the best solution and hence theorem 1 follows.

Algorithm 1 Cluster Formation.

$\mathcal{C}' \leftarrow \emptyset$, $S \leftarrow \mathcal{C}$, $ctr(j) = \emptyset\ \forall j \in S$.
while $S \neq \emptyset$ **do**
  Pick $j' \in S$ with the smallest radius $\mathcal{R}_{j'}$ in S, breaking ties arbitrarily.
  $S \leftarrow S \setminus \{j'\}$, $\mathcal{C}' \leftarrow \mathcal{C}' \cup \{j'\}$
  while $\exists j \in S$: $c(j',\ j) \leq 2\ell\hat{C}_j$ **do**
    $S \leftarrow S \setminus \{j\}$, $ctr(j) = j'$
  end while
end while
$\forall j' \in \mathcal{C}'$: let $\mathcal{N}_{j'} = \{i \in \mathcal{F} \mid \forall k' \in \mathcal{C}' : j' \neq k' \Rightarrow c(i,\ j') < c(i,\ k')\}$

2 Capacitated Knapsack Median Problem

In this section, we consider the capacitated knapsack median problem. CKnM can be formulated as the following integer program (IP):

$$
\begin{aligned}
&\textit{Minimize} \quad \mathcal{C}ostKnM(x,y) = \textstyle\sum_{j\in\mathcal{C}} \sum_{i\in\mathcal{F}} c(i,\ j)x_{ij} \\
&\textit{subject to} \quad \textstyle\sum_{i\in\mathcal{F}} x_{ij} = 1 && \forall\ j \in \mathcal{C} && (1)\\
&\textstyle\sum_{j\in\mathcal{C}} x_{ij} \leq u\ y_i && \forall\ i \in \mathcal{F} && (2)\\
&x_{ij} \leq y_i && \forall\ i \in \mathcal{F},\ j \in \mathcal{C} && (3)\\
&\textstyle\sum_{i\in\mathcal{F}} f_i y_i \leq \mathcal{B} && && (4)\\
&y_i,\ x_{ij} \in \{0,1\} && && (5)
\end{aligned}
$$

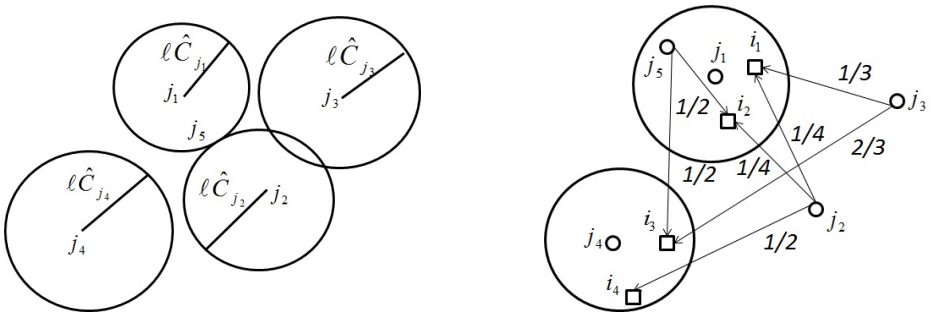
LP-Relaxation of the problem is obtained by allowing the variables $y_i, x_{ij} \in [0,1]$. Call it LP_1. To begin with, we guess the facility with maximum opening cost, f^*_{max}, in the optimal solution and remove all the facilities with facility cost $> f^*_{max}$ before applying the algorithm. For the easy exposition of ideas, we will give a weaker result, in section 2.4, in which we violate capacities by a factor of 3. Most of the ideas are captured in this section.

2.1 Simplifying the problem instance

We first simplify the problem instance by partitioning the sets of facilities and clients into clusters. This is achieved using the filtering technique of Lin and Vitter [24]. For an LP solution $\rho =< x, y >$ and a subset T of facilities, let $size(y,\ T) = \sum_{i\in T} y_i$ denote the total extent up to which facilities are opened in T under ρ.

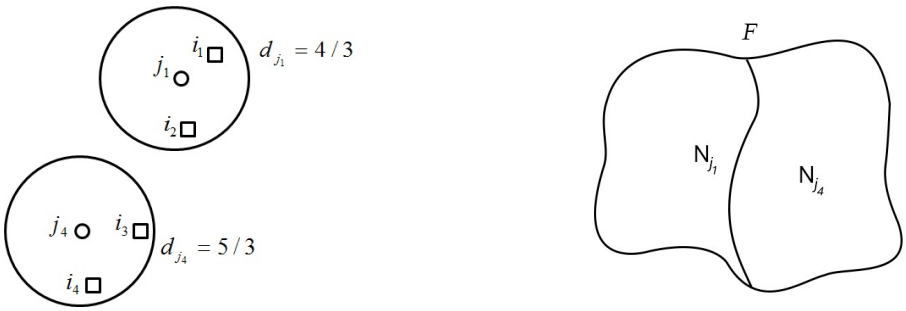
Partitioning the set of facilities into clusters and sparsifying the client set. Let $\rho^* =< x^*, y^* >$ denote the optimal LP solution. Let $\hat{C}_j$ denote the average connection cost of a client j in ρ^* i.e., $\hat{C}_j = \sum_{i\in\mathcal{F}} x^*_{ij} c(i,\ j)$. Let $\ell \geq 2$ be a fixed parameter and *ball(j)* be the set of facilities within a distance of $\ell\hat{C}_j$ of j *i.e.*, *ball(j)* $= \{i \in \mathcal{F} : c(i,\ j) \leq \ell\hat{C}_j\}$ (Figure 1(a)). Then, $size(y^*,\ ball(j)) \geq 1 - \frac{1}{\ell}$. Let $\mathcal{R}_j = \ell\hat{C}_j$ denote the *radius* of *ball(j)*. We identify a set $\mathcal{C}'$ of clients (Figure 1(b)) which will serve as the centers of the clusters using Algorithm 1. Note that *ball(j')* $\subseteq \mathcal{N}_{j'}$ and the sets $\mathcal{N}_{j'}$ partition $\mathcal{F}$. (Figure 2(b)).

Partitioning the demands. Let l_i denote the total demand of clients in $\mathcal{C}$ serviced by facility i i.e., $l_i = \sum_{j\in\mathcal{C}} x^*_{ij}$ and, $d_{j'} = \sum_{i\in\mathcal{N}_{j'}} l_i$ for $j' \in \mathcal{C}'$. Move the demand $d_{j'}$ to the center j' of the cluster (Figures 1-(b) and 2-(a)). For $j \in \mathcal{C}$, let $\mathcal{A}_{\rho^*}(j, \mathcal{N}_{j'})$ denote the total extent upto



(a) (b)

Figure 1 (a) The balls around the clients. (b) Reduced set of clients and assignment by LP solution.



(a) (b)

Figure 2 (a) Partitioning of demand. (b) Partition of $\mathcal{F}$.

which j is served by the facilities in $\mathcal{N}_{j'}$. Then, we can also write $d_{j'} = \sum_{j \in \mathcal{C}} \mathcal{A}_{\rho^*}(j, \mathcal{N}_{j'})$. Thus, after this step, unit demand of any $j \in \mathcal{C}$, is distributed to centers of all the clusters whose facilities serve j. In particular, it takes care of the demand of the clients that were removed during sparsification. Each cluster center is then responsible for the portion of demand of $j \in \mathcal{C}$ served by the facilities in its cluster.

The cost of moving the demand $d_{j'}$ to j' is bounded by $2(\ell + 1)LP_{opt}$ as shown in Corollary 6. Also, any two cluster centers j' and k' satisfy the *separation property*: $c(j',\ k') > 2\ell\ max\{\hat{C_{j'}}, \hat{C_{k'}}\}$. In addition, they satisfy Lemmas (5), (7) and (8).

▶ **Lemma 5.** *Let $j' \in \mathcal{C}'$ and $i \in \mathcal{N}_{j'}$ then, (i) For $k' \in \mathcal{C}'$, $c(j',\ k') \leq 2c(i,\ k')$, (ii) For $j \in \mathcal{C} \setminus \mathcal{C}'$, $c(j',\ j) \leq 2c(i,\ j) + 2\ell\hat{C}_j$ and (iii) For $j \in \mathcal{C}$, $c(i,\ j') \leq c(i,\ j) + 2\ell\hat{C}_j$.*

Proof.

i) By triangle inequality, $c(j', k') \leq c(i, j') + c(i, k')$. Since $i \in \mathcal{N}_{j'} \Rightarrow c(i, j') \leq c(i, k')$ and hence $c(j', k') \leq 2c(i, k')$.

ii) Since $j \notin \mathcal{C}'$, there exist a client $k' \in \mathcal{C}'$ such that $ctr(j) = k'$ and $c(j, k') \leq 2\ell\hat{C}_j$. Also, If $k' = j'$ then $c(i, j') = c(i, k')$ else $c(i, j') \leq c(i, k')$ because $i \in \mathcal{N}_{j'}$ and not $\mathcal{N}_{k'}$. Then, by triangle inequality, $c(i, k') \leq c(i, j) + c(j, k') \leq c(i, j) + 2\ell\hat{C}_j = c(i, j) + 2\mathcal{R}_j$. Therefore, $c(j', j) \leq c(i, j') + c(i, j) \leq 2c(i, j) + 2\mathcal{R}_j$.

iii) Consider two cases: $j \in \mathcal{C}'$ and $j \notin \mathcal{C}'$. In the first case, $c(i, j') \leq c(i, j)$ because $i \in \mathcal{N}_{j'}$ and not $\mathcal{N}_j$ and hence $c(i, j') \leq c(i, j) + 2\ell\hat{C}_j$. In the latter case, by triangle

inequality we have, $c(i,j') \leq c(i,j) + c(j',j)$. Since $j \notin \mathcal{C}' \Rightarrow c(j',j) \leq 2\ell\hat{C}_j$. Thus, $c(i,j') \leq c(i,j) + 2\ell\hat{C}_j$. ◀

▶ **Corollary 6.** $\sum_{j\in\mathcal{C}} \sum_{j'\in\mathcal{C}'} c(j',\ j)\mathcal{A}_{\rho^*}(j,\mathcal{N}_{j'}) \leq 2(\ell+1)LP_{opt}$.

▶ **Lemma 7.** *Let $j \in \mathcal{C}\setminus\mathcal{C}'$ and $j' \in \mathcal{C}'$ such that $c(j',\ j) \leq \mathcal{R}_{j'}$, then $\mathcal{R}_{j'} \leq 2\mathcal{R}_j$.*

Proof. Suppose, if possible, $\mathcal{R}_{j'} > 2\mathcal{R}_j$. Let $ctr(j) = k'$. Then, $c(j,\ k') \leq 2\mathcal{R}_j$. And, $c(k',\ j') \leq c(k',\ j) + c(j,\ j') \leq 2\mathcal{R}_j + \mathcal{R}_{j'} < 2\mathcal{R}_{j'} = 2\ell\hat{C}_{j'}$, which is a contradiction to separation property. ◀

▶ **Lemma 8.** $\sum_{j'\in\mathcal{C}'} d_{j'} \sum_{i\in\mathcal{F}} c(i,\ j')x^*_{ij'} \leq 3\sum_{j\in\mathcal{C}} \sum_{i\in\mathcal{F}} c(i,\ j)x^*_{ij} = 3LP_{opt}$.

Proof. $\sum_{j'\in\mathcal{C}'} d_{j'} \sum_{i\in\mathcal{F}} c(i,\ j')x^*_{ij'} = \sum_{j'\in\mathcal{C}'} \big(\sum_{j\in\mathcal{C}} \mathcal{A}_{\rho^*}(j,\mathcal{N}_{j'})\big)\hat{C}_{j'}$
$= \sum_{j'\in\mathcal{C}'} \big(\sum_{j\in\mathcal{C}:c(j',j)\leq\mathcal{R}_{j'}} \mathcal{A}_{\rho^*}(j,\mathcal{N}_{j'})\hat{C}_{j'} + \sum_{j\in\mathcal{C}:c(j',j)>\mathcal{R}_{j'}} \mathcal{A}_{\rho^*}(j,\mathcal{N}_{j'})\hat{C}_{j'}\big)$
Second term in the sum on RHS $< \frac{1}{\ell}\sum_{j'\in\mathcal{C}'} \sum_{j\in\mathcal{C}:c(j',j)>\mathcal{R}_{j'}} \mathcal{A}_{\rho^*}(j,\mathcal{N}_{j'})c(j',\ j)$
$\leq \frac{1}{\ell}\sum_{j\in\mathcal{C}} \sum_{j'\in\mathcal{C}':c(j',j)>\mathcal{R}_{j'}} \sum_{i\in\mathcal{N}_{j'}} x^*_{ij}(2c(i,\ j) + 2\ell\hat{C}_j)$ as $c(j',\ j) \leq 2c(i,\ j) + 2\ell\hat{C}_j$ by Lemma 5
$\leq \sum_{j\in\mathcal{C}} \sum_{j'\in\mathcal{C}':c(j',j)>\mathcal{R}_{j'}} \sum_{i\in\mathcal{N}_{j'}} x^*_{ij}(c(i,\ j) + 2\hat{C}_j)$. Thus the claim follows. ◀

Let $\mathcal{C}_S$ be the set of cluster centers $j' \in \mathcal{C}'$ for which $d_{j'} < u$ and $\mathcal{C}_D$ be the set of remaining centers in $\mathcal{C}'$. The clusters centered at $j' \in \mathcal{C}_S$ are called *sparse* and those centered at $j' \in \mathcal{C}_D$ *dense*. For $j' \in \mathcal{C}_D$, sufficient facilities are opened in $\mathcal{N}_{j'}$ so that its entire demand is served within the cluster itself and we say that j' is *self-sufficient*. Unfortunately, the same claim cannot be made for the sparse clusters i.e., we cannot guarantee to open even one facility in each sparse cluster (since $d_{j'} < u$, we need only one facility in each sparse cluster j'). Thus, in the next section, we define a routing tree that is used to route the unmet demand of a cluster to another cluster in bounded cost.

2.2 Constructing the Binary Routing Tree

First, we define a dependency graph $G = (V, E)$, similar to the one defined by Krishnaswamy et al [17], on cluster centers, i.e., $V = \mathcal{C}'$. For brevity of notation, we use j' to refer to the node corresponding to cluster center j' as well as to refer to the cluster center j' itself. For $j' \in \mathcal{C}_S$, let $\eta(j')$ be the nearest other cluster center in $\mathcal{C}'$ of j' i.e., $\eta(j') = k'(\neq j') \in \mathcal{C}' : k'' \in \mathcal{C}' \Rightarrow c(j',\ k') \leq c(j',\ k'')$ and for $j' \in \mathcal{C}_D,\ \eta(j') = j'$. The dependency graph consists of directed edges $c(j',\ \eta(j'))$. Each connected component of the graph is a tree except possibly for a 2-cycle at the root. We remove any edge arbitrarily from the two cycle. The resulting graph is then a forest. Note that, there is at most one dense cluster in a component and if present, it must be the root of the tree. The following lemma will be useful to bound the cost of sending the unserved demand of $j' \in \mathcal{C}_S$ to $\eta(j')$.

▶ **Lemma 9.** $\sum_{j'\in\mathcal{C}_S} d_{j'}(\sum_{i\in\mathcal{N}_{j'}} c(i,\ j')x^*_{ij'} + c(j',\ \eta(j'))(1 - \sum_{i\in\mathcal{N}_{j'}} x^*_{ij'})) \leq 6LP_{opt}$.

Proof. The second term of LHS $= \sum_{j'\in\mathcal{C}_S} d_{j'}\big(\sum_{i\notin\mathcal{N}_{j'}} c(j',\ \eta(j'))x^*_{ij'}\big)$
$\leq \sum_{j'\in\mathcal{C}_S} d_{j'}\big(\sum_{k'\in\mathcal{C}':k'\neq j'} \sum_{i\in\mathcal{N}_{k'}} c(j',\ k')x^*_{ij'}\big)$
$\leq \sum_{j'\in\mathcal{C}_S} d_{j'}\big(\sum_{k'\in\mathcal{C}':k'\neq j'} \sum_{i\in\mathcal{N}_{k'}} 2c(i,\ j')x^*_{ij'}\big)$. ◀

Unfortunately, the in-degree of a node in a tree may be unbounded and hence arbitrarily large amount of demand may accumulate at a cluster center, which may further lead to unbounded capacity violation at the facilities in its cluster.

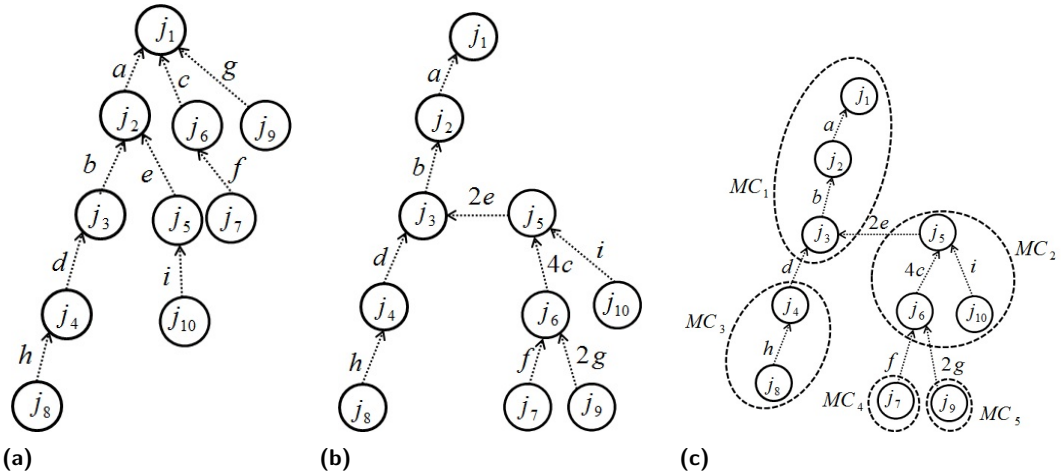


Figure 3 (a) A Tree T of unbounded in-degree. $a < b < d < h$, $a < c < g$, $b < e$. (b) A Binary Tree T' where each node has in-degree at most 2. (c) Formation of meta-clusters for $\ell = 3$.

Bounding the in-degree of a node in the dependency graph. We convert the dependency graph G into another graph G' where-in the in-degree of each node is bounded by 2 with in-degree of the root being 1. This is done as follows (Figure 3(a)-(b)): let $\mathcal{T}$ be a tree in G. $\mathcal{T}$ is converted into a binary tree using the standard procedure after sorting the children of node j' from left to right in non-decreasing order of distance from j' i.e., for each child k' (except for the nearest child) of j', add an edge to its left sibling with weight $2c(k',\ \eta(k'))$ and remove the edge (k', j'). There is no change in the outgoing edge of the leftmost child of j'. Let $\psi(j')$ be the parent of node j' in G'. Its easy to see that $c(j',\ \psi(j')) \leq 2c(j',\ \eta(j'))$. Henceforth whenever we refer to distances, we mean the new edge weights. Hence, we have the following:

$$\sum_{j' \in \mathcal{C}_S} d_{j'} \Big(\sum_{i \in \mathcal{N}_{j'}} c(i,\ j') x^*_{ij'} + c(j',\ \psi(j'))(1 - \sum_{i \in \mathcal{N}_{j'}} x^*_{ij'}) \Big) \leq 12 LP_{opt} \tag{6}$$

2.3 Constructing the Meta-clusters

If we could ensure that for every $j' \in \mathcal{C}_S$ for which no facility is opened in $\mathcal{N}_{j'}$, a facility is opened in $\psi(j')$, we are done (with 3 factor loss in capacities). But we do not know how to do that. However, for every such cluster center j', we will identify a set of centers which will be able to take care of the demand of j' and each one of them is within a distance of $O(\ell)c(j',\ \psi(j'))$ from j'.

We exploit the following observation to make groups of ℓ clusters: each cluster has facilities opened in it to an extent of at least $(1 - 1/\ell)$. Hence, every collection of ℓ clusters, has at least $\ell - 1$ facilities opened in it. Thus, we make groups (called meta-clusters), each consisting of ℓ clusters, if possible. For every tree $\mathcal{T}$ in G', MCs are formed by processing the nodes of $\mathcal{T}$ in a top-down greedy manner starting from the root as described in Algorithm 2. (Also see Figure 3(c)). There may be some MCs of size less than ℓ, towards the leaves of the tree.

Let G_r denote a MC with r being the root cluster of it. With a slight abuse of notation, we will use G_r to denote the collection of centers of the clusters in it as well as the set of clusters themselves. Let $\mathcal{H}(G_r)$ denote the subgraph of $\mathcal{T}$ induced by the nodes in G_r. $\mathcal{H}(G_r)$ is clearly a tree. We say that G_r is responsible for serving the demand in its clusters.

Algorithm 2 Meta-cluster Formation.

Meta-cluster(Tree $\mathcal{T}$)
$\mathcal{N} \leftarrow$ *set of nodes in* $\mathcal{T}$.
while there are non-grouped nodes in $\mathcal{N}$ **do**
  Pick a topmost non-grouped node, say k of $\mathcal{N}$: form a new MC, G_k.
  while G_k has fewer than ℓ nodes **do**
    If $\mathcal{N} = \emptyset$ then break and stop.
    Let $j = argmin_{u \in \mathcal{N}}\{c(u,\ v) : (u, v) \in \mathcal{T}, v \in G_k\}$, set $G_k = G_k \cup \{j\}$. $\mathcal{N} \leftarrow \mathcal{N} \setminus \{j\}$.
  end while
end while

With the guarantee of only $\ell - 1$ opening amongst ℓ clusters, there may be a cluster with no facility opened in it. If this cluster happens to be a sparse cluster at the root, its demand cannot be served within the MC. Thus we define a (routing) tree structure on MCs as follows: a tree consists of MCs as nodes and there is an edge from a MC G_r to another MC G_s if there is a directed edge from root r of G_r to some node $s' \in G_s$, G_s is then called the parent meta-cluster of G_r, G_r a child meta-cluster of G_s and the edge $(r,\ s')$ is called the *connecting edge* of the child MC G_r. If G_r is a root MC, add an edge to itself with cost $c(r,\ \ \psi(r))$. This edge is then called the *connecting edge* of G_r. Note that the cost of any edge in G_s is less than the cost of the connecting edge of G_r which is further less than the cost of any edge in G_r. Further, a dense cluster, if present, is always the root cluster of a root MC. We guarantee that the unmet demand of a MC is served in its parent MC.

2.4 3-factor capacity violation

In this section, we present the main contribution of our work. Inspired by the LP of Krishnaswamy *et al.* [17], we formulate a new LP and present an iterative rounding algorithm to obtain a solution with at most two fractionally opened facilities. Such a solution is called *pseudo-integral* solution. Modifying the LP of Krishnaswamy *et al.* [17] and obtaining a feasible solution of bounded cost for the capacitated scenario is non-trivial. The rounding algorithm is also non-trivial.

2.4.1 Formulating the new LP and obtaining a pseudo-integral solution

Sparse clusters have the nice property that they need to take care of small demand ($< u$ each) and dense clusters have the nice property that the total opening within each cluster is at least 1. These properties are exploited to define a new LP that opens sufficient number of facilities in each MC such that the opened facilities are well spread out amongst the clusters (we make sure that at most 1 (sparse) cluster has no facility opened in it) and demand of a dense cluster is satisfied within the cluster itself. We then present an iterative rounding algorithm that provides us with a solution having at most two fractionally opened facilities.

Let δ_r be the number of dense clusters and σ_r be the number of sparse clusters in a MC G_r. With at least $1 - 1/\ell$ opening in each sparse cluster, observing the fact that $\sigma_r \leq \ell$, we have at least $\sigma_r(1 - 1/\ell) \geq \sigma_r - 1$ total opening in σ_r sparse clusters of G_r. Also, at least $\lfloor d_{j_d}/u \rfloor$ opening is there in a dense cluster centered at j_d in G_r. Let $\alpha_r = \max\{0, \sigma_r - 1\}$. LP is defined so as to open at least $\lfloor d_{j_d}/u \rfloor + \alpha_r$ facilities in G_r. Let $\tau(j') = \{i \in \mathcal{N}_{j'} : c(i,\ j') \leq c(j',\ \psi(j'))\}$ if $j' \in \mathcal{C}_S$ (recall that $\psi(j')$ is the parent of j' in binary tree) and $\tau(j') = \mathcal{N}_{j'}$ if $j' \in \mathcal{C}_D$. Also, let $\mathcal{S}_r = G_r \cap \mathcal{C}_S$ and $s_r = \alpha_r$ for all MCs G_r,

$\tilde{\mathcal{F}} = \mathcal{F}$, $\tilde{\mathcal{B}} = \mathcal{B}$, $r_{j'} = \lfloor d_{j'}/u \rfloor$ $\forall j' \in \mathcal{C}_D$ and $\hat{\tau}(j') = \tau(j')$ $\forall j' \in \mathcal{C}'$. These sets are updated as we go from one iteration to the next iteration in our rounding algorithm, thereby giving a new (reduced) LP in each iteration. Let w_i denote whether facility i is opened in the solution or not. We now write an LP, called LP_2 with the objective of minimising the following function:

$$\mathcal{C}ostKM(w) = \sum_{j' \in \mathcal{C}_S} d_{j'}[\sum_{i \in \mathcal{N}_{j'}} c(i,\ j')w_i + c(j',\ \psi(j'))(1 - \sum_{i \in \mathcal{N}_{j'}} w_i)] + u \sum_{j' \in \mathcal{C}_D} \sum_{i \in \mathcal{N}_{j'}} c(i,\ j')w_i$$

$$\begin{aligned}
s.t. \quad & \textstyle\sum_{i \in \hat{\tau}(j')} w_i \leq 1 && \forall\ j' \in \mathcal{C}_S && (7)\\
& \textstyle\sum_{i \in \hat{\tau}(j')} w_i = r_{j'} && \forall\ j' \in \mathcal{C}_D && (8)\\
& \textstyle\sum_{j' \in \mathcal{S}_r} \sum_{i \in \hat{\tau}(j')} w_i \geq s_r && \forall\ r : G_r \text{ is a MC} && (9)\\
& \textstyle\sum_{i \in \tilde{\mathcal{F}}} f_i w_i \leq \tilde{\mathcal{B}} && && (10)\\
& 0 \leq w_i \leq 1\ \forall\ i \in \tilde{\mathcal{F}} && && (11)
\end{aligned}$$

Constraints (8) and (9) ensure that sufficient number of facilities are opened in a meta-cluster. Constraints (7) and (8) ensure that the opened facilities are well spread out amongst the clusters as no more than 1 and $\lfloor \frac{d_{j'}}{u} \rfloor$ facilities are opened in a sparse and dense cluster respectively. Constraint (8) also ensures that at least $\lfloor \frac{d_{j'}}{u} \rfloor$ facilities are opened in a dense cluster. This requirement is essential to make sure that the demand of a dense cluster is served within the cluster only. Hence, equality in constraint (8) is important.

▶ **Lemma 10.** *A feasible solution w' to LP_2 can be obtained such that $\mathcal{C}ostKM(w') \leq (2\ell + 13)LP_{opt}$.*

Proof. Refer to Appendix 5.1. ◀

For a vector $w \in \mathcal{R}^{|\mathcal{F}|}$ and $\mathcal{F}' \subseteq \mathcal{F}$, let $w^{\mathcal{F}'}$ denote the vector 'w restricted to $\mathcal{F}'$'. Also, let $s =< s_r >$, $S =< S_r >$ and $R =< r_{j'} >_{j' \in \mathcal{C}_D}$. Algorithm 3 presents an iterative rounding algorithm that solves LP_2 and returns a pseudo-integral solution $\tilde{w}$. A sparse cluster is removed from the scenario for the next iteration as and when a facility is integrally opened in it (lines 11, 12). In a dense cluster centered at j', the number of facilities to be opened by the LP ($r_{j'}$) is decremented by the number of integrally opened facilities in it (line 15) at every iteration and the cluster is removed when it becomes 0 (line 16). Similar treatment is done for $G_r \cap \mathcal{C}_S$ (line 12, 14)

▶ **Lemma 11.** *The solution $\tilde{w}$ given by Iterative Rounding Algorithm satisfies the following: i) $\tilde{w}$ is feasible, ii) $\tilde{w}$ has at most two fractional facilities and iii) $\mathcal{C}ostKM(\tilde{w}) \leq (2\ell+13)LP_{opt}$.*

Proof. Refer to Appendix 5.2. ◀

2.4.2 Obtaining an integrally open solution

The two fractionally opened facilities obtained in Section 2.4.1, if any, are opened integrally at a loss of additive f_{max} in the budget. Let $\hat{w}$ denote the solution obtained. Next lemma shows that $\hat{w}$ has sufficient number of facilities opened in each MC to serve the demand the MC is responsible for, except possibly for u units. Lemma (12) presents the assignments done within a MC and discusses their impact on the capacity and the cost bounds.

▶ **Lemma 12.** *Consider a meta-cluster G_r. Suppose the capacities are scaled up by a factor of $\max\{3, 2 + \frac{4}{\ell-1}\}$ for $\ell \geq 2$. Then, i) the dense cluster in G_r (if any) is* self-sufficient *i.e., its demand can be completely assigned within the cluster itself at a loss of at most factor 2 in cost. ii) There is at most one cluster with no facility opened in it and it is a sparse cluster.*

Algorithm 3 Obtaining a pseudo-integral solution.

pseudo-integral($\tilde{\mathcal{F}}$, $\tilde{\mathcal{B}}$, s, S, $\hat{\tau}()$, R)
$\tilde{w}_i^{\mathcal{F}} = 0 \ \forall i \in \mathcal{F}$
while $\tilde{\mathcal{F}} \neq \emptyset$ **do**
  Compute an extreme point solution $\tilde{w}^{\tilde{\mathcal{F}}}$ to LP_2.
  $\tilde{\mathcal{F}}_0 \leftarrow \{i \in \tilde{\mathcal{F}} : \tilde{w}_i^{\tilde{\mathcal{F}}} = 0\}$, $\tilde{\mathcal{F}}_1 \leftarrow \{i \in \tilde{\mathcal{F}} : \tilde{w}_i^{\tilde{\mathcal{F}}} = 1\}$.
  if $|\tilde{\mathcal{F}}_0| = 0$ and $|\tilde{\mathcal{F}}_1| = 0$ **then**
    Return $\tilde{w}^{\mathcal{F}}$. * exit when all variables are fractionally opened*\
  else
    For all MCs G_r{
    while $\exists j' \in S_r$ such that constraint (7) is tight over $\tilde{\mathcal{F}}_1$ i.e., $\sum_{i \in \hat{\tau}(j') \cap \tilde{\mathcal{F}}_1} \tilde{w}_i^{\tilde{\mathcal{F}}} = 1$ **do**
      Remove the constraint corresponding to j' from (7). * a facility in $\tau(j')$ has been opened*\
      set $S_r = S_r \setminus \{j'\}$, $s_r = \max\{0, s_r - 1\}$. * delete the contribution of j' in constraint (9)*\
    end while
    If $s_r = 0$, remove the constraint corresponding to $\mathcal{S}_r$ from (9). * $\sigma_r - 1$ facilities have been opened in $G_r \cap \mathcal{C}_S$ *\
    If $\exists j' \in G_r \cap \mathcal{C}_D$, set $r_{j'} \leftarrow r_{j'} - |\hat{\tau}(j') \cap \tilde{\mathcal{F}}_1|$. * decrement $r_{j'}$ by the number of integrally opened facilities in $\hat{\tau}(j')$ *\
    If $r_{j'} = 0$, remove the constraint corresponding to j' from (8). * $\lfloor d_{j'}/u \rfloor$ facilities have been integrally opened in $\tau(j')$ *\ }
  end if
  $\tilde{\mathcal{F}} \leftarrow \tilde{\mathcal{F}} \setminus (\tilde{\mathcal{F}}_0 \cup \tilde{\mathcal{F}}_1)$, $\tilde{\mathcal{B}} \leftarrow \tilde{\mathcal{B}} - \sum_{i \in \tilde{\mathcal{F}}_1} f_i \tilde{w}_i^{\tilde{\mathcal{F}}}$, $\hat{\tau}(j') \leftarrow \hat{\tau}(j') \setminus (\tilde{\mathcal{F}}_1 \cup \tilde{\mathcal{F}}_0) \ \forall j' \in \mathcal{C}'$.
end while
Return $\tilde{w}^{\mathcal{F}}$

iii) Any (cluster) center responsible for the unserved demand of $j' \in \mathcal{C}'$ is an ancestor of j' in $\mathcal{H}(G_r)$. iv) At most u units of demand in G_r remain un-assigned and it must be in the root cluster of G_r. Such a MC cannot be a root MC. v) Let $\beta_r = \lfloor d_{j_d}/u \rfloor + \max\{0, \sigma_r - 1\}$, where j_d is the center of the dense root cluster (if any) in G_r. Then, at least β_r facilities are opened in G_r. (vi) Total distance traveled by demand $d_{j'}$ of $j'(\neq r) \in G_r$ to reach the centers of the clusters in which they are served is bounded by $d_{j'} c(j',\ \psi(j'))$.

Proof. Refer to Appendix 5.3. ◀

Lemma (13) deals with the remaining demand that we fail to assign within a MC. Such demand is assigned in the parent MC. Lemma (13) discusses the cost bound for such assignments and the impact of the demand coming onto G_r from the children MCs along with the demand within G_r on capacity.

▶ **Lemma 13.** *Consider a meta-cluster G_r. The demand of G_r and the demand coming onto G_r from the children meta-clusters can be assigned to the facilities opened in G_r such that: i) capacities are violated at most by a factor of $max\{3, 2 + \frac{4}{\ell - 1}\}$ for $\ell \geq 2$. ii) Total distance traveled by demand $d_{j'}$ of $j' \in \mathcal{C}'$ to reach the centers of the clusters in which they are served is bounded by $\ell d_{j'} c(j',\ \psi(j'))$.*

Proof. Refer to Appendix 5.4. ◀

Choosing $\ell \geq 2$ such that $2 + \frac{4}{(\ell - 1)} = 3 \Rightarrow \ell = 5$. Lemma (14) bounds the cost of assigning the demands collected at the centers to the facilities opened in their respective clusters.

▶ **Lemma 14.** *The cost of assigning the demands collected at the centers to the facilities opened in their respective clusters is bounded by $O(1)LP_{opt}$.*

Proof. The proof follows from the observation that if $d_{j'}$ is served by a facility in $\tau(j''), j'' \in \mathcal{C}_S$ then $c(j'',\ i) \leq c(j'',\ \psi(j'')) \leq c(j',\ \psi(j'))$. This was the motivation to define $\tau(j')$ the way it was, while defining LP_2. For details, refer to Appendix 5.5. ◀

2.5 $(2+\epsilon)$ factor capacity violation

There is only one scenario in which we violate the capacities by a factor of 3 in the previous section. In all other scenarios capacities scaled up by a factor of $(2+\epsilon)$ are sufficient even to accommodate the demand of the children MCs. Consider this special scenario. Let j_d be the center of the dense cluster and j_s be its only child (sparse) cluster in the routing tree. Further let, $d_{j_d} = 1.99u$ and $d_{j_s} = .99u$. Then, we must have a total opening of more than 2 in the clusters of j_d and j_s taken together whereas LP_2 opens only 1. In such a scenario, if we treat j_s with j_d instead of considering it with the remaining sparse clusters of G_r, we can open 2 facilities in $\tau(j_d) \cup \tau(j_s)$ and they have to serve a total demand of at most $4u$ ($1.99u + .99u+$ at most u of the remaining sparse clusters) within the MC, thereby violating the capacities by a factor of at most 2. On the other hand, if $d_{j_d} = 1.01u$ and $d_{j_s} = .98u$, then we cannot guarantee to open 2 facilities in $\tau(j_d) \cup \tau(j_s)$. In this case, if we treated j_s with j_d and only 1 facility is opened in $\tau(j_d) \cup \tau(j_s)$, it will have to serve a total demand of (close to) $3u$ ($1.01u + .98u+$ at most u of the remaining sparse clusters) leading to violation of 3 in capacity. Note that first case corresponds to the scenario when the residual demand of j_d (viz. $.99u$ here) is large (close to u) and the second case corresponds to the scenario when the residual demand of j_d (viz. $.01u$ here) is small (close to 0). In the first case we treat j_s with j_d whereas in the second case, we treat it with the remaining sparse clusters. In Section 2.4, one can imagine that a MC G_r is partitioned into G_r^1 and G_r^2 where G_r^1 contained only the dense cluster of G_r and G_r^2 contained all the sparse clusters of G_r. We modify the partitions as follows: let $res(j_d) = d_{j_d}/u - \lfloor d_{j_d}/u \rfloor$: (i) if $res(j_d) < \epsilon$: set $G_r^1 = G_r \cap \mathcal{C}_D$, $G_r^2 = G_r \cap \mathcal{C}_S$, $\gamma_r = \lfloor d_{j_d}/u \rfloor$, $\sigma'_r = \sigma_r$. (This is same as above.) (ii) otherwise, $\epsilon \leq res(j_d) < 1$: set $G_r^1 = (G_r \cap \mathcal{C}_D) \cup \{j_s\}$, $G_r^2 = (G_r \cap \mathcal{C}_S) \setminus \{j_s\}$, $\gamma_r = \lfloor d_{j_d}/u \rfloor + |\{j_s\}|$ [4], $\sigma'_r = \max\{0, \sigma_r - 1\}$.

We modify our LP accordingly so as to open at least γ_r facilities in G_r^1 and $\alpha_r = \max\{0, \sigma'_r - 1\}$ facilities in G_r^2. Let $S_r^1 = G_r^1$, $s_r^1 = \gamma_r$ and $S_r^2 = G_r^2$, $s_r^2 = \alpha_r, \hat{\tau}(j') = \tau(j')\ \forall j'$. For $j' \in \mathcal{C}_D$, let $r_{j'} = \lfloor d_{j'}/u \rfloor$. Also, let $\tilde{\mathcal{F}} = \mathcal{F}$ and $\tilde{\mathcal{B}} = \mathcal{B}$. Let w_i denote whether facility i is opened in the solution or not. LP_2 is modified as follows:
LP_3 : *Min.* $\mathcal{C}ostKM(w)$

$$\text{subject to} \sum_{i \in \hat{\tau}(j')} w_i \leq 1 \quad \forall\ j' \in \mathcal{C}_S \tag{12}$$

$$\sum_{j' \in S_r^1} \sum_{i \in \hat{\tau}(j')} w_i \geq s_r^1 \quad \forall\ G_r^1 : s_r^1 \neq 0 \tag{13}$$

$$\sum_{j' \in S_r^2} \sum_{i \in \hat{\tau}(j')} w_i \geq s_r^2 \quad \forall\ G_r^2 : s_r^2 \neq 0 \tag{14}$$

$$\sum_{i \in \tilde{\mathcal{F}}} f_i w_i \leq \tilde{\mathcal{B}} \tag{15}$$

$$0 \leq w_i \leq 1 \qquad \forall i \in \tilde{\mathcal{F}} \tag{16}$$

▶ **Lemma 15.** *A feasible solution w' to LP_3 can be obtained such that $\mathcal{C}ostKM(w') \leq (2\ell + 13)LP_{opt}$.*

Proof. Proof is similar to the proof of Lemma (10). ◀

Algorithm 3 can be modified to obtain Algorithm 4 as follows: whenever a constraint corresponding to (12) gets tight over integrally opened facilities, it is removed from S_r^1 or S_r^2 wherever it belongs, in the same manner as line 12 of Algorithm 3.

[4] In case a component of dependency graph consists of a singleton dense cluster, j_s may not exist. This case causes no problem even if $res(j_d)$ is large as it must be a leaf MC in this case.

Algorithm 4 Obtaining a pseudo-integral solution.

pseudo-integral($\tilde{\mathcal{F}}$, $\tilde{\mathcal{B}}$, s^1, s^2, S^1, S^2, $\hat{\tau}()$, R')
$\tilde{w}_i^{\mathcal{F}} = 0 \ \forall i \in \mathcal{F}$
while $\tilde{\mathcal{F}} \neq \emptyset$ **do**
  Compute an extreme point solution $\tilde{w}^{\tilde{\mathcal{F}}}$ to LP_3.
  $\tilde{\mathcal{F}}_0 \leftarrow \{i \in \tilde{\mathcal{F}} : \tilde{w}_i^{\tilde{\mathcal{F}}} = 0\}$, $\tilde{\mathcal{F}}_1 \leftarrow \{i \in \tilde{\mathcal{F}} : \tilde{w}_i^{\tilde{\mathcal{F}}} = 1\}$.
  if $|\tilde{\mathcal{F}}_0| = 0$ and $|\tilde{\mathcal{F}}_1| = 0$ **then**
    Return $\tilde{w}^{\mathcal{F}}$.
  else
    For all MCs G_r\{
    while $\exists j' \in G_r \cap \mathcal{C}_S$ such that constraint (12) is tight over $\tilde{\mathcal{F}}_1$ i.e., $\sum_{i \in \hat{\tau}(j') \cap \tilde{\mathcal{F}}_1} \tilde{w}_i^{\tilde{\mathcal{F}}} = 1$ **do**
      Remove the constraint corresponding to j' from (12). $\ast$ a facility in $\tau(j')$ has been opened$\ast$\
      If $j' \in S_r^1$, set $S_r^1 = S_r^1 \setminus \{j'\}$, $s_r^1 = \max\{0, s_r^1 - 1\}$. \$\ast$ delete the contribution of j' in constraint (13) $\ast$\
      If $j' \in S_r^2$, set $S_r^2 = S_r^2 \setminus \{j'\}$, $s_r^2 = \max\{0, s_r^2 - 1\}$.\$\ast$ delete the contribution of j' in constraint (14) $\ast$\
      If $s_r^2 = 0$, remove the constraint corresponding to the MC from (14).\$\ast$ α_r facilities have been opened in $G_r \cap \mathcal{C}_S$ $\ast$\
    end while
    If $\exists j' \in G_r \cap \mathcal{C}_D$, set $s_r^1 = s_r^1 - |\hat{\tau}(j') \cap \tilde{\mathcal{F}}_1|$. \$\ast$ decrement s_r^1 by the number of integrally opened facilities in $\hat{\tau}(j')$ $\ast$\
    If $s_r^1 = 0$, remove the constraint corresponding to the MC from (13). \$\ast$ γ_r facilities have been opened in G_r^1 $\ast$\
  end if
  $\tilde{\mathcal{F}} \leftarrow \tilde{\mathcal{F}} \setminus (\tilde{\mathcal{F}}_0 \cup \tilde{\mathcal{F}}_1)$, $\tilde{\mathcal{B}} \leftarrow \tilde{\mathcal{B}} - \sum_{i \in \tilde{\mathcal{F}}_1} f_i \tilde{w}_i^{\tilde{\mathcal{F}}}$, $\hat{\tau}(j') \leftarrow \hat{\tau}(j') \setminus (\tilde{\mathcal{F}}_1 \cup \tilde{\mathcal{F}}_0) \ \forall j' \in \mathcal{C}'$.
end while
Return $\tilde{w}^{\mathcal{F}}$.

▶ **Lemma 16.** *The solution $\tilde{w}$ given by Iterative Rounding Algorithm satisfies the following: i) $\tilde{w}$ is feasible, ii) $\tilde{w}$ has at most two fractional facilities and iii) $\mathcal{C}ostKM(\tilde{w}) \leq (2\ell+13)LP_{opt}$.*

Proof. Proof is similar to the proof of Lemma (11). ◀

The two fractionally opened facilities, if any, are opened integrally as in Section 2.4.2 at a loss of additive f_{max} in the budget. Let $\hat{w}$ denote the integrally open solution.

In the next lemma, we show that $\hat{w}$ has sufficient number of facilities opened in each MC to serve the demand the MC is responsible for, except possibly for u units. Let M be the set of all meta clusters and M_1 be the set of meta clusters, each consisting of exactly one dense and one sparse cluster. MCs in M_1 need special treatment and will be considered separately. Lemma (17) presents the assignments done within a MC and discusses their impact on the capacity and the cost bounds.

▶ **Lemma 17.** *Consider a meta-cluster G_r. Suppose the capacities are scaled up by a factor of $2+\epsilon$ for $\ell \geq 1/\epsilon$. Then, (i) G_r^1 is* self-sufficient *i.e., its demand can be completely assigned within the cluster itself. (ii) There are at most two clusters, one in G_r^1 and one in G_r^2, with no facility opened in them and these clusters are sparse. (iii) Any (cluster) center responsible for the unserved demand of j' is an ancestor of j' in $\mathcal{H}(G_r)$. (iv) At most u units of demand in G_r remain un-assigned and it must be in the root cluster of G_r. Such a MC cannot be a root MC. (v) For $G_r \in M \setminus M_1$, let $\beta_r = \lfloor d_{j_d}/u \rfloor + \max\{0, \sigma_r - 1\}$, where j_d is the center of the dense root cluster in G_r. Then, at least β_r facilities are opened in G_r. (vi) For $G_r \in M_1$, let $\beta_r = \lfloor d_{j_d}/u \rfloor$ if $res(j_d) < \epsilon$ and $= \lfloor d_{j_d}/u \rfloor + 1$ otherwise. Then, at least β_r facilities are opened in G_r. (vii) Total distance traveled by demand $d_{j'}$ of $j' (\neq r) \in G_r$ to reach the centers of the clusters in which they are served is bounded by $2d_{j'}c(j', \psi(j'))$.*

Proof. Refer to Appendix 5.6. ◀

Lemma (18) deals with the remaining demand that we fail to assign within the MC. Such demand is assigned in the parent MC. Lemma (18) discusses the cost bound for such assignments and the impact of the demand coming onto G_r from the children MCs along with the demand within G_r on capacity.

▶ **Lemma 18.** *Consider a meta-cluster G_r. The demand of G_r and the demand coming onto G_r from the children meta-clusters can be assigned to the facilities opened in G_r such that: (i) capacities are violated at most by a factor of $(2+\frac{4}{\ell-1})$ for $\ell \geq 1/\epsilon$ and, (ii) Total distance traveled by demand $d_{j'}$ of $j' \in \mathcal{C}'$ to reach the centers of the clusters in which they are served is bounded by $\ell d_{j'} c(j',\ \psi(j'))$.*

Proof. Proof is similar to the proof of Lemma (13). ◀

▶ **Lemma 19.** *The cost of assigning the demands collected at the centers to the facilities opened in their respective clusters is bounded by $(2+\epsilon)(2\ell+1)LP_{opt}$.*

Proof. Proof is similar to the proof of Lemma (14). ◀

3 Capacitated k Facility Location Problem

Standard LP-Relaxation of the CkFLP can be found in Aardal *et al.* [1]. When $f_i = 0$, the problem reduces to the k-median problem and when $k = |\mathcal{F}|$ it reduces to the facility location problem. Our techniques for CKnM provide similar results for CkFLP in a straight forward manner i.e., $O(1/\epsilon^2)$ factor approximation, violating the capacities by a factor of $(2+\epsilon)$ and cardinality by plus 1. The violation of cardinality can be avoided by opening the facility with larger opening integrally while converting a pseudo integral solution into an integrally open solution. Thus, we obtain Theorem 2.

Proof of Theorem 3. Let $\rho^* =< x^*, y^* >$ denote the optimal LP solution. For sparse clusters, we open the cheapest facility i^* in *ball(j)*, close all facilities in the cluster and shift their demands to i^*. Let $\hat{\rho} =< \hat{x}, \hat{y} >$ be the solution so obtained. It is easy to see that we loose at most a factor of 2 in cardinality, and $\mathcal{C}ost\text{k}FLP(\hat{x}, \hat{y})$ is within $O(1)LP_{opt}$.

To handle dense clusters, we introduce the notion of cluster instances. For each cluster center $j' \in \mathcal{C}_D$, let $b^f_{j'} = \sum_{i\in\mathcal{N}_{j'}} f_i y^*_i$ and $b^c_{j'} = \sum_{i\in\mathcal{N}_{j'}} \sum_{j\in\mathcal{C}} x^*_{ij}[c(i,\ j) + 4\hat{C}_j]$. We define a cluster instance $\mathcal{S}_{j'}(j',\ \mathcal{N}_{j'},\ d_{j'},\ b^c_{j'}, b^f_{j'})$ as follows: Minimize $Cost_{CI}(z) = \sum_{i\in\mathcal{N}_{j'}}(f_i + uc(i,\ j'))z_i$ s.t. $u\sum_{i\in\mathcal{N}_{j'}} z_i \geq d_{j'}$ and $z_i \in [0,1]$. It can be shown that $z_i = \sum_{j\in\mathcal{C}} x^*_{ij}/u = l_i/u \leq y^*_i\ \forall i \in \mathcal{N}_{j'}$ is a feasible solution with cost at most $b^f_{j'} + b^c_{j'}$. An almost integral solution z' is obtained by arranging the fractionally opened facilities in z in non-decreasing order of $f_i + c(i,\ j')u$ and greedily transferring the total opening $size(z,\ \mathcal{N}_{j'})$ to them. Let $l'_i = z'_i u$. For a fixed $\epsilon > 0$, an integrally open solution $\hat{z}$ and assignment $\hat{l}$ (possibly fractional) is obtained as follows: let i_1 be the fractionally opened facility, if any. If $z'_{i_1} < \epsilon$, close i_1 and shift its demand to another integrally opened facility at a loss of factor $(1+\epsilon)$ in its capacity. Else $(z'_{i_1} \geq \epsilon)$, open i_1, at a loss of factor 2 in cardinality and $1/\epsilon$ in facility cost. The solution $\hat{z}$ satisfies the following: $\hat{l}_i \leq (1+\epsilon)\hat{z}_i u\ \forall i \in \mathcal{N}_{j'}$, $\sum_{i\in\mathcal{N}_{j'}} \hat{z}_i \leq 2\sum_{i\in\mathcal{N}_{j'}} z'_i\ \forall j' \in \mathcal{C}_D$ and $Cost_{CI}(\hat{z}) \leq \max\{1/\epsilon, 1+\epsilon\} Cost_{CI}(\hat{z})$. ◀

4 Conclusion

In this work, we presented the first constant factor approximation algorithm for uniform hard capacitated knapsack median problem violating the budget by a factor of $(1+\epsilon)$ and capacity by $(2+\epsilon)$. Two variety of results were presented for capacitated k-facility location problem

with a trade-off between capacity and cardinality violation: an $O(1/\epsilon^2)$ factor approximation violating capacities by $(2+\epsilon)$ and a $O(1/\epsilon)$ factor approximation, violating the capacity by a factor of at most $(1+\epsilon)$ using at most $2k$ facilities. As a by-product, we also gave a constant factor approximation for uniform capacitated facility location at a loss of $(1+\epsilon)$ in capacity from the natural LP. The result shows that the natural LP is not too bad.

It would be interesting to see if the capacity violation can be reduced to $(1+\epsilon)$ using the techniques of Byrka *et al.* [8]. Avoiding violation of budget will require strengthening the LP in a non-trivial way. Another direction for future work would be to extend our results to non-uniform capacities. Conflicting requirement of facility costs and capacities makes the problem challenging.

References

1 Karen Aardal, Pieter L. van den Berg, Dion Gijswijt, and Shanfei Li. Approximation algorithms for hard capacitated k-facility location problems. *EJOR*, 242(2):358–368, 2015. doi:10.1016/j.ejor.2014.10.011.

2 Ankit Aggarwal, Anand Louis, Manisha Bansal, Naveen Garg, Neelima Gupta, Shubham Gupta, and Surabhi Jain. A 3-approximation algorithm for the facility location problem with uniform capacities. *Journal of Mathematical Programming*, 141(1-2):527–547, 2013. doi:10.1007/s10107-012-0565-4.

3 Hyung-Chan An, Aditya Bhaskara, Chandra Chekuri, Shalmoli Gupta, Vivek Madan, and Ola Svensson. Centrality of trees for capacitated k-center. *Math. Program.*, 154(1-2):29–53, 2015. doi:10.1007/s10107-014-0857-y.

4 Hyung-Chan An, Mohit Singh, and Ola Svensson. LP-Based Algorithms for Capacitated Facility Location. In *FOCS, 2014*, pages 256–265, 2014. doi:10.1109/FOCS.2014.35.

5 Manisha Bansal, Naveen Garg, and Neelima Gupta. A 5-Approximation for Capacitated Facility Location. In *Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, pages 133–144, 2012. doi:10.1007/978-3-642-33090-2_13.

6 Jaroslaw Byrka, Krzysztof Fleszar, Bartosz Rybicki, and Joachim Spoerhase. Bi-Factor Approximation Algorithms for Hard Capacitated k-Median Problems. In *SODA 2015*, pages 722–736, 2015. doi:10.1137/1.9781611973730.49.

7 Jaroslaw Byrka, Thomas Pensyl, Bartosz Rybicki, Joachim Spoerhase, Aravind Srinivasan, and Khoa Trinh. An Improved Approximation Algorithm for Knapsack Median Using Sparsification. In *ESA 2015*, pages 275–287, 2015. doi:10.1007/978-3-662-48350-3_24.

8 Jaroslaw Byrka, Bartosz Rybicki, and Sumedha Uniyal. An Approximation Algorithm for Uniform Capacitated k-Median Problem with $(1+\epsilon)$ Capacity Violation. In *Integer Programming and Combinatorial Optimization - 18th International Conference, IPCO 2016, Liège, Belgium, June 1-3, 2016, Proceedings*, pages 262–274, 2016. doi:10.1007/978-3-319-33461-5_22.

9 Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS), New York, NY, USA*, pages 378–388, 1999.

10 Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for facility location problems. *SIAM Journal on Computing*, 34(4):803–824, 2005.

11 Moses Charikar, Sudipto Guha, Éva Tardos, and David B. Shmoys. A Constant-Factor Approximation Algorithm for the k-Median Problem (Extended Abstract). In *STOC, 1999*, pages 1–10, 1999. doi:10.1145/301250.301257.

12 Moses Charikar and Shi Li. A Dependent LP-Rounding Approach for the k-Median Problem. In *ICALP 2012*, pages 194–205, 2012. doi:10.1007/978-3-642-31594-7_17.

13 Marek Cygan, MohammadTaghi Hajiaghayi, and Samir Khuller. LP rounding for k-centers with non-uniform hard capacities. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 273–282, 2012. doi:10.1109/FOCS.2012.63.

14 H. Gökalp Demirci and Shi Li. Constant Approximation for Capacitated k-Median with $(1 + \epsilon)$ Capacity Violation. In *43rd ICALP 2016, July 11–15, 2016, Rome, Italy*, pages 73:1–73:14, 2016.

15 Samir Khuller and Yoram J. Sussmann. The Capacitated K-Center Problem. *SIAM J. Discrete Math.*, 13(3):403–418, 2000. doi:10.1137/S0895480197329776.

16 Madhukar R. Korupolu, C. Greg Plaxton, and Rajmohan Rajaraman. Analysis of a local search heuristic for facility location problems. *Journal of Algorithms*, 37(1):146–188, 2000. doi:http://dx.doi.org/10.1006/jagm.2000.1100.

17 Ravishankar Krishnaswamy, Amit Kumar, Viswanath Nagarajan, Yogish Sabharwal, and Barna Saha. The Matroid Median Problem. In *SODA, 2011*, pages 1117–1130, 2011. doi:10.1137/1.9781611973082.84.

18 Ravishankar Krishnaswamy, Shi Li, and Sai Sandeep. Constant approximation for k-median and k-means with outliers via iterative rounding. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 646–659, 2018. doi:10.1145/3188745.3188882.

19 Amit Kumar. Constant Factor Approximation Algorithm for the Knapsack Median Problem. In *SODA, 2012*, pages 824–832. SIAM, 2012. URL: http://dl.acm.org/citation.cfm?id=2095116.2095182.

20 Retsef Levi, David B. Shmoys, and Chaitanya Swamy. LP-based approximation algorithms for capacitated facility location. *Journal of Mathematical Programming*, 131(1-2):365–379, 2012. doi:10.1007/s10107-010-0380-8.

21 Shanfei Li. An Improved Approximation Algorithm for the Hard Uniform Capacitated k-median Problem. In Klaus Jansen, José D. P. Rolim, Nikhil R. Devanur, and Cristopher Moore, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014), Germany*, volume 28 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 325–338. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/LIPIcs.APPROX-RANDOM.2014.325.

22 Shi Li. On Uniform Capacitated k-Median Beyond the Natural LP Relaxation. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 696–707, 2015. doi:10.1137/1.9781611973730.47.

23 Shi Li. Approximating capacitated k-median with $(1 + \epsilon)k$ open facilities. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 786–796, 2016. doi:10.1137/1.9781611974331.ch56.

24 Jyh-Han Lin and Jeffrey Scott Vitter. epsilon-Approximations with Minimum Packing Constraint Violation (Extended Abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 771–782, 1992. doi:10.1145/129712.129787.

25 David B. Shmoys, Éva Tardos, and Karen Aardal. Approximation Algorithms for Facility Location Problems (Extended Abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA,*, pages 265–274, 1997. doi:10.1145/258533.258600.

26 Chaitanya Swamy. Improved Approximation Algorithms for Matroid and Knapsack Median Problems and Applications. In *APPROX/RANDOM, 2014*, pages 403–418, 2014. doi:10.4230/LIPIcs.APPROX-RANDOM.2014.403.

5 Appendix

5.1 Proof of Lemma 10

Define a feasible solution to the LP_2 as follows: let $j' \in \mathcal{C}_D,\ i \in \tau(j')$, set $w'_i = \frac{l_i}{d_{j'}}\lfloor d_{j'}/u \rfloor = \frac{l_i}{u}\frac{\lfloor d_{j'}/u \rfloor}{d_{j'}/u} \leq \frac{l_i}{u} \leq y^*_i$. For $j' \in \mathcal{C}_S$, we set $w'_i = min\{x^*_{ij'},\ y^*_i\} = x^*_{ij'} \leq y^*_i$ for $i \in \tau(j')$ and $w'_i = 0$ for $i \in \mathcal{N}_{j'} \setminus \tau(j')$. We will next show that the solution is feasible.

For $j' \in \mathcal{C}_S$, $\sum_{i\in\tau(j')} w'_i \leq \sum_{i\in\mathcal{N}_{j'}} w'_i = \sum_{i\in\mathcal{N}_{j'}} x^*_{ij'} \leq 1$.

Next, let $j' \in \mathcal{C}_D$, then $\sum_{i\in\tau(j')} w'_i = \sum_{i\in\mathcal{N}_{j'}} \frac{l_i}{u}\frac{\lfloor d_{j'}/u \rfloor}{d_{j'}/u} = \lfloor d_{j'}/u \rfloor$ as $\sum_{i\in\mathcal{N}_{j'}} l_i = d_{j'}$. Note that $\sum_{i\in\tau(j')} w'_i \geq 1$ as $d_{j'} \geq u$.

For a meta-cluster G_r, we have $\sum_{j'\in G_r}\sum_{i\in\tau(j')} w'_i = \sum_{j'\in G_r\cap\mathcal{C}_S}\sum_{i\in\tau(j')} x^*_{ij'} \geq \sum_{j'\in G_r\cap\mathcal{C}_S}(1 - 1/l) = \max\{0, \sigma_r - 1\} = \alpha_r$.

Since for each $i \in \mathcal{F}$ we have $w'_i \leq y^*_i \Rightarrow \sum_{i\in\mathcal{F}} f_i w'_i \leq \sum_{i\in\mathcal{F}} f_i y^*_i \leq \mathcal{B}$.

Next, consider the objective function. For $j' \in \mathcal{C}_D$, we have $\sum_{i\in\tau(j')} u\ c(i,\ j')w'_i = u\sum_{i\in\mathcal{N}_{j'}} c(i,\ j')(\frac{\sum_{j\in\mathcal{C}} x^*_{ij}}{u}) = \sum_{i\in\mathcal{N}_{j'}}\sum_{j\in\mathcal{C}} c(i,\ j')x^*_{ij} \leq \sum_{i\in\mathcal{N}_{j'}}\sum_{j\in\mathcal{C}}\left(c(i,\ j) + 2\ell\hat{C}_j\right)x^*_{ij}$. Summing over all $j' \in \mathcal{C}_D$ we get, $\sum_{j'\in\mathcal{C}_D}\sum_{i\in\mathcal{N}_{j'}}\sum_{j\in\mathcal{C}} x^*_{ij}[c(i,\ j) + 2\ell\hat{C}_j] \leq (2\ell+1)LP_{opt}$.

Now consider the part of objective function for $\mathcal{C}_S$. $\sum_{j'\in\mathcal{C}_S} d_{j'}(\sum_{i\in\mathcal{N}_{j'}} c(i,\ j')w'_i + c(j',\ \psi(j'))(1 - \sum_{i\in\mathcal{N}_{j'}} w'_i)) = \sum_{j'\in\mathcal{C}_S} d_{j'}(\sum_{i\in\tau(j')} c(i,\ j')w'_i + \sum_{i\in\mathcal{N}_{j'}\setminus\tau(j')} c(i,\ j')w'_i + c(j',\ \psi(j'))(1-\sum_{i\in\tau(j')} w'_i-\sum_{i\in\mathcal{N}_{j'}\setminus\tau(j')} w'_i)) = \sum_{j'\in\mathcal{C}_S} d_{j'}(\sum_{i\in\tau(j')} c(i,\ j')x^*_{ij'}+c(j',\ \psi(j'))(1 - \sum_{i\in\tau(j')} x^*_{ij'}))$

$< \sum_{j'\in\mathcal{C}_S} d_{j'}(\sum_{i\in\tau(j')} c(i,\ j')x^*_{ij'} + c(j',\ \psi(j'))(1 - \sum_{i\in\tau(j')} x^*_{ij'}))$
$+\sum_{j'\in\mathcal{C}_S} d_{j'}(\sum_{i\in\mathcal{N}_{j'}\setminus\tau(j')}(c(i,\ j')-c(j',\ \psi(j')))x^*_{ij'})$ as $c(i,\ j') > c(j',\ \psi(j'))\ \forall i \in \mathcal{N}_{j'}\setminus\tau(j')$

$= \sum_{j'\in\mathcal{C}_S} d_{j'}(\sum_{i\in\mathcal{N}_{j'}} c(i,\ j')x^*_{ij'} + c(j',\ \psi(j'))(1 - \sum_{i\in\mathcal{N}_{j'}} x^*_{ij'}))$. Thus, by equation (6), we get $\sum_{j'\in\mathcal{C}_S} d_{j'}(\sum_{i\in\mathcal{N}_{j'}} c(i,\ j')w'_i + c(j',\ \psi(j'))(1 - \sum_{i\in\mathcal{N}_{j'}} w'_i)) \leq 12LP_{opt}$.

Thus, the solution w' is feasible and $\mathcal{C}ostKM(w')$,

$$\sum_{j'\in\mathcal{C}_S} d_{j'}\left[\sum_{i\in\mathcal{N}_{j'}} c(i,\ j')w'_i + c(j',\ \psi(j'))\left(1 - \sum_{i\in\mathcal{N}_{j'}} w'_i\right)\right] + u\sum_{j'\in\mathcal{C}_D}\sum_{i\in\mathcal{N}_{j'}} c(i,\ j')w'_i \leq (2\ell + 13)LP_{opt}.$$

5.2 Proof of Lemma 11

i) We will prove the claim by induction. Let $LP^{(t)}$ denote the LP at the beginning of the t^{th} iteration and $\tilde{w}^{(t)}$ denote the solution at the end of the t^{th} iteration. We will show that if $\tilde{w}^{(t)}$ is a feasible solution to LP_2, then $\tilde{w}^{(t+1)}$ is also a feasible solution to LP_2. Since $\tilde{w}^{(1)}$ is feasible (extreme point solution), the feasibility of the solution follows. Let $\tilde{\mathcal{F}}^{(t)}, \tilde{\mathcal{B}}^{(t)}, s^{(t)}, S^{(t)}, \hat{\tau}()^{(t)}, R^{(t)}$ denote the values at the beginning of the t^{th} iteration. Then, $\tilde{w}_i^{(t+1)} = \tilde{w}_i^{(t)}\ \forall i \in \mathcal{F} \setminus \tilde{\mathcal{F}}^{(t+1)}$.

Consider a constraint that was not present in $LP^{(t+1)}$. In any iteration, we remove a constraint only when none of the facilities in its corresponding clusters is fractionally opened. That is all the facilities in $\tau(j')$ appearing on the left hand side of a constraint are integral. Thus $\tilde{w}_i^{(t+1)} = \tilde{w}_i^{(t)}$ for all such facilities. Hence if they are satisfied by $\tilde{w}^{(t)}$ then they are satisfied by $\tilde{w}^{(t+1)}$. So, we consider only those constraints that were present in $LP^{(t+1)}$. For $j' \in \mathcal{C}_S$, since $\hat{\tau}(j')^{(t+1)} = \tau(j') \setminus \tilde{\mathcal{F}}_0^{(t)}$ $\forall t$, therefore, $\sum_{i\in\hat{\tau}(j')^{(t+1)}} \tilde{w}_i^{(t+1)} = \sum_{i\in\tau(j')} \tilde{w}_i^{(t+1)}$ $\forall t$. Thus, we will omit (t) and use $\tau()$ instead of $\hat{\tau}()$ for brevity of notation.

Consider constraints (7) that were not removed in t^{th} iteration. Since $\tau(j') \subseteq \tilde{\mathcal{F}}^{(t+1)}$ for $j' \in \mathcal{C}_S$, the feasibility of the constraint follows as $\tilde{w}^{(t+1)}$ is an extreme point solution of the reduced LP over the set $\tilde{\mathcal{F}}^{(t+1)}$.

Next, consider constraints (8). Let $\mathcal{F}_1^{(t)}$ denote the set of facilities that are opened integrally in $\tilde{w}^{(t)}$ i.e., $\tilde{w}_i^{(t)} = 1$ $\forall i \in \mathcal{F}_1^{(t)}$ then the corresponding constraint in $LP^{(t+1)}$ is $\sum_{i\in\tau(j')\setminus\mathcal{F}_1^{(t)}} w_i = \lfloor \frac{d_{j'}}{u} \rfloor - |\mathcal{F}_1^{(t)}|$. Since $\tilde{w}^{(t+1)}$ is an extreme point solution of $LP^{(t+1)}$, it satisfies this constraint i.e., $\sum_{i\in\hat{\tau}(j')\setminus\mathcal{F}_1^{(t)}} \tilde{w}_i^{(t+1)} = \lfloor \frac{d_{j'}}{u} \rfloor - |\mathcal{F}_1^{(t)}|$. Since $w_i^{(t+1)} = w_i^{(t)} = 1$ $\forall i \in \mathcal{F}_1^{(t)}$, adding $\mathcal{F}_1^{(t)}$ on both the sides, we get the desired feasibility.

Consider constraints (9). Since $\tilde{w}^{(t)}$ is feasible for LP_2, we have $\sum_{j'\in G_r\cap\mathcal{C}_S} \sum_{i\in\tau(j')} \tilde{w}_i^{(t)} \geq \alpha_r$ and since $\tilde{w}^{(t+1)}$ is feasible for $LP^{(t+1)}$, we have $\sum_{j'\in S_r^{(t+1)}} \sum_{i\in\tau(j')} \tilde{w}_i^{(t+1)} \geq s_r^{(t+1)}$. Then, $\sum_{j'\in G_r\cap\mathcal{C}_S} \sum_{i\in\tau(j')} \tilde{w}_i^{(t+1)} = \sum_{j'\in(G_r\cap\mathcal{C}_S)\setminus S_r^{(t+1)}} \sum_{i\in\tau(j')} \tilde{w}_i^{(t+1)} +$ $\sum_{j'\in S_r^{(t+1)}} \sum_{i\in\tau(j')} \tilde{w}_i^{(t+1)} \geq \sum_{j'\in(G_r\cap\mathcal{C}_S)\setminus S_r^{(t+1)}} \sum_{i\in\tau(j')} \tilde{w}_i^{(t)} + s_r^{(t+1)}$ $= \sum_{j'\in(G_r\cap\mathcal{C}_S)\setminus S_r^{(t+1)}} 1 + s_r^{(t+1)}$ (as these clusters must have been removed as they got tight) $= |(G_r \cap \mathcal{C}_S) \setminus S_r^{(t+1)}| + s_r^{(t+1)} = \alpha_r$.

Next, consider constraint (10). Since $\tilde{w}^{(t)}$ is feasible for LP_2, we have $\sum_{i\in\mathcal{F}} f_i\tilde{w}_i^{(t)} \leq \mathcal{B}$ and since $\tilde{w}^{(t+1)}$ is feasible for $LP^{(t+1)}$, we have $\sum_{i\in\tilde{\mathcal{F}}^{(t+1)}} f_i\tilde{w}_i^{(t+1)} \leq \tilde{\mathcal{B}}^{(t+1)}$. Also, we have $w_i^{(t+1)} = w_i^{(t)}$ $\forall i \in \mathcal{F} \setminus \tilde{\mathcal{F}}^{(t+1)}$. Consider $\sum_{i\in\mathcal{F}} f_i\tilde{w}_i^{(t+1)} = \sum_{i\in\mathcal{F}\setminus\tilde{\mathcal{F}}^{(t+1)}} f_i\tilde{w}_i^{(t+1)} +$ $\sum_{i\in\tilde{\mathcal{F}}^{(t+1)}} f_i\tilde{w}_i^{(t+1)} \leq \sum_{i\in\mathcal{F}\setminus\tilde{\mathcal{F}}^{(t+1)}} f_i\tilde{w}_i^{(t)} + \tilde{\mathcal{B}}^{(t+1)}$.

And since $\tilde{\mathcal{B}}^{(t+1)} = \mathcal{B} - \sum_{i\in\mathcal{F}\setminus\tilde{\mathcal{F}}^{(t+1)}} f_i\tilde{w}_i^{(t)}$, we have $\sum_{i\in\mathcal{F}} f_i\tilde{w}_i^{(t+1)} \leq \mathcal{B}$. Thus, the solution $\tilde{w}^{(t+1)}$ is feasible.

ii) Consider the last iteration of the algorithm. The iteration ends either at step $(3-4)$ or at step $(9-10)$. In the former case, the solution clearly has no fractionally opened facility. Suppose we are in the latter case. Let the linearly independent tight constraints corresponding to (7), (8) and (9) be denoted as $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ respectively. Let A and B be set of variables corresponding to some constraint in $\mathcal{X}$ and $\mathcal{Z}$ respectively such that $A \cap B \neq \emptyset$. Then, $A \subseteq B$. Imagine deleting A from B and subtracting 1 from s_r. Repeat the process with another such constraint in $\mathcal{X}$ until there is no more constraint in $\mathcal{X}$ whose variable set has a non-empty intersection with B. At this point, $s_r \geq 1$ and the number of variables in B is at least 2. Number of variables in any set corresponding to a tight constraint in $\mathcal{X}$ (or $\mathcal{Y}$) is also at least 2. Thus, the total number of variables is at least $2|\mathcal{X}|+2|\mathcal{Y}|+2|\mathcal{Z}|$ and the number of tight constraints is at most $|\mathcal{X}|+|\mathcal{Y}|+|\mathcal{Z}|+1$. Thus, we get $|\mathcal{X}| + |\mathcal{Y}| + |\mathcal{Z}| \leq 1$ and hence there at most two (fractional) variables.

iii) Note that no facility is opened in $\mathcal{N}_{j'} \setminus \tau(j') : j' \in \mathcal{C}_S$ for if $i \in \mathcal{N}_{j'} \setminus \tau(j') : j' \in \mathcal{C}_S$ is opened, then it can be shut down and the demand $d_{j'}\tilde{w}_i$, can be shipped to $\psi(j')$, decreasing the cost as $c(j',\ \psi(j')) < c(i,\ j')$. Then, the claim follows as we compute extreme point solution in step (7) in the first iteration and the cost never increases in subsequent calls.

5.3 Proof of Lemma 12

i) Let $j_d \in \mathcal{C}_D \cap G_r$. Total demand d_{j_d} of j_d can be distributed to the opened facilities ($\geq \lfloor d_{j_d}/u \rfloor$) at a loss of factor 2 in capacity and cost both, as $d_{j_d}/u - \lfloor d_{j_d}/u \rfloor < 1 \leq \lfloor d_{j_d}/u \rfloor$.
For $\sigma_r = 0$, (ii) - (v) hold vacuously. So, let $\sigma_r \geq 1$.

ii) LP_2 opens $\alpha_r = \max\{0, \sigma_r - 1\}$ facilities in $G_r \cap \mathcal{C}_S$. Constraint (7) ensures that at most one facility is opened in each sparse cluster. Thus, there is at most one cluster in $G_r \cap \mathcal{C}_S$ with no facility opened in it.

iii) and iv) Let $j' \in G_r \cap \mathcal{C}_S$ such that no facility is opened in $\tau(j')$. If j' is not the root of G_r or G_r is a root MC, then LP_2 must have opened a facility in $\tau(\psi(j'))$. Demand of j' is assigned to this facility at a loss of maximum 2 factor in capacity if $\psi(j') \in \mathcal{C}_S$ and 3 if $\psi(j') \in \mathcal{C}_D$: $d_{\psi(j')} = 1.99u$ and $d_{j'} = .99u$. Otherwise (if j' is the root of G_r and G_r is not a root MC), at most u units of demand of G_r remain unassigned within G_r. (v) holds as$\lfloor d_{j_d}/u \rfloor$ facilities are opened in the cluster centered at j_d and $\alpha_r = \max\{0, \sigma_r - 1\}$ facilities are opened in $G_r \cap \mathcal{C}_S$ by constraints (8) and (9) respectively. (vi) Since the demand $d_{j'}$ of $j' \in G_r$ is served either within its own cluster or in the cluster centered at $\psi(j')$, total distance traveled by demand $d_{j'}$ of j' to reach the centers of the clusters in which they are served is bounded by $d_{j'}c(j',\ \psi(j'))$.

5.4 Proof of Lemma 13

After assigning the demands of the clusters within G_r as explained in Lemma (12), demand coming from all the children meta-clusters are distributed proportionately to facilities within G_r utilizing the remaining capacities. Next, we will show that this can be done within the claimed capacity bound.

i) Let G_r be a non leaf meta-cluster with a dense cluster $j' \in \mathcal{C}_D$ at the root, if any. Also, let t_r be the total number of clusters in G_r, i.e., $t_r = \delta_r + \sigma_r$. The total demand to be served in G_r is at most $u(\lfloor d_{j'}/u \rfloor + 1 + \sigma_r) + u(t_r + 1) \leq (\beta_r + 2)u + (t_r + 1)u$ whereas the total available capacity is at least $\beta_r u$ by Lemma (12). Thus, the capacity violation is bounded by $\frac{(\beta_r+2)u+(t_r+1)u}{\beta_r u} \leq \frac{(\beta_r+2)u+(\beta_r+2)u}{\beta_r u} = 2 + 4/\beta_r \leq 2 + 4/(\ell - 1)$ (as $\lfloor d_{j'}/u \rfloor \geq \delta_r$ we have $\beta_r \geq \sigma_r - 1 + \delta_r = t_r - 1 = \ell - 1$ for a non-leaf MC).
The capacity violation of factor 3 can happen in the case when no facility is opened in $\tau(j')$ for $j' \in \mathcal{C}_S$ and $\psi(j') \in \mathcal{C}_D$ as explained in Lemma (12).
Leaf meta-clusters may have length less than l but they do not have any demand coming onto them from the children meta-cluster, thus capacity violation is bounded as explained in Lemma (12).

ii) Let j' belongs to a MC G_r such that its demand is not served within G_r. Then, j' must be the root of G_r and its demand is served by facilities in clusters of the parent MC, say G_s. Since the edges in G_s are no costlier than the connecting edge $(j', \psi(j'))$ of G_r and there are at most $\ell - 1$ edges in G_s, the total distance traveled by demand $d_{j'}$ of j' to reach the centers of the clusters in which they are served is bounded by $\ell d_{j'}c(j',\ \psi(j'))$.

5.5 Proof of Lemma 14

Let $j' \in \mathcal{C}'$. Let $\lambda(j')$ be the set of centers j'' such that facilities in $\tau(j'')$ serve the demand of j'. Note that if some facility is opened in $\tau(j')$, then $\lambda(j')$ is $\{j'\}$ itself and if no facility is opened in $\tau(j')$, then $\lambda(j') = \{j'' : \exists i \in \tau(j'')$ such that demand of j' is served by i as per the assignments done in Lemmas (12) and (13)$\}$.

The cost of assigning a part of the demand $d_{j'}$ to a facility opened in $\lambda(j') \cap \mathcal{C}_S$ is bounded differently from the part assigned to facilities in $\lambda(j') \cap \mathcal{C}_D$.

Let $j'' \in \mathcal{C}_S \cap \lambda(j')$, $i \in \tau(j'')$. Then, $c(j'', i) \leq c(j'', \psi(j'')) \leq c(j', \psi(j'))$. Last inequality follows as: either j'' is above j' in the same MC (say G_r) (by Lemma (12.3)) or j'' is in the parent MC (say G_s) of G_r. In the first case, the edge $(j'', \psi(j''))$ is either in G_r or is the connecting edge of G_r. The inequality follows as edge costs are non-increasing as we go up the tree. In the latter case, edge $(j'', \psi(j''))$ is either in G_s or it is the connecting edge of G_s: in either case, $c(j'', \psi(j'')) \leq c(j', \psi(j'))$ as the connecting edge of G_s is no costlier than the edges in G_s which are no costlier than the connecting edge of G_r (possibly $c(j', \psi(j'))$) which are no costlier than the edges in G_r. Summing over all $j', j'' \in \mathcal{C}_S$, we see that this cost is bounded by $O(1)LP_{opt}$.

Next, let $j'' \in \mathcal{C}_D \cap \lambda(j'), i \in \mathcal{N}_{j''}$. Further, let g_i be the total demand served by a facility i. Since $g_i \leq 3u$, the cost of transporting $3u$ units of demand from j'' to i is $3u\hat{w}_i c(i, j'')$. Summing it over all $i \in \mathcal{N}_{j''}$, $j'' \in \mathcal{C}_D$, and then over all $j' \in \mathcal{C}'$, we get that the total cost for $\mathcal{C}_D$ is bounded by $O(1)LP_{opt}$.

5.6 Proof of Lemma 17

i) Let $j_d \in \mathcal{C}_D \cap G_r^1$. Consider the case when $res(j_d) < \epsilon$. The total demand $(\lfloor d_{j_d}/u \rfloor + res(j_d))u \leq (\lfloor d_{j_d}/u \rfloor + \epsilon)u$ of G_r^1 can be distributed to the opened facilities ($\geq \lfloor d_{j_d}/u \rfloor$) at a loss of factor 2 in capacity as $\lfloor d_{j_d}/u \rfloor \geq 1$.
When $\epsilon \leq res(j_d) < 1$, the demand of G_r^1 is at most $(\lfloor d_{j_d}/u \rfloor + res(j_d) + 1)u \leq (\lfloor d_{j_d}/u \rfloor + 2)u$. The available opening is $\lfloor d_{j_d}/u \rfloor + 1$. Thus, the capacity violation is at most $(\lfloor d_{j_d}/u \rfloor + 2)u/(\lfloor d_{j_d}/u \rfloor + 1)u < 2$ as $\lfloor d_{j_d}/u \rfloor \geq 1$. Hence G_r^1 is self-sufficient.
For $\sigma_r = 0$, (ii) - (vi) hold vacuously. Thus, now onwards we assume that $\sigma_r \geq 1$.

ii) LP_2 opens $\max\{0, \sigma'_r - 1\}$ facilities in G_r^2 where σ'_r is the number of clusters in G_r^2. Constraint (12) ensures that at most one facility is opened in each cluster. Thus, there is at most one cluster in G_r^2 with no facility opened in it and it is a sparse cluster. Next consider G_r^1 with a sparse cluster in it, i.e., $G_r^1 = \{j_d, j_s\}$, it is possible that all the γ_r facilities are opened in $\tau(j_d)$ and no facility is opened in $\tau(j_s)$. Thus, there are at most two clusters with no facility opened in them and these clusters are sparse.

iii) and iv) Let $j' \in G_r^2$ such that no facility is opened in $\tau(j')$. If $\psi(j') \in G_r^2$, then LP_2 must have opened a facility in $\tau(\psi(j'))$. Demand of j' is assigned to this facility at a loss of maximum 2 factor in capacity. If $\psi(j') \notin G_r^2$ then either G_r^1 is empty or $\psi(j') \in G_r^1$. In the former case j' must be the root of G_r and G_r cannot be the root MC. Clearly, at most u units of demand of G_r remain unassigned within G_r. In the latter case i.e., $\psi(j') \in G_r^1$, then $\psi(j')$ is either j_d or j_s.

v) and vi) We will next show that demand of j' will be absorbed in $\tau(j_d) \cup \tau(j_s)$ in the claimed bounds along with claims (v) and (vi) of the lemma.

1. $res(j_d) < \epsilon$, we have $G_r^1 = \{j_d\}$, $\gamma_r = \lfloor d_{j_d}/u \rfloor$, $G_r^2 = G_r \cap \mathcal{C}_S$, $\sigma'_r = \sigma_r$, and $\beta_r = \lfloor d_{j_d}/u \rfloor + \sigma_r - 1$. In this case, $j' = j_s$ and $\psi(j') = j_d$. LP_2 must have opened at least $\lfloor d_{j_d}/u \rfloor \geq 1$ facilities in $\tau(j_d)$ Total demand $(\lfloor d_{j_d}/u \rfloor + res(j_d) + 1))u$ of j_d and j' can be distributed to the facilities opened in $\tau(j_d)$ ($\geq \lfloor d_{j_d}/u \rfloor$) at a loss of factor $2 + \epsilon$ in capacity, as $res(j_d) < \epsilon$ and $1 \leq \lfloor d_{j_d}/u \rfloor$.
2. $\epsilon \leq res(j_d) < 1$, we have $G_r^1 = \{j_d, j_s\}$, $\gamma_r = \lfloor d_{j_d}/u \rfloor + 1$, $G_r^2 = G_r \cap \mathcal{C}_S \setminus \{j_s\}$, $\sigma'_r = \sigma_r - 1$ and $\beta_r = \lfloor d_{j_d}/u \rfloor + \sigma_r - 1$ if $\sigma_r \geq 2$ and $= \lfloor d_{j_d}/u \rfloor + 1$ if $\sigma_r = 1$. In this case, $\psi(j') = j_s$. In the worst case, no facility is opened in $\tau(j_s)$. LP_2 must have opened at least $\lfloor d_{j_d}/u \rfloor + 1 \geq 2$ facilities in $\tau(j_d) \cup \tau(j_s)$. Total demand

$(\lfloor d_{j_d}/u \rfloor + res(j_d) + 1 + 1))u$ of j_d, j_s and j' can be distributed to the facilities opened in $\tau(j_d) \cup \tau(j_s)$ $(\geq \lfloor d_{j_d}/u \rfloor + 1)$ at a loss of factor 2 in capacity, as $\lfloor d_{j_d}/u \rfloor + 1 \geq 2$.

vii) Clearly, $c(j',\ j_d) \leq 2c(j',\ \psi(j'))$. (2) above also handles the case when no facility is opened in a sparse cluster in G_r^1.

A 5-Approximation for Universal Facility Location

Manisha Bansal
Indraprastha College for Women, University of Delhi, India
mbansal@ip.du.ac.in

Naveen Garg
Indian Institute of Technology Delhi, India
naveen@cse.iitd.ac.in

Neelima Gupta
University of Delhi, India
ngupta@cs.du.ac.in

Abstract

In this paper, we propose and analyze a local search algorithm for the *Universal facility location problem*. Our algorithm improves the approximation ratio of this problem from 5.83, given by Angel et al., to 5. A second major contribution of the paper is that it gets rid of the expensive `multi` operation that was a mainstay of all previous local search algorithms for capacitated facility location and universal facility location problem. The only operations that we require to prove the 5-approximation are `add, open`, and `close`. A `multi` operation is basically a combination of the `open` and `close` operations. The 5-approximation algorithm for the capacitated facility location problem, given by Bansal et al., also uses the `multi` operation. However, on careful observation, it turned out that `add, open`, and `close` operations are sufficient to prove a 5-factor for the problem. This resulted into an improved algorithm for the universal facility location problem, with an improved factor.

2012 ACM Subject Classification General and reference → Reference works, General and reference → General conference proceedings, Theory of computation → Facility location and clustering

Keywords and phrases Facility location, Approximation Algorithms, Local Search

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.24

1 Introduction

In a *facility location problem* we are given a set of clients $C = \{1, \ldots, m\}$ and a set of facilities $F = \{1, \ldots, n\}$. A client j has a demand d_j which needs to be serviced by some facilities, i.e., the demand is splittable. The cost of servicing a client $j \in C$ by a facility i is given by c_{ij} (the service cost). The service costs form a metric. Further let, for $i, i' \in F$ $i \neq i'$, $c_{ii'}$ be the cost of the shortest path between i and i', i.e. $c_{ii'} = \min_{j \in C}(c_{ji} + c_{ji'})$. For the sake of simplicity, we consider the case when demand of a client $j \in C$ is one. Arbitrary demands can be easily handled by doing slight modifications, details of which can be found in Pal et al. [4] and Mahdian et al. [3].

In the classical uncapacitated facility location problem (UFL), we are also given $f : F \to \Re^+$, and f_i is the cost of opening a facility at location i. In the capacitated version of the facility location problem, besides the cost of opening a facility at location i we are also given an upper bound u_i on the number of clients that can be served at location i. The Universal facility location (UniFL) problem is a further generalization of the capacitated facility location problem. Now the cost of opening a facility at $i \in F$ depends on the number of clients that this facility would serve and is given by a cost function $f_i(.)$, which is a monotonically non-decreasing function of the capacity allocated to facility i. Thus if u_i is the

capacity allocated to facility i, then $f_i(u_i)$ is its facility opening cost. The aim is to determine a capacity allocation vector $U = \langle u_1, u_2, \cdots, u_n \rangle$ such that the total cost of opening facilities and the cost of serving clients by the open facilities, while respecting capacity constraints, is minimized. Once the allocation vector U is known it is easy to determine the assignment of clients by solving a mincost flow problem. Therefore the capacity allocation vector U completely determines the solution. Note that if $f_i(u_i) = \infty$ for $u_i > c_i$ and $f_i(u_i) = f_i$ otherwise, then we have an instance of the capacitated facility location problem where c_i is the fixed capacity of facility i.

The Universal facility location problem was introduced by Mahdian and Pal [3] who gave a 7.88-approximation algorithm which was improved by Vygen [5] to a 6.702-approximation. This has been further improved to a 5.83-approximation by Angel et al. [1]. Our main result in this paper, is a 5-approximation algorithm for this problem. Our algorithm extends our earlier 5-approximation algorithm for the capacitated facility location problem [2] and borrows heavily from that work. Bansal et al. [2] gave a local search algorithm for capacitated facility location problem that uses the operations `add`, `mopen`, `mclose`, and `mmulti`. Our algorithm too uses the operations `add`, `mopen`, and `mclose` (which we call `open`, and `close` in this paper) but doesn't require the `mmulti` operation. The `mmulti` operation which is a combination of an `open` and `close` operation is an expensive operation to perform and has appeared in some form in all previous works on universal facility location problem and capacitated facility location problem. By getting rid of this expensive operation we hope that our algorithm would be simpler to implement and faster in practice. This paper, thus, not only extends but also simplifies the result in [2]. Bansal et al. [2] argue that the locality gap of any procedure for the capacitated facility location problem that uses the operations `add, mopen, mclose, and mmulti` is atleast 5. This lower bound also applies in our setting since we consider a subset of these operations for a more general problem. When analyzing the cost of an operation we sometimes assign clients fractionally to the facilities. This can be done as it is well known that a fractional assignment cannot be better than the integral optimum assignment, in an assignment problem.

The remainder of this paper is organized as follows: We begin with some preliminaries and in section 3 we present the local search steps. In Section 4 we analyse the local search algorithm by identifying a suitable set of inequalities and finally in Section 5 we put the various pieces together to prove our main theorem. As mentioned earlier, our paper borrows heavily from ideas developed in [2, 6] and other previous work. At many places we have rephrased key arguments to keep the paper self-contained.

2 Preliminaries

A solution to the UniFL problem consists of a capacity allocation vector and an assignment of the clients to the facilities which obey capacity constraints.

Let us consider an allocation vector $U = \langle u_1, u_2, \ldots, u_n \rangle$ for a given instance. We abuse notation and use U to denote both the solution and the allocation vector. The cost of a solution U is denoted by $c(U) = c_f(U) + c_s(U)$, where $c_f(U)$ is the facility cost and $c_s(U)$ is the service cost of the solution U.

Let U be a locally optimal solution and U^* be an optimum solution. For each $s \in F$, u_s (respectively u^*_s) denotes the capacity allocated to s in the locally optimal (respectively optimum) solution. Let F_U (respectively F_{U^*}) be the set of facilities for which u_s (respectively u^*_s) is greater than zero. It is no loss of generality to assume that u_s is the number of clients served by s in U, and u^*_s the no. of clients served by s in U^*.

Let $\sigma(j), \tau(j)$ be the facilities serving client j in solutions U, U^* respectively. Construct a bipartite graph, $G = (C \cup F, E)$ where E contains edges $(\sigma(j), j)$ and $(j, \tau(j))$. Thus each client has one incoming and one outgoing edge while a facility s has u_s outgoing edges and u^*_s incoming edges. The graph G is now decomposed into a set of maximal paths, $\mathcal{P}$, and cycles, $\mathcal{C}$. A path $P \in \mathcal{P}$ is a sequence of vertices $s = s_0, j_0, s_1, j_1, \ldots, s_k, j_k, s_{k+1} = o$, which starts at a vertex $s \in F_U$ and ends at a vertex $o \in F_{U^*}$. Let head(P) denote the client served by s and tail(P) the client served by o on this path. Note that $\{s_1, s_2, \ldots, s_k\} \subseteq F_U \cap F_{U^*}$. Similarly all facilities on a cycle are from $F_U \cap F_{U^*}$.

The *length* of a path P is given by

$$\text{length}(P) = \sum_{j \in C \cap P} (U^*_j + U_j)$$

where U^*_j(respectively U_j) is the service cost of client j in the solution U^*(respectively U). Note that

$$\sum_{P \in \mathcal{P}} \text{length}(P) + \sum_{Q \in \mathcal{C}} \text{length}(Q) = c_s(U) + c_s(U^*).$$

A *shift* along a path P is a reassignment of clients so that j_i which was earlier assigned to s_i is now assigned to s_{i+1}. As a consequence of this shift, facility s serves one client less while facility o serves one client more. shift(P) denotes the increase in service cost due to a shift along P i.e.

$$\text{shift}(P) = \sum_{j \in C \cap P} (U^*_j - U_j).$$

For a cycle in $\mathcal{C}$ the increase in service cost equals the sum of $U^*_j - U_j$ for all clients j in the cycle and since the assignment of clients to facilities is done optimally in our solution and in the global optimum, this sum is zero. Thus

$$\sum_{Q \in \mathcal{C}} \sum_{j \in Q} (U^*_j - U_j) = 0.$$

Let N^o_s be the set of paths that begin at s and end at o. Define $\texttt{out}(s) = \cup_o N^o_s$ as the set of paths starting from s and $\texttt{in}(o) = \cup_s N^o_s$ as the set of paths ending at o. Since the paths chosen are maximal, for any s, at least one of the two sets $\texttt{in}(s)$,$\texttt{out}(s)$ is empty. Let S be the set of facilities for which $\texttt{in}(s)$ is empty and O the set of facilities for which $\texttt{out}(s)$ is empty. Hence, $S \cap O = \phi$. Note that for $s \in S$, $|\texttt{out}(s)| = u_s - u^*_s$ and for $s \in O$, $|\texttt{in}(s)| = u^*_s - u_s$.

Define a capacity function, $\hat{u}$, on the facilities as follows: $\hat{u}(s)$ equals $|\texttt{out}(s)| = u_s - u^*_s$ if $s \in S$, it equals $|\texttt{in}(s)| = u^*_s - u_s$ if $s \in O$ and is 0 for $s \in F \setminus (S \cup O)$. To bound the cost of facilities in S, we reduce the capacity of each facility $s \in S$ from u_s to u^*_s and reassign $u_s - u^*_s = \hat{u}(s)$ clients served by s to facilities in O. We refer to this step as *closing* facility s. Similarly, *opening* a facility $o \in O$ implies increasing its capacity by $u^*_o - u_o = \hat{u}(o)$. `open` and `close` operations are explained in the next section.

To formulate a suitable set of inequalities we formulate an assignment problem where each node $s \in S$ has supply $\hat{u}(s)$ and a node $o \in O$ has demand $\hat{u}(o)$. When a client served by s is transferred to o the increase in service cost is at most c_{so} and hence this is the cost of sending one unit of flow from node s to o. Let $y(s, o)$ be the flow from s to o in an optimum solution to this assignment problem.

▶ **Lemma 1.** *The cost of an optimum flow y is at most $c_s(U) + c_s(U^*)$.*

Proof. Consider a solution $\hat{y}$ defined as $\hat{y}(s,o) = |N_s^o|$. It is easy to check that this is a feasible solution to the assignment problem. Note that for every path $P \in N_s^o$, length$(P) \geq c_{so}$. Hence

$$\begin{aligned}\sum_s \sum_o c_{so}\hat{y}(s,o) &\leq \sum_s \sum_o \sum_{P \in N_s^o} \text{length}(P) \\ &= \sum_{P \in \mathcal{P}} \text{length}(P) \\ &\leq c_s(U) + c_s(U^*)\end{aligned}$$

◀

It is easy to argue that there is an optimum flow y where the edges carrying non-zero flow form an acyclic subgraph. We call this subgraph as exchange graph. Let $G' = (S \cup O, E')$ where $E' = \{(s,o) | y(s,o) > 0\}$ and suppose G' is not acyclic. Let C be the edges on a cycle. Take alternate edges of C to form sets C_1, C_2. Let γ be the minimum flow on an edge in C. Consider two operations - one in which we increase the flow on edges in C_1 and decrease the flow on edges in C_2 by an amount γ and the other in which we do the inverse. In one of these operations the total cost $\sum_{s \in S, o \in O} c_{so} y(s,o)$ would not increase and the flow on one of the edges would reduce to zero thereby removing it from the graph. This process is continued till the graph becomes acyclic. Note that the total flow from nodes of S to a node $o \in O$ is equal to $\hat{u}(o)$ i.e., $\sum_{s \in S} y(s,o) = \hat{u}(o)$ and total flow from a node $s \in S$ to nodes in O is equal to $\hat{u}(s)$, i.e., $\sum_{o \in O} y(s,o) = \hat{u}(s)$.

3 The local search operations

Starting with a feasible solution U, we perform *add, open* and *close* operations to improve the solution U if possible. Given a solution U, we can assume that for each facility $i \in U$, u_i is exactly equal to the number of clients it is serving for if it is not true then we can reduce u_i and hence the cost of the solution. U is locally optimal if none of these operations improve the cost of the solution and at this point the algorithm stops. The *add* operation is the same as given by Mahdian and Pal [3] while the *open* and *close* are almost the same as in [2].

add(s, δ). In this operation the capacity allocated at a facility s is increased by an amount $\delta > 0$. A mincost flow problem is then solved to find the best assignment of clients to the facilities. As a consequence of this operation the cost increases by: $f_s(u_s + \delta) - f_s(u_s) + c_s(U') - c_s(U)$ where U' is the new solution after increasing the capacity of s. This operation can be performed in polynomial time [3] .

open(t, δ_1, δ_2). This operation is best viewed as a combination of two operations. In the first operation the capacity allocated at $t \in F$ is increased by δ_2 and the total capacity of a set T, to be determined as a part of the operation, is decreased by the same amount. The second operation is $add(t, \delta_1 - \delta_2)$. Our procedure for implementing this operation is as follows.

1. We create an instance of the knapsack problem where the sack has capacity δ_2. For each facility $i \in F, i \neq t$, we have for all $0 < j < u_i$, an object of weight j and profit $f_i(u_i) - f_i(u_i - j) - j \cdot c_{it}$. Picking such an object into the knapsack corresponds to reducing the capacity of facility i by j units and the profit is a lower bound on the savings we get and is obtained by reassigning j clients served by i to t. The knapsack

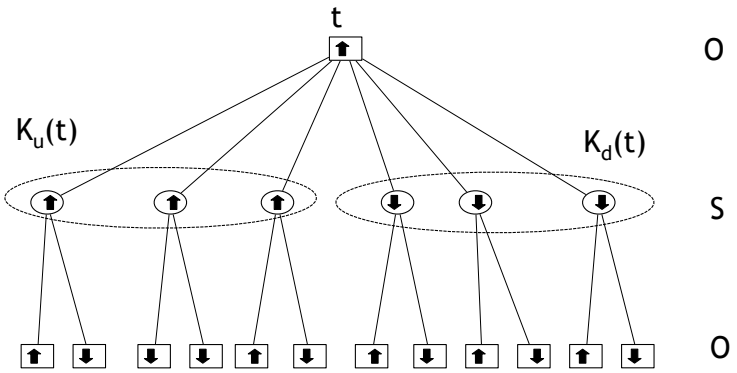


Figure 1 A subtree of height 2 showing up-facilities and down-facilities. The square facilities are in the optimum solution while the circular facilities are in the locally optimum solution. The arrow in the facility identifies it as an up/down facility.

problem is to maximise profit under the constraint that we can pick at most one object corresponding to each facility. Thus, by solving the knapsack problem, we find a set T and for each facility $i \in T$, a quantity j by which the capacity at i is decreased.

2. Independently of the above knapsack procedure we find, by solving a mincost flow problem, the maximum savings in service cost if an additional $\delta_1 - \delta_2$ clients are assigned to t.

The profit in step 1 and the savings in step 2, when reduced by $(f_t(u_t + \delta_1) - f_t(u_t))$ is an estimate of the savings obtained by this operation and if this quantity is positive we have a local improvement. Step 2 can be performed in polynomial time. To perform step 1 in polynomial time, a dynamic programming solution, similar to Mahdian and Pal [3] is used, and values of j are taken to be non-negative integers.

close(t, δ_1, t^*). This operation too is best viewed as a combination of two operations. The first operation is $add(t^*, \delta_2 - \delta_1)$. In the second operation the capacity allocated at $t \in F$ is decreased by δ_1 and the total capacity of a set $T, t^* \in T$, to be determined during the operation, is increased by an amount δ_2, $\delta_2 > \delta_1$. Assuming $\delta_2 \geq 0$ is known, the operation can be implemented as follows.

1. We solve a mincost flow problem to compute the maximum savings in service cost when $\delta_2 - \delta_1$ clients are assigned to t^*.

2. Next we create a knapsack instance with sack capacity δ_1. For all $0 < j < \delta_1$, for each facility $i \in F, i \notin \{t, t^*\}$, we have an object of weight j and profit $f_i(u_i) - f_i(u_i + j) - j \cdot c_{it}$ while for facility $i = t^*$, we have an object of weight j and profit $f_i(u_i + \delta_2 - \delta_1) - f_i(u_i + j + \delta_2 - \delta_1) - j \cdot c_{it}$. As before, the knapsack problem is to maximise profit under the constraint that we can pick at most one object corresponding to each facility.

The savings in step 1 and the profit in step 2, when increased by $(f_t(u_t) - f_t(u_t - \delta_1)) - (f_{t^*}(u_{t^*} + \delta_2 - \delta_1) - f_{t^*}(u_{t^*}))$ is an estimate of the savings obtained by this operation and if this quantity is positive we have a local improvement. To perform step 2 in polynomial time, a dynamic programming solution, similar to Mahdian and Pal [3] is used with values of j being non-negative integers.

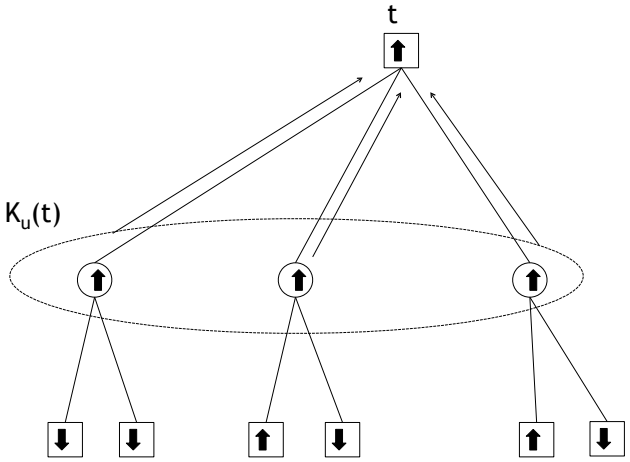


Figure 2 *open* operation considered for handling facilities in $K_u(t)$ when t is an up-facility.

4 Bounding the cost of our solution

Recall that U is a locally optimal solution and U^* is an optimum solution. Also, G' is an exchange graph and y defines an optimum flow on the edges in this graph. We consider potential local improvement steps and using the fact that U is a locally optimal solution, formulate suitable inequalities which help us bound the cost of our solution. The inequalities are written such that

1. each facility in S is closed once.
2. each facility in O is opened at most five times.
3. the total cost of reassigning clients is bounded by

$$2 \sum_{s \in S, o \in O} c_{so} y(s,o) + 3 \sum_{o \in O} \sum_{P \in \mathtt{in}(o)} \text{shift}(P).$$

Every tree in the forest G' is rooted at a facility in O. Consider a subtree T of height 2 having root $t \in O$(Figure 1). For a facility i, let $p(i)$ be the parent and $K(i)$ the children of i. A facility i is an *up-facility* if $y(i,p(i)) \geq \sum_{j \in K(i)} y(i,j)$ and a *down-facility* otherwise. $K_u(i)$ (respectively $K_d(i)$) denote the children of i which are up-facilities (respectively down-facilities).

Our choice of operations, considered for the purpose of analysis, is different from the ones considered in [2] and ensure that for a facility $o \in O$:

1. If o is an *up-facility*, it is opened at most twice in operations involving facilities which are descendants of o in the tree and is opened at most twice in other operations.
2. If o is a *down-facility*, it is opened at most four times in operations involving facilities which are descendants of o in the tree and is opened at most once in other operations.

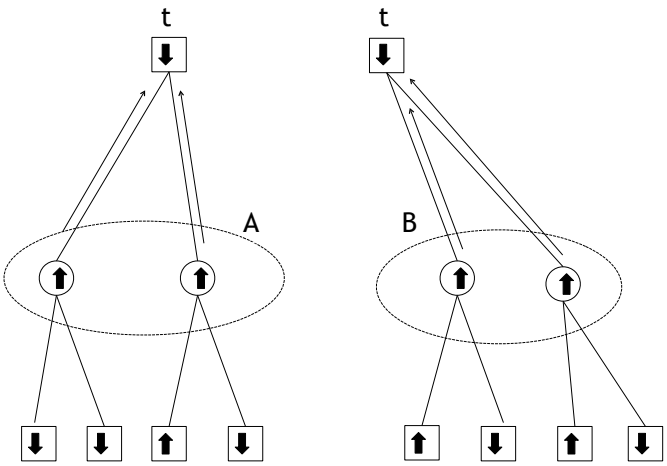


Figure 3 *open* operations considered for facilities in $K_u(t) \setminus \{h\}$ when t is a down facility.

4.1 Closing children of t which are up-facilities

Consider the children of t which are *up-facilities*. We first consider the easier case when t is an up-facility(Figure 2). Now,

$$\hat{u}(t) = \sum_{s \in K(t)} y(s,t) + y(p(t),t) \geq \sum_{s \in K_u(t)} y(s,t) + y(p(t),t) \geq \sum_{s \in K_u(t)} 2y(s,t) \geq \sum_{s \in K_u(t)} \hat{u}(s)$$

where the second last inequality is due to the fact that t is an *up-facility* and the last inequality holds since for all $s \in K_u(t)$ we have

$$\hat{u}(s) = y(s,t) + \sum_{o \in K(s)} y(s,o) \leq 2y(s,t).$$

Thus all facilities $s \in K_u(t)$ can be closed (i.e.,their capacity reduced by $\hat{u}(s)$) in a single operation $open(t, \hat{u}(t), \sum_{s \in K_u(t)} \hat{u}(s))$. We now bound the cost of reassignment of clients as a result of this operation.

1. $\hat{u}(s)$ clients of a facility $s \in K_u(t)$ are assigned to t. Since for $s \in K_u(t)$, $\hat{u}(s) \leq 2y(s,t)$, this reassignment cost is at most $2y(s,t)c_{st}$.
2. We can assign an additional $\hat{u}(t) - \sum_{s \in K_u(t)} \hat{u}(s)$ clients to t. One way of doing this is by shifting to an extent $(1 - \sum_{s \in K_u(t)} \hat{u}(s)/\hat{u}(t))$ along each of the $\hat{u}(t)$ paths in $\mathtt{in}(t)$. Since U is locally optimal, this operation will not improve the cost of U. This operation then yields the inequality

$$f_t(u_t^*) - f_t(u_t) - \sum_{s \in K_u(t)} (f_s(u_s) - f_s(u_s^*)) +$$
$$\sum_{s \in K_u(t)} 2y(s,t)c_{st} + \left(1 - \sum_{s \in K_u(t)} \hat{u}(s)/\hat{u}(t)\right) \sum_{P \in \mathtt{in}(t)} \text{shift}(P) \geq 0 \qquad (1)$$

We next consider the case when t is a down-facility (Figure 3) and begin by noting that

$$\sum_{s \in K_u(t)} \hat{u}(s) \leq \sum_{s \in K_u(t)} 2y(s,t) \leq 2 \sum_{s \in K(t)} y(s,t) \leq 2\hat{u}(t).$$

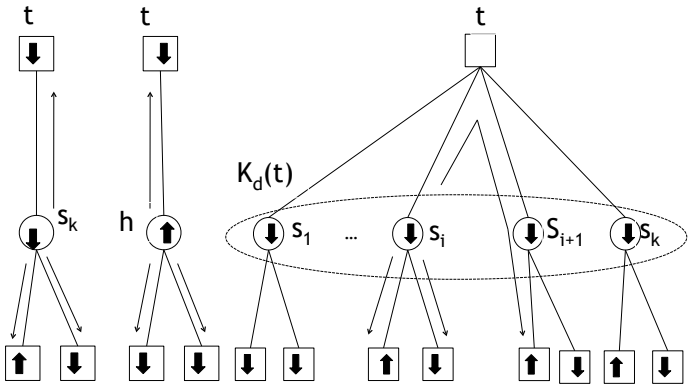


Figure 4 *close* operations for facilities in $K_d(t)$ and facility h showing the reassignment of clients when one of these facilities are closed.

Let $h = \arg\max_{s \in K_u(t)} y(s,t)$ and partition the facilities in $K_u(t) \setminus h$ into two sets A and B such that $\sum_{s \in A} \hat{u}(s) \leq \hat{u}(t)$ and $\sum_{s \in B} \hat{u}(s) \leq \hat{u}(t)$. The facilities in sets A and B are closed in two *open* operations $open(t, \hat{u}(t), \sum_{s \in A} \hat{u}(s))$ and $open(t, \hat{u}(t), \sum_{s \in B} \hat{u}(s))$ respectively; see Figure 3.

The extra capacity available at t in each of these open operations is used to assign additional clients to t in the same manner as done earlier.

The facility h is handled together with the facilities in $K_d(t)$ using *close* operations as discussed next.

4.2 Closing facility h and down-facilities which are children of t

Now we discuss the operations to close facilities $s \in K_d(t) \cup \{h\}$ and refer to Figure 4. Consider the facilities in $K_d(t)$. As in [2, 5], for every $s \in K_d(t)$ we define $rem(s) = y(s,t) - \sum_{o \in K_d(s)} y(s,o)$ and rename the facilities in $K_d(t)$ so that $rem(s_1) \leq rem(s_2) \leq \cdots rem(s_k)$.

We can close facility $s_i \in K_d(t), i < k$ by reassigning $\hat{u}(s_i)$ of its clients to facilities in $K(s_i) \cup K_u(s_{i+1})$ as follows:

1. $y(s_i, o)$ clients are reassigned to $o \in K_u(s_i)$.
2. $2y(s_i, o)$ clients are reassigned to $o \in K_d(s_i)$. Since o is a down-facility, $y(s_i, o) \leq \sum_{s' \in K(o)} y(s', o)$ and hence $2y(s_i, o) \leq \hat{u}(o)$.
3. This leaves $rem(s_i) = y(s_i, t) - \sum_{o \in K_d(s_i)} y(s_i, o)$ clients, which are reassigned to facilities in $K_u(s_{i+1})$. Doing so is feasible since

$$rem(s_i) \leq rem(s_{i+1}) = y(s_{i+1}, t) - \sum_{o \in K_d(s_{i+1})} y(s_{i+1}, o) \leq \sum_{o \in K_u(s_{i+1})} y(s_{i+1}, o).$$

We denote by $z(s_i, o)$ the number of clients reassigned to $o \in K(s_i) \cup K_u(s_{i+1})$ in the above argument.

We can formulate the following inequality, w.r.t.closing of facility s_i:

▶ **Lemma 2.**

$$f_{s_i}(u_{s_i}) - f_{s_i}(u^*_{s_i}) - \sum_{o \in K(s_i) \cup K_u(s_{i+1})} (f_o(u^*_o) - f_o(u_o))$$
$$\leq \sum_{o \in K(s_i) \cup K_u(s_{i+1})} \left(z(s_i, o) c_{s_i o} + \sum_{P \in \boldsymbol{in}(o)} \left(1 - \frac{z(s_i, o)}{\hat{u}(o)}\right) \mathit{shift}(P) \right) \tag{2}$$

Proof. Denote the set $K(s_i) \cup K_u(s_{i+1})$ by T. Consider the facilities of T in decreasing order of $z(s_i, o)/\hat{u}(o)$ and keep including them into a set T' until the total capacity of the facilities in T', i.e. $\sum_{o \in T'} \hat{u}(o)$, exceeds $\hat{u}(s_i)$. Let t^* be the last facility to be included into T' and $k = \hat{u}(s_i) - \sum_{o \in T' \setminus \{t^*\}} \hat{u}(o)$ be the number of clients reassigned from s_i to t^*. Then a $\mathit{close}(s_i, \hat{u}(s_i), t^*)$ operation which reassigns $\hat{u}(o)$ clients from s_i to $o \in T' \setminus \{t^*\}$, k clients from s_i to t^* and an additional $\hat{u}(t^*) - k$ clients to t^* by performing a shift along each path in $\mathtt{in}(t^*)$ to an extent $1 - k/\hat{u}(t^*)$ yields the inequality

$$f_{s_i}(u_{s_i}) - f_{s_i}(u^*_{s_i}) - \sum_{o \in T'} (f_o(u^*_o) - f_o(u_o))$$
$$\leq \sum_{o \in T' \setminus \{t^*\}} \hat{u}(o) c_{s_i o} + k c_{s_i t^*} + (1 - k/\hat{u}(t^*)) \sum_{P \in \mathtt{in}(t^*)} \mathrm{shift}(P) \tag{3}$$

We now build a linear combination by taking inequality 3 to an extent of ξ, reduce $z(s_i, o)$ by $\xi \cdot \hat{u}(o)$ for all facilities $o \in T' \setminus \{t^*\}$ and reduce $z(s_i, t^*)$ by $\xi \cdot k$, where ξ is the largest value such that $z(s_i, o) \geq 0, o \in T'$. We keep building the linear combination in this manner till $\xi = 0$. This process can be viewed as sending $\xi \cdot \hat{u}(s_i)$ units of flow from s_i to facilities in T' with facility $o \in T' \setminus \{t^*\}$ receiving $\xi \cdot \hat{u}(o)$ flow and facility t^* receiving $\xi \cdot k$ flow. The edges (s_i, o) have capacity $z(s_i, o)$ which is reduced by the amount of flow sent in each step. Initially the total capacity of all edges $\sum_{o \in T} z(s_i, o)$ equals the amount of flow $\hat{u}(s_i)$ that needs to be sent and this property is maintained at each step. By picking the facilities with the largest values of $z(s_i, o)/\hat{u}(o)$ we are ensuring that the maximum of these quantities never exceeds the fraction of the flow that remains to be sent. This implies that when the procedure terminates all $z(s_i, o)$ are zero and $\hat{u}(s_i)$ units of flow have been sent.

If a facility $o \in T$ was opened to an extent λ_o in the above process, then its contribution in the linear combination would be $\lambda_o(f_o(u^*_o) - f_o(u_o)) + z(s_i, o) c_{s_i o} + (\lambda_o - z(s_i, o)/\hat{u}(o)) \sum_{P \in \mathtt{in}(o)} \mathrm{shift}(P)$. We add a $1 - \lambda_o$ multiple of the inequality

$$f_o(u^*_o) - f_o(u_o) + \sum_{P \in \mathtt{in}(o)} \mathrm{shift}(P) \geq 0 \tag{4}$$

which corresponds to the operation $\mathtt{add}(o, \hat{u}(o))$, to the linear combination to match the contribution of o in Inequality 2. ◀

Note that due to the above process, s_i is closed to an extent of one and $o \in T$ is opened to an extent of one.

We now consider operations of the kind $\mathit{close}(s_k, \hat{u}(s_k), t^*)$ where $t^* \in K(s_k) \cup \{t\}$ to handle s_k. As before we build a suitable linear combination of the inequalities arising from these operations while ensuring that the total number of clients reassigned from s_k to $o \in K(s_k) \cup \{t\}$ is $z(s_k, o) = y(s_k, o)$. The inequality corresponding to this linear combination

is given by

$$(f_{s_k}(u_{s_k}) - f_{s_k}(u^*_{s_k})) - \sum_{o \in K(s_k) \cup \{t\}} (f_o(u^*_o) - f_o(u_o))$$
$$\leq \sum_{o \in K(s_k) \cup \{t\}} z(s_k, o) c_{s_k o} + \sum_{o \in K(s_k) \cup \{t\}} \sum_{P \in \mathtt{in}(o)} (1 - z(s_k, o)/\hat{u}(o)) \text{shift}(P) \tag{5}$$

Note that in the above process, s_k is closed to an extent of one and $o \in K(s_k) \cup \{t\}$ is opened to an extent of one.

The following lemma bounds the cost of reassigning clients (excluding the cost of shifting along paths in $\mathcal{P}$) in the close operations on facilities in $K_d(t)$.

▶ **Lemma 3.**

$$\sum_{i=1}^{k} \sum_{o} z(s_i, o) c_{s_i o} \leq 2 \sum_{i=1}^{k} \sum_{o \in K(s_i) \cup \{t\}} y(s_i, o) c_{s_i o}$$

Proof. We begin by observing that since edge costs form a metric, $c_{s_i o}, o \in K_u(s_{i+1})$ is at most $c_{s_i t} + c_{t s_{i+1}} + c_{s_{i+1} o}$.

The contribution of the edge $(s_i, t), i \neq 1, k$ to the reassignment cost is at most $(\mathtt{rem}(s_i) + \mathtt{rem}(s_{i-1})) c_{s_i t}$. Since both $\mathtt{rem}(s_{i-1}) \leq \mathtt{rem}(s_i) \leq y(s_i, t)$ the total contribution of this edge is at most $2y(s_i, t) c_{s_i t}$. The contribution of the edge (s_1, t) to the reassignment cost is at most $\mathtt{rem}(s_1) c_{s_1 t} \leq y(s_1, t) c_{s_1 t}$ while the contribution of the edge (s_k, t) to the reassignment cost is at most $(\mathtt{rem}(s_{k-1}) + y(s_k, t)) c_{s_k t} \leq 2y(s_k, t) c_{s_k t}$.

The contribution of the edge $(s_i, o), o \in K_d(s_i)$ is at most $2y(s_i, o) c_{s_i o}$ since $2y(s_i, o)$ clients are assigned to o when s_i is closed.

The contribution of the edge $(s_i, o), o \in K_u(s_i)$, $i \neq 1$ is at most $2y(s_i, o) c_{s_i o}$ since at most $y(s_i, o)$ clients are assigned to j once when s_i is closed and once when s_{i-1} is closed. The contribution of the edge $(s_1, o), o \in K_u(s_1)$ is at most $y(s_1, o) c_{s_1 o}$. ◀

Finally, by considering operations $close(h, \hat{u}(h), t^*)$ where $t^* \in K(h) \cup \{t\}$ and taking a suitable linear combination we obtain an inequality similar to inequality 5 with h replacing s_k. Note that in this operation, the contribution of an edge (h, o), $o \in K(h) \cup \{t\}$ is at most $y(h, o) c_{ho}$. Also note that h is closed to an extent of one and $o \in K(h) \cup \{t\}$ is opened to an extent of one in this process.

5 Putting Things Together

In all the operations discussed in the previous section, a facility $o \in O$ is opened at most 5 times and cost of reassignment of clients in all these operations is small. We prove these facts in the following lemmas.

▶ **Lemma 4.** *A facility $o \in O$ is opened at most 5 times and is assigned a total of at most $2 \sum_s y(s, o) \leq 2\hat{u}(o)$ clients, from the facilities closed in the respective operations, over all the operations considered.*

Proof.

When o is an up-facility: While considering the facilities of S which are descendants of o, o would be opened twice, once when it is part of *close* operations $close(s_k, \hat{u}(s_k), t)$, $s_k \in K_d(o), t \in K(s_k) \cup \{o\}$ and once when it is part of an *open* operation $open(o, \hat{u}(o)$,

$\sum_{s\in K_u(o)} \hat{u}(s)$). o is assigned at most $2\sum_{s\in K_u(o)} y(s,o)+y(s_k,o)$ clients where $s_k \in K_d(o)$. Note that this is at most $2\sum_{s\in K(o)} y(s,o)$.

While considering the facilities of S which are not descendants of o, if the parent of o, $p(o)$, is an up-facility, o would be opened once if $p(o) = h$. In this case a `close` operation involving $p(o)$ assigns at most $y(p(o), o)$ clients to o. If $p(o) \neq h$ then o is not opened and no client is assigned to it.

If $p(o)$ is a down-facility then o would be opened at most twice and would be assigned at most $2y(p(o), o)$ clients. This can be argued in a straightforward manner by considering the 3 cases: $p(o) = s_1; p(o) = s_i, i \neq 1, k; p(o) = s_k$.

When o is a down-facility: While considering the facilities of S which are descendants of o, o would be opened four times: once when it is part of *close* operations $close(s_k, \hat{u}(s_k), t)$ where $s_k \in K_d(o), t \in K(s_k)\cup\{o\}$, once when it is part of *close* operations $close(h, \hat{u}(h), t)$ where $h \in K_u(o), t \in K(h) \cup \{o\}$, and twice as a part of two *open* operations in which sets $A, B \subseteq K_u(o)$ are closed. The number of clients assigned to o in these operations is $2\sum_{s\in A} y(s,o)$, $2\sum_{s\in B} y(s,o)$, $y(h,o)$ and $y(s_k,o)$ respectively. Since $A\cup B\cup\{h\} = K_u(o)$ and $s_k \in K_d(o)$, the total number of clients assigned to o in these four operations is at most $2\sum_{s\in K(o)} y(s,o)$.

o would be opened at most once while considering the facilities of S, which are not descendants of o irrespective of whether $p(o)$ is an up-facility or a down-facility. If the parent of o, $p(o)$, is an up-facility then o would be assigned at most $y(p(o), o)$ clients in a *close* operation involving $p(o)$. If $p(o)$ is a down-facility then o would be assigned at most $2y(p(o), o)$ clients and once again this can be argued by considering 2 cases: $p(o) = s_i, i \neq k; p(o) = s_k$. Therefore the total number of clients assigned to o when o is a down-facility is at most $2\sum_s y(s,o)$. ◀

▶ **Lemma 5.** *The total reassignment cost of all the operations is bounded by*

$$2\sum_{s\in S, o\in O} c_{so}y(s,o) + 3\sum_{o\in O}\sum_{P\in N_{U^*}(o)} shift(P)$$

Proof. The first term in the required expression follows from the fact that in all the operations considered, the contribution of an edge (s,o) of the exchange graph is at most $2c_{so}y(s,o)$.

When all the inequalities are added, the term $\sum_{P\in \mathtt{in}(o)} \text{shift}(P)$ for a facility $o \in O$ appears to the extent of $\alpha - \beta/\hat{u}(o)$ where α is the number of times o is opened and β is the total number of clients assigned to o from the facilities whose capacity allocation decreases in the operation in which o is opened. From Lemma 4, β is at most $2\hat{u}(o)$ and α is at most 5. If a facility $o \in O$ is opened less than five times in these operations then we add the inequality corresponding to $add(o, \hat{u}(o))$ to our linear combination so that each facility is now opened exactly five times. Therefore, the coefficient of the term $\sum_{P\in \mathtt{in}(o)} \text{shift}(P)$ is at least 3. If its greater than 3 for some $o \in O$ then we will reduce the coefficient of $\sum_{P\in \mathtt{in}(o))} \text{shift}(P)$ in some of the inequalities involving o to make this contribution exactly 3. ◀

From Lemma 4 and Lemma 5, we can conclude that

$$-\sum_{s\in S}(f_s(u_s) - f_s(u_s^*)) + 5\sum_{o\in O}(f_o(u_o^*) - f_o(u_o))$$
$$+ 2\sum_{s\in S, o\in O} c_{so}y(s,o) + 3\sum_{o\in O}\sum_{P\in N_{U^*}(o)} \text{shift}(P) \geq 0$$

Rearranging terms, we get

$$-\sum_{i\in F} f_i(u_i) + 5\sum_{i\in F} f_i(u_i^*) + 2\sum_{s\in S, o\in O} c_{so}y(s,o) + 3\sum_{o\in O}\sum_{P\in \mathtt{in}(o)} \text{shift}(P) \geq 0$$

The third term in the above inequality can be bounded by $2(c_s(U) + c_s(U^*))$ (Lemma 1). The fourth term can be written as

$$3\sum_{o\in O}\sum_{P\in \mathtt{in}(o)} \text{shift}(P) = 3\sum_{P\in\mathcal{P}}\sum_{j\in P}(U_j^* - U_j) + 3\sum_{Q\in\mathcal{C}}\sum_{j\in Q}(U_j^* - U_j) = 3\sum_{j\in C}(U_j^* - U_j)$$

where second term in the middle equality is due to the fact that $\sum_{j\in Q:\, Q\in\mathcal{C}}(U_j^* - U_j) = 0$. Thus,

$$-c_f(U) + 5c_f(U^*) + 2(c_s(U) + c_s(U^*)) + 3(c_s(U^*) - c_s(U)) \geq 0$$

which implies the following bound on the cost of our solution

$$c_f(U) + c_s(U) \leq 5c_f(U^*) + 5c_s(U^*) \tag{6}$$

To ensure that the local search procedure has a polynomial running time we need to modify the local search procedure so that a step is performed only when the cost of the solution decreases by at least $(\epsilon/4n)c(U)$. This modification gives rise to an extra term of at most $(4\epsilon/4)c(U)$ in the above inequality. This implies that the cost of the solution U is at most $5c(U^*) + \epsilon \cdot c(U)$.

Thus we arrive at our main result:

▶ **Theorem 6.** *The local search procedure with operations* add, open and close *yields a locally optimum solution that is a* $(5+\epsilon)$*-approximation to the optimum solution.*

References

1 Eric Angel, Nguyen Kim Thang, and Damien Regnault. Improved Local Search for Universal Facility Location. In *Proceedings of 19th International Conference on Computing and Combinatorics (COCOON), Hangzhou, China*, pages 316–324, 2013. doi:10.1007/978-3-642-38768-5_29.

2 Manisha Bansal, Naveen Garg, and Neelima Gupta. A 5-Approximation for Capacitated Facility Location. In *Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, pages 133–144, 2012.

3 Mohammad Mahdian and Martin Pál. Universal Facility Location. In *Proceedings of the 11th Annual European Symposium on Algorithms (ESA), Budapest, Hungary*, volume 2832 of *Lecture Notes in Computer Science*, pages 409–421, 2003. doi:10.1007/978-3-540-39658-1_38.

4 M. Pál, É. Tardos, and T. Wexler. Facility Location with Nonuniform Hard Capacities. In *FOCS '01: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, page 329, Washington, DC, USA, 2001. IEEE Computer Society.

5 Jens Vygen. From stars to comets: Improved local search for universal facility location. *Journal of Operations Research Letters*, 35(4):427–433, 2007. doi:10.1016/j.orl.2006.08.004.

6 Jiawei Zhang, Bo Chen, and Yinyu Ye. A Multiexchange Local Search Algorithm for the Capacitated Facility Location Problem. *Math. Oper. Res.*, 30(2):389–403, 2005. doi:10.1287/moor.1040.0125.

On Fair Division for Indivisible Items

Bhaskar Ray Chaudhury
MPI for Informatics, Saarland Informatics Campus, Germany
s8bhrayc@stud.uni-saarland.de

Yun Kuen Cheung[1]
Singapore University of Technology and Design, Singapore
yunkuen_cheung@sutd.edu.sg

Jugal Garg[2]
Department of Industrial and Enterprise Systems Engineering
University of Illinois at Urbana-Champaign, USA
jugal@illinois.edu

Naveen Garg
Department of Computer Science, IIT Delhi, India
naveen@cse.iitd.ac.in

Martin Hoefer
Institut für Informatik, Goethe-Universität Frankfurt am Main, Germany
mhoefer@cs.uni-frankfurt.de

Kurt Mehlhorn
MPI for Informatics, Saarland Informatics Campus, Germany
mehlhorn@mpi-inf.mpg.de

Abstract

We consider the task of assigning indivisible goods to a set of agents in a fair manner. Our notion of fairness is Nash social welfare, i.e., the goal is to maximize the geometric mean of the utilities of the agents. Each good comes in multiple items or copies, and the utility of an agent diminishes as it receives more items of the same good. The utility of a bundle of items for an agent is the sum of the utilities of the items in the bundle. Each agent has a utility cap beyond which he does not value additional items. We give a polynomial time approximation algorithm that maximizes Nash social welfare up to a factor of $e^{1/e} \approx 1.445$. The computed allocation is Pareto-optimal and approximates envy-freeness up to one item up to a factor of $2 + \varepsilon$.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Fair Division, Indivisible Goods, Envy-Free

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.25

Acknowledgements We want to thank Hannaneh Akrami for a careful reading of the paper.

[1] Most work done while the author was at MPI for Informatics, Saarland Informatics Campus. The author would like to acknowledge NRF 2018 Fellowship NRF-NRFF2018-07.
[2] Supported by NSF CRII Award 1755619.

38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018).
Editors: Sumit Ganguly and Paritosh Pandya; Article No. 25; pp. 25:1–25:17

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

1 Introduction

We consider the task of dividing indivisible goods among a set of n agents in a fair manner. More precisely, we consider the following scenario. We have m distinct goods. Goods are available in several copies or items; there are k_j items of good j. The agents have decreasing utilities for the different items of a good, i.e., for all i and j

$$u_{i,j,1} \geq u_{i,j,2} \geq \ldots \geq u_{i,j,k_j}.$$

An allocation assigns the items to the agents. For an allocation x, x_i denotes the multi-set of items assigned to agent i, and $m(j, x_i)$ denotes the multiplicity of good j in x_i. Of course, $\sum_i m(j, x_i) = k_j$ for all j. The total utility of bundle x_i for agent i is given by

$$u_i(x_i) = \sum_j \sum_{1 \leq \ell \leq m(j,x_i)} u_{i,j,\ell}.$$

Each agent has a utility cap c_i. The capped utility of bundle x_i for agent i is defined as

$$\bar{u}_i(x_i) = \min(c_i, u_i(x_i)).$$

Our notion of fairness is *Nash social welfare* (NSW) [13], i.e., the goal is to maximize the geometric mean

$$\mathrm{NSW}(x) = \left(\prod_{1 \leq i \leq n} \bar{u}_i(x_i) \right)^{1/n}$$

of the capped utilities. All utilities and caps are assumed to be integers. We give a polynomial-time approximation algorithm with approximation guarantee $e^{1/e} + \varepsilon \approx 1.445 + \varepsilon$ for any positive ε.

The problem has a long history. For divisible goods, maximizing Nash Social Welfare (NSW) for any set of valuation functions can be expressed via an Eisenberg-Gale program [8]. Notably, for *additive valuations* ($c_i = \infty$ for each agent i and $k_j = 1$ for each good j) this is equivalent to a Fisher market with identical budgets. In this way, maximizing NSW is achieved via the well-known fairness notion of competitive equilibrium with equal incomes (CEEI) [12].

For indivisible goods, the problem is NP-complete [14] and APX-hard [10]. Several constant-factor approximation algorithms are known for the case of additive valuations. They use different approaches.

The first one was pioneered by Cole and Gkatzelis [6] and uses spending-restricted Fisher markets. Each agent comes with one unit of money to the market. Spending is restricted in the sense that no seller wants to earn more than one unit of money. If the price p of a good is higher than one in equilibrium, only a fraction $1/p$ of the good is sold. Cole and Gkatzelis showed how to compute a spending restricted equilibrium in polynomial time and how to round its allocation to an integral allocation with good NSW. In the original paper they obtained an approximation ratio of $2e^{1/e} \approx 2.889$. Subsequent work [5] improved the ratio to 2.

The second approach is via stable polynomials. Anari et al. [1] obtained an approximation factor of e.

The third approach is via integral allocations that are Pareto-optimal and envy-free up to one good. It was introduced by Barman et al. [3]. An allocation is envy-free up to one good if for any two agents i and k there is a good j such that $u_i(x_k - j) \leq u_i(x_i)$, i.e., after

removal of one good from k's bundle its utility for i is no larger than the utility of i's bundle for i. Caragiannis et al. [4] have shown that an allocation maximizing NSW is Pareto-optimal and envy-free up to one good. For a price vector p for the goods, the price $P(x_i)$ of a bundle is the sum of the prices of the goods in the bundle. An allocation is almost price-envy-free up to one good (ε-p-EF1) if $P(x_k - j) \leq (1+\varepsilon)P(x_i)$ for all agents i and k and some good j, where ε is an approximation parameter. An allocation is MBB (maximum bang per buck) if $j \in x_i$ implies $u_{ij}/p_j = \max_\ell u_{i\ell}/p_\ell$ for all j and i. Barman et al. [3] studied allocations that are Pareto-optimal, almost price-envy-free up to one good, and MBB. They showed that such allocations are almost envy-free up to one good[3] and approximate NSW up to a factor $e^{1/e} + \varepsilon \approx 1.445 + \varepsilon$. They also showed how to compute such an allocation in polynomial time.

There are also constant-factor approximation algorithms beyond additive utilities.

Garg et al. [9] studied budget-additive utilities ($k_j = 1$ for all goods j and arbitrary c_i). They showed how to generalize the Fisher market approach and obtained an $2e^{1/2e} \approx 2.404$-approximation.

Anari et al. [2] investigated multi-item concave utilities ($c_i = \infty$ for all i and k_j arbitrary). They generalized the Fisher market and the stable polynomial approach and obtained approximation factors of 2 and e^2, respectively.

We show that the price-envy-free allocation approach can handle both generalizations combined. We obtain an approximation ratio of $e^{1/e}+\varepsilon \approx 1.445+\varepsilon$. The allocation computed by our algorithm is Pareto-optimal and guarantees $u_i(x_k - j) \leq (2+\varepsilon)u_i(x_i)$ for any two agents i and k, i.e., it approximates envy-freeness up to one item up to a factor of essentially two. The approach via price-envy-freeness does not only yield better approximation ratios, it is, in our opinion, also simpler to state and simpler to analyze.

The paper is structured as follows. In Section 2 we give the algorithm and analyze its approximation ratio (Section 2.3), guarantee to individual agents (Section 2.4), and running time (Section 2.5). In Section 3 we show that the analysis is essential tight by establishing a lower bound of 1.44 on the approximation ratio of the algorithm, in Section 4 we discuss certification of the approximation ratio, and in Section 5 we show that for the multi-copy case and the capped case optimal allocations are not necessarily envy-free up to one good.

2 Algorithm and Analysis

Let us recall the setting. Items are indivisible. There are n agents and m goods. There are k_j items or copies of good j. Let $M = \sum_j k_j$ be the total number of items. The agents have decreasing utilities for the different items of a good, i.e., for all i and j

$$u_{i,j,1} \geq u_{i,j,2} \geq \ldots \geq u_{i,j,k_j}.$$

For an allocation x, x_i denotes the multi-set of items assigned to agent i, and $m(j, x_i)$ denotes the multiplicity of good j in x_i. The total utility of bundle x_i for agent i is given by

$$u_i(x_i) = \sum_j \sum_{1 \leq \ell \leq m(j,x_i)} u_{i,j,\ell}.$$

[3] Consider two bundles x_k and x_i and assume $P(x_k - j) \leq (1+\varepsilon)P(x_i)$ for some $j \in x_k$. Let $\alpha_i = \max_\ell u_{i\ell}/p_\ell$. Then $u_i(x_k - j) = \sum_{\ell \in x_k - j} u_{i\ell} \leq \alpha_i \sum_{\ell \in x_k - j} p_\ell \leq (1+\varepsilon)\alpha_i \sum_{\ell \in x_i} p_\ell = (1+\varepsilon)\sum_{\ell \in x_i} u_{i\ell}$.

Each agent has a utility cap c_i. The capped utility of bundle x_i for agent i is defined as

$$\bar{u}_i(x_i) = \min(c_i, u_i(x_i)).$$

Following [9], we assume w.l.o.g. $u_{i,j,\ell} \leq c_i$ for all i, j, and ℓ. In the algorithm, we ensure this assumption by capping every $u_{i,*,*}$ at c_i. All utilities and caps are assumed to be integers.

2.1 A Reduction to Rounded Utilities and Caps

Let $r \in (1, 3/2]$. For every non-zero utility $u_{i,j,\ell}$ let $v_{i,j,\ell}$ be the next larger power of r. For zero utilities v and u agree. Similarly, for c_i let d_i be the next larger power of r. It is well-known that it suffices to solve the rounded problem with a good approximation guarantee.

▶ **Lemma 1.** *Let x approximate the NSW for the rounded problem up to a factor of γ. Then x approximates the NSW for the original problem up to a factor γr.*

Proof. Let x^* be an optimal allocation for the original problem. Let us write $\mathrm{NSW}(x^*, u, c)$ for the Nash social welfare of the allocation x^* with respect to the utilities u and caps c. Define $\mathrm{NSW}(x, u, c)$, $\mathrm{NSW}(x^*, v, d)$, and $\mathrm{NSW}(x, v, d)$ analogously. We need to upper bound $\mathrm{NSW}(x^*, u, c)/\mathrm{NSW}(x, u, c)$. Since $u \leq v$ and $c \leq d$ componentwise, $\mathrm{NSW}(x^*, u, c) \leq \mathrm{NSW}(x^*, v, d)$. Since x approximates the NSW for the rounded problem up to a factor γ, $\mathrm{NSW}(x^*, v, d) \leq \gamma \mathrm{NSW}(x, v, d)$. Since $v \leq ru$ and $d \leq rc$ componentwise, $\mathrm{NSW}(x, v, d) \leq r\mathrm{NSW}(x, u, c)$. Thus

$$\frac{\mathrm{NSW}(x^*, u, c)}{\mathrm{NSW}(x, u, c)} \leq \frac{\gamma \mathrm{NSW}(x, v, d)}{\mathrm{NSW}(x, v, d)/r} = \gamma r. \qquad \blacktriangleleft$$

2.2 The Algorithm

Barman et al. [3] gave a highly elegant approximation algorithm for the case of a single copy per good and no utility caps. We generalize their approach. The algorithm uses an approximation parameter $\varepsilon \in (0, 1/4]$. Let $r = 1 + \varepsilon$. The nonzero utilities are assumed to be powers of r.

The algorithm maintains an integral assignment x, a price p_j for each good, and an MBB-ratio[4] α_i for each agent. Of course, $\sum_i m(j, x_i) = k_j$ for each good j. The prices, MBB-ratios, and multiplicity of goods in bundles are related through the following inequalities:

$$\frac{u_{i,j,m(j,x_i)+1}}{p_j} \leq \alpha_i \leq \frac{u_{i,j,m(j,x_i)}}{p_j}, \tag{1}$$

i.e., if $u_{i,j,\ell}/p_j > \alpha_i$, then at least ℓ copies of j are allocated to agent i and if $u_{i,j,\ell}/p_j < \alpha_i$, then less than ℓ copies of j are allocated to agent i. If no copy of good j is assigned to i, the upper bound for α_i is infinity. If all copies of good j are assigned to i, the lower bound for α_i is zero. Note that if α_i is equal to its upper bound in (1), we may take one copy of j away from i without violating the inequality as the upper bound becomes the new lower bound. Similarly, if α_i is equal to its lower bound in (1), we may assign an additional copy of j to i without violating the inequality as the lower bound becomes the new upper bound.

[4] In the case of one copy per good, $\alpha_i = u_{i,j}/p_j$ whenever (the single copy of) good j is assigned to i and $\alpha_i \geq u_{i,\ell}/p_\ell$ for all goods ℓ. Thus α_i is the maximum utility per unit of money (maximum bang per buck (MBB)) that agent i can get.

Since (1) must hold for every good j, α_i must lie in the intersection of the intervals for the different goods j, i.e.,

$$\max_j \frac{u_{i,j,m(j,x_i)+1}}{p_j} \leq \alpha_i \leq \min_j \frac{u_{i,j,m(j,x_i)}}{p_j}.$$

The value of bundle x_i for i is given by[5]

$$P_i(x_i) = \frac{u_i(x_i)}{\alpha_i} = \frac{1}{\alpha_i} \sum_j \sum_{1 \leq \ell \leq m(j,x_i)} u_{i,j,\ell}. \tag{2}$$

Definitions (1) and (2) are inspired by Anari et al [2]. We say that α_i is equal to the upper bound for the pair (i,j) if α_i is equal to its upper bound in (1) and that α_i is equal to the lower bound for the pair (i,j) if α_i is equal to its lower bound in (1).

An agent i is *capped* if $u_i(x_i) \geq c_i$ and is *uncapped* otherwise.

The algorithm starts with a greedy assignment. For each good j, it assigns each copy to the agent that values it most. The price of each good is set to the utility of the assignment of its last copy and all MBB-values are set to one. Note that this setting guarantees (1) for every pair (i,j). Also, all initial prices and MBB-values are powers of r. It is an invariant of the algorithm that prices are powers of r. Only the final price increase in the main-loop may destroy this invariant.

After initialization, the algorithm enters a loop. We need some more definitions. An agent i is a *least spending* uncapped agent if it is uncapped and $P_i(x_i) \leq P_k(x_k)$ for every other uncapped agent k. An agent i *ε-p-envies* agent k up to one item if $P_k(x_k - j) > (1+\varepsilon) \cdot P_i(x_i)$ for every good $j \in x_k$. Recall that x_k is a multi-set. In the multi-set $x_k - j$, the number of copies of good j is reduced by one, i.e., $m(j, x_k - j) = m(j, x_k) - 1$. Therefore $P_k(x_k - j) = P_k(x_k) - u_{k,j,m(j,x_k)}/\alpha_k$. An allocation is *$\varepsilon$-p-envy free up to one item (ε-p-EF1)* if for every uncapped agent i and every other agent k there is a good j such that $P_k(x_k - j) \leq (1+\varepsilon)P_i(x_i)$.

We also need the notion of the *tight graph*. It is a directed bipartite graph with the agents on one side and the goods on the other side. We have a directed edge (i,j) from agent i to good j if $\alpha_i = u_{ijm(j,x_i)+1}/p_j$, i.e., α_i is at its lower bound for the pair (i,j). We have a directed edge (j,i) from good j to agent i if $\alpha_i = u_{ijm(j,x_i)}/p_j$, i.e., α_i is at its upper bound for the pair (i,j). Note that necessarily $m(j,x_i) \geq 1$ in the latter case, since otherwise good j does not impose an upper bound for α_i.

An *improving path* starting at an agent i is a simple path $P = (i = a_0, g_1, a_1, \ldots, g_h, a_h)$ in the tight graph starting at i and ending at another agent a_h such that $P_{a_h}(x_{a_h} - g_h) > (1+\varepsilon)P_i(x_i)$ and $P_{a_\ell}(x_{a_\ell} - g_\ell) \leq (1+\varepsilon)P_i(x_i)$ for $1 \leq \ell < h$.

Let i be the least spending uncapped agent. We perform a breadth-first search in the tight graph starting at i. If the BFS discovers an improving path starting at i, we use the shortest such path to improve the allocation. Note that if i *ε-p-envies* some node that is reachable from i in the tight graph then the BFS will discover an improving path.

In the main loop, we distinguish cases according to whether BFS discovers an improving path starting at i or not.

Assume first that BFS discovers the improving path $P = (i = a_0, g_1, a_1, \ldots, g_h, a_h)$. We take g_h away from a_h and assign it to a_{h-1}. If we now have $P_{a_{h-1}}(x_{a_{h-1}} + g_h - g_{h-1}) \leq$

[5] In the case of one copy per good, $P_i(x_i) = u_i(x_i)/\alpha_i = \sum_{j \in x_i} p_j$ is the total price of the goods in the bundle. We reuse the letter P for the value of a bundle, although $P_i(x_i) = 1/\alpha_i \cdot \sum_j \sum_{1 \leq \ell \leq m(j,x_i)} u_{i,j,\ell}$ is no longer the total price of the goods in the bundle.

Algorithm 1: Approximate Nash Social Welfare for Multi Item Concave Utilities with Caps.

Input : Fair Division Problem given by utilities $u_{ij\ell}$, $i \le n$, $j \le m$, $\ell \le k_j$, utility caps c_i, and approximation parameter $\varepsilon \in (0, 1/4]$. Let $r = 1 + \varepsilon$. Nonzero u_{ij}'s and c_i's are powers of r.

Output : Price vector p and 4ε-p-EF1 integral allocation x

1 **for** i, j, ℓ **do**
2 $u_{i,j,\ell} \leftarrow \min(c_i, u_{i,j,\ell})$
3 **for** $j \in G$ **do**
4 **for** $\ell \in [k_j]$ *in increasing order* **do**
5 assign the ℓ-th copy of j to $i_0 = \operatorname{argmax}_i u_{ij,m(j,x_i)+1}$;
6 Set $p_j \leftarrow u_{i_0,j,m(j,x_{i_0})}$, where i_0 is the agent to which the k_j-th copy of j was assigned
7 **for** $i \in A$ **do**
8 $\alpha_i = 1$
9 **while** *true* **do**
10 **if** *allocation x is ε-p-EF1* **then**
11 **break** from the loop and terminate
12 Let i be a least spending uncapped agent
13 Perform a BFS in the tight graph starting at i
14 **if** *the BFS-search discovers an improving path starting in i, let $P = (i = a_0, g_1, a_1, \ldots, g_h, a_h)$ be a shortest such path* **then**
15 Set $\ell \leftarrow h$
16 **while** $\ell > 0$ *and* $P_{a_\ell}(x_{a_\ell} - g_\ell) > (1+\varepsilon)P_i(x_i)$ **do**
17 remove g_ℓ from x_{a_ℓ} and assign it to $a_{\ell-1}$; $\ell \leftarrow \ell - 1$
18 **else**
19 Let S be the set of goods and agents that can be reached from i in the tight graph
20 $\beta_1 \leftarrow \min_{k \in S;\ j \notin S} \alpha_k/(u_{k,j,m(j,x_k)+1}/p_j)$ (add a good to S)
21 $\beta_2 \leftarrow \min_{k \notin S;\ j \in S} (u_{k,j,m(j,x_k)}/p_j)/\alpha_k$ (add an agent to S)
22 $\beta_3 \leftarrow \frac{1}{r^2 P_i(x_i)} \max_{k \notin S} \min_{j \in x_k} P_k(x_k - j)$ (i is happy)
23 $\beta_4 \leftarrow r^s$, where s is the smallest integer such that $r^{s-1} \le P_h(x_h)/P_i(x_i) < r^s$ and h is the least spending uncapped agent outside S (new least spender)
24 $\beta \leftarrow \min(\beta_1, \beta_2, \max(1, \beta_3), \beta_4)$
25 multiply all prices of goods in S by β and divide all MBB-values of agents in S by β
26 **if** $\beta_3 \le \min(\beta_1, \beta_2, \beta_4)$ **then**
27 break from the while-loop

$(1+\varepsilon)P_i(x_i)$ we stop. Otherwise, we take g_{h-1} away from a_{h-1} and assign it to a_{h-2}. If we now have $P_{a_{h-2}}(x_{a_{h-2}} + g_{h-1} - g_{h-2}) \le (1+\varepsilon)P_i(x_i)$ we stop. Otherwise, ... We continue in this way until we stop or assign g_1 to a_0. In other words, let $h' < h$ be maximum such that $P_{a_{h'}}(x_{a_{h'}} + g_{h'+1} - g_{h'}) \le (1+\varepsilon)P_i(x_i)$. If h' exists, then we take a copy of g_ℓ away from a_ℓ and assign it to $a_{\ell-1}$ for $h' < \ell \le h$. If h' does not exist, we do so for $1 \le \ell \le h$. Let us call the above a sequence of swaps.

▶ **Lemma 2.** *Consider an execution of lines (15) to (17) and let h' be the final value of ℓ (this agrees with the definition of h' in the preceding paragraph). Let x' be the resulting allocation. Then $x'_\ell = x_\ell$ for $0 \le \ell < h'$, $x'_{h'} = x_{h'} + g_{h'+1}$, $x'_\ell = x_\ell + g_{\ell+1} - g_\ell$ for $h' < \ell < h$, and $x'_h = x_h - g_h$. Also,*

- $P_{a_h}(x_{a_h}) \ge P_{a_h}(x'_{a_h}) > (1+\varepsilon)P_i(x_i)$,

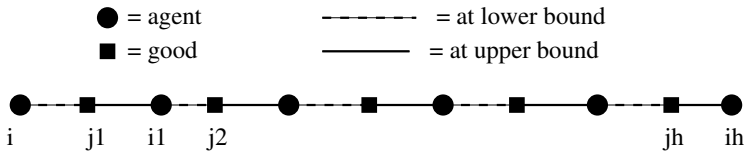


Figure 1 An improving path. Agents and goods alternate on the path and the path starts and ends with an agent. For the solid edges (j,i), α_i is at its upper bound for the pair (i,j) and for the dashed edges (i,j), α_i is at its lower bound for the pair (i,j).

- $P_{a_{h'}}(x'_{a_{h'}} - g_{h'}) = P_{a_{h'}}(x_{a_{h'}} + g_{h'+1} - g_{h'}) \leq (1+\varepsilon)P_i(x_i)$ *if* $h' \geq 1$
- $P_{a_0}(x'_{a_0} - g_1) = P_{a_0}(x_{a_0}) \leq (1+\varepsilon)P_i(x_i)$ *if* $h' = 0$.
- $P_{a_\ell}(x'_{a_\ell}) = P_{a_\ell}(x_{a_\ell} + g_{\ell+1} - g_\ell) > (1+\varepsilon)P_i(x_i)$ *and* $P_{a_\ell}(x'_{a_\ell} - g_{\ell+1}) = P_{a_\ell}(x_{a_\ell} - g_\ell) \leq (1+\varepsilon)P_i(x_i)$ *for* $h' < \ell < h$.
- $P_{a_\ell}(x'_{a_\ell} - g_\ell) = P_{a_\ell}(x_{a_\ell} - g_\ell) \leq (1+\varepsilon)P_i(x_i)$ *for* $0 \leq \ell < h'$.

Proof. Immediate from the above. ◀

If i is still the least spending uncapped agent after an execution of lines (15) to (17), we search for another improving path starting from i. We will show below that i can stay the least spending agent for at most n^2M iterations. Intuitively this holds because for any agent (factor n) and any fixed length shortest improving path (factor n), we can have at most M iterations for which the shortest improving path ends in this particular agent.

We come to the else-case, i.e., BFS does not discover an improving path starting at i. This implies that i does not ε-p-envy any agent that it can reach in the tight graph. We then increase some prices and decrease some MBB-values. Let S be the set of agents and goods that can be reached from i in the tight graph.

▶ **Lemma 3.** *If a good j belongs to S and α_k is at its upper bound for the pair (k,j), then k belongs to S. If an agent k belongs to S and α_k is at its lower bound for the pair (k,j), then j belongs to S.*

Proof. Consider any good $j \in S$. Since j belongs to S, there is an alternating path starting in i and ending in j. If the path contains k, k belongs to S. If the path does not contain k, we can extend the path by k. In either case, k belongs to S.

Consider any agent $k \in S$. Since k belongs to S, there is an alternating path starting in i and ending in k. If the path contains j, j belongs to S. If the path does not contain j, we can extend the path by j. In either case, j belongs to S. ◀

We multiply all prices of goods in S and divide all MBB-values of agents in S by a common factor $t \geq 1$. What is the effect?

- Let $u_{k,j,(j,x_k)+1}/p_j \leq \alpha_k \leq u_{k,j,m(j,x_k)}/p_j$ be the inequality (1) for the pair (k,j). The endpoints do not move if $j \notin S$ and are divided by t for $j \in S$. Similarly, α_k does not move if $k \notin S$ and are divided by t if $k \in S$. So in order to preserve the inequality, we must have: If α_k is equal to the upper endpoint and p_j moves, i.e., $j \in S$, then α_k must also move. If α_k is equal to the lower endpoint and α_k moves then p_j must also move. Both conditions are guaranteed by Lemma 3.
- If k and j are both in S, then α_k and the endpoints of the interval for (k,j) move in sync. So agents and goods reachable from i in the tight graph, stay reachable.
- If $k \notin S$, there might be a $j \in S$ such that α_k becomes equal to the right endpoint of the interval for (k,j). Then k is added to S.

- If $k \in S$, there might be a $j \notin S$ such that α_k becomes equal to the left endpoint of the interval for (k, j). Then j is added to S.
- For agents in S, $P_k(x_k)$ is multiplied by t. For agents outside S, $P_k(x_k)$ stays unchanged.

How is the common factor t chosen? There are four limiting events. Either S grows and this may happen by the addition of a good (factor β_1) or an agent (factor β_2); or $P_i(x_i)$ comes close to the largest value of $\min_{j \in x_k} P_k(x_k - j)$ for any other agent (factor β_3), or $P_i(x_i)$ becomes larger than $P_h(x_h)$ for some uncapped agent h outside S (factor β_4). Since we want prices to stay powers of r, β_4 is chosen as a power of r. The factor β_3 might be smaller than one. Since we never want to decrease prices, we take the maximum of 1 and β_3.

▶ **Lemma 4.** *Prices and MBB-values are powers of r, except maybe at termination.*

Proof. This is true initially, since prices are utility values and utility values are assumed to be powers of r and since MBB-values are equal to one. If prices and MBB-values are powers of r before a price update, β_1, β_2, and β_4 are powers of r. Thus prices and MBB-values are after the price update, except maybe when the algorithm terminates. ◀

We next show that the algorithm terminates with an allocation that is almost price-envy-free up to one item.

▶ **Lemma 5.** *Assume $\varepsilon \leq 1/4$. When the algorithm terminates, x is a 4ε-p-EF1 allocation.*

Proof. Let q be the price vector after the price increase and let h be the least spending uncapped agent after the increase; $h = i$ is possible. We first show that that $Q_i(x_i) \leq rQ_h(x_h)$. This is certainly true if $h = i$. If $h \notin S$, since the price increase is limited by β_4, we have

$$Q_i(x_i) = \beta P_i(x_i) \leq \beta_4 P_i(x_i) = r \cdot r^{s-1} \cdot P_i(x_i) \leq rP_h(x_h) = rQ_h(x_h).$$

So in either case, we have $Q_i(x_i) \leq rQ_h(x_h)$. Moreover, $Q_h(x_h) \leq Q_i(x_i)$ because h is a least spending uncapped agent after the price increase.

If the algorithm terminates, we have $\beta_3 \leq \beta_4$. Consider any agent k. Then, for $k \in S$,

$$Q_k(x_k - j_k) \leq (1+\varepsilon)Q_i(x_i) \leq (1+\varepsilon) \cdot r \cdot Q_h(x_h)$$

and, for $k \notin S$,

$$Q_k(x_k - j_k) = P_k(x_k - j_k) \leq \beta_3(1+\varepsilon)rP_i(x_i) = (1+\varepsilon)rQ_i(x_i) \leq (1+\varepsilon) \cdot r^2 \cdot Q_h(x_h).$$

Thus we are returning an allocation that is $((1+\varepsilon)r^2 - 1)$-q-EF1. Finally, note that $(1+\varepsilon)r^2 = (1+\varepsilon)^3 \leq (1+4\varepsilon)$ for $\varepsilon \leq 1/4$. ◀

▶ Remark. We want to point out the differences to the algorithm by Barman et al. Our definition of alternating path is more general than theirs since it needs to take into account that the number of items of a particular good assigned to an agent may change. For this reason, we need to maintain the MBB-ratio explicitly. In the algorithm by Barman et al. the MBB ratio of agent i is equal to the maximum utility to price ratio $\max_j u_{ij}/p_j$ and only MBB goods can be assigned to an agent. As a consequence, if a good belongs to S, the agent owning it also belongs to S. In price changes, there is no need for the quantity β_2. In the definition of β_3, we added an additional factor r^2 in the denominator. We cannot prove polynomial running time without this factor. Finally, we start the search for an improving path from the least uncapped agent and not from the least agent.

2.3 Analysis of the Approximation Factor

The analysis refines the analysis given by Barman et al. Let (x^{alg}, p, α) denote the allocation and price and MBB vector returned by the algorithm. Recall that x^{alg} is γ-p-EF1 with $\gamma = 4\varepsilon$ with respect to p and (1) holds for every i. We scale all the utilities of agent i and its utility cap by α_i, i.e., we replace $u_{i,j,\ell}$ by $u_{i,j,\ell}/\alpha_i$ and c_i by c_i/α_i and use $u_{i,j,\ell}$ and c_i also for the scaled utilities and scaled utility cap. The scaling does not change the integral allocation maximizing Nash Social Welfare. Inequality (1) becomes

$$\frac{u_{i,j,m(j,x_i^{alg})+1}}{p_j} \leq 1 \leq \frac{u_{i,j,m(j,x_i^{alg})}}{p_j}, \tag{3}$$

i.e., the items allocated to i have a utility to price ratio of one or more and the items that are not allocated to i have a ratio of one or less. Also, the value of bundle x_i for i is now equal to its utility for i and is given by

$$P_i(x_i^{alg}) = u_i(x_i^{alg}) = \sum_j \sum_{1 \leq \ell \leq m(j,x_i^{alg})} u_{i,j,\ell}. \tag{4}$$

All $u_{i,*,*}$ are at most c_i.

Let A_c and A_u be the set of capped and uncapped agents in x^{alg}, let $c = |A_c|$ and $n - c = |A_u|$ be their cardinalities. We number the uncapped agents such that $u_1(x_1^{alg}) \geq u_2(x_2^{alg}) \geq \ldots \geq u_{n-c}(x_{n-c}^{alg})$. Let $\ell = u_{n-c}(x_{n-c}^{alg})$ be the minimum utility of a bundle assigned to an uncapped agent. The capped agents are numbered $n - c + 1$ to n. Let x^* be an integral allocation maximizing Nash social welfare.

We define an auxiliary problem with $\sum_j k_j$ goods and one copy of each good. The goods are denoted by triples (i, j, ℓ), where $1 \leq \ell \leq m(j, x_i^{alg})$. The utility of good (i, j, ℓ) is uniform for all agents and is equal to $u_{i,j,\ell}$. Formally,

$$v_{*,(i,j,\ell)} = u_{i,j,\ell}, \tag{5}$$

where v is the utility function for the auxiliary problem. The cap of agent i is c_i. Since v is uniform, we can write $v(x_i)$ instead of $v_i(x_i)$. The capped utility of x_i for agent i is $\bar{v}_i(x_i) = \min(c_i, v(x_i))$. Note that v is uniform, but $\bar{v}$ is not. Let x^{optaux} be an optimal allocation for the auxiliary problem.

▶ **Lemma 6.** *We have:*

(a) $\sum_i u_i(x_i^*) \leq \sum_i u_i(x_i^{alg}) = \sum_{i,j,1 \leq \ell \leq m(j,x_i^{alg})} v_{*,(i,j,\ell)}$.

(b) $\mathrm{NSW}(x^*) = \left(\prod_i \bar{u}_i(x_i^*)\right)^{1/n} \leq \left(\prod_i \bar{v}_i(x_i^{optaux})\right)^{1/n} = \mathrm{NSW}(x^{optaux})$.

(c) *x^{alg} is Pareto-optimal.*

Proof. We can obtain x^* from x^{alg} by moving copies of goods.

Set $x \leftarrow x^{alg}$. Consider any good j. As long as the multiplicities of j in the bundles of x and x^* are not the same, identify two agents i and k, where x_i contains more copies of j than x_i^* and x_k contains fewer copies of j than x_k^*, and move a copy of j from i to k. Each copy taken away has a utility of at least p_j, each copy assigned additionally has a utility of at most p_j. Thus the total utility cannot go up by reassigning. This proves (a).

For part (b), we interpret x^{alg} as an allocation for the auxiliary problem; goods (i, j, ℓ) with $1 \leq \ell \leq m(j, x_i^{alg})$ are allocated to agent i. We then move goods exactly as in (a). We obtain an allocation $\hat{x}$ for the auxiliary problem with $u_i(x_i^*) \leq v(\hat{x}_i)$ for all i.

For part (c), assume that x^{alg} is not Pareto-optimal. Then there is an integral allocation y with $u_i(y_i) \geq u_i(x_i^{alg})$ for all i and at least one strict inequality. These inequalities are not affected by our scaling of the utilties. However, the reasoning of part (a) applied to y and x^{alg} shows $\sum_i u_i(y_i) \leq \sum_i u_i(x_i^{alg})$ for all i for the scaled utilities. ◀

We stress that Lemma 6 refers to the scaled utilities. For the scaled utilities x^{alg} maximizes social welfare. It does not do so for the unscaled utilities.

For any agent i, let $b_i \in x_i^{alg}$ be such that $u_i(x_i^{alg} - b_i) \leq (1+\gamma)\ell$. Note that $u_i(x_i^{alg} - b_i) = u_i(x_i^{alg}) - u_{i,b_i,m(b_i,x_i^{alg})}$. Let $B = \{\, (i, b_i, m(b_i, x_i^{alg})) \,;\, 1 \leq i \leq n \,\}$ be the goods in the auxiliary problem corresponding to the b_i's. We now consider allocations for the auxiliary problem that are allowed to be partially fractional. We require that the goods in B are allocated integrally and allow all other goods to be assigned fractionally. For convenience of notation, let $g_i = (i, b_i, m(b_i, x_i^{alg}))$. The following lemma is crucial for the analysis.

▶ **Lemma 7.** *There is an optimal allocation for the relaxed auxiliary problem in which good g_i is allocated to agent i.*

Proof. Assume otherwise. Among the allocations maximizing Nash social welfare for the relaxed auxiliary problem, let x^{optrel} be the one that maximizes the number of agents i that are allocated their own good g_i.

Assume first that there is an agent i to which no good in B is allocated. Then g_i is allocated to some agent k different from i. Since $b_i \in x_i^{alg}$, $v(g_i) = u_{i,b_i,m(b_i,x_i^{alg})} \leq c_i$. The inequality holds since utilities $u_{i,*,*}$ are capped at c_i during initialization. We move g_i from k to i and $\min(v(g_i), v(x_i^{optrel}))$ value from i to k. This is possible since only divisible goods are allocated to i. If we move $v(g_i)$ from i to k, the NSW does not change. If $v(g_i) > v(x_i^{optrel})$ and hence $c_i \geq v(g_i) > v(x_i^{optrel})$, the product $\bar{v}_i(x_i) \cdot \bar{v}_k(x_k)$ changes from

$$\min(c_i, v(x_i^{optrel})) \cdot \min(c_k, v(x_k^{optrel} - g_i + g_i)) = \\ \min(c_k v(x_i^{optrel}), v(x_k^{optrel} - g_i) v(x_i^{optrel}) + v(g_i) v(x_i^{optrel}))$$

to

$$\min(c_i, v(g_i)) \cdot \min(c_k, v(x_k^{optrel} - g_i + x_i^{optrel})) = \\ \min(c_k v(g_i), v(x_k^{optrel} - v(g_i)) v(g_i) + v(x_i^{optrel}) v(g_i)).$$

The arguments of the min in the lower line are componentwise larger than those of the min in the upper line. We have now modified x^{optrel} such that the NSW did not decrease and the number of agents owning their own good increased. The above applies as long as there is an agent owning no good in B.

So assume every agent i owns a good in B, but not necessarily g_i. Let i be such that $v(g_i)$ is largest among all goods g_i that are not allocated to their i. Then g_i is allocated to some agent k different from i. The value of the good g_ℓ allocated to i is at most $v(g_i)$ since $\ell \neq i$ and by the choice of i. We move g_i from k to i and $\min(v(g_i), v(x_i^{optrel}))$ value from i to k. This is possible since $v(g_\ell) \leq v(g_i)$ and all other goods assigned to i are divisible. We have now modified x^{optrel} such that the NSW did not decrease and the number of agents owning their own good increased. We continue in this way until g_i is allocated to i for every i. ◀

Let x^{optrel} be an optimal allocation for the relaxed auxiliary problem in which good g_i is contained in the bundle x_i^{optrel} for every i. Let α be such that

$$\alpha\ell = \min\{\, v(x_i^{optrel}) \,;\, v(x_i^{optrel}) < c_i \,\}$$

is the minimum value of any agent that is uncapped in x^{optrel}. Let $\alpha = \infty$, if every agent is capped in x^{optrel}. Let A_c^{optrel} and A_u^{optrel} be the set of capped and uncapped agents in x^{optrel}. Let h be such that $u_h(x_h^{alg}) > \alpha\ell \geq u_{h+1}(x_{h+1}^{alg})$.

▶ **Lemma 8.** *For $i \leq h$, $v(x_i^{optrel}) \leq u_i(x_i^{alg})$. For all i, $u_i(x_i^{alg}) \leq v(x_i^{optrel}) + (1+\gamma)\ell$. For $i \in A_u \cap A_c^{optrel}$, $c_i \leq \alpha\ell$ and $i \notin [h]$.*

Proof. Consider any $i \leq h$. $v(x_i^{optrel}) \leq u_i(x_i^{alg})$ is obvious, if $v(x_i^{optrel}) \leq \alpha\ell$. If $v(x_i^{optrel}) > \alpha\ell$, then $\alpha < \infty$ and hence A_u^{optrel} is non-empty. We claim that $x_i^{optrel} = \{g_i\}$, i.e., x_i^{optrel} is a singleton consisting only of g_i. Assume otherwise, then also some divisible goods are assigned to i. We can move some of them to an agent that is uncapped in x^{optrel} and has value $\alpha\ell$. This increases the NSW, a contradiction.

For the upper bound, we observe that $g_i \in x_i^{optrel}$ and $u_i(x_i^{alg} - b_i) \leq (1+\gamma)\ell$.

Consider next any $i \in A_u \cap A_c^{optrel}$. Assume $c_i > \alpha\ell$. If x^{optrel} assigns divisible goods to i, we can move some of them to an agent that is uncapped in x^{optrel} and has value $\alpha\ell$. This increases the NSW. Thus x_i^{optrel} consists only of g_i. But then $v(g_i) \leq u_i(x_i^{alg}) < c_i$ and i does not belong to A_c^{optrel}. This shows $c_i \leq \alpha\ell$. Then also $i \notin [h]$ because otherwise $c_i < u_i(x_i^{alg})$ and hence i would be capped in x^{alg}. ◀

▶ **Lemma 9.**

$$\mathrm{NSW}(x^*) \leq \mathrm{NSW}(x^{optrel}) \leq \left((\alpha\ell)^{n-c-h-|A_u \cap A_c^{optrel}|} \cdot \prod_{i \in A_c \cup (A_u \cap A_c^{optrel})} c_i \cdot \prod_{1 \leq i \leq h} u_i(x_i^{alg}) \right)^{\frac{1}{n}}.$$

Moreover, $c_i \leq \alpha\ell$ for any $i \in A_u \cap A_c^{optrel}$.

Proof. If $v(x_i^{optrel}) \neq \alpha\ell$ then either $i \in A_c$ or $i \in A_u \cap A_c^{optrel}$ or $i \in A_u \setminus A_c^{optrel}$. In the first case, $v(x_i^{optrel}) \leq c_i$. In the second case, $v(x_i^{optrel}) = c_i \leq \alpha\ell$ and $i \notin [h]$ by Lemma 8. In the third case, $v(x_i^{optrel}) \leq u_i(x_i^{alg})$ for $i \leq h$. So assume $i > h$. Then $v(g_i) \leq u_i(x_i^{alg}) \leq \alpha\ell$ and hence all value in $v(x_i^{optrel})$ above $\alpha\ell$ would be by fractional goods. They could be reassigned for an increase in NSW. We conclude that for the agents $i \in A_u \setminus A_c^{optrel}$ with $i > h$, we have $v(x_i^{optrel}) = \alpha\ell$. ◀

We next bound $\mathrm{NSW}(x^{alg})$ from below. We consider assignments x for the auxiliary problem that agree with x^{alg} for the agents in $A_c \cup [h]$ and reassign the value $\sum_{i \in A_u - [h]} u_i(x_i^{alg})$ fractionally. Note that for any $i \in A_u - [h]$, $\ell \leq u_i(x_i^{alg}) \leq \min(c_i, \alpha\ell)$. The former inequality follows from $i \in A_u$ and the latter inequality follows from the definition of h and $i \in A_u$. We reallocate value so as to move $u_i(x_i)$ towards the bounds ℓ and $\min(c_i, \alpha\ell)$. As long as there are two agents whose value is not at one of their bounds, we shift value from the smaller to the larger. This decreases NSW. We end when all but one agent have an extreme allocation, either ℓ or $\min(c_i, \alpha\ell)$. One agent ends up with an allocation $\beta\ell$ with $\beta \in [1, \alpha]$.

Let us introduce some more notation. Write $A_u \cap A_c^{optrel}$ as $S \cup T$, where the agents $i \in T$ end up at c_i and the agents in S end up at ℓ. Also let s and t be the number of agents in $A_u \setminus A_c^{optrel}$ that end up at ℓ and $\alpha\ell$ respectively. Then

$$\mathrm{NSW}(x^{alg}) \geq \left(\prod_{i \in A_c} c_i \cdot \prod_{1 \leq i \leq h} u_i(x_i^{alg}) \cdot \ell^s \cdot (\alpha\ell)^t \cdot (\beta\ell) \cdot \prod_{i \in T} c_i \cdot \ell^{|S|} \right)^{1/n}.$$

Note that $n - c - h = s + t + 1 + |S| + |T|$. Therefore

$$\frac{\mathrm{NSW}(x^*)}{\mathrm{NSW}(x^{alg})} \leq \left(\alpha^s \cdot \frac{\alpha}{\beta} \cdot \prod_{i \in S} \frac{c_i}{\ell}\right)^{1/n} \leq \left(\left(\frac{s\alpha + \frac{\alpha}{\beta} + \sum_{i \in S} \frac{c_i}{\ell}}{s + 1 + |S|}\right)^{s+1+|S|}\right)^{1/n},$$

where we used the inequality between geometric mean and arithmetic mean for the second inequality.

The total mass allocated by x^{optrel} to the agents in $A_u - [h]$ is $(s+t+1)\alpha\ell + \sum_{i \in S \cup T} c_i$. The allocation x^{alg} wastes up to $(1+\gamma)\ell$ for each $i \in A_c \cup [h]$ and uses $s\ell + t\alpha\ell + \beta\ell + \sum_{i \in T} c_i + |S|\ell$ on the agents in $A_u - [h]$. Therefore

$$(s + t + 1)\alpha\ell + \sum_{i \in S \cup T} c_i \leq (|A_c| + h)(1 + \gamma)\ell + s\ell + t\alpha\ell + \beta\ell + \sum_{i \in T} c_i + |S|\ell$$

and hence after rearranging, dividing by ℓ and adding α/β on both sides

$$\begin{aligned} s\alpha + \frac{\alpha}{\beta} + \sum_{i \in S} \frac{c_i}{\ell} &\leq (1 + \gamma)(|A_c| + h) + s + |S| + \frac{\alpha}{\beta} + \beta - \alpha \\ &\leq (1 + \gamma)(|A_c| + h) + s + |S| + 1 \leq (1 + \gamma)n. \end{aligned}$$

Note that $\beta + \alpha/\beta - \alpha \leq 1$ for $\beta \in [1, \alpha]$, since the expression is one at $\beta = 1$ and $\beta = \alpha$ and it second derivative as function of β is positive. Thus

$$\begin{aligned} \frac{\mathrm{NSW}(x^{optrel})}{\mathrm{NSW}(x^{alg})} &\leq \left(\left(\frac{(1 + \gamma)(|A_c| + h) + s + |S| + 1}{s + 1 + |S|}\right)^{s+1+|S|}\right)^{1/n} \\ &\leq \left(\frac{(1 + \gamma)n}{s + 1 + |S|}\right)^{(s+1+|S|)/n} \leq e^{e^{-1/(1+\gamma)}}, \end{aligned}$$

since the maximum of $((1 + \gamma)\delta)^{1/\delta}$ is attained for $\delta = \frac{1}{(1+\gamma)} e^{1/(1+\gamma)}$ and is equal to $\exp(\exp(-1/(1 + \gamma)))$. The following table contains concrete values for small non-negative values of γ.

$1 + \gamma$	1.00	1.01	1.02	1.03	1.04
$\exp(\exp(-1/(1 + \gamma)))$	1.44467	1.44997	1.45523	1.46046	1.46566

▶ Remark. The paper [5] introduces a mathematical program for maximizing NSW in the case of additive valuations. The program has an integrality gap of $e^{1/e}$. We believe that the fact that the same expression $e^{1/e}$ appears at two places is coincidence and does not point to some hidden relationship. In particular, Barman et al.'s algorithm computes the optimal allocation for the instances which [5] uses to demonstrate the integrality gap.

2.4 Guarantees for Individual Agents

The allocation computed by our algorithm is Pareto-optimal and maximizes NSW up to a factor 1.45. It also gives any uncapped agent i the guarantee $\min_{j \in x_k} P_k(x_k - j) \leq (1+\varepsilon)P_i(x_i)$ for every other agent k. This guarantee is not meaningful for agent i. We now show that it implies $\min_{j \in x_k} u_i(x_k - j) \leq (2 + \varepsilon)u_i(x_i)$, i.e., the utility for i of k's bundle minus one item is essentially bounded by twice the utility of i's bundle for i. The proof shows that the additional utility for i of the items that k has in excess of i up to one item is bounded by $(1 + \varepsilon)u_i(x_i)$. In the case of one copy per good, x_k and x_i are disjoint and hence any item in x_k is in excess of i's possession of the same good.

▶ **Theorem 10.** *The allocation computed by the algorithm satisfies* $\min_{j\in x_k} u_i(x_k - j) \le (2+\varepsilon)u_i(x_i)$ *for any agent* k *and any uncapped agent* i*.*

Proof. Let g be such that $u_k(x_k - g) = \min_{j\in x_k} u_k(x_k - j)$. Then

$$\begin{aligned}
u_i(x_k - g) &\le u_i(x_i \cup x_k - g) && \text{more never harms}\\
&= u_i(x_i) + \sum_j \sum_{\ell=m(j,x_i)+1}^{m(j,x_k\cup x_i-g)} u_{i,j,\ell}\\
&\le u_i(x_i) + \sum_j \sum_{\ell=m(j,x_i)+1}^{m(j,x_k\cup x_i-g)} \alpha_i p_j && \text{since } u_{i,j,\ell}/p_j \le \alpha_i \text{ for } \ell > m(j,x_i)\\
&\le u_i(x_i) + \sum_j \sum_{\ell=1}^{m(j,x_k-g)} \alpha_i p_j\\
&\le u_i(x_i) + \sum_j \sum_{\ell=1}^{m(j,x_k-g)} \alpha_i \frac{u_{k,j,\ell}}{\alpha_k} && \text{since } u_{k,j,\ell}/p_j \ge \alpha_k \text{ for } k \le m(j,x_k)\\
&\le u_i(x_i) + \alpha_i p_k(x_k - g) && \text{definition of } P_k(x_k - g)\\
&\le u_i(x) + \alpha_i(1+\varepsilon)P_i(x_i) && \text{since } P_k(x_g - g) \le P_i(x_i)\\
&= (2+\varepsilon)u_i(x_i) && \text{since } u_i(x_i) = \alpha_i P_i(x_i). \quad ◀
\end{aligned}$$

2.5 Polynomial Running Time

The analysis follows Barman et al. with one difference. Lemma 12 is new. For its proof, we need the revised definition of β_3.

▶ **Lemma 11.** *The price of the least spending uncapped agent is non-decreasing.*

Proof. This is clear for price increases. Consider a sequence of swaps along an improving path $P = (i = a_0, g_1, a_1, \ldots, g_h, a_h)$, where the agent a_h loses a good, the agents a_ℓ, $h' < \ell < h$, lose and gain a good, and the agent $a_{h'}$ gains a good. By Lemma 1, all agents a_ℓ with $h' < \ell \le h$ have a price of at least $(1+\varepsilon)P_i(x_i)$ after the swap. Also the price of agent $a_{h'}$ does not decrease. ◀

▶ **Lemma 12.** *For any agent* k*, let* j_k *be a highest price item in* x_k*. Then* $\max_k P_k(x_k - j_k)$ *does not increase in the course of the algorithm as long as this value is above* $(1+\varepsilon)\min_{uncapped\ i} P_i(x_i)$*. Once* $\max_k P_k(x_k - j_k) \le (1+\varepsilon)\min_{uncapped\ i} P_i(x_i)$*, the algorithm terminates.*

Proof. We first consider price increases and then a sequence of swaps.

Consider any price increase which is not the last. Then $\beta_4 \le \beta_3$. Let h be the least uncapped spender after the price increase and q be the price vector after the increase. Then $Q_h(x_h) \le Q_i(x_i) \le rQ_h(x_h)$. For $k \in S$, we have $min_j Q_k(x_k - j) \le (1+\varepsilon)Q_i(x_i) \le (1+\varepsilon)rQ_h(x_h)$, i.e., agents in S can become violators but we can bound how bad they can become. For the agent $k \notin S$ defining β_3, we have

$$\min_j P_k(x_\ell - j) = \beta_3(1+\varepsilon)rP_i(x_i) \ge (1+\varepsilon)rQ_i(x_i) \ge (1+\varepsilon)rQ_h(x_h)$$

and hence the worst violator stays outside S. We used the equality $r = 1+\varepsilon$ and the inequality $Q_i(x_i) = \beta P_i(x_i) \le \beta_3 P_i(x_i)$ in this derivation.

Consider next a sequence of swaps. We have an improving path from i to k, say $P = (i = a_0, g_1, a_1, \ldots, g_h, a_h = k)$. Let x' be the allocation after the sequence of swaps. Then $\min_j P_k(x'_k - j) \leq \min_j P_k(x_k - j)$ since k looses a good and $\min_j P_\ell(x'_\ell - j) \leq (1+\varepsilon)P_i(x_i)$ for all $\ell \in [0, h-1]$ by Lemma 2. ◀

▶ **Lemma 13.** *The number of subsequent iterations with no change of the least spending agent and no price increase is bounded by n^2M.*

Proof. Let i be the least spending agent. We count for any other agent k, how often the improving path can end in k. For each fixed length of the improving path, this can happen at most M times (for details see [3]). The argument is similar to the argument used in the strongly polynomial algorithms for weighted matchings [7]. ◀

▶ **Lemma 14.** *If the least spending uncapped agent changes after a price increase, the value of the old least spending uncapped agent increases by a factor of at least r.*

Proof. The least uncapped spender changes if $\beta = \beta_4$ and β_4 is at least r. So $P_i(x_i)$ increases by at least r. ◀

▶ **Theorem 15.** *The number of iterations is bounded by $n^3M^2 \log_r MU$.*

Proof. Divide the execution into maximum subsequences with the same least spender. Consider any fixed agent i and the subsequences where i is the least spender. At the end of each subsequence, i receives an additional item, or we have a price increase. In the latter case, $P_i(x_i)$ is multiplied by at least r. Consider the subsequences between price increases. At the end of a subsequence i receives an additional item. It may or may not keep this item until the beginning of the next subsequence. If there are more than M subsequences with i being the least spender, there must be two subsequences such that i looses an item between these subsequences. According to Lemma 2, the value of i after the swap is at least r times the minimum price of any bundle and hence at least r times the price of bundle i when i was least spender for the last time. Thus $P_i(x_i)$ increases by a factor of at least r.

We have now shown: After at most $M \cdot n^2M$ iterations with i being the least spender, $P_i(x_i)$ is multiplied by a factor r. Thus there can be at most $n^2M^2 \log_r MU$ such iterations. Multiplication by n yields the bound on the number of iterations. ◀

3 A Lower Bound on the Approximation Ratio of the Algorithm

We show that the performance of the algorithm is no better than 1.44. Let k, s and K be positive integers with $K \geq k$ which we fix later. Consider the following instance. We have $h = s(k-1)$ goods of value K and $n = h + s$ goods of value 1. There is one copy of each good. The number of agents is n and all agents value the goods in the same way.

The algorithm may construct the following allocation. There are h agents that are allocated a good of value 1 and a good of value K and there are s agents that are allocated a good of value 1. This allocation can be constructed during initialization. The prices are set to the values and the algorithm terminates.

The optimal allocation will allocate a good of value K to h players and spread the $h + s = sk$ goods of value 1 across the remaining s agents. So s agents get value k each. Thus

$$\frac{\mathrm{NSW}(OPT)}{\mathrm{NSW}(ALG)} = \left(\frac{K^h k^s}{(K+1)^h}\right)^{1/(h+s)} = \left(\left(\frac{K}{K+1}\right)^{(k-1)s} k^s\right)^{1/ks} = \left(\frac{K}{K+1}\right)^{(k-1)/k} k^{1/k}.$$

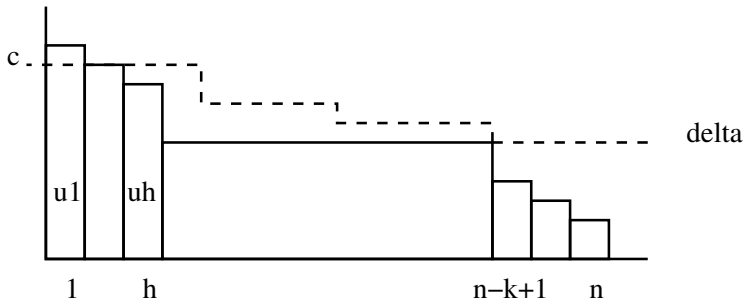


Figure 2 The allocation constructed in the proof of Theorem 16. The dashed line above agents 1 to $n-k$ indicates the utility caps. The solid rectangles visualize the values of the bundles.

The term involving K is always less than one. It approaches 1 as K goes to infinity. The second term $k^{1/k}$ has it maximal value at $k = e$. However, we are restricted to integral values. We have $2^{1/2} = 1.41$ and $3^{1/3} = 1.442$. For $k = 3$, $(K/(K+1))^{2/3} = \exp(\frac{2}{3}\ln(1-1/(K+1))) \approx \exp(-\frac{2}{3(K+1)}) \approx 1 - \frac{2}{3(K+1)}$. So for $K = 666$, the factor is less than $1 - 1/1000$ and therefore $\mathrm{NSW}(OPT)/\mathrm{NSW}(ALG) \geq 1.440$.

4 Certification of the Approximation Ratio

How can a user of an implementation of the algorithm be convinced that the solution returned has a NSW no more than 1.445 times the optimum? She may read this paper and convince herself that the program indeed implements the algorithm described in this article. This is unsatisfactory [11]. In this section, we describe an alternative certificate.

The algorithm returns an allocation x^{alg}, prices p_j for the goods, and MBB-ratios α_i for the agents. After scaling all utilities and the utility gap of agent i by α_i, we have (3). The user needs to understand that this scaling has no effect on the optimal allocation. As in Section 2.3, we introduce the auxiliary problem with $M = \sum_j k_j$ goods and one copy of each good. The goods have uniform utilities. The user needs to understand that the NSW of the auxiliary problem is an upper bound (Lemma 6). We are left with the task of convincing the user of an upper bound on the NSW of the auxiliary problem.

▶ **Theorem 16.** *Let $c_1 \geq c_2 \geq \ldots \geq c_n$ be the utility caps of the agents, let $u_1 \geq u_2 \geq \ldots \geq u_M$ be the utilities of the M goods of the auxiliary problem, and let x^{optaux} be an optimal allocation for the auxiliary problem. Then*

$$\mathrm{NSW}(x^{optaux}) \leq \left(\prod_{1 \leq i \leq h} \min(c_i, u_i) \cdot \delta^{n-h-k} \cdot \prod_{n-k+1 \leq i \leq n} c_i \right)^{1/n},$$

where $\delta = \left(\sum_{h+1 \leq j \leq M} u_j - \sum_{n-k+1 \leq i \leq n} c_i\right)/(n-h-k)$ and h and k are such that $h < n-k$ and $c_{n-k+1} \leq \delta < c_{n-k}$ and $\delta < u_h$. The right hand side is illustrated in Figure 2.

Proof. We insist that the goods 1 to h are allocated integrally and allow the remaining goods to be allocated fractionally. Clearly, we cannot allocate more than c_i to any agent, in particular, not to agents $n-k+1$ to n and to agents 1 to h. The optimal way to distribute value $\sum_{h+1 \leq j \leq M} u_j$ to agents $h+1$ to n is clearly to allocate δ each to agents $h+1$ to $n-k$ which all have a cap of more than δ and to the assign their cap to agents $n-k+1$ to n. The items u_1 to u_h of value more than δ are best assigned to the agents with the largest utility

caps. Assume that two such items, say u_ℓ and u_k, are allocated to the same agent. Then one of the first h agents is allocated no such item; let v be the value allocated to this agent. Moving u_k to this agent and value $\min(u_k, v)$ from this agent in return, does not decrease the NSW. Also, if any fractional items are assigned in addition to the first h agents, we move them to agents $h+1$ to $n-k$ and increase the NSW. This establishes the upper bound. ◀

The upper bound can be computed in time $O(n^2 + M)$. We conjecture that it can be computed in linear time $O(n + M)$. We also conjecture that the bound is never worse than the bound used in the analysis of the algorithm. It can be better as the following example shows. We have two uncapped agents and three goods of value $u_1 = 3$, $u_2 = 1$ and $u_3 = 1$, respectively. The algorithm may assign the first two goods to the first agent and the third good to the second agent. The set B in the analysis of the algorithm consists of the first good and the last good. Then $\ell = 1$. The optimal allocation allocates 3 to the first agent and 2 to the second agent. Thus $\alpha\ell = 2$. The analysis uses the upper bound $\sqrt{4 \cdot 2}$ for the NSW of the optimal allocation. The theorem above gives the upper bound $\sqrt{3 \cdot 2}$; note that $h = 1$, $k = 0$, and $\delta = 2$.

5 Envy-Freeness up to one Copy

For the case of additive valuations and one copy of each good, the optimal allocation is envy-free up to one good as shown in [4]. Also the allocation constructed by the algorithm by Barman et al. [3] is envy-free up to one good. In this section, we show that these properties hold neither for the multi-copy case nor for the capped case.

We first give an example for the multi-copy uncapped case. There are two agents and two goods. Good 1 has 5 copies, and good 2 has 2 copies. For the first agent, the utility vector for good 1 is $(1, 1, 0, 0, 0)$ and for good 2 is $(\delta, 0)$, where $\delta = 1/4$. For the second agent, the utility vector for good 1 is $(1, 1, 1, 0, 0)$ and for good 2 is $(1, 1)$. Then at the optimal NSW allocation, the first agent is allocated two copies of good 1 and none of good 2, while the second agent is allocated three copies of good 1 and two copies of good 2. Clearly, the first agent envies the second agent even after removing one copy (of either good) from the allocation of the second agent. However, $u_1(x_2) = 2 + \delta$.

What does the algorithm do? The initial assignment constructs the optimal assignment and sets $p_1 = p_2 = \alpha_1 = \alpha_2 = 1$. Agent 1 is the least spending uncapped agent. The constraints on α_1 are $[0, 1]$ by the first good and $[\delta, 1]$ by the second good. The tight graph consists only of agent 1. We enter the else-case of the main loop with $S = 1$. Then $\beta_1 = 1/\delta$, $\beta_2 = \infty$, $\beta_3 = 4/(2r^2) = 2/r^2$ and $\beta_4 = r^{1+\lfloor \log_r 5/2 \rfloor} \geq \beta_3$. Thus $\beta = \beta_3$. We decrease $\alpha_1$to $r^2/2 \approx 1/2$ and terminate. The optimal allocation is now ε-p-envy free up to one copy.

For the linear capped case, again we have two agents, and this time we have four goods with one copy each. The utility vectors of both agents are $(1, 1, 1, 1)$, but the first agent is capped at $1 + \delta$, while the second agent is uncapped. Again $\delta = 1/4$. Then the optimal NSW allocation allocates one good to the first agent and three goods to the second agent. Clearly, the first agent envies the second agent, even after removing one good from the allocation of the second agent.

What does the algorithm do? It may construct the optimal assignment during initialization; the prices of all four goods and both α-values are set to one. Agent 1 is the least spending uncapped agent. The tight graph consists of the edges from agent 1 to the goods owned by agent 2 and from these goods to agent 1. An improving path exists and one of these goods is reassigned to agent 1. The algorithm terminates with an allocation in which both agents own two goods.

References

1 Nima Anari, Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. Nash Social Welfare, Matrix Permanent, and Stable Polynomials. In *ITCS*, pages 36:1–36:12, 2017. doi:10.4230/LIPIcs.ITCS.2017.36.

2 Nima Anari, Tung Mai, Shayan Oveis Gharan, and Vijay V. Vazirani. Nash Social Welfare for Indivisible Items under Separable, Piecewise-Linear Concave Utilities. In *SODA*, pages 2274–2290, 2018. doi:10.1137/1.9781611975031.147.

3 Siddharth Barman, Sanath Kumar Krishna Murthy, and Rohit Vaish. Finding Fair and Efficient Allocations. *CoRR*, abs/1707.04731, 2017. to appear in EC 2018. arXiv:1707.04731.

4 Ioannis Caragiannis, David Kurokawa, Hervé Moulin, Ariel D. Procaccia, Nisarg Shah, and Junxing Wang. The Unreasonable Fairness of Maximum Nash Welfare. In *EC*, pages 305–322, 2016. doi:10.1145/2940716.2940726.

5 Richard Cole, Nikhil R. Devanur, Vasilis Gkatzelis, Kamal Jain, Tung Mai, Vijay V. Vazirani, and Sadra Yazdanbod. Convex Program Duality, Fisher Markets, and Nash Social Welfare. In *EC*, pages 459–460, 2017. doi:10.1145/3033274.3085109.

6 Richard Cole and Vasilis Gkatzelis. Approximating the Nash Social Welfare with Indivisible Items. In *STOC*, pages 371–380, 2015. doi:10.1145/2746539.2746589.

7 J. Edmonds and R.M. Karp. Theoretical Improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19:248–264, 1972.

8 E. Eisenberg and D. Gale. Consensus of subjective probabilities: The Pari-Mutuel method. *The Annals Math. Statist.*, 30:165–168, 1959.

9 Jugal Garg, Martin Hoefer, and Kurt Mehlhorn. Approximating the Nash Social Welfare with Budget-Additive Valuations. In *SODA 2018*, pages 2326–2340, 2018. doi:10.1137/1.9781611975031.150.

10 Euiwoong Lee. APX-hardness of maximizing Nash social welfare with indivisible items. *Inf. Process. Lett.*, 122:17–20, 2017. doi:10.1016/j.ipl.2017.01.012.

11 R.M. McConnell, K. Mehlhorn, S. Näher, and P. Schweitzer. Certifying algorithms. *Computer Science Review*, 5(2):119–161, 2011. doi:10.1016/j.cosrev.2010.09.009.

12 Hervé Moulin. *Fair division and collective welfare*. MIT Press, 2003.

13 J. Nash. The Bargaining Problem. *Econometrica*, 18:155–162, 1950.

14 Nhan-Tam Nguyen, Trung Thanh Nguyen, Magnus Roos, and Jörg Rothe. Computational complexity and approximability of social welfare optimization in multiagent resource allocation. *Autonomous Agents and Multi-Agent Systems*, 28(2):256–289, 2014. doi:10.1007/s10458-013-9224-2.

Combinatorial Algorithms for General Linear Arrow-Debreu Markets

Bhaskar Ray Chaudhury
MPI for Informatics, Saarland Informatics Campus, Graduate School of Computer Science
Saarbrücken, Germany
braycha@mpi-inf.mpg.de

Kurt Mehlhorn
MPI for Informatics, Saarland Informatics Campus, Saarbrücken, Germany
mehlhorn@mpi-inf.mpg.de

Abstract

We present a combinatorial algorithm for determining the market clearing prices of a general linear Arrow-Debreu market, where every agent can own multiple goods. The existing combinatorial algorithms for linear Arrow-Debreu markets consider the case where each agent can own all of one good only. We present an $\tilde{\mathcal{O}}((n+m)^7 \log^3(UW))$ algorithm where n, m, U and W refer to the number of agents, the number of goods, the maximal integral utility and the maximum quantity of any good in the market respectively. The algorithm refines the iterative algorithm of Duan, Garg and Mehlhorn using several new ideas. We also identify the hard instances for existing combinatorial algorithms for linear Arrow-Debreu markets. In particular we find instances where the ratio of the maximum to the minimum equilibrium price of a good is $U^{\Omega(n)}$ and the number of iterations required by the existing iterative combinatorial algorithms of Duan, and Mehlhorn and Duan, Garg, and Mehlhorn are high. Our instances also separate the two algorithms.

2012 ACM Subject Classification Theory of computation → Mathematical optimization

Keywords and phrases Linear Exchange Markets, Equilibrium, Combinatorial Algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.26

Related Version A full version of the paper is available at [2], `https://arxiv.org/abs/1810.01237`.

1 Introduction

In a linear Arrow-Debreu market, there is a set B of n agents and a set G of m divisible goods. We will refer to an individual agent by b_i for $i \in [n]$ and to an individual good by g_j for $j \in [m]$. Each agent comes with a basket of goods to the market, more precisely, agent b_i owns $w_{ij} \geq 0$ units of good g_j. The total supply of g_j is then $\sum_i w_{ij}$ units. Moreover, the agents have utilities over the goods and $u_{ij} \geq 0$ is the utility derived by agent b_i from one unit of good g_j.

The goal is to find a positive price vector $p \in \mathbb{R}^m_{\geq 0}$ and a non-zero flow $f \in \mathbb{R}^{n\times m}_{\geq 0}$ such that:

a) For all $j \in [m]$: $\sum_i f_{ij} = \sum_i w_{ij} p_j$ (all goods are completely sold)
b) For all $i \in [n]$: $\sum_j w_{ij} p_j = \sum_j f_{ij}$ (agents spend all their income)
c) $f_{ij} > 0$ implies $u_{ij}/p_j = \max_{\ell\in[m]} u_{i\ell}/p_\ell$ (only bang-for-buck spending)

Such a price vector is called a vector of equilibrium prices. In a), the left hand side is the total flow (of money) into g_j and the right hand side is the total value of all units of good g_j. In b), the left hand side is the income of agent b_i under prices p and the right hand side is his spending. In c), $\max_{\ell\in[m]} u_{i\ell}/p_\ell$ is the maximum ratio of utility to price (bang-for-buck) that agent b_i can achieve. Agents spend only money on goods that give them the maximum bang-for-buck. For agent b_i and good g_j, we use $x_{ij} = f_{ij}/p_j$ for the amount of g_j allocated to agent b_i. A price vector p and an flow f as above is called a *market equilibrium*.

We make the standard assumption that every agent likes at least one good, i.e., for all i, $\max_j u_{ij} > 0$, and that each good is liked by some agent, i.e., for all j, $\max_i u_{ij} > 0$. We also make the nonstandard assumption that for every proper subset $B' \subset B$ there is at least one good g_j that is not completely owned by the agents in B' and such that at least some $b_i \in B'$ is interested in g_j, i.e., $w_{kj} > 0$ for some $b_k \notin B'$ and $u_{ij} > 0$ for some $b_i \in B'$. In other words, there is no subset of agents that are only interested in the goods completely owned by them. References [15, 7, 11] show how to remove the nonstandard assumption. We assume that utilities u_{ij} and weights w_{ij} are integral and use $U = \max_{i\in[n],j\in[m]} u_{ij}$ to denote the maximum utility and $W = \max_{j\in[m]} \max_{i\in[n]} w_{ij}$ to denote the maximum weight. Then the budget available to any agent is bounded by $\sum_j w_{ij}p_j \le nW \max_j p_j$.

Linear Exchange markets were introduced by Walras [19] back in 1874. Walras also argued that equilibrium prices exist. The first rigorous proof for the existence of equilibirum under strong assumptions was given by Wald [18]. Arrow and Debreu [1] gave the proof for the existence of equilibrium when the utility functions are concave. There has been substantial algorithmic research put into determining the equilibrium prices since the 60s. Codenotti et al. [3] gives a surveys the algorithmic literature before 2004. While there are strongly polynomial approximation schemes for determining the equilibrium prices [16, 14, 9], the existence of a strongly polynomial exact algorithms still remains as an open question. There have been exact finite algorithms [12, 13], exact weakly polynomial time algorithms [15, 20, 11, 10] and the characterization of the equilibrium prices as a solution set of a convex program [7, 17].

A market equilibrium can be found in time polynomial in n, m, $\log U$ and $\log W$ by a number of different algorithms. Jain [15] and Ye [20] gave algorithms based on the ellipsoid and the interior point method, respectively, and Duan and Mehlhorn [11] and Duan, Garg, and Mehlhorn [10] described combinatorial algorithms. The algorithm by Ye has a running time of $\mathcal{O}(\max(n,m)^8(\log UW)^2)$, see [10, footnote on page 2].

The combinatorial algorithms actually only solve a special case: $m = n$ and each agent is the sole owner of a good, i.e., $w_{ii} = 1$, and $w_{ij} = 0$ for $i \neq j$. The algorithm in [10] solves the special case in time $\mathcal{O}(n^7 \log^3(nU))$. A reduction for reducing the general case to the special case is known, see Section 2. However, it turns a general problem with n agents and m goods into a special problem with nm goods and hence leads to a running time of $\tilde{\mathcal{O}}((nm)^7\text{poly}(\log(U)))$ (ignoring poly logarithmic dependencies on n). In an unpublished note, Darwish and Mehlhorn [6, 4] have shown how to extend the algorithm in [11] to the general problem without going through the reduction. The resulting running time is $\mathcal{O}(\max(n,m)^{10}\log^2(\max(n,m)UW))$. They were unable to generalize the approach in [10].

Our Contribution. Our contribution is twofold: a combinatorial algorithm for the general problem and examples that are difficult for the algorithms in [11, 10]. We give a combinatorial algorithm for the general problem with running time $\mathcal{O}((n+m)^7 \log^3(nmUW))$. The algorithm refines the algorithm in [10] by several new ideas. We discuss them in Section 2. In particular, in [10], the number of iterations compared to [11] is reduced by a factor of

$\Omega(n)$ by a modified price update rule . The modified update rule is subtle and heavily relies on the fact that there is a one to one correspondence between an agent and a good (one agent owns all of one good only) and this is not true in the general scenario and several of their crucial arguments break down. We come up with a novel price update rule that also highlights some new structure in the problem.

We also give examples that are difficult for the algorithms in [11, 10] amd where the equilibrium prices are exponential in U. Both algorithms are iterative and need $\mathcal{O}(n^5 \log(U))$ and $\mathcal{O}(n^4 \log(nU))$ iterations respectively. The examples force the algorithms into $\tilde{\Omega}(n^{4+\frac{1}{3}} \log(U))$ and $\tilde{\Omega}(n^4 \log(U))$ iterations respectively (ignoring poly logarithmic dependencies on n). They separate the two algorithms.

2 Determining the equilibrium price vector of the general linear Arrow-Debreu market

For completeness we first give the reduction from the general case to the special case (where each agent owns all of one good only). This reduction is well-known. For each positive w_{ij} we create an agent b_{ij} and a good g_{ij} owned by this agent. There is one unit of good g_{ij}. If $w_{ij} = 0$, there is no good g_{ij} and no agent b_{ij}. We interpret g_{ij} as the goods g_j owned by b_i and b_{ij} as a copy of agent b_i. We define the utility derived by the agent b_{ij} from one unit of good $g_{\ell k}$ as $\tilde{u}_{ij,\ell k} = w_{\ell k} \cdot u_{ik}$; here the factor u_{ik} reflects that b_{ij} is a copy of agent b_i and $g_{\ell k}$ is a copy of g_k and the factor $w_{\ell k}$ reflects that b_ℓ owns $w_{\ell k}$ units of good g_k but there is only copy of good $g_{\ell k}$.

▶ **Lemma 1.** *Let p and f be the market clearing price vector and the corresponding money flow for the above instance of the special case with nm agents and goods. Then $\frac{p_{\ell k}}{w_{\ell k}}$ does not depend on ℓ, but only on k. Let $\hat{p}_k = \frac{p_{\ell k}}{w_{\ell k}}$ and $\hat{f}_{ik} = \sum_{j\in[m]} \sum_{\ell\in[n]} f_{ij,\ell k}$. Then $\hat{p}$ and $\hat{f}$ are the market clearing price vector and corresponding money flow for general case with utility matrix u and weight matrix w.*

Proof. Assume $\frac{w_{\ell k}}{p_{\ell k}} > \frac{w_{hk}}{p_{hk}}$. Let b_{ij} be any arbitrary agent. Then

$$\frac{\tilde{u}_{ij,\ell k}}{p_{\ell k}} = \frac{w_{\ell k} \cdot u_{ik}}{p_{\ell k}} > \frac{w_{hk} \cdot u_{ik}}{p_{hk}} = \frac{\tilde{u}_{ij,hk}}{p_{hk}}.$$

Hence agent b_{ij} prefers good $g_{\ell k}$ over good g_{hk}. Since b_{ij} is arbitrary, g_{hk} will not be sold at all, a contradiction. Thus $\frac{w_{\ell k}}{p_{\ell k}} = \frac{w_{hk}}{p_{hk}}$ for all ℓ and h. Now, it is easy to verify that every agent invests in goods that give him maximum utility to price ratio. Assume $\hat{f}_{ik} > 0$. Then for any arbitrary k' and ℓ', there is a j and ℓ, such that

$$\frac{u_{ik}}{\hat{p}_k} = \frac{\tilde{u}_{ij,\ell k}}{w_{\ell k} \cdot \hat{p}_k} = \frac{\tilde{u}_{ij,\ell k}}{p_{\ell k}} \geq \frac{\tilde{u}_{ij,\ell' k'}}{p_{\ell' k'}} = \frac{\tilde{u}_{ij,\ell' k'}}{w_{\ell' k'} \cdot \hat{p}_{k'}} = \frac{u_{ik'}}{\hat{p}_{k'}}.$$

The equilibrium flow constraints are also easily verifiable,

$$\sum_{i\in[n]} \hat{f}_{ik} = \sum_{i\in[n]} \sum_{j\in[m]} \sum_{\ell\in[n]} f_{ij,\ell k} = \sum_{\ell\in[n]} p_{\ell k} = \sum_{\ell\in[n]} w_{\ell k} \cdot \hat{p}_k$$

$$\sum_{k\in[m]} \hat{f}_{ik} = \sum_{k\in[m]} \sum_{j\in[m]} \sum_{\ell\in[n]} f_{ij,\ell k} = \sum_{j\in[m]} p_{ij} = \sum_{j\in[m]} w_{ij} \cdot \hat{p}_j$$ ◀

We now give our algorithm that does not rely on this reduction.

Algorithm 1 Combinatorial algorithm for determining the equilibirum prices in the general linear Arrow-Debreu market.

Set $p_i \leftarrow 1 \quad \forall j \in [n]$.
Set $\varepsilon \leftarrow 1/(8 \cdot (n+m)^{4(n+m)}(UW)^{3(n+m)})$.
while $\|r_f\|_2 > \varepsilon$, where f is a balanced flow in N_p **do**
    Let S be the set of high-surplus agents w.r.t f in N_p.
    $x \leftarrow \min(x_{\text{eq}}, x_{23}, x_{24}, x_{13}, x_2, x_{\max})$.
    Multiply prices of goods in $\Gamma(S)$ by x and update f, p to f' and p' as in (1) - (2) and N_p to $N_{p'}$.
    Let f'' be the balanced flow in $N_{p'}$.
    Set $p \leftarrow p'$ and $f \leftarrow f''$.
end while
Round p to equilibrium prices.

2.1 The Algorithm

Algorithm 1 shows the algorithm. Similar to the algorithms in [8, 11, 10], the algorithm is iterative and flow based. For the description of the algorithm, we need the concepts of an equality network, of a balanced flow, of the set of high-surplus buyers, and of the flow- and price-update.

Equality Network N_p. For a price vector p the equality network N_p is a flow network with vertices $s \cup t \cup B \cup G$ and edges

- (s, b_i) with capacity $\sum_{j\in[m]} w_{ij} \cdot p_j$ for all $i \in [n]$.
- (g_j, t) with capacity $\sum_{i\in[n]} w_{ij} \cdot p_j$ for all $j \in [m]$.
- (b_i, g_j) with capacity ∞ iff $u_{ij}/p_j \geq u_{ik}/p_k$ for all $k \in [m]$.

For any $B' \subseteq B$ let $\Gamma(B') = \{g_j \mid (b_i, g_j) \in N_p \text{ for some } b_i \in B'\}$ denote the neighborhood of B' in N_p (all the goods agents in B' may invest on).

Surpluses and Surplus vector r_f. Let f be a valid flow in the equality network N_p, and let f_{ij}, f_{si} and f_{jt} denote the flow along the edge (b_i, g_j), (s, b_i) and (g_j, t) respectively. We define the surplus $r_f(b_i)$ of the agent b_i and $r_f(g_j)$ of the good g_j, as $r_f(b_i) = \sum_{j\in[m]} w_{ij} \cdot p_j - f_{si}$ and $r_f(g_j) = \sum_{i\in[n]} w_{ij} \cdot p_j - f_{jt}$ respectively. Surpluses are always non-negative. We define the surplus vector $r_f \in \mathbb{R}^n$ as $\langle r_f(b_1), r_f(b_2), \ldots r_f(b_n)\rangle$.

Balanced Flow: A balanced flow f is a valid flow in N_p with minimum norm $\|r_f\|_2^2$. Every balanced flow is a maximum flow. Additionally, if f is a balanced flow and f_{ij} and $f_{i'j}$ are positive, then the surpluses of the agents b_i and $b_{i'}$ are the same and if (b_i, g_j) and $(b_{i'}, g_j)$ are edges of N_p and $r_f(b_i) > r_f(b_{i'})$ then $f_{i'j} = 0$. This essentially follows from the fact that the L2 norm of a vector reduces as the components move closer to each other in magnitude (while the L1 norm remain constant). The following algorithmic property of balanced flows will be useful.

▶ **Lemma 2.** *Balanced Flows can be computed with at most n max flow computations [8] and by one parameterized flow computation [5].*

High Surplus Agents S and Goods in High Demand $\Gamma(S)$. Let f be a balanced flow and assume that there is surplus. The high surplus edges and high demand goods w.r.t. f in

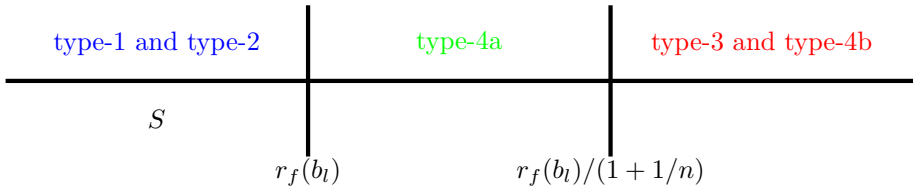


Figure 1 The horizontal line indicates the agents ordered by decreasing surplus from left to right. b_ℓ is the agent of smallest surplus in S. All agents with surplus in $r_f(b_\ell), r_f(b_\ell)/(1+1/n)]$ are type-4a agents. They have no outflow and do not own goods in $\Gamma(S)$.

N_p are defined exactly as in [10]. Renumber the agents in order of decreasing surplus so that b_1 has the highest surplus and b_n has the lowest surplus. Let ℓ be minimal such that $r_f(b_\ell) > r_f(b_{\ell+1})$ and for every k such that $r_f(b_\ell) > r_f(b_k) \geq r_f(b_\ell)/(1+1/n)$, $f_{sk} = 0$ and $w_{kj} = 0$ for every $g_j \in \Gamma(S)$, where $S = \{b_1, \ldots, b_\ell\}$. If no such ℓ exists, let $\ell = n$. We refer to S as the set of high surplus agents and to $\Gamma(S)$ as the set of high demand goods. The surplus of the goods in $\Gamma(S)$ is zero since the agents in S have positive surplus. There is no flow on edges $(b_i, g_j) \in N_p$ with $b_i \notin S$ and $g_j \in \Gamma(S)$.

The algorithm in [10] that determines S in time $\mathcal{O}(n^2)$ can be easily generalized and we do not discuss it here.

Price and Flow Update. Like the earlier combinatorial algorithms, our algorithm is a multiplicative price update algorithm. It works in phases and in each phase we compute the balanced flow f in N_p and determine the high surplus agents S and high demand goods $\Gamma(S)$. We increase the prices of the goods in $\Gamma(S)$ as well as the money flow into them by the same factor $x > 1$ (so we change the flow f in N_p to f' in $N_{p'}$). Formally,

$$f'_{ij} = \begin{cases} x \cdot f_{ij} & g_j \in \Gamma(S) \\ f_{ij} & g_j \notin \Gamma(S) \end{cases} \qquad f'_{si} = \begin{cases} x \cdot f_{si} & b_i \in S \\ f_{si} & b_i \notin S \end{cases} \tag{1}$$

$$f'_{jt} = \begin{cases} x \cdot f_{jt} & g_j \in \Gamma(S) \\ f_{jt} & g_j \notin \Gamma(S) \end{cases} \qquad p'_i = \begin{cases} x \cdot p_i & g_j \in \Gamma(S) \\ p_i & g_j \notin \Gamma(S) \end{cases} \tag{2}$$

The Factor x. We define $x = \min(x_{\text{eq}}, x_{23}, x_{24}, x_{13}, x_2, x_{\max})$, where the quantities on the right are defined below.

Since we increase the prices of all the goods in $\Gamma(S)$ by the same factor x, the only equality edges that may disappear are the ones that connect an agent from $B \setminus S$ to a good from $\Gamma(S)$. Since f is a balanced flow and the agents in S have strictly higher surplus than the ones in $B \setminus S$, the edges from agents in $B \setminus S$ to goods in $\Gamma(S)$ in N_p carry no flow and hence them disappearing will not lead to a violation of the flow constraints. The new edges that appear will connect an agent in S to a good in $G \setminus \Gamma(S)$. Therefore we define,

$$x_{\text{eq}} = \min \left\{ \frac{u_{ij}}{p_j} \cdot \frac{p_k}{u_{ik}} \;\middle|\; b_i \in S, (b_i, g_j) \in N_p, g_k \notin \Gamma(S) \right\}$$

This is the minimum x at which a new equality edge appears in the network.

Next we consider how the surpluses of the agents are affected. Observe that $r_{f'}(b_i) = r_f(b_i) + x \cdot (\sum_{g_j \in \Gamma(S)} (w_{ij} p_j - f_{ij}))$. Therefore all the surpluses vary linearly with x. We now, introduce 5 classes of agents similar to the ones in [10]. Figure 1 illustrates this definition.

- Type-1 agent: An agent is a type-1 agent if it belongs to S and its surplus increases by the price change. Formally, b_i is type-1 if $b_i \in S$ and $\sum_{g_j \in \Gamma(S)} w_{ij} p_j > \sum_{g_j \in \Gamma(S)} f_{ij}$.

- Type-2 agent: An agent is a type-2 agent if it belongs to S and its surplus does not increase by the price change. Formally, b_i is type-2 if $b_i \in S$ and $\sum_{g_j \in \Gamma(S)} w_{ij} p_j \leq \sum_{g_j \in \Gamma(S)} f_{ij}$.
- Type-3 agent: An agent is type-3 agent if it does not belong to S and its surplus increases by the price change, i.e., it partially owns a good in $\Gamma(S)$. Formally, $b_i \notin S$ and $w_{ij} > 0$ for some $g_j \in \Gamma(S)$.
- Type-4a agent: An agent is a type-4a agent if it does not belong to S and has no ownership in a good in $\Gamma(S)$ and its surplus is at least $r_{\min}/(1+1/n)$, where $r_{\min}$ is the minimum surplus of any agent in S. Formally, $b_i \notin S$, $w_{ij} = 0$ for all $j \in \Gamma(S)$ and $r_f(b_i) \geq r_{\min}/(1+1/n)$. By definition of S such b_i have no outflow, i.e., $f_{si} = 0$, and no ownership of any good in $\Gamma(S)$, i.e., $w_{ij} = 0$ for any $j \in \Gamma(S)$.
- Type-4b agent: An agent is a type-4b agent if it does not belong to S and has no ownership in a good in $\Gamma(S)$ and its surplus is strictly less than $r_{\min}/(1+1/n)$, where $r_{\min}$ is the minimum surplus of any agent in S. Formally, $b_i \notin S$, $w_{ij} = 0$ for all $j \in \Gamma(S)$ and $r_f(b_i) < r_{\min}/(1+1/n)$.

We abbreviate the above sets of agents of each type as T_1, T_2, T_3, T_{4a} and T_{4b}. Notice that the surpluses of agents in $T_{4a} \cup T_{4b}$ remain unaffected by increase of price of goods in $\Gamma(S)$. We now define x_{23}, x_{24}, x_{13} as the minimal x such that $r_{f'}(b_i) = r_{f'}(b_j)$ for some $b_i \in T_2, b_j \in T_3$ and $b_i \in T_2, b_j \in T_{4b}$ and $b_i \in T_1, b_j \in T_3$ respectively. We also define x_2 as the minimal x when $r_{f'}(b_i) = 0$ where $b_i \in T_2$.

Finally,

$$x_{\max} = \begin{cases} 1 + \frac{1}{Rn^3} & \text{when } \min_{g_j \in \Gamma(S)} p_j < Rn^4 mW \\ 1 + \frac{1}{Rkn^2} & \text{when } \min_{g_j \in \Gamma(S)} p_j \geq Rn^4 mW, \end{cases}$$

where k is the number of agents in S that partially own a good in $\Gamma(S)$ and $R = 8e^2$. Note that k is at least the number of type-1 agents.[1] We distinguish light and heavy iterations. An iteration is light, if $p_j < Rn^4 mW$ for some good $g_j \in \Gamma(S)$ (some good in demand is not heavily priced), and heavy if $p_j \geq Rn^4 mW$ for all goods $g_j \in \Gamma(S)$ (all goods in demand are heavily priced).

Effect of an Iteration. We multiply the prices of the goods in $\Gamma(S)$ by x and update f, p to f' and p' as in (1)-(4) and N_p to $N_{p'}$. Note that the goods completely sold w.r.t. f in N_p are also completely sold with respect to $N_{p'}$. Maximizing and balancing the flow will not increase the surplus of a good from zero to a positive value. Thus all goods that are completely w.r.t. f in N_p are also completely sold w.r.t. the balanced flow f'' in $N_{p'}$. The 2-norm of $r_{f''}$ is at most the $L2$ norm of $r_{f'}$.

We stop updating the prices once the $L2$ norm of the surplus vector is at most $\varepsilon = 1/(8(n+m)^{4(n+m)}(U \cdot W)^{3(n+m)})$.

---

[1] In [10], $x_{\max}$ is defined as

$$x_{max} = \begin{cases} 1 + \frac{1}{Rn^3} & \text{if there are type-3 agents} \\ 1 + \frac{1}{Rkn^2} & \text{otherwise, where } k = \text{number of type-1 agents.} \end{cases}$$

The revised definition we change the classification of iterations. We do not classify them based on the type of the agents in S, but by looking at the smallest price of a good in $\Gamma(S)$ and we have a larger k (k is at least the number of type-1 agents).

2.2 Analysis of the Algorithm

Before we present the analysis of our algorithm we briefly indicate why the price-update scheme used in [10] does not generalize. The algorithm in [10] reduces the number of iterations of the algorithm in [11] by a multiplicative factor of $\Omega(n)$ by performing a more careful selection of the set S and crucial change in the price update rule. In particular they set

$$x_{max} = \begin{cases} 1 + \frac{1}{Rn^3} & \text{if there are type-3 agents} \\ 1 + \frac{1}{Rkn^2} & \text{otherwise, where } k = \text{number of type-1 agents.} \end{cases}$$

Since in the special case, every agent owns only all of one good, k equals the number of type-1 agents in S and the number of goods in $\Gamma(S)$ in all iterations that do not involve a type-3 agent. We enlist two crucial arguments that are necessary (not sufficient) to reduce the number of iterations by $\Omega(n)$ from that in [11] and their dependencies on k as follows,

1. There is a decrease in the $L2$ norm of the surplus vector in every balancing iteration without a type-3 agent. This argument crucially relies on k being equal to the number of type-1 agents.
2. The total multiplicative increase of the $L2$ norm of the surplus vector in all x_{max} iterations without a type-3 agent is at most $(nU)^{\mathcal{O}(n)}$. This claim relies on the fact that k equals the number of goods in $\Gamma(S)$.

In the general scenario we do not have such one to one correspondence between the agents and the goods. Therefore if we choose k to be either the number of type-1 agents in S or the number of goods in $\Gamma(S)$, then one of the claims from above will fail. Thus we need at least a different classification of the iterations or a different choice of k than the ones used in [10]. In our algorithm we do both and thereby highlighten more hidden structure in the problem.

Like the algorithms in [11, 10], in every iteration Algorithm 1 only increases the prices of goods that are completely sold (since f is a balanced flow and the agents in S have positive surplus, the goods in $\Gamma(S)$ will have zero surplus). Since the sum of surpluses of the agents equals the sum of surpluses of the goods, both are therefore non-decreasing during a price update. Also the goods completely sold w.r.t. f in N_p also remain sold w.r.t. f'' in $N_{p'}$ in every iteration. Therefore the sum of surpluses of the agents is non-increasing throughout the algorithm. Initially (when the price of every good g_j is 1) this sum is at most nmW.

▶ **Observation 3.** *The $L1$ norm of the surplus vector is non-increasing throughout the algorithm and is at most nmW.*

2.2.1 An Upper Bound on the Maximum Price

Observe that in every iteration of the algorithm there is a good with unit price. This follows from the fact that the goods that are completely sold stay completely sold and since only the prices of goods that are completely sold are increased, the price of goods that are not completely sold is equal to the initial price and hence equal to one. We now derive an upper bound on the maximum price of a good.

▶ **Lemma 4.** *At any time during the course of the algorithm the maximum price is at most $\max(2,U)^{m-1} \cdot W^{2m-2}$ and the maximum budget is at most $\max(2,U)^m \cdot W^{2m}$.*

Proof. Let us renumber the goods in increasing order of their prices. We show that $p_i \leq \max(2,U)^{i-1} \cdot W^{2i-2}$ by induction on i. The smallest price is 1 and this establishes the induction base. Consider an arbitrary i and let $\hat{G}$ denote the set of goods $\{g_{i+1}, ..., g_m\}$ and $\hat{B}$ be the agents investing on the goods in $\hat{G}$. We will derive a bound on p_{i+1}. We distinguish two cases.

Assume first that there is an agent in $b_h \in \hat{B}$ that is also interested in a good in $G \setminus \hat{G}$; say b_h invests on good $g_j \in \hat{G}$ and is interested in good $g_\ell \in G \setminus \hat{G}$. Then $u_{hj}/p_j \geq u_{h\ell}/p_\ell$ and hence $p_{i+1} \leq p_j \leq U \cdot p_\ell \leq U \cdot p_i \leq U \cdot \max(2,U)^{i-1} \cdot W^{2i-2} \leq \max(2,U)^i \cdot W^{2i}$.

Assume next that the agents in $\hat{B}$ are only interested in the goods in $\hat{G}$. Then there must be a good in $\hat{G}$, say g_k, that is partially owned by an agent in $B \setminus \hat{B}$. Otherwise, the agents in $\hat{B}$ will only be interested in goods completely owned by them. Therefore let $b_h \in B \setminus \hat{B}$ be the agent that partially owns g_k. The budget m_h of b_h is at least $w_{hk}p_k$ and hence at least p_k. Since b_h invests only in goods in $G \setminus \hat{G}$, its budget is at most the total value of the goods in $G \setminus \hat{G}$. Thus

$$m_h \leq W \cdot \sum_{j \in [i]} p_j \leq W \cdot \sum_{j \in [i]} \left(\max(2,U)^{j-1} W^{2j-2}\right) \leq W \max(2,U)^i W^{2i-1} = \max(2,U)^i W^{2i}$$

and hence $p_{i+1} \leq p_k \leq \max(2,U)^i W^{2i}$. The maximum budget of an agent is at most W times the total price of all goods and hence is bounded by $W \cdot \sum_{j \in [m]} \left(\max(2,U)^{j-1} \cdot W^{2j-2}\right) \leq \max(2,U)^m \cdot W^{2m}$. ◀

In particular, when $n = m$ and w is an identity matrix, we have an upper bound of $\max(2,U)^{n-1}$ for the highest price of a good in contrast to $\max(n,U)^n$ in [11, 10]. Also note that the maximum price of a good and the maximum budget of an agent is independent of the number of agents. We now separately bound the $x_{\max}$-iterations ($x = x_{\max}$) and balancing iterations ($x < x_{\max}$).

2.2.2 Bounding the number of $x_{\max}$-iterations

In this section we will be bounding the light and the heavy $x_{\max}$-iterations separately. For bounding both classes of iterations, we use upper bounds on the prices of the goods. A more aggressive price update scheme is used for the heavy $x_{\max}$-iterations as the prices of all goods in $\Gamma(S)$ in such iterations are high. Such aggressive price update may apparently result in a significant multiplicative increase in the $L2$ norm of the surplus vector. We address this concern in the next subsection. We first show that $x_{\max}$-iterations where there is at least one type-3 agent are light.

▶ **Lemma 5.** *$x_{\max}$-iterations with at least one type-3 agent are light.*

Proof. Let b_i be a type-3 agent. Then there is a $g_j \in \Gamma(S)$ with $w_{ij} > 0$. The additive increase in the surplus of a type-3 agent b_i during an $x_{\max}$-iteration is at least $w_{ij} \cdot p_j / Rn^3$. Since the total surplus is always at most nmW (by Observation 3), the increase in the surplus of any agent is at most nmW. This immediately implies that $p_j \leq Rn^4 mW$. ◀

Now we bound the number of light $x_{\max}$-iteration.

▶ **Lemma 6.** *The number of light $x_{\max}$-iterations is at most $10R^2n^3m\log(nmW)$.*

Proof. Assume otherwise. Then there exists a good g_j that is the minimum priced good in $\Gamma(S)$ in more than $10R^2n^3\log(nmW)$ iterations. Before the last such iteration, $p_j > (1+\frac{1}{Rn^3})^{10R^2n^3\log(nmW)} > e^{5R\log(nmW)} = n^{5R}m^{5R}W^{5R} > Rn^4mW$, which is a contradiction. The second to last inequality uses the fact that $1 + x > e^{\frac{x}{2}}$ for $0 < x \leq 1$. ◀

We turn to heavy $x_{\max}$-iterations. For such iterations, there exists no type-3 agent and hence the goods in $\Gamma(S)$ are completely owned by the agents in $T_1 \cup T_2$. Thus

$$\sum_{b_i \in T_1 \cup T_2} \sum_{g_j \in \Gamma(S)} f_{ij} = \sum_{g_j \in \Gamma(S)} \sum_{i \in T_1 \cup T_2} w_{ij}p_j = \sum_{b_i \in T_1} \sum_{g_j \in \Gamma(S)} w_{ij} \cdot p_j + \sum_{b_i \in T_2} \sum_{g_j \in \Gamma(S)} w_{ij} \cdot p_j,$$

and hence

$$\sum_{b_i \in T_1} \sum_{g_j \in \Gamma(S)} (w_{ij} \cdot p_j - f_{ij}) = \sum_{b_i \in T_2} \sum_{g_j \in \Gamma(S)} (f_{ij} - w_{ij} \cdot p_j). \tag{3}$$

Note that type-1 and type-2 agents can own goods outside of $\Gamma(S)$. However the above relation will help us prove that this "excess budget" is at most nmW. In fact the following lemma plays a pivotal role to bound the multiplicative increase in the $L2$ norm of the surplus vector in the next subsubsection.

▶ **Lemma 7.** *For every agent $b_i \in S$ in a heavy $x_{\max}$-iteration, $\sum_{g_j \notin \Gamma(S)} w_{ij}p_j \leq nmW$.*

Proof. For $b_i \in S$, we have $r_f(b_i) = \sum_{g_j \notin \Gamma(S)} w_{ij}p_j + \sum_{g_j \in \Gamma(S)}(w_{ij}p_j - f_{ij})$. For $b_i \in T_1$, this implies $\sum_{g_j \notin \Gamma(S)} w_{ij}p_j \leq r_f(b_i) \leq nmW$. For $b_i \in T_2$, using (3)

$$\begin{aligned}\sum_{g_j \notin \Gamma(S)} w_{ij}p_j &= r_f(b_i) + \sum_{g_j \in \Gamma(S)} (f_{ij} - w_{ij}p_j) \leq r_f(b_i) + \sum_{b_h \in T_2} \sum_{g_j \in \Gamma(S)} (f_{hj} - w_{hj}p_j) \\ &= r_f(b_i) + \sum_{b_h \in T_1} \sum_{g_j \in \Gamma(S)} (w_{hj}p_j - f_{hj}) \leq r_f(b_i) + \sum_{b_h \in T_1} r_f(b_h) \leq nmW. \end{aligned}$$ ◀

In a heavy $x_{\max}$-iteration, the price of any good in $\Gamma(S)$ is at least Rn^4mW and hence the budget of any agent that partially owns a good in $\Gamma(S)$ is at least that much. By the above, the ownership of the goods outside $\Gamma(S)$ contribute very little to the budget of such agents. Thus any multiplicative increment on the prices of the goods in $\Gamma(S)$ will inflict an almost equal multiplicative increase in the budget of such agents.

▶ **Lemma 8.** *The number of heavy $x_{\max}$-iterations is $\mathcal{O}(n^3m \cdot \log(WU))$.*

Proof. Consider any heavy $x_{\max}$ iteration and let m_i denote the budget of agent b_i. For any agent b_i that partially owns a good in $\Gamma(S)$, $m_i \geq Rn^4mW$. The budget m_i of any agent b_i that partially owns a good in $\Gamma(S)$, increases as follows (new budget denoted by m_i'):

$$\begin{aligned} m_i' &= \sum_{g_j \notin \Gamma(S)} w_{ij}p_j + (1 + \frac{1}{Rkn^2}) \cdot \sum_{g_j \in \Gamma(S)} w_{ij}p_j \\ &\geq (1 + \frac{1}{Rkn^2}) \cdot (1 + \frac{1}{n^3})^{-1} \cdot (1 + \frac{1}{n^3}) \cdot \sum_{g_j \in \Gamma(S)} w_{ij}p_j \\ &= (1 + \frac{1}{Rkn^2}) \cdot (1 + \frac{1}{n^3})^{-1} \cdot \left(\sum_{g_j \in \Gamma(S)} w_{ij}p_j + \frac{\sum_{g_j \in \Gamma(S)} w_{ij}p_j}{n^3} \right) \\ &\geq (1 + \frac{1}{Rkn^2}) \cdot (1 + \frac{1}{n^3})^{-1} \cdot \left(\sum_{g_j \in \Gamma(S)} w_{ij}p_j + nmW \right) \qquad \text{since} \sum_{g_j \in \Gamma(S)} w_{ij}p_j \geq Rn^4mW \\ &\geq (1 + \frac{1}{Rkn^2}) \cdot (1 + \frac{1}{n^3})^{-1} \cdot \left(\sum_{g_j \in \Gamma(S)} w_{ij}p_j + \sum_{g_j \notin \Gamma(S)} w_{ij}p_j \right) \qquad \text{since} \sum_{g_j \notin \Gamma(S)} w_{ij}p_j \leq nmW \\ &\geq (1 + \Omega(\frac{1}{kn^2})) \cdot m_i. \end{aligned}$$

Let $M = \prod_{i \in [n]} m_i$. Since $m_i \leq (\max(2, U)W^2)^m$ we have $\log(M) \leq nm \log(UW)$ always. Also $\log M \geq 0$ initially. At any heavy $x_{\max}$ iteration, $\log(M)$ increases by a additive factor of $\log((1 + \Omega(\frac{1}{kn^2}))^k) \in \Omega(\frac{1}{n^2})$. Since $\log(M)$ is non-decreasing in every iteration of the algorithm (since the prices of the goods and the budgets of the agents only increase), the number of heavy $x_{\max}$-iterations is $\mathcal{O}(n^3m \log(WU))$. ◀

We have now bounded the total number of $x_{\max}$-iterations by $\mathcal{O}(n^3m \log(nmUW))$.

2.2.3 On the Increase in the $L2$-Norm of the Surplus Vector in the $x_{\max}$-Iterations

The $L2$ norm of the surplus vector is minimal for the balanced flow. We just look at the difference in the $L2$ norm of the surpluses with respect to the flows f in N_p and f' in $N_{p'}$ (Updated flow as in 1 - (2)) . Note that this difference in surplus is at least as large as the difference between the $L2$ norm of the surplus vector with respect to f in N_p and f'' in $N_{p'}$ (balanced flow in $N_{p'}$ in Algorithm 1). This suffices as we are upper bounding the difference in the $L2$ norm of the surplus vector in this section.

In a light $x_{\max}$-iteration, the $L2$ norm of the surplus vector increases at most by a factor of $(1+\mathcal{O}(\frac{1}{n^3}))$ and thus the total multiplicative increase in the $L2$ norm of the surplus vector resulting from such iterations is $(1+\mathcal{O}(\frac{1}{n^3}))^{\mathcal{O}(n^3 m\cdot\log(nmW))} = (nmW)^{\mathcal{O}(m)}$.

We now bound the multiplicative increase resulting from heavy $x_{\max}$-iterations. Despite the more aggressive price update scheme in heavy $x_{\max}$-iterations, we can assure the same multiplicative increase. As in [10] we wish to prove that the ratio of the highest to the lowest surplus of the agents in S is at most $1+\mathcal{O}(\frac{k}{n})$). One possible approach is to show that the number of distinct surpluses in S is $\mathcal{O}(k)$ (in that case the ratio will be $(1+\frac{1}{n})^{\mathcal{O}(k)} = 1+\mathcal{O}(\frac{k}{n})$). In [10], this is relatively easy to argue, as goods and agents are in one-to-one correspondence and all agents having positive outflow to a good g have same surplus (by the property of balanced flow). This immediately implies that there are at most $2k$ distinct surpluses of the agents in S (additional k for agents with zero outflow that own one of the goods in $\Gamma(S)$). This argument does not hold in the general scenario as the number of goods can be much larger $|S|$. However Lemma 7 gives us a useful structure in the equality network.

▶ **Lemma 9.** *The total multiplicative increase resulting in the $L2$ norm of the surplus vector in heavy $x_{\max}$-iterations is $(WU)^{\mathcal{O}(m)}$.*

Proof. Let S' be the set of agents in S that partially own some good in $\Gamma(S)$ and $k = |S'|$. Let S'' be the set of agents in S with positive outflow. Any agent in S' has a budget of at least Rn^4mW (follows from the definition of heavy x_{max} iteration) and therefore has positive outflow (since its surplus is at most nmW by Observation 3). Thus $S' \subseteq S''$. By Lemma 7, the budget of any agent in $S \setminus S'$ is at most nmW and hence the total outflow from agents in $S \setminus S'$ is at most n^2mW. Therefore any good in $\Gamma(S)$ must have inflow from an agent in S' and hence the surplus of any agent in S'' is equal to the surplus of some agent in S'.

Let $r_1 > r_2 > \ldots > r_h$ with $h \leq k$ be the distinct surplus values of the agents in S''. Agents in $S \setminus S''$ have no outflow and no ownership of any good in $\Gamma(S)$. Therefore $r_{i+1} \geq r_i/(1+1/n)$ by definition of S for $1 \leq i < h$.

From $r_i \leq (1+\frac{1}{n})r_{i+1}$ for all i, we conclude $r_1 \leq (1+\frac{1}{n})^k r_h \leq (1+\frac{2k}{n})r_h$. Therefore for any type-1 agent b_i, we can claim that $r_f(b_i) < (1+\frac{2k}{n})r_h$. Let $r_{f'}$ be the surplus vector after the $x_{\max}$-iteration. Since there are no type-3 agents in this iteration, only the surpluses of the type-1 and type-2 agents belonging to S'' are affected (Agents belonging to $S \setminus S''$ have no ownership of goods in $\Gamma(S)$ and no outflow also, so their surpluses are unchanged when we change f to f') . Let δ_i denote the increase in the surplus of a type-1 agent b_i and let μ_j denote the decrease of surplus of a type-2 agent b_j. Note that $\sum_{b_i\in T_1}\delta_i = \sum_{b_i\in T_2}\mu_i \leq \frac{1}{Rkn^2}\sum_{b_i\in T_1} r_f(b_i)$. Then,

$$\begin{aligned}
\|r_{f'}\|_2^2 - \|r_f\|_2^2 &= \sum_{b_i \in T_1} ((r_f(b_i) + \delta_i)^2 - r_f(b_i)^2) - \sum_{b_i \in T_2} (r_f(b_i)^2 - (r_f(b_i) - \mu_i)^2) \\
&= 2 \sum_{b_i \in T_1} r_f(b_i)\delta_i - 2 \sum_{b_i \in T_2} r_f(b_i)\mu_i + \sum_{b_i \in T_1} \delta_i^2 + \sum_{b_i \in T_2} \mu_i^2 \\
&\leq 2r_1 \sum_{b_i \in T_1} \delta_i - 2r_h \sum_{b_i \in T_2} \mu_i + \sum_{b_i \in T_1} \delta_i^2 + \sum_{b_i \in T_2} \mu_i^2 \\
&\leq 2(r_1 - r_h) \sum_{b_i \in T_1} \delta_i + 2 \left(\sum_{b_i \in T_1} \delta_i \right)^2 \\
&\leq 2((1 + \frac{2k}{n})r_h - r_h) \sum_{b_i \in T_1} \frac{1}{Rkn^2} \cdot r_f(b_i) + 2\frac{1}{R^2k^2n^4} \left(\sum_{b_i \in T_1} r_f(b_i) \right)^2 \\
&\leq \frac{4}{Rn^3} \sum_{b_i \in T_1} r_f(b_i)^2 + n \cdot \frac{2}{R^2k^2n^4} \sum_{b_i \in T_1} r_f(b_i)^2,
\end{aligned}$$

Thus $\|r_{f'}\|_2^2 \in (1 + \mathcal{O}(\frac{1}{n^3}))\|r_f\|_2^2$. Therefore the multiplicative increase in any heavy $x_{\max}$-iterations is $1 + \mathcal{O}(\frac{1}{n^3})$. Thus the total multiplicative increase in the $L2$ norm of the surplus vector in all heavy $x_{\max}$ iterations is at most $(1 + \mathcal{O}(\frac{1}{n^3}))^{\mathcal{O}(n^3 m \log(WU))} \in \mathcal{O}(WU)^{\mathcal{O}(m)}$. ◀

Thus the total multiplicative increase in all $x_{\max}$-iterations is at most $(nmUW)^{\mathcal{O}(m)}$.

2.2.4 Balancing Iterations

In the balancing iterations $x < x_{\max}$. First we discuss the case when $x = \min(x_{23}, x_{24}, x_{13}, x_2)$. Since the $L1$ norm of the surplus vector is non-increasing during such an iteration (by Observation 3), the total decrease in the surplus of the type-2 agents is at least the total increase in the surpluses of the type-1 and type-3 agents. So now we quantify the decrease in the $L2$ norm of the surplus during such an iteration. Let $r_{\min}$ denote the lowest surplus of an agent in S and $r_{\max}$ denote the highest surplus of a type-3 or type-4b agent. Notice that the highest surplus of any agent and hence any agent in S is at most $e \cdot r_{min}$ and that $r_{\max} \leq r_{\min}/(1 + 1/n)$.

Each type-1 agent's surplus increases at most by a multiplicative factor of $1 + 1/Rkn^2$. Every type-1 agent partially owns at least one good in $\Gamma(S)$ and therefore, k is at least the number of type-1 agents in $B(S)$. Thus the total increase in the sum of squares of the surpluses as a result of increase in the surpluses of the type-1 agents is $\sum_{b_i \in T_1} (1/Rkn^2) \cdot r_f(b_i)^2$ which is at most $e^2 r_{min}^2/Rn^2$.

The surplus of the type-2 and the type-3 agents move closer to each other and the decrease in the former is at least as large as the increase in the latter. Therefore, the sum of their surpluses does not increase. We now quantify the decrease in the sum of squares of their surpluses. Let δ_i denote the decrease in the surplus of a type-2 agent b_i and μ_j the increase in surplus of a type-3 agent b_j. Let $r_{f'}$ be the new surplus vector (w.r.t flow f'). Then the

change in the sum of squares of type-2, type-3 and type-4 agents is

$$\begin{aligned}
&\sum_{b_i \in T_2} ((r_f(b_i) - \delta_i)^2 - r_f(b_i)^2)) + \sum_{b_i \in T_3} ((r_f(b_i) + \mu_i)^2 - r_f(b_i)^2)) \\
&\quad = \sum_{b_i \in T_2} (-2r_f(b_i)\delta_i + \delta_i^2) + \sum_{b_i \in T_3} (2r_f(b_i)\mu_i + \mu_i^2)) \\
&\quad = \sum_{b_i \in T_2} -r_f(b_i)\delta_i + \sum_{b_i \in T_3} r_f(b_i)\mu_i - \sum_{b_i \in T_2} \delta_i(r_f(b_i) - \delta_i) + \sum_{b_i \in T_3} \mu_i(r_f(b_i) + \mu_i)
\end{aligned}$$

For any balancing iteration we have that $\min_{b_i \in T_2}(r_f(b_i) - \delta_i) \geq \max_{b_i \in T_3}(r_f(b_i) + \mu_i)$ and $\sum_{b_i \in T_2} \delta_i \geq \sum_{b_i \in T_3} \mu_i$. This implies that $\sum_{b_i \in T_2} \delta_i(r_f(b_i) - \delta_i) \geq \sum_{b_i \in T_3} \mu_i(r_f(b_i) + \mu_i)$. Notice that $r_{\min}$ is $\min_{b_i \in T_1 \cup T_2} r_f(b_i)$ and $r_{\max}$ is $\max_{b_i \in T_3 \cup T_{4b}} r_f(b_i)$. Therefore, we may continue

$$\begin{aligned}
&\leq -r_{\min} \sum_{b_i \in T_2} \delta_i + r_{\max} \sum_{b_i \in T_3} \mu_i \\
&\leq -(r_{\min} - r_{\max}) \cdot \sum_{b_i \in T_2} \delta_i \leq -(r_{\min} - r_{\max}) \cdot \frac{(\sum_{b_i \in T_2} \delta_i + \sum_{b_i \in T_3} \mu_i)}{2}.
\end{aligned}$$

Now, whenever $x = min(x_{23}, x_{24}, x_{13}, x_2)$, $\sum_{b_i \in T_2} \delta_i + \sum_{b_i \in T_3} \mu_i \geq r_{\min} - r_{\max}$. Thus

$$\leq -\frac{(r_{\min} - r_{\max})^2}{2} \leq -\frac{r_{\min}^2}{2(n+1)^2} \leq -\frac{r_{\min}^2}{4n^2}.$$

Therefore $\|r_{f'}\|_2^2 - \|r_f\|_2^2 \leq \frac{e^2 r_{\min}^2}{Rn^2} - \frac{r_{\min}^2}{4n^2} = -\frac{r_{\min}^2}{4n^2}$ (Recall that $R = 8e^2$). Since $\|r_f\|_2^2 \leq ne^2 r_{\min}^2$, we have

$$\|r_{f'}\|_2^2 \leq (1 - \Omega(\frac{1}{n^3}))\|r_f\|_2^2.$$

Now we look into the case when $x = x_{\text{eq}}$. We update the flow exactly the same way as in [10]. The new equality edge will involve a type-1 agent or a type-2 agent. But after we update the flow, we are in a similar situation as above, with a possibility that the surplus of a few type-1 agents may even decrease and be equal to that of a few type-3 agents. Like earlier the sum of surpluses of the agents is non-increasing here too and all arguments for the decrease in the sum of squares of the agents remain the same. The flow adjustment is exactly identical to the one in [10].

▶ **Lemma 10.** *The total number of balancing iterations is* $\mathcal{O}(n^3 \max(n, m) \log(nmUW))$

Proof. Every balancing iteration results in a multiplicative decrease of $1 - \Omega(\frac{1}{n^3})$ in the $L2$ norm of the surplus. The total multiplicative increase as a result of $x_{\max}$-iterations is $(nmUW)^{\mathcal{O}(\max(n.m))}$. Initially $\|r_f\|_2$ is at most $\sqrt{n(mW)^2}$ and the algorithm terminates with $\|r_f\|_2$ being ε. Therefore the total number of balancing iterations is at most

$$\log_{1-\Omega(\frac{1}{n^3})}(\frac{1}{\varepsilon} \cdot \sqrt{n(mW)^2} \cdot (nmUW)^{\mathcal{O}(m)}) = \mathcal{O}(n^3 \max(n, m) \log(nmUW)). \quad \blacktriangleleft$$

So now we have bounded all the iterations of our algorithm.

▶ **Theorem 11.** *The total number of iterations is* $\mathcal{O}(n^3 \max(n, m) \cdot \log(nWU))$.

2.2.5 Extraction of Equilibrium Prices and Perturbation of Utilities

In [11] is was shown for the special case ($n = m$ and w the identity matrix) that once the total surplus is sufficiently small, the equality network for the current price vector p is the equality network for the equilibrium price vector $\hat{p}$. The equilibrium price vector can then be extracted by solving a linear system. Darwish [4] showed that the same approach also works for the general case.

▶ **Theorem 12.** *Consider any instance of the general linear Arrow-Debreu market with n agents, m goods, utility matrix u and weight matrix w. If p is a price vector such that $\|r_f\|$ at most $1/(8 \cdot (n+m)^{4(n+m)}(UW)^{3(n+m)})$ for any balanced flow f in N_p, then the equilibrium price vector p^* can be determined in $\mathcal{O}((n+m)^4 \cdot \log(UW))$.*

[10] achieves $\mathcal{O}(n^2)$ time for determining the balanced flow in every iteration by keeping the Equality Network acyclic at every point in time in the algorithm. This improvement, after minor adaptions, also applies to the general case. More details are given in the full version of the paper in [2].

2.3 Summary

▶ **Theorem 13.** *The market clearing price vector for the general Arrow-Debreu market can be determined with $\mathcal{O}((n+m)^6 \log(nmWU))$ arithmetic operations.*

Proof. We perturb the utility matrix following the same perturbation as in [10]. This perturbation ensures that the equality network is acyclic at any point in time in the algorithm. Thereafter we run Algorithm 1 until $\|r_f\| < 1/(8 \cdot (n+m)^{4(n+m)}(UW)^{3(n+m)})$. This involves $\mathcal{O}((n)^3 m \log(nWU))$ iterations. In each iteration we can determine x_{23}, x_{24}, x_{13}, x_2 and x_{eq} in $\mathcal{O}(n^2)$ comparisons. The balanced flow can also be determined by $n+m$ max flow calls in N_p. Since N_p is acyclic (due to the perturbation), we can compute each max flow in $\mathcal{O}(n+m)$ arithmetic operations as in [10]. Thus every iteration involves $\mathcal{O}((n+m)^2)$ arithmetic operations and comparisons. Therefore Algorithm 1 terminates performing $\mathcal{O}((n+m)^6 \log(nmWU))$ arithmetic operations and comparisons. Thereafter we perform extraction as in Theorem 12 in time $\mathcal{O}((n+m)^4 \log(nmUW))$ and determine the equilibrium prices for the perturbed utilities. We then determine equilirbium prices corresponding to the original utilities from the equilibrium prices of the perturbed utilities performing $\mathcal{O}(m^3)$ arithmetic operations in $\mathcal{O}(m^4 \log(UW))$ time (details in the full version of the paper in [2]) . Overall we perform $\mathcal{O}((n+m)^6 \log(nmWU))$ arithmetic operations and comparisons. ◀

To achieve the polynomial running time we can follow the same strategy used in [11]. where we restrict the prices and the update factor to powers of $1+1/L$ where L has polynomial bit length (linear in $n+m$). This guarantees that all arithmetic is done on rationals of polynomial bitlength. This can be adapted to the perturbation as well [10].

3 Lower Bounds for the Algorithms in [11, 10]

In this section, we construct non-trivial instances that make the equilibrium prices exponential in U and forces the algorithms in [11, 10] to execute large number of iterations. We construct an I_n comprises of a set of agents $B = \{b_1, b_2, ..., b_n\}$ and a set of goods $G = \{g_1, g_2, ..., g_n\}$, and n is *even*. There is exactly one unit of each good ($W = 1$) and agent b_i only owns one unit of good g_i. We now define the utility matrix as follows: $u_{i,i-1} = U \quad \text{for } 2 \leq i \leq n,$

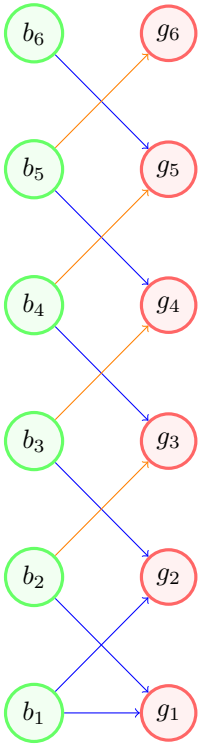


Figure 2 Utility matrix for $n = 6$. The agents are the green nodes on the left and the goods are the red nodes on the right. Agent b_i owns good g_i. The blue edges represent utility U and the orange ones represent utility one.

$u_{i,i+1} = 1$ for $2 \leq i \leq n-1$, and $u_{1,1} = u_{1,2} = U$. $u_{i,j} = 0$ for every other pair of (i, j). See Figure 2.

▶ **Theorem 14.** *For the instance I_n we have,*

1. *The ratio of the maximum to the minimum price of a good at equilirbium is $\Omega(U^{\Omega(n)})$.*
2. *The algorithms in [11, 10] execute $\Omega(n^4 \log(U))$ iterations.*

Proof. We first show (1). Let p be the market clearing price vector with p_i denoting the price of good g_i, and f be the money flow at equilibrium. Since the only agent interested in g_n is b_{n-1} , $p_{n-1} \geq p_n$. We now discuss two disjoint scenarios,

- $p_n = p_{n-1}$. In this case we claim that for every even i, $p_{i-1} = p_i \geq U \cdot p_{i+1} = U \cdot p_{i+2}$ and $f_{i,i-1} = f_{i-1,i} = p_i = p_{i-1}$. For the base case, $i = n$ we have $p_n = p_{n-1}$ and $f_{n,n-1} = f_{n-1,n} = p_n = p_{n-1}$. For the inductive step we assume that the claim holds for $i + 2$. Since g_{i+2} is a bang per buck good for b_{i+1}, $p_i \geq U \cdot p_{i+2} = U \cdot p_{i+1}$. Since the only other agent interested in g_i is b_{i-1} we may conclude that $p_{i-1} \geq p_i$ (b_{i-1} is the only agent investing in g_i). But then again, since the only good b_i invests in is g_{i-1}, $p_{i-1} \leq p_i$. This implies that $p_{i-1} = p_i \geq U \cdot p_{i+1} = U \cdot p_{i+2}$ and $f_{i,i-1} = f_{i-1,i} = p_i = p_{i-1}$.
- $p_{n-1} > p_n$. In this case we claim that for every even i, $p_i < p_{i-1}$ and $p_{i-1} \geq U \cdot p_{i+1}$. For the base case $i = n$, this trivially holds. For the inductive step we assume that our claim holds for $i + 2$. Since $p_{i+1} > p_{i+2}$, the agent b_{i+1} must invest in the good g_i. This implies that $p_i \leq U \cdot p_{i+2}$. Since $p_{i+2} < p_{i+1}$, agent b_i must invest in good g_{i+1}. Therefore g_{i+1} is a bang per buck good for agent b_i, implying that $p_{i-1} \geq U \cdot p_{i+1} > U \cdot p_{i+2} \geq p_i$.

In either case, we have $p_{i-1} \geq Up_{i+1}$ for even i. Thus there are goods with price ratio equal to $U^{\frac{n}{2}-1}$. For (2), note that the algorithm in [11] never increases the price of a good more than a multiplicative factor of $(1 + \mathcal{O}(1/n^3))$. Thus the number of iterations is $\log_{1+\mathcal{O}(1/n^3)} U^{\Omega(n)} \in \Omega(n^4 \log(U))$. To prove that the algorithm in [11, 10] takes $\Omega(n^4 \log(U))$ iterations, we first need to understand the details of how the sets S (high surplus agents) and $\Gamma(S)$ (high demand goods) evolve throughout the iterations of the algorithm in [11, 10]. In particular we show that there exists a good g_i and its price is increased by a multiplicative factor of $U^{\Omega(n)}$ in $x_{\max}$ iterations with $\Omega(n)$ type-1 agents. Note that the price of any good is increased at most by a multiplicative factor of $1 + \mathcal{O}(1/n^3)$ in any $x_{\max}$ iteration with $\Omega(n)$ type-1 agents. Since the total price increase in such iterations is $U^{\Omega(n)}$, the number of such iterations is $\Omega(\log_{1+\mathcal{O}(1/n^3)} U^{\Omega(n)}) \in \Omega(n^4 \log(U))$. For the detailed proof we refer to the full version of the paper in [2]. ◀

We next claim that there is an instance I'_n that separates the two algorithms in [11, 10].

▶ **Theorem 15.** *The number of iterations executed by the algorithm in [11] on the instance I'_n is $\Omega((n^{4+\frac{1}{3}} \log(U))/\log(n)^4)$.*

We refer to the full version of the paper [2] for the exact construction and the detailed proof of Theorem 15.

References

1. Kenneth J. Arrow and Gérard Debreu. Existence of an equilibrium for a competitive economy. *Econometrica*, 22:265–290, 1954. URL: `http://www.jstor.org/stable/1907353`.
2. B. R. Chaudhury and K. Mehlhorn. Combinatorial Algorithms for General Linear Arrow-Debreu Markets. *ArXiv e-prints*, October 2018. `arXiv:1810.01237`.
3. Bruno Codenotti, Benton McCune, and Kasturi R. Varadarajan. Market equilibrium via the excess demand function. In *STOC*, pages 74–83. ACM, 2005.
4. Omar Darwish. Market Equilibrium Computation for the Linear Arrow-Debreu Model. Master's thesis, Fachbereich Informatik, Saarland University, 2016.
5. Omar Darwish and Kurt Mehlhorn. Improved Balanced Flow Computation Using Parametric Flow. *Information Processing Letters*, pages 560–563, 2016.
6. Omar Darwish and Kurt Mehlhorn. Linear Arrow-Debreu Markets: The General Case. Unpublished, 2016.
7. Nikhil R. Devanur, Jugal Garg, and László A. Végh. A Rational Convex Program for Linear Arrow-Debreu Markets. arXiv:1307.803, 2013.
8. Nikhil R. Devanur, Christos H. Papadimitriou, Amin Saberi, and Vijay V. Vazirani. Market equilibrium via a primal–dual algorithm for a convex program. *J. ACM*, 55(5):22:1–22:18, November 2008. `doi:10.1145/1411509.1411512`.
9. Nikhil R. Devanur and Vijay V. Vazirani. An improved approximation scheme for computing Arrow-Debreu prices for the linear case. In *FSTTCS*, pages 149–155, 2003.
10. Ran Duan, Jugal Garg, and Kurt Mehlhorn. An improved combinatorial algorithm for linear Arrow-Debreu markets. In *SODA*, pages 90–106, 2016.
11. Ran Duan and Kurt Mehlhorn. A Combinatorial Polynomial Algorithm for the Linear Arrow-Debreu Market. *Information and Computation*, 243:112–132, 2015. a preliminary version appeared in ICALP 2013, LNCS 7965, pages 425-436.
12. B. Curtis Eaves. A Finite Algorithm for the Linear Exchange Model. *Journal of Mathematical Economics*, 3:197–203, 1976.

13 Jugal Garg, Ruta Mehta, Milind Sohoni, and Vijay V. Vazirani. A Complementary Pivot Algorithm for Markets Under Separable, Piecewise-linear Concave Utilities. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 1003–1016, New York, NY, USA, 2012. ACM. doi:10.1145/2213977.2214068.

14 Rahul Garg and Sanjiv Kapoor. Auction Algorithms for Market Equilibrium. In *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*, STOC '04, pages 511–518, New York, NY, USA, 2004. ACM. doi:10.1145/1007352.1007430.

15 Kamal Jain. A Polynomial Time Algorithm for Computing an Arrow-Debreu Market Equilibrium for Linear Utilities. *SIAM J. Comput.*, 37(1):303–318, April 2007. doi:10.1137/S0097539705447384.

16 Kamal Jain, Mohammad Mahdian, and Amin Saberi. Approximating Market Equilibria, 2003.

17 E. I. Nenakov and M. E. Primak. One algorithm for finding solutions of the Arrow-Debreu model. *Kibernetica*, 3:127–128, 1983.

18 A. Wald. Über einige Gleichungssysteme der mathematischen Ökonomie. *Zeitschrift für Nationalökonomie*, 7:637–670, 1936. Translated: Econometrica, Vol. 19(4), p.368–403, 1951.

19 Léon Walras. *Elements of Pure Economics, or the theory of social wealth.* Porcupine Press, 1874.

20 Yinyu Ye. A path to the Arrow-Debreu competitive market equilibrium. *Math. Program.*, 111(1):315–348, June 2007. doi:10.1007/s10107-006-0065-5.

On the Welfare of Cardinal Voting Mechanisms

Umang Bhaskar[1]
Tata Institute of Fundamental Research, Mumbai, India
umang@tifr.res.in

Abheek Ghosh
The University of Texas at Austin, TX, USA
abheek@cs.utexas.edu

Abstract

A voting mechanism is a method for preference aggregation that takes as input preferences over alternatives from voters, and selects an alternative, or a distribution over alternatives. While preferences of voters are generally assumed to be cardinal utility functions that map each alternative to a real value, mechanisms typically studied assume coarser inputs, such as rankings of the alternatives (called ordinal mechanisms). We study cardinal mechanisms, that take as input the cardinal utilities of the voters, with the objective of minimizing the distortion – the worst-case ratio of the best social welfare to that obtained by the mechanism.

For truthful cardinal mechanisms with m alternatives and n voters, we show bounds of $\Theta(mn)$, $\Omega(m)$, and $\Omega(\sqrt{m})$ for deterministic, unanimous, and randomized mechanisms respectively. This shows, somewhat surprisingly, that even mechanisms that allow cardinal inputs have large distortion. There exist ordinal (and hence, cardinal) mechanisms with distortion $O(\sqrt{m \log m})$, and hence our lower bound for randomized mechanisms is nearly tight. In an effort to close this gap, we give a class of truthful cardinal mechanisms that we call randomized hyperspherical mechanisms that have $O(\sqrt{m \log m})$ distortion. These are interesting because they violate two properties – localization and non-perversity – that characterize truthful ordinal mechanisms, demonstrating non-trivial mechanisms that differ significantly from ordinal mechanisms.

Given the strong lower bounds for truthful mechanisms, we then consider approximately truthful mechanisms. We give a mechanism that is δ-truthful given $\delta \in (0, 1)$, and has distortion close to 1. Finally, we consider the simple mechanism that selects the alternative that maximizes social welfare. This mechanism is not truthful, and we study the distortion at equilibria for the voters (equivalent to the Price of Anarchy, or PoA). While in general, the PoA is unbounded, we show that for equilibria obtained from natural dynamics, the PoA is close to 1. Thus relaxing the notion of truthfulness in both cases allows us to obtain near-optimal distortion.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases computational social choice, voting rules, cardinal mechanisms, price of anarchy, distortion

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.27

1 Introduction

A society or a group of people may have different views and preferences but want to make a collective decision that will impact the entire group. For example, the people of India may have conflicting opinions on which party should win the Lok Sabha elections, and who should be the Prime Minister. This is the problem of preference aggregation, and the methods

[1] Research funded in part by a Ramanujan fellowship.

38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018).
Editors: Sumit Ganguly and Paritosh Pandya; Article No. 27; pp. 27:1–27:22

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

of achieving this aggregation are called voting mechanisms – functions that map the given preferences of voters over a set of alternatives to a single alternative or a distribution over alternatives, without money being exchanged.

Central to the question of preference aggregation is the question of how preferences are perceived by the voters, and how they are expressed to the mechanism. In classical social choice theory, particularly when a voting mechanism is randomized, i.e., can output a distribution over the set of alternatives, voter preferences are assumed to be von Neumann-Morgenstern utility functions, that map each alternative to a real-valued utility. We assume that the total utility each voter has for the alternatives is 1. This is the unit-sum assumption, though other normalizations such as unit-range are also studied. A voter then prefers distributions over the alternatives that maximize her expected utility. These utility functions may be latent and hidden from the mechanism but are required for the voter to rationally compare distributions or lotteries over the set of outcomes. In contrast to these cardinal utility functions of the voters, frequently the mechanisms studied in the literature, and used in practice, have coarser inputs, such as a ranking of the alternatives, or simply a vote for the alternative with the highest utility (called ordinal and plurality voting respectively).

If utility functions are cardinal, then an understanding of cardinal voting mechanisms, where voters give as input their utility functions, seems fundamental to understanding the problem of preference aggregation. Though less popular than other voting mechanisms owing to their complexity, cardinal voting mechanisms find use in many areas. For example, they are motivated by automated agents in recommender systems that use exact numeric values for making decisions, and hence naturally have easily expressible cardinal utilities. The use of these automated agents in a movie recommendation system is described by Ghosh et al. [14] (cf. [24]). Hillinger further argues for the use of cardinal voting mechanisms, especially since they do not artificially restrict the freedom of expression of voters [17].

Given an input format for voter preferences, how then should the mechanism choose an alternative? A widely studied property is incentive compatibility or truthfulness – a voter should maximize her expected utility by truthfully expressing her preferences, irrespective of the votes of others. Truthful mechanisms are desirable since voters need not strategize or seek information on the behavior of other voters. Other properties for mechanisms that are studied include Pareto-efficiency and polynomial-time computation. A natural objective for the voting mechanism, given the cardinal utilities of voters, is to maximize the social welfare or the total utility of the voters. This has been a mechanism objective in a number of recent papers (e.g., [7, 13]). The objective of social welfare assumes that the utilities allow for interpersonal comparison: that a unit of voter 1's utility is equivalent to a unit of agent 2's utility. Such comparisons may not be generally applicable, but even then, aggregate utility (or disutility) is frequently used as a quantitative measure, e.g., man-hours required for a project, or total time spent in traffic. The motivation for studying social welfare from classical economic theory, as well as further uses in modern recommendation systems, is also described by Boutilier et al. [7].

The social welfare of a voting mechanism is measured by the distortion – the ratio of the welfare of the best alternative to the expected welfare obtained by the mechanism, in the worst case over all instances [24].[2] Unfortunately, combined with the requirement that the

[2] In their paper, Procaccia and Rosenschein use distortion to measure the loss due to the embedding of cardinal utilities into the space ordinal preferences. However, they also define distortion as stated here. Other papers use 'approximation ratio' for this quantity (e.g., [13]). We find the term distortion to be more natural and descriptive, and hence use it here for cardinal mechanisms as well.

mechanism be truthful, for the input formats typically studied, the distortion is known to be large – for ordinal mechanisms, that take as input a ranking of the alternatives, the distortion is $\Theta(\sqrt{m \log m})$ [5], where m is the number of alternatives. Even for ordinal mechanisms that are not truthful, the distortion is $\Omega(\sqrt{m})$ [7], suggesting that most of the loss in welfare is caused by the incongruity between the cardinal utilities experienced by the voters, and ordinal preferences given as input to the mechanism. This begs the question: for truthful, cardinal mechanisms, where the input to the mechanisms are cardinal utility functions, how large is the distortion? Note that in this case, there is no incongruity between the preferences experienced and expressed: both are cardinal.

Our goal in this paper is to address this question, and to obtain bounds on the social welfare obtainable by cardinal mechanisms. While cardinal mechanisms are less popular than mechanisms with simpler input formats, we believe that an understanding of cardinal mechanisms is crucial in many ways to understanding other preference aggregation mechanisms. Firstly, lower bounds obtained for cardinal mechanisms are lower bounds for mechanisms with other input formats as well. Secondly, studying cardinal mechanisms helps disentangle the effects of various constraints on the mechanism, since the input format is no longer a constraint. Thirdly, as noted above, particularly in the case of automated agents, cardinal mechanisms are of practical use. Lastly, we note that cardinal mechanisms have received less attention than ordinal mechanisms, and theoretically these present several challenging questions. For example, strong characterizations of truthful ordinal mechanisms are known [16, 1], while truthful cardinal mechanisms are only partially characterized (e.g., [18, 3]). This is a long-standing open question, and we hope that the perspective of distortion of cardinal mechanisms may present useful insights for this question as well.

We study truthful, nearly-truthful, as well as manipulable cardinal mechanisms, and provide bounds on the social welfare obtainable. In the last case, we study the social welfare at equilibrium for the deterministic cardinal mechanism that simply returns the alternative with maximum social welfare for the reported utility functions. While for truthful cardinal mechanisms we show strong and nearly tight lower bounds, for the last two cases we are able to show that with some reasonable restrictions, the distortion achieved is nearly 1, in sharp contrast to the case of truthful cardinal mechanisms, where the distortion is $\Omega(\sqrt{m})$.

Our Contribution. In this paper, we study the distortion of cardinal mechanisms with n unit-sum voters and $m \geq 3$ alternatives. Unit-sum assumes that the sum of utilities of each voter for the alternatives is 1. This assumption implies that all the voters have equal weight, and no voter is more important than the other in contributing to the social welfare. We first focus on truthful cardinal mechanisms, and show that for deterministic cardinal mechanisms, the distortion is $\Theta(mn)$. Note that the trivial randomized mechanism that picks an alternative uniformly without looking at the utilities of the voters has distortion $O(m)$. A natural property for mechanisms is Pareto-optimality (e.g., [16]), which states that for any alternative a chosen by the mechanism with positive probability, there is no alternative b for which all voters have higher utility. A significantly weaker property is unanimity, which states that if there is an alternative that has maximum utility for all voters, then the mechanism should pick this with probability 1. Unfortunately, even for this weaker property, we show that any truthful unanimous mechanism has distortion $\Omega(m)$. Underlying our results are strong previous characterizations of unanimous mechanisms which show that such mechanisms must be random dictatorships, which pick a voter at random and return the maximum utility alternative for the voter [10, 18, 21].

We then focus on truthful randomized mechanisms. There exist randomized ordinal (and hence cardinal) mechanisms with distortion $O(\sqrt{m \log m})$. We show that, perhaps surprisingly, this is nearly the best possible for cardinal mechanisms as well, showing a bound of $\Omega(\sqrt{m})$ on the distortion of cardinal mechanisms. Note that if voters reported their utility truthfully, a mechanism with distortion 1 is trivial. This implies that the loss in welfare due to truthfulness is already large, apart from that caused by the information loss due to the input format.[3]

We leave the problem of closing the gap between the upper and lower bounds ($O(\sqrt{m \log m})$ and $\Omega(\sqrt{m})$) open. We instead address the following question: can cardinal mechanisms with low distortion be very different from ordinal mechanisms? In particular, the characterization by Gibbard of truthful ordinal mechanisms requires such mechanisms to satisfy non-perversity and localization (these are defined in the next section) [16]. Must mechanisms that violate these properties be trivial, with large distortion, or do there exist cardinal mechanisms that violate these properties, and yet have good distortion? We give a mechanism that we call a randomized hyperspherical mechanism that violates these properties, but has distortion $O(\sqrt{m \log m})$, matching the best known upper bound. Spherical mechanisms were previously studied by Feige and Tennenholtz [12], but these are much simpler mechanisms with distortion $\Omega(m)$. We view our mechanism as a significant extension of these. The mechanism we introduce may be of independent technical interest as well. One of the steps involved in the mechanism is to project a point onto the intersection of the standard simplex and a hypersphere of given radius, for which we give a polynomial time algorithm.

Given the strong lower bounds, we then consider two kinds of mechanisms that may incentivize strategic behavior (called manipulable). We first study approximately-truthful cardinal mechanisms, where a mechanism is δ-truthful if no voter can increase her expected utility by more than δ by reporting her utilities untruthfully. Here we show surprisingly good results: for any $\delta \in (0, 1)$, we give a cardinal mechanism that has distortion that approaches 1 as the number of voters increases, and is 2δ-truthful. Thus slightly relaxing the notion of truthfulness allows us to obtain near optimal bounds on the distortion. It is instructive to compare our results with those of Birrell and Pass, who show that approximately truthful ordinal mechanisms can be used to approximate any deterministic ordinal mechanism, in a formal sense [6]. However, the distortion of any deterministic ordinal mechanism is $\Omega(m)$ [24].

We lastly consider the natural, but manipulable, deterministic mechanism that for any utilities given as input, simply returns the alternative that has maximum social welfare according to these utilities. Since we can no longer rely on voters being truthful, we instead consider the pure Nash equilibrium for this mechanism, i.e., utility profiles where no single voter can report a different utility function and improve her utility. Here, the distortion is equivalent to the Price of Anarchy (PoA), defined as the ratio of the welfare of the best alternative to the expected welfare of the mechanism, in the worst case over all equilibria and all instances. Simple examples show the PoA is in general unbounded, and even natural refinements studied previously have unbounded PoA. Instead, we consider equilibria reachable from natural iterative voting, where in each step, a voter changes her input to the mechanism to improve her utility. Iterative voting has been considered in a number of previous papers (e.g., [20, 25]), though usually for ordinal or other mechanisms with restricted input formats. We show that under certain natural restrictions on the allowable deviations, iterative voting converges, and the PoA approaches 1 as the number of voters increases.

[3] Ariel Procaccia in a conversation mentioned that he had also obtained this bound independently, but had not written it up.

Related Work. We focus here on literature directly related to our work, and refer to the book on computational social choice [8] and the survey by Barbera [2] for a wider discussion. For voting mechanisms, truthfulness has been an important consideration, epitomized by the well-known impossibility result by Gibbard and Satterthwaite for deterministic ordinal voting mechanisms which states that any such truthful mechanism must be a dictatorship [15, 27]. Allowing randomization alleviates this, and such mechanisms were characterized by Gibbard [16] as being distributions over unilateral and duple mechanisms, each of which must be localized and non-perverse. This characterization was further refined by Barbera [1].

In comparison, for characterizing truthful cardinal mechanisms, only partial results are known. Characterizations are known for twice-differentiable truthful cardinal mechanisms [3]. Further, truthful cardinal mechanisms which are unanimous are a convex combination of dictatorial schemes [18, 10, 21].

The social welfare of voting mechanisms was explicitly studied by Boutilier et al. [7], who showed that for randomized ordinal mechanisms, the distortion was $O(\sqrt{m}\log^* m)$ and $\Theta(\sqrt{m})$. The concept of distortion as a measure of loss of welfare by a mechanism was introduced earlier and studied for many well-known deterministic voting mechanisms including Borda, Plurality, and Veto [24]. In general, the distortion is shown to be unbounded. For highly structured utility functions, the authors obtain positive results. The mechanisms studied in these papers are not truthful. For truthful ordinal mechanisms for voters with unit-sum utilities, the distortion is known to be $\Theta(\sqrt{m \log m})$ [5].

Filos-Ratsikas and Miltersen study the social welfare of truthful cardinal mechanisms with voters that have unit-range utilities, i.e., for each voter i, $\max_{a\in A} u_i(a) = 1$ and $\min_{a\in A} u_i(a) = 0$ [13]. They obtain bounds of $O(m^{3/4})$ and $\Omega(\log\log m/\log m)$ for truthful cardinal mechanisms, and a bound of $\Omega(m^{2/3})$ for ordinal mechanisms and some generalizations.

Relaxing truthfulness, it is known that approximately truthful randomized ordinal mechanisms can approximate any deterministic ordinal voting rule, in that the output obtained by the randomized mechanism could have been obtained by the deterministic ordinal voting rule by changing a small number of votes [6]. This does not give us bounds on the distortion, since any deterministic ordinal mechanism has distortion $\Omega(m)$ (e.g., [24]).

The social welfare at equilibrium – with the ratio well-known as the Price of Anarchy (PoA) – is known to be bad if the set of equilibria is not restricted. Hence a number of papers study equilibria reachable by natural best-response dynamics, called iterative voting. For the plurality voting rule, best-response dynamics is shown to converge in $O(mn)$ steps [20]. Convergence for Veto, Borda, and other voting rules is also studied [19, 26]. Further, the social welfare for equilibria obtained through best-response dynamics for plurality is known to be at most 1 less than the optimal, however, this can be small (and hence the PoA can be large) for Veto and Borda. Finally, further restrictions on the class of equilibria obtainable have also been studied for plurality voting, such as strong equilibria or equilibria with truth-biased voters [11, 23]. In particular, Rabinovich et al. [25] characterize the set of equilibria obtainable from plurality voting with truth-biased voters.

2 Preliminaries and Notation

A population of n voters (or agents) $N = \{1, 2, \ldots, n\}$ want to select an alternative (or candidate) from a set A of size m. Each voter i has a utility function $u_i : A \to [0, 1]$. We will sometimes abuse notation and think of u_i as vector in $\mathbb{R}^m$. We assume that the voters

are unit-sum voters, i.e., the total utility of a voter is 1 ($\sum_{a\in A} u_i(a) = 1$ for each voter i). We will also assume that no two alternatives have the same utility for a voter, thus if $u_i(a) = u_i(b)$, then $a = b$. This is a mild technical assumption which allows us to obtain a total order over alternatives for each voter, and simplify our proofs. Let $\vec{u} := (u_i)_{i\in N}$ be a vector of utility functions, or a utility profile. Define $\vec{u}/_i u_i'$ to be the utility profile $\vec{u}$ with the utility function for the ith voter replaced by u_i'.

A (cardinal) mechanism μ is defined as a map, possibly randomized, from input utility profiles to distributions over alternatives. For our lower bounds, we assume that the mechanisms we study have at least 3 feasible alternatives, i.e., there exist three alternatives each of which is chosen with positive probability for some input utility profile. We will sometimes compare our mechanisms to ordinal mechanisms, which are similarly possibly randomized maps from rankings over the set of alternatives provided by voters, to distributions over the alternatives. A plurality mechanism is one where each voter selects a single alternative, and the mechanism chooses an alternative that is selected by the maximum number of voters. Given a distribution $(p_a)_{a\in A}$ over the alternatives, the expected utility of a voter is $\sum_{a\in A} u_i(a)p_a$. We assume that all voters are expected utility maximizers, and note that to maximize her utility a voter i may report a different utility function $u_i' \neq u_i$ to the mechanism. For clarity, we call the input provided to a mechanism by a voter her strategy, which may not be her true utility. A mechanism is truthful (more formally, truthful in expectation) if each voter obtains maximum expected utility by reporting her true utility profile u_i, irrespective of the strategies of the other voters. Thus a cardinal mechanism is truthful if for each voter i with utility u_i, and each strategy profile $\vec{u'}$, $\mathbb{E}_{a\sim\mu(\vec{u'})}[u_i(a)] \leq \mathbb{E}_{a\sim\mu(\vec{u'}/_i u_i)}[u_i(a)]$. A mechanism is manipulable if it is not truthful. Further, relaxing truthfulness, in a δ-truthful mechanism each voter can improve her expected utility by at most δ by choosing a strategy that is not her true utility. In both these cases, we assume that voters vote truthfully, since the incentive to misreport utility functions is small.

For an alternative a in an instance with utility profile $\vec{u}$, the utilitarian social welfare (or simply welfare) is $\mathrm{sw}(a) = \sum_i u_i(a)$, the sum of utilities of all the voters for that alternative. We study mechanisms that maximize welfare, and hence our primary measure of a mechanism is its distortion, defined informally as the worst-case ratio of the maximum utility of an alternative, to the expected utility obtained by the mechanism [4, 24]. The distortion of a mechanism is thus at least 1. Under the assumption that voters choose their utility u_i as strategy, the distortion for a mechanism μ is defined formally as:

$$\mathrm{dist}(\mu) := \sup_{\vec{u}} \frac{\max_{a\in A} \mathrm{sw}(a)}{\mathbb{E}_{a\sim\mu(\vec{u})}\mathrm{sw}(a)}\,.$$

In Section 5, we study the simple deterministic mechanism that chooses the alternative with maximum welfare, for the strategy profile reported by the voters. This mechanism is not truthful, and we will be interested in welfare at the pure Nash equilibrium for the voters. In this case, the distortion is equivalent to the well-studied Price of Anarchy. Formally, let $\vec{u}$ be the utility profile of the voters, and let $\vec{u'}$ be a strategy profile. Then $\vec{u'}$ is a pure Nash equilibrium if for every voter $i \in N$ and every strategy u_i'',

$$\mathbb{E}_{a\sim\mu(\vec{u'})} u_i(a) \geq \mathbb{E}_{a\sim\mu(\vec{u'}/_i u_i'')} u_i(a)\,.$$

The Price of Anarchy (PoA) of a mechanism μ is defined as the worst-case ratio over all possible instances, of the maximum welfare of an alternative to the lowest welfare of an alternative chosen by the mechanism at an equilibrium.

$$\sup_{\vec{u}} \sup_{\text{equilibrium strategy profiles } \vec{u'}} \frac{\max_{a\in A} \sum_{i\in N} u_i(a)}{\mathbb{E}_{a\sim\mu(\vec{u'})} \sum_{i\in N} u_i(a)}$$

The following properties for ordinal mechanisms were introduced by Gibbard [16].

▶ **Definition 1.** An ordinal mechanism is non-perverse if increasing the ranking of an alternative by a voter, while leaving the relative order of the other alternatives unchanged, does not decrease the probability that the alternative is selected. An ordinal mechanism is localized if, for two rankings by a voter where the set of top k alternatives are the same (but individual rankings may not be preserved), the total probability mass on these k alternatives is also unchanged.

We extend these properties to cardinal mechanisms as follows.

▶ **Definition 2.** Let utility functions v, v' be such that $v'(a) > v(a)$ for some alternative a, and for all other alternatives $b, c \neq a$, $v(b) \geq v(c)$ iff $v'(b) \geq v'(c)$. That is, the relative order of the other alternatives remains unchanged. Cardinal mechanism μ is non-perverse if for all utility profiles $\vec{u}$ and voters i, $\mu(\vec{u}/_i v')(a) \geq \mu(\vec{u}/_i v)(a)$.

▶ **Definition 3.** Given utility functions v, v' and a permutation π so that $v(a_1) \geq \cdots \geq v(a_m)$ and $v'(a_{\pi(1)}) \geq \cdots \geq v'(a_{\pi(m)})$, cardinal mechanism μ is localized if for every $k \leq m$ such that (i) $\{a_1, \ldots, a_k\} = \{a_{\pi(1)}, \ldots, a_{\pi(k)}\}$ and (ii) $\sum_{i=1}^{k} v(a_i) = \sum_{i=1}^{k} v(a_{\pi(i)})$, the probability mass on these k alternatives remains unchanged. That is, for every utility profile $\vec{u}$ and voter i, $\sum_{i=1}^{k} \mu(\vec{u}/_i v)(a_i) = \sum_{i=1}^{k} \mu(\vec{u}/_i v')(a_{\pi(i)})$.

All missing proofs are given in the appendix.

3 Truthful Mechanisms

In this section, we obtain bounds on the distortion of truthful mechanisms. Disappointingly, but perhaps unsurprisingly, we first show that deterministic mechanisms have distortion $\Theta(mn)$ (Theorem 7). Further, mechanisms that are unanimous – i.e., if there exists an alternative that has maximum utility for each voter, then this alternative must be selected – have distortion $\Omega(m)$ (Theorem 9).[4] In fact, we show that truthfulness is in general expensive – all truthful cardinal mechanisms, even randomized, have distortion $\Omega(\sqrt{m})$ (Theorem 10). The lower bound is disappointing, since it shows that even truthful mechanisms that do not restrict the input format have large distortion. Our lower bound is nearly tight, since there are truthful ordinal mechanisms that have distortion $O(\sqrt{m \log m})$, implying that the loss from restricting the input format does not impose a significant additional burden.

The gap between the upper and lower bounds ($O(\sqrt{m \log m})$ and $\Omega(\sqrt{m})$) remains open. However, the seminal work by Gibbard [16] shows that an ordinal mechanism is truthful iff it is localized and non-perverse. An interesting question is if there exist mechanisms which approach the distortion lower bound, and violate these properties, extended to cardinal mechanisms. We show that indeed such mechanisms exist, and give one such mechanism with distortion $O(\sqrt{m \log m})$, matching the best known upper bound.

We will use the following definition and characterization for our proofs. Recall that we assume that for each voter, no two alternatives have the same utility. Since we assume truthfulness in this section, this extends to their strategies as well.

▶ **Definition 4.** A mechanism is a dictatorship if there exists voter i so that for any strategy profile $\vec{u}$, $\mu(\vec{u}) = \arg\max_{a \in A} u_i(a)$. Voter i is said to decide the mechanism in this case.

[4] A mechanism that is Pareto-optimal must be unanimous, hence the lower bound holds for all Pareto-optimal mechanisms as well.

▶ **Definition 5.** Cardinal mechanism μ is unanimous if whenever there exists an alternative $a^* \in A$ such that $\arg\max_{a \in A} u_i(a) = a^*$ for all voters i, $\mu(\vec{u})$ selects a^* with probability 1.

▶ **Theorem 6** ([18, 10])**.** *A unanimous truthful cardinal mechanism is a randomization over dictatorial mechanisms.*

A similar result was also shown for ordinal mechanisms by Gibbard [16]. We first show tight bounds for deterministic mechanisms.

▶ **Theorem 7.** *Deterministic truthful cardinal mechanisms have distortion* $\Theta(mn)$.

Proof. For the upper bound, consider the mechanism that picks the maximum utility alternative for voter 1. This mechanism has welfare at least $1/m$, while the maximum welfare obtainable is n, giving us the upper bound.

For the lower bound, we first assume that for every $a \in A$, there is some strategy profile for which the mechanism returns a. This is without loss of generality, since if there is some alternative a that is never chosen, then when all agents have utility 1 for a and 0 for all other alternatives, the distortion is infinite. We next show in the following claim that a truthful deterministic cardinal mechanism must be unanimous.

▶ **Claim 8.** *A truthful deterministic cardinal mechanism must be unanimous.*

Proof. Suppose the truthful deterministic mechanism μ is not unanimous. Then for some alternative a, $\vec{u}$ is a utility profile where the maximum utility alternative for every voter is a, but the mechanism returns $b \neq a$. Since a is feasible, there exists $\vec{u'}$ such that $\mu(\vec{u'}) = a$. Let the voters deviate, one by one, from their strategy in $\vec{u}$ to $\vec{u'}$. For some voter i, the mechanism chooses a after the player deviates, and not before. This player then has an incentive to report her utility as in $\vec{u'}$ when her actual utility is as in $\vec{u}$, when the other voters have utilities as in the utility profile before (and also after) the deviation by i. The mechanism thus cannot be truthful. ◀

We can now complete the proof of the theorem. By Theorem 6 and Claim 8, any truthful deterministic cardinal mechanism must be a dictatorship. For such a mechanism, let i be the voter that decides the mechanism. Consider the utility profile where voter i has utility $1/m + \epsilon$ for candidate a and utility $1/m - \epsilon/(m-1)$ for the other alternatives. All the other voters have utility 1 for some candidate $b \neq a$. Then the maximum welfare is about $(n-1)$ while the mechanism obtains welfare $1/m + \epsilon$, giving us the required distortion. ◀

▶ **Theorem 9.** *Unanimous truthful cardinal mechanisms have distortion* $\Omega(m)$.

Proof. Let the unanimous mechanism be μ, by Theorem 6 this must be a randomization over dictatorships. Select $a^* \in A$, and consider the utility profile $\vec{u}$ where all agents have utility $0.5 - \epsilon$ for a^* and $0.5 + \epsilon$ for some other alternative uniformly selected from $A \setminus \{a^*\}$. Then $\mu(\vec{u})$ selects a^* with probability 0, and gets expected welfare $n/2(m-1)$, while alternative a^* has welfare $n/2$ (we assume $\epsilon \to 0$), giving distortion $\Omega(m)$. ◀

We now show a lower bound for all truthful cardinal mechanisms.

▶ **Theorem 10.** *Any truthful cardinal mechanism has distortion* $\Omega(\sqrt{m})$.

Proof. We will assume that $\sqrt{m}$ is an integer and n is divisible by $\sqrt{m}$. This helps simplify the proof but is not required for it to hold. Let μ be a truthful mechanism. Let $\{a_1, a_2, \ldots, a_{\sqrt{m}}\} = A^* \subseteq A$ be a subset of alternatives of size $\sqrt{m}$. Partition the set of agents N into $\sqrt{m}$ sets of equal size $n/\sqrt{m}$, say $N_1, N_2, \ldots, N_{\sqrt{m}}$. Let $n' := n - (n/\sqrt{m})$.

Create a utility profile where for each $i \in \{1, 2, \ldots, \sqrt{m}\}$, the agents in set N_i have utility 1 for a_i and utility 0 for the other alternatives. Call this profile $\vec{u}^0$. For this profile, at least one of the alternatives in A^* is selected by μ with probability $\leq 1/\sqrt{m}$, say alternative $a_{\sqrt{m}}$. Let the probability it gets be $p_0 \leq 1/\sqrt{m}$.

Now, for all voters $i \in N \setminus N_{\sqrt{m}} = \{1, \ldots, n'\}$ we will make the utility uniform among the alternatives in $A \setminus \{a_{\sqrt{m}}\}$, one voter at a time. Formally, let v be a utility function where $a_{\sqrt{m}}$ has utility 0 and all other alternatives have utility $1/(m-1)$. Let utility profile $\vec{u}^i = \vec{u}^{i-1}/_i v$, and let p_i be the probability that alternative $a_{\sqrt{m}}$ gets for profile $\vec{u}^i$, for $i \in \{1, 2, \ldots, n'\}$.

Observe that only the utility for the ith agent changes when we change the strategy profile from $\vec{u}^{i-1}$ to $\vec{u}^i$. Suppose voter i has utility function v. Then the truthfulness condition requires that the probability mass on alternatives other than $a_{\sqrt{m}}$ should be maximum when it reports truthfully, and hence $1 - p_i \geq 1 - p_{i-1}$, or $p_i \leq p_{i-1}$. Thus, $p_{n'} \leq p_0 \leq 1/\sqrt{m}$. Now for the bound on the distortion, for the last utility profile $\vec{u}^{n'}$ when only voters in $N_{\sqrt{m}}$ have positive utility 1 for $a_{\sqrt{m}}$ and all other voters divide their utility equally among the other alternatives, $a_{\sqrt{m}}$ has maximum social welfare equal to $n/\sqrt{m}$, and this is picked with probability at most $1/\sqrt{m}$. The distortion bound follows from simple calculations. ◀

Randomized Hyperspherical Mechanisms

We now describe a truthful cardinal mechanism that has distortion $O(\sqrt{m \log m})$, matching the best known distortion upper bound, and which violates the properties of localization and non-perversity. We first present the mechanism and its analysis, and then show an example for which the mechanism violates these properties. For a dimension m, let $\mathbb{1}$ be the all-ones vector. The standard $(m-1)$-simplex $\{x \in \mathbb{R}^m_{\geq 0} : \|x\|_1 = 1\}$ is denoted $\triangle_m$. Before we describe our mechanism, for a fixed radius $R \geq 0$ and dimension m, consider the following sets:

$$S^1_m(R) = \left\{ p \in \mathbb{R}^m : \left\| p - \frac{1}{m}\mathbb{1} \right\|_2 \leq R,\ \|p\|_1 = 1 \right\}, \quad S^2_m(R) = S^1_m \cap \mathbb{R}^m_{\geq 0}\,.$$

$S^1_m(R)$ is the set of points in $\mathbb{R}^m$ whose coordinates (possibly negative) sum to 1, and are at distance at most R from $(1/m, \ldots, 1/m)$. $S^2_m(R)$ is the set of points that lie in the intersection of the standard simplex with the ball of radius R with center $\frac{1}{m}\mathbb{1}$. Note that both sets are convex. Given $x \in \mathbb{R}^m$, there is a boundary point p that maximizes $p^T x$ in either of these sets, since the objective is linear.

We now describe our mechanism. Let $\vec{u}$ be the given strategy profile. Let μ_1 be the mechanism that picks an alternative with uniform probability and returns it. Let μ_2 be the mechanism that selects a radius R uniformly from the set

$$\Gamma = \left\{ \frac{1}{\sqrt{m(m-1)}}, \frac{2}{\sqrt{m(m-1)}}, \frac{4}{\sqrt{m(m-1)}}, \ldots, \frac{m-1}{\sqrt{m(m-1)}} \right\}$$

and selects a voter i from N with uniform probability. Mechanism μ_2 returns the point $p \in S^2_m(R)$ that maximizes $p^T u_i$. Since p lies in the standard simplex, it is a distribution. Finally, our randomized hyperspherical mechanism runs μ_1 with probability 1/2, and μ_2 with probability 1/2.

Analysis. The truthfulness of the mechanism is evident since for μ_2, the voter i and radius R are chosen independently from the input $\vec{u}$, and we choose $p \in S^2_m(R)$ that maximizes the expected utility $p^T u_i$ for voter i. There are thus two things we need to show: that the

Algorithm 1 Compute-p.

Input: Dimension $m \in \mathbb{Z}_+$, vector $x \in \triangle_m$, radius $R \geq 0$.
Output: Distribution $p \in S_m^2(R)$ that maximizes $p^T x$.

1: $p \leftarrow \frac{1}{m}\mathbb{1} + R\frac{x - \frac{1}{m}\mathbb{1}}{\|x - \frac{1}{m}\mathbb{1}\|_2}$
2: **if** $p \geq 0$ **then**
3: **return** p
4: **else**
5: $x' \leftarrow \frac{x[1:m-1]}{\sum_{i=1}^{m-1} x_i}$, $R' \leftarrow \sqrt{R^2 - \frac{1}{m(m-1)}}$, $m' \leftarrow m-1$.
6: $p' \leftarrow$ Compute-$p(m', x', R')$. **return** $(p'\ 0)$.

distribution p that maximizes $p^T u_i$ over $S_m^2(R)$ can be obtained efficiently, and that the mechanism has distortion $O(\sqrt{m \log m})$.

We first give an algorithm for computing p. Let $x = u_i$ be the utility for the voter chosen. We reindex the alternatives so that $x_1 \geq \cdots \geq x_m$. We use $x[1:k] = (x_1, \ldots, x_k)$ to denote the vector consisting of the first k components of x.

Theorem 11 shows that the algorithm finds the point $p \in S_m^2(R)$ that maximizes $p^T x$, as required. The algorithm first finds the point in $p \in S_m^1(R)$ that maximizes $p^T x$. If p is nonnegative, then $p \in S_m^2(R)$, and this is returned. If not, then we show that in the optimal distribution, it must be the case that $p_m = 0$. In this case, let x' and R' be x and R as modified in Line 5. It can be checked that $S_{m-1}^2(R')$ is the intersection of $S_m^2(R)$ with the hyperplane $p_m = 0$. In this case, we focus on the first $m-1$ coordinates of x, and recursively find the point in p' in $S_{m-1}^2(R')$ that maximizes $p'^T x'$. We show in the proof that the distribution $p = (p'\ 0)$ is the point in $S_m^2(R)$ that maximizes $p^T x$.

▶ **Theorem 11.** *Algorithm 1 correctly returns $p \in S_m^2(R)$ that maximizes $p^T x$.*

Proof. The proof of the theorem follows from these claims.

▶ **Claim 12.** *The point p obtained in Line 1 is the point $q \in S_m^1(R)$ that maximizes $q^T x$.*

Proof. It can be checked from the steps in the algorithm that $\sum_i p_i = 1$, and $\|p - \mathbb{1}/m\|_2 = R$, hence $p \in S_m^1(R)$. Secondly, $q \in S_m^1(R)$ maximizes $q^T x$ iff q maximizes $(q - \mathbb{1}/m)^T x = \|q - \mathbb{1}/m\|_2 \|x\|_2 \cos\theta$, where θ is the angle between $q - \mathbb{1}/m$ and x. The point p is chosen such that $\theta = 0$ and $\|q - \mathbb{1}/m\|_2 = R$, hence it maximizes $q^T x$. ◀

Clearly, if $p \geq 0$, then $p \in S_m^2(R)$ and we are done. Else, since $x_1 > x_2 > \ldots > x_m$, $p_1 \geq p_2 \geq \ldots \geq p_m$ (else permuting the coordinates of p to obtain these inequalities would give us a point in $S_m^1(R)$ with higher value for $p^T x$). Hence suppose $p_m < 0$.

▶ **Claim 13.** *If $p_m < 0$, then there is a point $p' \in S_m^2(R)$ that maximizes $q^T x$ over all points q in $S_m^2(R)$, and has $p'_m = 0$.*

Proof. Let q' be a point in $S_m^2(R)$ that maximizes $q^T x$ over such points, and suppose $q'_m > 0$. Then $x^T q' \leq x^T p$, and since $q'_m > 0$, $p_m < 0$, and the coordinates for each of these vectors is nonincreasing in the indices, there is a point p' on the line joining q' and p that lies in $S_m^1(R)$ (since both these points lie in $S_m^1(R)$, and this set is convex) so that $x^T p' \geq x^T q'$ and $p'_m = 0$, and with coordinates nonincreasing in the indices. Hence $p' \geq 0$, and hence $p' \in S_m^2(R)$, which is the required point. ◀

Now let $\lambda = \sum_{i=1}^{m-1} x_i$, $x' = (x_1, x_2, \ldots, x_{m-1})/\lambda$, and $R' = \sqrt{R^2 - \frac{1}{m(m-1)}}$, as we use in Line 5. Let q' be a point in $S^2_{m-1}(R')$ that maximizes $q^T x[1:m-1]$ over all such points q.

▶ **Claim 14.** *If $p_m < 0$, then $(q'\ 0) \in S^2_m(R)$ maximizes $q^T x$ over all such points q.*

Proof. Since $q' \in S^2_{m-1}(R')$, $(q'\ 0)$ is in $S^2_m(R)$. For the second part, let p' be as obtained in Claim 13, and let $q'' = (p'_1, \ldots, p'_{m-1})$. Since $p'_m = 0$, $q'' \in S^2_{m-1}(R')$. Then $\lambda {q'}^T x' = (q'\ 0)^T x$, and similarly $\lambda {q''}^T x' = (q''\ 0)^T x = {p'}^T x$. Suppose for a contradiction that $(q'\ 0)^T x = \lambda {q'}^T x' < {p'}^T x$. Then $\lambda {q'}^T x' < \lambda {q''}^T x'$, which contradicts the optimality of q'. ◀

Thus, if the point $p \geq 0$, then this is a point in $S^2_m(R)$ that maximizes $p^T x$, and is correctly returned. If not, then $p_m < 0$. In this case, by Claim 13, $p'_m = 0$, and by Claim 14 it is sufficient to compute the point $q' \in S^2_{m-1}(R')$ that maximizes $q^T x$ over all such points q, and return the vector $(q'\ 0)$, which is the iterative step in our algorithm as well. ◀

We now show the bound on the distortion.

▶ **Theorem 15.** *The randomized hyperspherical mechanism has distortion $O(\sqrt{m \log m})$.*

Proof. Let $a^* \in A$ be the alternative with maximum social welfare. Observe that mechanism μ_1, that picks an alternative with uniform probability, has expected welfare of n/m. If $\mathrm{sw}(a^*) \leq n\sqrt{\frac{\log m}{m}}$, then since μ_1 is picked w.p. $1/2$, $\mathrm{dist}(\mu) \leq 2\,\mathrm{dist}(\mu_1) \leq 2\sqrt{m \log m}$.

Else, assume μ_2 is the mechanism picked. Let $p(a)$ be the probability that alternative a is picked, and $p_i(a)$ be the probability that agent i and alternative a are picked. Then $p(a) = \sum_{i \in N} p_i(a)$. For any $i \in N$ and corresponding utility function u_i, there is hypersphere with radius R between $\|u_i - \mathbb{1}/m\|_2$ and $\|u_i - \mathbb{1}/m\|/2$. The point p on this hypersphere that maximizes $p^T u_i$ is the point on the line joining $\mathbb{1}/m$ with u_i, which clearly lies in the simplex. Since this point is at least halfway to u_i, the coordinate corresponding to alternative a has value $\lambda u_i(a) + (1-\lambda)\frac{1}{m}$ for $\lambda \geq 1/2$, and hence

$$p_i(a) \geq \frac{1}{n} \frac{1}{\log m} \frac{u_i(a)}{2}$$

and hence $p(a) \geq \sum_i p_i(a) \geq \mathrm{sw}(a)/(2n \log m)$. Since $\mathrm{sw}(a^*) \geq n\sqrt{\frac{\log m}{m}}$, the distortion is

$$\mathrm{dist}(\mu) \leq 2\,\mathrm{dist}(\mu_2) \leq \frac{2\mathrm{sw}(a^*)}{\mathrm{sw}(a^*)p(a^*)} \leq \frac{4n \log m}{sw(a^*)} \leq \frac{4n \log m}{n\sqrt{\log m/m}} = 4\sqrt{m \log m}.$$ ◀

We now show that the randomized hyperspherical mechanism violates the properties of non-perversity and locality. Let there be 3 alternatives and 1 voter. The mechanism randomizes over hyperspheres with radii $1/\sqrt{6}$ and $2/\sqrt{6}$ with probability $1/4$ each, and selects an alternative uniformly with probability $1/2$.

- Perverse. Consider the utility profiles $u = (1/4 - \epsilon, 1/2, 1/4 + \epsilon)$ and $u' = (0, 3/4, 1/4)$, where $\epsilon \to 0^+$. The relative ordering of the alternatives for both the profiles is same. Running the mechanism returns the distributions $p \approx (0.208, 0.584, 0.208)$ and $p' \approx (0.187, 0.579, 0.234)$ for u and u', respectively. The mechanism is perverse, since the probability of the second alternative decreases, despite its utility increasing and the ordering of the alternatives remaining unchanged.

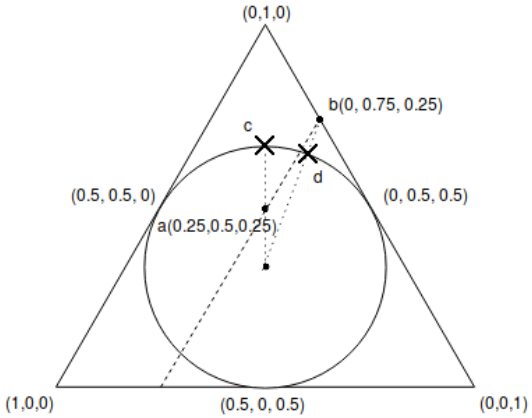


Figure 1 The hyperspherical mechanism for a single sphere of radius $1/\sqrt{6}$, with three alternatives.

- Not localized. Consider the utility profiles $u = (1-\epsilon, \epsilon, 0)$ and $u' = (3/4, 1/4, 0)$, where $\epsilon \to 0^+$. The total utility for the first two alternatives is same for both profiles and is equal to 1. Running the mechanism returns the distributions $p \approx (0.584, 0.208, 0.208)$ and $p' \approx (0.579, 0.234, 0.187)$ for u and u' respectively. The mechanism is not localized, since the total probability given to the first two alternatives is different: 0.792 and 0.813, despite having the same total utility.

Figure 1 shows the projection of the two utility vectors in the first example onto the second hypersphere of radius $1/\sqrt{6}$.

4 Approximately Truthful Mechanisms

Given the strong lower bounds on distortion with truthful mechanisms, we now consider approximately truthful mechanisms. A mechanism is δ-truthful if no voter can increase her expected utility by more than an additive δ by misreporting her utilities. In this case, perhaps surprisingly, we are able to show mechanisms that obtain near-optimal distortion. Our mechanism takes a parameter $\delta \in (0, 1)$ as input. The resulting mechanism is 2δ-truthful, and has distortion that goes to 1 as the number of voters increases. In particular, if $\delta = 25m/n < 1$, then the mechanism has distortion almost 1.01.

The mechanism proceeds as follows. It first elicits the strategy profile $\vec{u}$ of the voters. For alternative $j \in [m]$, define $s_j = \mathrm{sw}(j) = \sum_{i \in N} u_i(j)$. Assume (or re-index the set of alternatives) that $s_1 \geq s_2 \geq \cdots \geq s_m$. Define

$$\lambda := \max \left\{ k \in [m] : \sum_{i=1}^{k} (s_i - s_k) < \frac{1}{\delta} \right\}. \tag{1}$$

Note that $\lambda \geq 1$. Then the mechanism returns the probability distribution defined as follows:

$$p_k = \begin{cases} \frac{1}{\lambda}\left(1 - \delta \sum_{i=1}^{\lambda} (s_i - s_k)\right) & \text{for } k \leq \lambda \\ 0 & \text{for } k > \lambda \end{cases} \tag{2}$$

Since $\sum_{k=1}^{\lambda} \sum_{i=1}^{\lambda} (s_i - s_k) = 0$, the sum of probabilities $\sum_{i=1}^{m} p_i = \sum_{i=1}^{\lambda} p_i = 1$. Further,

for all $k \leq \lambda$ the sum

$$\sum_{i=1}^{\lambda}(s_i - s_k) \;=\; \sum_{i=1}^{k}(s_i - s_k) + \sum_{i=k+1}^{\lambda}(s_i - s_k) \;\leq\; \sum_{i=1}^{k}(s_i - s_k) \;\leq\; 1/\delta\,,$$

where the first inequality is by the indexing of the alternatives and the second is by definition of λ. Hence all the probabilities are non-negative. It can also be shown by a quick calculation that the probability distribution returned by the mechanism can also be written as:

$$p_k = \begin{cases} p_1 - \delta(s_1 - s_k) & \text{for } k \leq \lambda \\ 0 & \text{for } k > \lambda \end{cases} \tag{3}$$

with p_1 chosen so that the sum of the probabilities is 1. We now show the required properties for this mechanism.

▶ **Theorem 16.** *Given parameter $\delta \in (0,1)$, the described mechanism has distortion $< \frac{s_1}{s_1 - \frac{1}{4\delta}}$.*

Proof. Clearly, the maximum welfare obtainable is s_1. The expected welfare obtained by the mechanism is $\sum_{j=1}^{m} p_j s_j$, and replacing for s_j from (3), this gives us $s_1 - (p_1 - \sum_{j=1}^{\lambda} p_j^2)/\delta$. Optimizing over the p_j's, we find that the worst social welfare is obtained when $p_1 = (1 + 1/\lambda)/2$, $p_j = 1/(2\lambda)$ for $2 \leq j \leq \lambda$, and the expected welfare in this case is at least $s_1 - (1/4\delta)$. The bound on the distortion follows immediately. ◀

▶ **Theorem 17.** *Given parameter $\delta \in (0,1)$, the described mechanism is 2δ-truthful.*

Proof. We show the following stronger property: give two strategy profiles $\vec{u}$, $\vec{u'}$, let $\vec{s} := (\sum_{i \in N} u_i(a))_{a \in A}$ and $\vec{s'} := (\sum_{i \in N} u'_i(a))_{a \in A}$ be the respective social welfare vectors. Let $\alpha := \|\vec{s} - \vec{s'}\|_1$ be the L_1 distance between the two welfare vectors. Then the distributions returned by the mechanism given inputs $\vec{u}$ and $\vec{u}'$ differs in any component by at most $\alpha\delta$. The theorem then follows, since by deviating, a single player can change the total welfare by at most 2, and hence the distribution changes in any component by at most 2δ. We first state the following claim, which states that if λ remains unchanged for $\vec{s}$ and $\vec{s'}$, then the property described holds. Let p and p' be the probability distributions returned by the mechanism for $\vec{u}$ and $\vec{u'}$ (with welfare vectors $\vec{s}$ and $\vec{s'}$ respectively). The proof follows immediately from (2).

▶ **Claim 18.** *If λ is the same for $\vec{s}$ and $\vec{s'}$, then the distributions p and p' differ in each component by at most $\delta\|\vec{s} - \vec{s'}\|_1$.*

The next claim shows that if there exists an index k so that $\sum_{i=1}^{k}(s_i - s_k) = \frac{1}{\delta}$, then including this in λ does not change the distribution. Another way of viewing the claim is that it shows that the probability distribution changes continuously with s. In particular, the strict inequality in the definition of λ can be replaced by an inequality without changing the distribution.

▶ **Claim 19.** *Given a strategy profile $\vec{u}$, let $\vec{s} := \sum_{i \in N} u_i$. Let λ be as defined previously, and define λ' as any index k so that $\sum_{i=1}^{k}(s_i - s_k) = \frac{1}{\delta}$ (if it exists). Then the distribution p is unchanged if we replace λ by λ' in (2).*

Proof. Assume λ' exists, else the claim is trivially true. Let p be as defined in (2), and p' be the distribution obtained from (2) with λ replaced by λ'. By definition, $\lambda' > \lambda$. Let $r := \lambda' - \lambda$. Then since $1/\delta = \sum_{i=1}^{\lambda+k}(s_i - s_{\lambda+k})$ for all $k \in \{1, \ldots, r\}$, it must be true that

$s_{\lambda+1} = \cdots = s_{\lambda'}$. Hence, it must also be true that $\sum_{i=1}^{\lambda}(s_i - s_{\lambda'}) = \sum_{i=1}^{\lambda'}(s_i - s_{\lambda'})$. We will use this last equality later in the proof. For any $k \leq \lambda'$, we get the probability distribution

$$\begin{aligned} p'_k &= \frac{1}{\lambda'}\left(1 - \delta\sum_{i=1}^{\lambda'}(s_i - s_k)\right) \\ &= \frac{1}{\lambda'}\left(1 - \delta\sum_{i=1}^{\lambda'}(s_i - s_{\lambda'}) - \delta\lambda'(s_{\lambda'} - s_k)\right) \\ &= \frac{1}{\lambda'}\left(\delta\lambda'(s_k - s_{\lambda'})\right) = \delta(s_k - s_{\lambda'}) \end{aligned}$$

Note that (i) for $\lambda < k \leq \lambda'$, since $s_k = s_{\lambda'}$ as discussed above, $p'_k = 0 = p_k$, and similarly for $k > \lambda'$, by definition $p'_k = 0 = p_k$. Hence it remains to show that for $k \leq \lambda$, $p_k = \delta(s_k - s_{\lambda'})$. Simple calculations yield that for $k \leq \lambda$,

$$p_k - \delta(s_k - s_{\lambda'}) = \frac{1}{\lambda}\left(1 - \delta\sum_{i=1}^{\lambda}(s_i - s_{\lambda'})\right) = 0\,,$$

where the last equality follows from the discussion in the first paragraph. ◀

We now complete the proof of the theorem. Instead of the strategy profiles, we consider directly the resulting welfare vectors. Consider the straight line from $\vec{s}$ to $\vec{s'}$. Let λ and λ' be defined as in (1) for s and s' respectively. If $\lambda = \lambda'$, then by Claim 18, the theorem holds. Suppose instead that $\lambda' = \lambda + r$, for some $r \geq 1$. Let $\vec{s^0} = \vec{s}$ and $\vec{s^r} = \vec{s'}$. We segment the path from $\vec{s}$ to $\vec{s'}$ into r segments $[\vec{s^0}, \vec{s^1}), [\vec{s^1}, \vec{s^2}), \ldots, [\vec{s^{r-1}}, \vec{s^r}]$ so that (i) λ remains unchanged at each point within a segment, and (ii) at the ith breakpoint $\vec{s^i}$, $1/\delta = \sum_{j=1}^{\lambda+i}(s_j - s_{\lambda+i})$. It follows from the previous claims that for any component, the change in probability between $\vec{s}$ and $\vec{s'}$ is at most $\delta\sum_{i=1}^{r}\|\vec{s^i} - \vec{s^{i-1}}\|_1 = \delta\|\vec{s'} - \vec{s}\|_1$, as required to complete the proof. ◀

5 Convergence and Price of Anarchy in Iterative Voting

In this section, we focus on the deterministic mechanism μ that given a strategy profile $\vec{u'}$, chooses the alternative that maximizes the reported social welfare $\sum_{i\in N} u'_i(a)$. The alternatives are indexed, and ties are resolved in favour of the alternative with lower index. All results are presented for this mechanism. It is easily seen that even for two voters, this mechanism is not truthful. Hence we focus on the stable outcomes of strategic voting, in particular strategy profiles that are at a pure Nash equilibrium. As before, we are concerned with the social welfare of the alternative chosen by the mechanism at equilibrium. The distortion in this case is equivalent to the Price of Anarchy, and we refer to it as such here. The PoA is in general unbounded, and so are certain refinements. Instead, we consider equilibria which arise as a result of iterative voting dynamics, when starting from the initial truthful utility profile, a voter deviates at each step in a manner that improves her utility, until the voters reach an equilibrium. In contrast to truthful mechanisms, we show a strong positive result. We show that a particular and natural class of iterative voting dynamics converges quickly to an equilibrium. Further, the price of anarchy for the class of equilibria thus obtained is close to 1, as the number of voters increases. We note that while previous papers have studied either the convergence of iterative voting (e.g., [19, 22, 25]) or the PoA obtained for mechanisms such as plurality and veto (e.g., [9]), ours is the first to obtain results for the PoA of outcomes obtained by iterative voting for a natural cardinal mechanism.

The PoA over all equilibria is unbounded; consider a simple example where all voters have utility 1 for candidate a and 0 for candidate b, but all choose the strategy with utility 1 for b. This is clearly an odd example, and there seem to be two possible remedies. Firstly, we could consider strong equilibria, where $\vec{u'}$ is a strong equilibrium if no set (or coalition) of players can deviate to improve the expected utility of all the players in the set. Secondly, we could consider truth-biased agents, who prefer to vote truthfully if it does not reduce their utility (for which positive results are known in some cases, see [20, 23]). Unfortunately, in both cases we show that the PoA continues to be unbounded.

▶ **Theorem 20.** *The PoA with truth-biased agents is unbounded. Further, an equilibrium may not exist.*

▶ **Theorem 21.** *The PoA of strong Nash equilibria is unbounded.*

Given these negative results, we focus on equilibrium outcomes that are obtained as a result of iterative voting. As stated, we fix the mechanism μ that given a strategy profile $\vec{u'}$, chooses the alternative that maximizes the reported social welfare $\sum_{i \in N} u'_i(a)$. The alternatives are indexed, and ties are resolved in favour of the alternative with lower index. We assume that initially, all voters report their utilities truthfully. At each step, a single voter chooses a different strategy that improves her utility. We say that a particular iterative voting dynamics converges if in finite time, the strategy profile is an equilibrium. We are interested in the PoA of equilibria that are obtained as a result of iterative voting.

Again, we show in the appendix that without further restrictions, the PoA for equilibria obtained can be unbounded, even if the deviating player at each step strictly improves her utility. Let us instead consider the iterative voting process where the deviation by the player at each step satisfies the following properties:

(A) The utility of the deviating player must strictly increase after the deviation.

(B) The deviating player can increase the reported utility for a single alternative, and this alternative must be chosen by the mechanism after the deviation.

With these restrictions, the PoA for the class of equilibria obtained is nearly 1.

▶ **Theorem 22.** *The PoA for iterative voting with restrictions (A) and (B) is* $\frac{\max_{a \in A} \mathrm{sw}(a)}{\max_{a \in A} \mathrm{sw}(a) - 2\log_2 m}$.

Proof. Let $\vec{v^t}$ be the strategy profile in the tth time step. Then $\vec{v^0} = \vec{u}$, where $\vec{u}$ is the utility profile for the voters. We define $\mathrm{sw}^t(a) := \sum_{i \in N} u_i^t(a)$ as the welfare of alternative a according to the strategy profile at step t. Then $\mathrm{sw}^0(a) = \mathrm{sw}(a)$ since iterative voting starts with the true utility as strategy, and we index the alternatives so that $\mathrm{sw}(a_1) \geq \mathrm{sw}(a_2) \geq \cdots \geq \mathrm{sw}(a_m)$. In particular, the maximum-welfare candidate is a_1. We say an alternative wins at time t if it maximizes $\mathrm{sw}^t(a)$, and among all such alternatives, has the lowest index.

Fix any $j \in \{2, \ldots, m\}$, and let t be the first time that an alternative a_k with $k \geq j$ wins, hence $\mathrm{sw}(a_j) \geq \mathrm{sw}(a_k)$ by our indexing. Further, since this is the first time that a_k wins, it is also the first time that any voter raises its utility for a_k, and hence $\mathrm{sw}(a_k) \geq \mathrm{sw}^{t-1}(a_k)$. Lastly, since the voter that deviates at time t changes it's utility by at most 1 for any alternative, and a_k wins at time t, it must be true that $\mathrm{sw}^{t-1}(a_k) \geq \max_{r \leq m} \mathrm{sw}^{t-1}(a_r) - 2$. Putting these together,

$$\mathrm{sw}(a_j) \;\geq\; \mathrm{sw}(a_k) \;\geq\; \mathrm{sw}^{t-1}(a_k) \;\geq\; \max_{r \leq m} \mathrm{sw}^{t-1}(a_r) - 2 \;\geq\; \frac{1}{j-1} \sum_{r=1}^{j-1} \mathrm{sw}^{t-1}(a_r) - 2$$

where the last inequality is simply because the maximum of set of numbers is at least its average. Now observe that step t is the first step when an alternative with index at least j

has won, and hence by the dynamics restriction, this is the first step when an alternative with index at least j had its utility increased. Hence $\sum_{r\geq j} \mathrm{sw}^{t-1}(a_r) \leq \sum_{r\geq j} \mathrm{sw}(a_r)$. Since the sum of utilities of a strategy profile is always n, $\sum_{r<j} \mathrm{sw}^{t-1}(a_r) \geq \sum_{r<j} \mathrm{sw}(a_r)$. Plugging this into the previous inequality yields

$$\mathrm{sw}(a_j) \geq \frac{1}{j-1} \sum_{r=1}^{j-1} \mathrm{sw}(a_r) - 2$$

Let a_j be the highest indexed alternative to win during the dynamics. Then the above inequality is valid for all $k \in \{2, \ldots, j\}$, giving us a recurrence relation. To solve this recurrence, we can check that the hypothesis that $\mathrm{sw}(a_j) \geq \mathrm{sw}(a_1) - 2\log_2 j$ is correct. Our proof thus shows the stronger property that if alternative a wins at any time step in the dynamics, then $\mathrm{sw}(a) \geq \mathrm{sw}(a_1) - 2\log_2 m$. ◀

Unfortunately, it turns out that iterative voting even with these restrictions may not converge (Theorem 24, in the Appendix). However, with one further restriction on the allowable deviations, we can prove convergence.

(C) The deviating player can decrease the reported utility for a single alternative, and this alternative must be chosen by the mechanism before the deviation.

It is not hard to see that the number of steps required for convergence depends upon the least value by which a voter can change her score. As an example, consider two alternatives and two voters with utilities $(0.5 - \epsilon, 0.5 + \epsilon)$ and $(0.5 + \epsilon, 0.5 - \epsilon)$. Let δ be the least value by which an voter can change her utility. Then each time a voter increases the reported utility of her preferred alternative by δ, the alternative chosen by the mechanism changes, and hence convergence takes $\Omega(1/\delta)$ steps. In fact, a convergence bound of $O(mn/\delta)$ for iterative voting with restrictions (A), (B) and (C) can easily be shown, by observing that in each move a voter shifts her stated utility from a less preferred alternative to a more preferred alternative by at least δ. A voter can thus move at most $\frac{m}{\delta}$ times, and hence after $\frac{mn}{\delta}$ steps the iterative voting process must reach a Nash equilibrium.

We can obtain even better convergence bounds for iterative voting, where apart from the initial votes, in all subsequent strategies of a voter, exactly one alternative is given utility 1. These subsequent votes are then plurality votes, for which Meir et al. [20] show a bound of $O(mn)$ on the convergence. Hence every $O(mn)$ steps a new voter must change to plurality voting from her initial utility, and hence this iterative voting process must converge in $O(mn^2)$ steps. If the dynamics also satisfies restrictions (A) and (B), then the equilibrium obtained has PoA as shown in Theorem 22.

References

1 Salvador Barbera. Nice decision schemes. In *Decision theory and social ethics*, pages 101–117. Springer, 1978.

2 Salvador Barbera. An introduction to strategy-proof social choice functions. *Soc Choice Welfare 18*, pages 619–653, 2001.

3 Salvador Barbera, Anna Bogomolnaia, and Hans Van Der Stel. Strategy-proof probabilistic rules for expected utility maximizers. *Mathematical Social Sciences*, 35(2):89–103, 1998.

4 Gerdus Benade, Swaprava Nath, Ariel D. Procaccia, and Nisarg Shah. Preference Elicitation For Participatory Budgeting. In *Association for Advancement of Artificial Intelligence (AAAI), February 4 - 9, 2017, San Francisco, California, USA*, 2017. Forthcoming.

5 Umang Bhaskar, Varsha Dani, and Abheek Ghosh. Truthful and Near-Optimal Mechanisms for Welfare Maximization in Multi-Winner Elections. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*, 2018.

6 Eleanor Birrell and Rafael Pass. Approximately strategy-proof voting. In *IJCAI*, pages 67–72, 2011.

7 Craig Boutilier, Ioannis Caragiannis, Simi Haber, Tyler Lu, Ariel D. Procaccia, and Or Sheffet. Optimal social choice functions: A utilitarian view. *Artif. Intell.*, 227:190–213, 2015. doi:10.1016/j.artint.2015.06.003.

8 Felix Brandt, Vincent Conitzer, Ulle Endriss, Ariel D Procaccia, and Jérôme Lang. *Handbook of computational social choice*. Cambridge University Press, 2016.

9 Simina Brânzei, Ioannis Caragiannis, Jamie Morgenstern, and Ariel D Procaccia. How bad is selfish voting? In *AAAI*, volume 13, pages 138–144, 2013.

10 Bhaskar Dutta, Hans Peters, and Arunava Sen. Strategy-proof cardinal decision schemes. *Social Choice Welfare*, 28:163–179, 2007.

11 Edith Elkind, Evangelos Markakis, Svetlana Obraztsova, and Piotr Skowron. Equilibria of plurality voting: Lazy and truth-biased voters. In *International Symposium on Algorithmic Game Theory*, pages 110–122. Springer, 2015.

12 Uriel Feige and Moshe Tennenholtz. Responsive Lotteries. In *Algorithmic Game Theory - Third International Symposium, SAGT 2010, Athens, Greece, October 18-20, 2010. Proceedings*, pages 150–161, 2010.

13 Aris Filos-Ratsikas and Peter Bro Miltersen. Truthful approximations to range voting. *International Conference on Web and Internet Economics*, pages 175–188, 2014.

14 Sumit Ghosh, Manisha Mundhe, Karina Hernandez, and Sandip Sen. Voting for movies: the anatomy of a recommender system. In *Proceedings of the third annual conference on Autonomous Agents*, pages 434–435. ACM, 1999.

15 Allan Gibbard. Manipulation of voting schemes: a general result. *Econometrica: journal of the Econometric Society*, pages 587–601, 1973.

16 Allan Gibbard. Manipulation of schemes that mix voting with chance. *Econometrica: Journal of the Econometric Society*, pages 665–681, 1977.

17 Claude Hillinger. The case for utilitarian voting. *SSRN*, 2005. doi:10.2139/ssrn.732285.

18 Aanund Hylland. Strategy proofness of voting procedures with lotteries as outcomes and infinite sets of strategies. *Unpublished paper, University of Oslo.[341, 349]*, 1980.

19 Omer Lev and Jeffrey S Rosenschein. Convergence of iterative voting. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 611–618. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

20 Reshef Meir, Maria Polukarov, Jeffrey S Rosenschein, and Nicholas R Jennings. Convergence to Equilibria in Plurality Voting. In *AAAI*, volume 10, pages 823–828, 2010.

21 Shasikanta Nandeibam. An alternative proof of Gibbard's random dictatorship result. *Social Choice and Welfare*, 15(4):509–519, 1998.

22 Svetlana Obraztsova, Evangelos Markakis, Maria Polukarov, Zinovi Rabinovich, and Nicholas R. Jennings. On the Convergence of Iterative Voting: How Restrictive Should Restricted Dynamics Be? In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 993–999, 2015.

23 Svetlana Obraztsova, Evangelos Markakis, and David RM Thompson. Plurality voting with truth-biased agents. In *International Symposium on Algorithmic Game Theory*, pages 26–37. Springer, 2013.

24 Ariel D Procaccia and Jeffrey S Rosenschein. The distortion of cardinal preferences in voting. In *International Workshop on Cooperative Information Agents*, pages 317–331. Springer, 2006.

25 Zinovi Rabinovich, Svetlana Obraztsova, Omer Lev, Evangelos Markakis, and Jeffrey S. Rosenschein. Analysis of Equilibria in Iterative Voting Schemes. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 1007–1013, 2015.

26 Reyhaneh Reyhani and Mark Wilson. Best-reply dynamics for scoring rules. In *20th European Conference on Artificial Intelligence*. IOS Press, 2012.

27 Mark Allen Satterthwaite. Strategy-proofness and Arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of economic theory*, 10(2):187–217, 1975.

6 Appendix

In the appendix, we give the missing proofs from Section 5. We first prove that the PoA with truth-biased voters is unbounded, and in fact a pure Nash equilibrium may not exist.

Proof of Theorem 20. We will give an example for which the price of anarchy $\to \infty$. There are 3 alternatives $\{a, b, c\}$ and $2n+1$ agents. The utility profile is:

Agents	a	b	c
$1, \dots, n$	1	0	0
$n+1, \dots, 2n+1$	0	$1-\epsilon$	ϵ
Social Welfare	n	$(n+1)(1-\epsilon)$	$(n+1)\epsilon$

The strategy profile in Nash equilibrium:

Agents	a	b	c
$1, \dots, n$	1	0	0
$n+1$	0	$1-\epsilon$	ϵ
$n+2, \dots, 2n+1$	0	0	1
Total	n	$1-\epsilon$	$n+\epsilon$

The winner is alternative c. Observe that no agent can increase her utility by deviating. The PoA is $\frac{(n+1)(1-\epsilon)}{(n+1)\epsilon} = \frac{1-\epsilon}{\epsilon} \to \infty$ as $\epsilon \to 0$.

We now give an example for which there is no PNE. There are 2 alternatives $\{a, b\}$ and 2 agents. The mechanism is deterministic and ties are broken in favour of a. The utility profile is given below:

Agents	a	b
1	0.75	0.25
2	0.25	0.75

None of the three exhaustive cases below allow an equilibrium.

- If $u_1(a) < u_2(b)$ then b is the winner. For voter 1, this is not an equilibrium because she can increase her utility for a and make a win.
- If $1 > u_1(a) \geq u_2(b)$ then a is the winner. Now, for voter 2 this is not an equilibrium because she can increase her utility for b and make b win.

- If $u_1(a) = 1$ then a is the winner. Voter 2 cannot make b win, and as voters are truth-biased she will give her true input. Now, voter 1 is also truth-biased, and will give her true input, and a will remain the winner. This puts us in the second case above. ◀

We now show that the PoA even restricted to strong Nash equilibria is unbounded.

Proof of Theorem 21. We will give an example for which the price of anarchy $\to \infty$. There are 4 alternatives and $4n-1$ agents. Ties are broken lexicographically. The utility profile is given below:

Agents →	a	b	c	d
$1, \ldots, 2n-1$	1	0	0	0
$2n, \ldots, 3n-1$	0	$1-\epsilon$	0	ϵ
$3n, \ldots, 4n-1$	0	0	$1-\epsilon$	ϵ
Social Welfare	$2n-1$	$n(1-\epsilon)$	$n(1-\epsilon)$	$2n\epsilon$

The strong pure Nash equilibrium strategy profile is:

Agents	a	b	c	d
$1, \ldots, 2n-1$	1	0	0	0
$2n, \ldots, 3n-1$	0	0	0	1
$3n, \ldots, 4n-2$	0	0	0	1
$4n-1$	0	0	$1-\epsilon$	ϵ
Total	$2n-1$	0	$1-\epsilon$	$2n-1+\epsilon$

The winner is alternative d. Observe that no agent can increase her utility by deviating. The PoA is $\frac{2n-1}{2n\epsilon} \to \infty$ as $\epsilon \to 0$. ◀

Iterative Voting

We give the proofs related to the PoA and convergence of iterative voting dynamics. We first restate the restrictions on dynamics from the main paper.

(A) The utility of the deviating player must strictly increase after the deviation.
(B) The deviating player can increase the reported utility for a single alternative, and this alternative must be chosen by the mechanism after the deviation.
(C) The deviating player can decrease the reported utility for a single alternative, and this alternative must be chosen by the mechanism before the deviation.

We first show that just restriction (A) is insufficient to ensure bounded PoA.

▶ **Theorem 23.** *The price of anarchy of iterative voting with restriction (A) is unbounded.*

Proof. There are 5 alternatives $\{a, b, c, d, e\}$, and $2n+4$ agents. Ties broken lexicographically. $\epsilon \to 0$ is much smaller than $1/n$. The utility profile is as given.

Agents	a	b	c	d	e
A: $1, \ldots, n$	0.5	$0.5-\epsilon$	ϵ	0	0
B: $n+1, \ldots, 2n$	$0.5-\epsilon$	0.5	0	ϵ	0
$2n+1, 2n+2$	0	0	ϵ	0	$1-\epsilon$
$2n+3, 2n+4$	0	0	0	ϵ	$1-\epsilon$
Social Welfare	$n-n\epsilon$	$n-n\epsilon$	$(n+2)\epsilon$	$(n+2)\epsilon$	$4-4\epsilon$

Initially, alternative a is the winner. But an agent in B, say agent $n+1$, can deviate and make b win. Say the agent deviates to: $0.5+\epsilon$ to b, $0.5-\epsilon$ to d, and 0 to others. Now, an agent in a, say agent 1, deviates and gives input: $0.5+\epsilon$ to a, $0.5-\epsilon$ to c, and 0 to others. Now, alternative a wins again. Repeating this for all agents in A and B, after n steps the strategy profile is given below. Currently, a is the winner.

Agents	a	b	c	d	e
$1,\dots,n$	$0.5+\epsilon$	0	$0.5-\epsilon$	0	0
$n+1,\dots,2n$	0	$0.5+\epsilon$	0	$0.5-\epsilon$	0
$2n+1, 2n+2$	0	0	ϵ	0	$1-\epsilon$
$2n+3, 2n+4$	0	0	0	ϵ	$1-\epsilon$
Total	$n(0.5+\epsilon)$	$n(0.5+\epsilon)$	$n(0.5-\epsilon)+2\epsilon$	$n(0.5-\epsilon)+2\epsilon$	$4-4\epsilon$

Now, the agent $2n+3$ makes a move and sets d's score to 1, then the agent $2n+1$ makes a move and sets c's score to 1. The same moves are then repeated by agents $2n+4$ and $2n+2$. This makes c the current winner. The strategy profile after these deviations is given below.

Agents	a	b	c	d	e
$1,\dots,n$	$0.5+\epsilon$	0	$0.5-\epsilon$	0	0
$n+1,\dots,2n$	0	$0.5+\epsilon$	0	$0.5-\epsilon$	0
$2n+1, 2n+2$	0	0	1	0	0
$2n+3, 2n+4$	0	0	0	1	0
Total	$n(0.5+\epsilon)$	$n(0.5+\epsilon)$	$n(0.5-\epsilon)+2$	$n(0.5-\epsilon)+2$	0

Alternatives a and b cannot win the election by a deviation by the agents in A or B, so the agents in A and B start competing for c and d to reach the final equilibrium strategy profile given below. Alternative c is the final winner.

Agents	a	b	c	d	e
$1,\dots,n$	0	0	1	0	0
$n+1,\dots,2n$	0	0	0	1	0
$2n+1, 2n+2$	0	0	1	0	0
$2n+3, 2n+4$	0	0	0	1	0
Total	0	0	$n+2$	$n+2$	0

No agent has a move that can increase her utility, and hence this is an equilibrium. The PoA is $\frac{n(1-\epsilon)}{(n+2)\epsilon} \to \infty$. Observe that this proof works even for best response iterative voting dynamics: The deviating agent plays a move that makes the most preferred alternative win the election, among the alternatives that can win the election after a move by the agent. ◀

With just restriction (A), as shown, we obtain unbounded PoA. Theorem 22 shows that with both (A) and (B) we get a near-optimal bound on the PoA. We now show, however, that the two restrictions (A) and (B) that ensured small PoA are not enough to ensure convergence to a Nash equilibrium.

▶ **Theorem 24.** *Iterative voting with restrictions (A) and (B) may not converge to a PNE.*

Proof. We will give a utility profile with 4 alternatives $A = \{a, b, c, d\}$ and 4 agents $N = \{1, 2, 3, 4\}$, and a sequence of steps taken by the agents that will create a cycle. For ease of writing, we normalize the utilities to sum up to 6 rather than 1. In the tables below the winner is denoted by $*$. The steps are:

1. Utility profile is

Alternatives →	$a*$	b	c	d
Agent 1	1	1	0	4
Agent 2	2	2	0	2
Agent 3	0	2	3	1
Agent 4	4	0	2	0

2. Agent 1 plays. Utility increases from 1 to 4.

Alternatives →	a	b	c	$d*$
Agent 1	0	0	0	6
Agent 2	2	2	0	2
Agent 3	0	2	3	1
Agent 4	4	0	2	0

3. Agent 4 plays. Utility increases from 0 to 2.

Alternatives →	a	b	$c*$	d
Agent 1	0	0	0	6
Agent 2	2	2	0	2
Agent 3	0	2	3	1
Agent 4	0	0	6	0

4. Agent 1 plays. Utility increases from 0 to 1.

Alternatives →	a	$b*$	c	d
Agent 1	0	5	0	1
Agent 2	2	2	0	2
Agent 3	0	2	3	1
Agent 4	0	0	6	0

5. Agent 3 plays. Utility increases from 2 to 3.

Alternatives →	a	b	$c*$	d
Agent 1	0	5	0	1
Agent 2	2	2	0	2
Agent 3	0	0	5	1
Agent 4	0	0	6	0

6. Agent 4 plays. Utility increases from 2 to 4.

Alternatives →	$a*$	b	c	d
Agent 1	0	5	0	1
Agent 2	2	2	0	2
Agent 3	0	0	5	1
Agent 4	6	0	0	0

7. Agent 3 plays. Utility increases from 0 to 2.

Alternatives →	a	$b*$	c	d
Agent 1	0	5	0	1
Agent 2	2	2	0	2
Agent 3	0	2	3	1
Agent 4	6	0	0	0

8. Agent 1 plays. Utility increases from 1 to 4.

Alternatives →	a	b	c	$d*$
Agent 1	0	0	0	6
Agent 2	2	2	0	2
Agent 3	0	2	3	1
Agent 4	6	0	0	0

9. Agent 4 plays. Utility increases from 0 to 2.

Alternatives →	a	b	$c*$	d
Agent 1	0	0	0	6
Agent 2	2	2	0	2
Agent 3	0	2	3	1
Agent 4	0	0	6	0

Observe that the strategy profile in 3 and 9 are same, giving us a cycle. This proof also works for best-response dynamics. ◀

Symbolic Approximation of Weighted Timed Games

Damien Busatto-Gaston
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
damien.busatto-gaston@lis-lab.fr

Benjamin Monmege
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
benjamin.monmege@univ-amu.fr
https://orcid.org/0000-0002-4717-9955

Pierre-Alain Reynier
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
pierre-alain.reynier@univ-amu.fr

Abstract

Weighted timed games are zero-sum games played by two players on a timed automaton equipped with weights, where one player wants to minimise the accumulated weight while reaching a target. Weighted timed games are notoriously difficult and quickly undecidable, even when restricted to non-negative weights. For non-negative weights, the largest class that can be analysed has been introduced by Bouyer, Jaziri and Markey in 2015. Though the value problem is undecidable, the authors show how to approximate the value by considering regions with a refined granularity. In this work, we extend this class to incorporate negative weights, allowing one to model energy for instance, and prove that the value can still be approximated, with the same complexity. In addition, we show that a symbolic algorithm, relying on the paradigm of value iteration, can be used as an approximation schema on this class.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory, Theory of computation → Timed and hybrid models, Theory of computation → Quantitative automata

Keywords and phrases Weighted timed games, Real-time systems, Game theory, Approximation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.28

Funding This work has been funded by the DeLTA project (ANR-16-CE40-0007).

1 Introduction

The design of programs verifying some real-time specifications is a notoriously difficult problem, because such programs must take care of delicate timing issues, and are difficult to debug a posteriori. One research direction to ease the design of real-time software is to automatise the process. The situation may be modelled into a timed game, played by a *controller* and an antagonistic *environment*: they act, in a turn-based fashion, over a *timed automaton* [2], namely a finite automaton equipped with real-valued variables, called clocks, evolving with a uniform rate. A simple, yet realistic, objective for the controller is to reach a target location. We are thus looking for a *strategy* of the controller, that is a recipe dictating how to play so that the target is reached no matter how the environment plays. Reachability timed games are decidable [4], and EXPTIME-complete [18].

Weighted extensions of these games have been considered in order to measure the quality of the winning strategy for the controller [9, 1]: when the controller has several winning strategies in a given reachability timed game, the quantitative version of the game helps choosing a good one with respect to some metrics. This means that the game now takes place over a *weighted (or priced) timed automaton* [5, 3], where transitions are equipped with weights, and locations with rates of weights (the cost is then proportional to the time spent in this location, with the rate as proportional coefficient). While solving the optimal reachability problem on weighted timed automata has been shown to be PSPACE-complete [6] (i.e. the same complexity as the non-weighted version), weighted timed games are known to be undecidable [12]. This has led to many restrictions in order to regain decidability, the first and most interesting one being the class of strictly non-Zeno cost with only non-negative weights (in transitions and locations) [9]: this hypothesis requires that every execution of the timed automaton that follows a cycle of the region automaton has a weight far from 0 (in interval $[1, +\infty)$, for instance).

Negative weights are crucial when one wants to model energy or other resources that can grow or decrease during the execution of the system to study. In [16], we have recently extended the strictly non-Zeno cost restriction to weighted timed games in the presence of negative weights in transitions and/or locations. We have described there the class of *divergent weighted timed games* where each execution that follows a cycle of the region automaton has a weight far from 0, i.e. in $(-\infty, -1] \cup [1, +\infty)$. We were able to obtain a doubly-exponential-time algorithm to compute the values and almost-optimal strategies, while deciding the divergence of a weighted timed game is PSPACE-complete. These complexity results match the ones that could be obtained in the non-negative case from [9, 1].

The techniques used to obtain the results of [16] cannot be extended if the conditions are slightly relaxed. For instance, if we add the possibility for an execution of the timed automaton following a cycle of the region automaton to have weight *exactly 0*, the decision problem is known to be undecidable [10], even with non-negative weights only. For this extension, in the presence of non-negative weights only, it has been proposed an approximation schema to compute arbitrarily close estimates of the optimal value [10]. To this end, the authors consider regions with a refined granularity so as to control the precision of the approximation. In this work, our contribution is two-fold: first, we extend the class considered in [10] to the presence of negative weights; second, we show that the approximation can be obtained using a symbolic computation, based on the paradigm of value iteration.

More precisely, we define the class of *almost-divergent weighted timed games* where, for each strongly connected component (SCC) of the region automaton, executions following a cycle of this SCC have weights either all in $(-\infty, -1] \cup \{0\}$, or all in $\{0\} \cup [1, +\infty)$. In contrast, the *divergent* condition is equivalent to the same property on the strongly connected components, but without the presence of singleton $\{0\}$. Given an almost-divergent weighted timed game, an initial configuration c and a threshold ε, we compute a value that we guarantee to be ε-close to the optimal value when the play starts from c. Moreover, we prove that deciding if a weighted timed game is almost-divergent is a PSPACE-complete problem.

In order to approximate almost-divergent weighted timed games, we first adapt the approximation schema of [10] to our setting. At the very core of their schema is the notion of *kernels* that collect all cycles of weight exactly 0 in the game. Then, a semi-unfolding of the game (in which kernels are not unfolded) of bounded depth is shown to be equivalent to the original game. Adapting this schema to negative weights requires to address new issues:

- The definition and the approximation of these kernels is much more intricate in our setting (see Sections 4 and 6). Indeed, with only non-negative weights, a cycle of weight 0 only encounters locations and transitions with weight 0. It is no longer the case with

arbitrary weights, both for discrete weights on transitions (that could alternate between weight $+1$ and -1, e.g.) and continuous rates on locations: for this continuous part, this requires to keep track of the real-time dynamics of the game.

- Some configurations may have value $-\infty$. While it is undecidable in general whether a configuration has value $-\infty$, we prove that it is decidable for almost-divergent weighted timed games (see Lemma 9).
- The identification of an adequate bound to define an equivalent semi-unfolding of bounded depth is more difficult in our setting, as having guarantees on weight accumulation is harder (we can lose accumulated weight). We deal with this by evaluating how large the value of a configuration can be, provided it is not infinite. This is presented in Section 5.

We also develop, in Section 7, a more symbolic approximation schema, in the sense that it avoids the a priori refinement of regions. Instead, all computations are performed in a symbolic way using the techniques developed in [1]. This allows to mutualise as much as possible the different computations: comparing these schemas with the evaluation of MDPs or quantitative games like mean-payoff or discounted-payoff, it is the same improvement as when using *value iteration* techniques instead of techniques based on the unfolding of the model into a finite tree which can contain many times the same location.

2 Weighted timed games

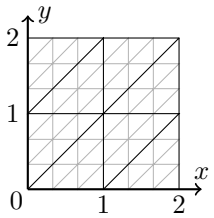
Clocks, guards and regions. We let X be a finite set of variables called clocks. A valuation of clocks is a mapping $\nu\colon X \to \mathbb{R}_{\geqslant 0}$. For a valuation ν, $d \in \mathbb{R}_{\geqslant 0}$ and $Y \subseteq X$, we define the valuation $\nu + d$ as $(\nu + d)(x) = \nu(x) + d$, for all $x \in X$, and the valuation $\nu[Y := 0]$ as $(\nu[Y := 0])(x) = 0$ if $x \in Y$, and $(\nu[Y := 0])(x) = \nu(x)$ otherwise. The valuation $\mathbf{0}$ assigns 0 to every clock. A guard on clocks of X is a conjunction of atomic constraints of the form $x \bowtie c$, where $\bowtie \in \{\leqslant, <, =, >, \geqslant\}$ and $c \in \mathbb{Q}$ (we allow for rational coefficients as we will refine the granularity in the following). Guard $\overline{g}$ is the closed version of a satisfiable guard g where every open constraint $x < c$ or $x > c$ is replaced by its closed version $x \leqslant c$ or $x \geqslant c$. A valuation $\nu\colon X \to \mathbb{R}_{\geqslant 0}$ satisfies an atomic constraint $x \bowtie c$ if $\nu(x) \bowtie c$. The satisfaction relation is extended to all guards g naturally, and denoted by $\nu \models g$. We let $\mathsf{Guards}(X)$ denote the set of guards over X. We rely on the crucial notion of regions, as introduced in the seminal work on timed automata [2]: intuitively, a region is a set of valuations that are all time-abstract bisimilar. We will need some refinement of regions, with respect to a granularity $1/N$, with $N \in \mathbb{N}$. Formally, with respect to the set X of clocks and a constant M, a $1/N$-region r is a subset of valuations characterised by the vector $(\iota_x)_{x\in X} = (\min(MN, \lfloor \nu(x)N \rfloor))_{x\in X} \in [0, MN]^X$ and the order of fractional parts of $\nu(x)N$, given as a partition $X = X_0 \uplus X_1 \uplus \cdots \uplus X_m$ of clocks: a valuation ν is in this $1/N$-region r if

(i) $\lfloor \nu(x)N \rfloor = \iota_x$, for all clocks $x \in X$;

(ii) $\nu(x) = 0$ for all $x \in X_0$;

(iii) all clocks $x \in X_i$ satisfy that $\nu(x)N$ have the same fractional part, for all $1 \leqslant i \leqslant m$.

We denote by $\mathsf{Reg}_N(X, M)$ the set of $1/N$-regions, and we write $\mathsf{Reg}(X, M)$ as a shorthand for $\mathsf{Reg}_1(X, M)$. We recover the traditional notion of region for $N = 1$. E.g., the figure below depicts regions $\mathsf{Reg}(\{x, y\}, 2)$ as well as their refinement $\mathsf{Reg}_3(\{x, y\}, 2)$.



For any integer guard g, either all valuations of a given $1/N$-region satisfy g, or none of them do. A $1/N$-region r' is said to be a time successor of the $1/N$-region r if there exist $\nu \in r$, $\nu' \in r'$, and $d > 0$ such that $\nu' = \nu + d$. Moreover, for $Y \subseteq X$, we let $r[Y := 0]$ be the $1/N$-region where clocks of Y are reset.

Weighted timed games. A weighted timed game (WTG) is then a tuple $\mathcal{G} = \langle L = L_{\mathsf{Min}} \uplus L_{\mathsf{Max}}, \Delta, \mathsf{wt}, L_T, \mathsf{wt}_T \rangle$ where L_{Min} and L_{Max} are finite disjoint subsets of locations belonging to Min and Max, respectively, $\Delta \subseteq L \times \mathsf{Guards}(X) \times 2^X \times L$ is a finite set of transitions, $\mathsf{wt} \colon \Delta \uplus L \to \mathbb{Z}$ is the weight function, associating an integer weight with each transition and location, $L_T \subseteq L_{\mathsf{Min}}$ is a subset of target locations for player Min, and $\mathsf{wt}_T \colon L_T \times \mathbb{R}^X_{\geqslant 0} \to \mathbb{R}_\infty$ is a function mapping each target location and valuation of the clocks to a final weight of $\mathbb{R}_\infty = \mathbb{R} \uplus \{-\infty, +\infty\}$ (possibly 0, $+\infty$, or $-\infty$). The addition of target weights is not standard, but we will use it in the process of solving those games: anyway, it is possible to simply map each target location to the weight 0, allowing us to recover the standard definition. Without loss of generality, we suppose the absence of deadlocks except on target locations, i.e. for each location $\ell \in L \backslash L_T$ and valuation ν, there exists $(\ell, g, Y, \ell') \in \Delta$ such that $\nu \models g$, and no transitions start in L_T.

The semantics of a WTG $\mathcal{G}$ is defined in terms of a game played on an infinite transition system whose vertices are configurations of the WTG. A configuration is a pair (ℓ, ν) with a location and a valuation of the clocks. Configurations are split into players according to the location. A configuration is final if its location is a target location of L_T. The alphabet of the transition system is given by $\mathbb{R}_{\geqslant 0} \times \Delta$ and will encode the delay that a player wants to spend in the current location, before firing a certain transition. For every delay $d \in \mathbb{R}_{\geqslant 0}$, transition $\delta = (\ell, g, Y, \ell') \in \Delta$ and valuation ν, there is an edge $(\ell, \nu) \xrightarrow{d,\delta} (\ell', \nu')$ if $\nu + d \models g$ and $\nu' = (\nu + d)[Y := 0]$. The weight of such an edge e is given by $d \times \mathsf{wt}(\ell) + \mathsf{wt}(\delta)$. An example is depicted on Figure 1.

A *finite play* is a finite sequence of consecutive edges $\rho = (\ell_0, \nu_0) \xrightarrow{d_0,\delta_0} (\ell_1, \nu_1) \xrightarrow{d_1,\delta_1} \cdots (\ell_k, \nu_k)$. We denote by $|\rho|$ the length k of ρ. The concatenation of two finite plays ρ_1 and ρ_2, such that ρ_1 ends in the same configuration as ρ_2 starts, is denoted by $\rho_1\rho_2$. We let $\mathsf{FPlays}_{\mathcal{G}}$ be the set of all finite plays in $\mathcal{G}$, whereas $\mathsf{FPlays}^{\mathsf{Min}}_{\mathcal{G}}$ (resp. $\mathsf{FPlays}^{\mathsf{Max}}_{\mathcal{G}}$) denote the finite plays that end in a configuration of Min (resp. Max). A *play* is then a maximal sequence of consecutive edges (it is either infinite or it reaches L_T).

A *strategy* for Min (resp. Max) is a mapping $\sigma \colon \mathsf{FPlays}^{\mathsf{Min}}_{\mathcal{G}} \to \mathbb{R}_{\geqslant 0} \times \Delta$ (resp. $\sigma \colon \mathsf{FPlays}^{\mathsf{Max}}_{\mathcal{G}} \to \mathbb{R}_{\geqslant 0} \times \Delta$) such that for all finite plays $\rho \in \mathsf{FPlays}^{\mathsf{Min}}_{\mathcal{G}}$ (resp. $\rho \in \mathsf{FPlays}^{\mathsf{Max}}_{\mathcal{G}}$) ending in non-target configuration (ℓ, ν), there exists an edge $(\ell, \nu) \xrightarrow{\sigma(\rho)} (\ell', \nu')$. A play or finite play $\rho = (\ell_0, \nu_0) \xrightarrow{d_0,\delta_0} (\ell_1, \nu_1) \xrightarrow{d_1,\delta_1} \cdots$ conforms to a strategy σ of Min (resp. Max) if for all k such that (ℓ_k, ν_k) belongs to Min (resp. Max), we have that $(d_k, \delta_k) = \sigma((\ell_0, \nu_0) \xrightarrow{d_0,\delta_0} \cdots (\ell_k, \nu_k))$. A strategy σ is *memoryless* if for all finite plays ρ, ρ' ending in the same configuration, we have that $\sigma(\rho) = \sigma(\rho')$. For all strategies σ_{Min} and σ_{Max} of players Min and Max, respectively, and for all configurations (ℓ_0, ν_0), we let $\mathsf{play}_{\mathcal{G}}((\ell_0, \nu_0), \sigma_{\mathsf{Max}}, \sigma_{\mathsf{Min}})$ be the outcome of σ_{Max} and σ_{Min}, defined as the only play conforming to σ_{Max} and σ_{Min} and starting in (ℓ_0, ν_0).

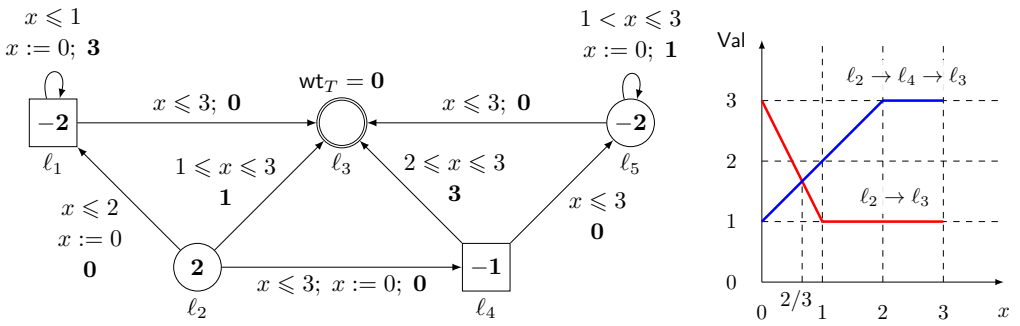


Figure 1 On the left, a weighted timed game. Locations belonging to Min (resp. Max) are depicted by circles (resp. squares). The target location is ℓ_3, whose output weight function is null. It is easy to observe that location ℓ_1 (resp. ℓ_5) has value $+\infty$ (resp. $-\infty$). As a consequence, the value in ℓ_4 is determined by the edge to ℓ_3, and depicted in blue on the right. In location ℓ_2, the value associated with the transition to ℓ_3 is depicted in red, and the value in ℓ_2 is obtained as the minimum of these two curves. Observe the intersection in $x = 2/3$ requiring to refine the regions.

The objective of Min is to reach a target configuration, while minimising the accumulated weight up to the target. Hence, we associate to every finite play $\rho = (\ell_0, \nu_0) \xrightarrow{d_0,\delta_0} (\ell_1, \nu_1) \xrightarrow{d_1,\delta_1} \cdots (\ell_k, \nu_k)$ its cumulated weight, taking into account both discrete and continuous costs: $\mathsf{wt}_\Sigma(\rho) = \sum_{i=0}^{k-1} \mathsf{wt}(\ell_i) \times d_i + \mathsf{wt}(\delta_i)$. Then, the weight of a play ρ, denoted by $\mathsf{wt}_\mathcal{G}(\rho)$, is defined by $+\infty$ if ρ is infinite (does not reach L_T), and $\mathsf{wt}_\Sigma(\rho) + \mathsf{wt}_T(\ell_T, \nu)$ if it ends in (ℓ_T, ν) with $\ell_T \in L_T$. Then, for all locations ℓ and valuation ν, we let $\mathsf{Val}_\mathcal{G}(\ell, \nu)$ be the value of $\mathcal{G}$ in (ℓ, ν), defined as $\mathsf{Val}_\mathcal{G}((\ell, \nu)) = \inf_{\sigma_{\mathsf{Min}}} \sup_{\sigma_{\mathsf{Max}}} \mathsf{wt}_\mathcal{G}(\mathsf{play}((\ell, \nu), \sigma_{\mathsf{Max}}, \sigma_{\mathsf{Min}}))$, where the order of the infimum and supremum does not matter, since WTGs are known to be determined[1]. We say that a strategy $\sigma^\star_{\mathsf{Min}}$ of Min is ε-optimal if, for all (ℓ, ν), and all strategies σ_{Max} of Max, $\mathsf{wt}_\mathcal{G}(\mathsf{play}((\ell, \nu), \sigma_{\mathsf{Max}}, \sigma^\star_{\mathsf{Min}})) \leqslant \mathsf{Val}_\mathcal{G}(\ell, \nu) + \varepsilon$. It is said optimal if this holds for $\varepsilon = 0$. A symmetric definition holds for optimal strategies of Max. If the game is clear from the context, we may drop the index $\mathcal{G}$ from all previous notations.

As usual in related work [1, 9, 10], we assume that the input WTGs have guards where all constants are integers, and all clocks are *bounded*, i.e. there is a constant $M \in \mathbb{N}$ such that every transition of the WTG is equipped with a guard g such that $\nu \models g$ implies $\nu(x) \leqslant M$ for all clocks $x \in X$. We denote by $w^L_{\max}$ (resp. $w^\Delta_{\max}$, $w^e_{\max}$) the maximal weight in absolute values of locations (resp. of transitions, edges) of $\mathcal{G}$, i.e. $w^L_{\max} = \max_{\ell \in L} |\mathsf{wt}(\ell)|$ (resp. $w^\Delta_{\max} = \max_{\delta \in \Delta} |\mathsf{wt}(\delta)|$, $w^e_{\max} = M w^L_{\max} + w^\Delta_{\max}$). We also assume that the output weight functions are piecewise linear with a finite number of pieces and are continuous on each region. Notice that the zero output weight function satisfies this property. Moreover, the computations we will perform in the following maintain this property as an invariant, and use it to prove their correctness.

Region and corner abstractions. The region automaton, or region game, $\mathcal{R}_N(\mathcal{G})$ (abbreviated as $\mathcal{R}(\mathcal{G})$ when $N = 1$) of a game $\mathcal{G} = \langle L = L_{\mathsf{Min}} \uplus L_{\mathsf{Max}}, \Delta, \mathsf{wt}, L_T, \mathsf{wt}_T \rangle$ is the WTG with locations $S = L \times \mathsf{Reg}_N(X, M)$ and all transitions $((\ell, r), g'', Y, (\ell', r'))$ with $(\ell, g, Y, \ell') \in \Delta$ such that the model of guard g'' (i.e. all valuations ν such that $\nu \models g''$) is a $1/N$-region r'',

[1] The determinacy result is stated in [13] for WTG (called priced timed games) with one clock, but the proof does not use the assumption on the number of clocks.

time successor of r such that r'' satisfies the guard g, and $r' = r''[Y := 0]$. Distribution of locations to players, final locations and weights are taken according to $\mathcal{G}$. We call *path* a finite or infinite sequence of transitions in this automaton, and we denote by π the paths. A play ρ in $\mathcal{G}$ is projected on a path π in $\mathcal{R}_N(\mathcal{G})$, by replacing every edge $(\ell, \nu) \xrightarrow{d, \delta=(\ell,g,Y,\ell')} (\ell', \nu')$ by the transition $((\ell, r), g, Y, (\ell', r'))$, where r (resp. r') is the $1/N$-region containing ν (resp. ν'): we say that ρ *follows* the path π. It is important to notice that, even if π is a *cycle* (i.e. starts and ends in the same location of the region game), there may exist plays following it in $\mathcal{G}$ that are not cycles, due to the fact that regions are sets of valuations. By projecting away the region information of $\mathcal{R}_N(\mathcal{G})$, we simply obtain:

▶ **Lemma 1.** *For all $\ell \in L$, $1/N$-regions r, and $\nu \in r$, $\mathsf{Val}_{\mathcal{G}}(\ell, \nu) = \mathsf{Val}_{\mathcal{R}_N(\mathcal{G})}((\ell, r), \nu)$.*

On top of regions, we will need the corner-point abstraction techniques introduced in [8]. A valuation v is said to be a corner of a $1/N$-region r, if it belongs to the topological closure $\overline{r}$ and has coordinates multiple of $1/N$ ($v \in (1/N)\mathbb{N}^X$). We call corner state a triple (ℓ, r, v) that contains information about a location (ℓ, r) of the region-game $\mathcal{R}_N(\mathcal{G})$, and a corner v of the $1/N$-region r. Every region has at most $|X| + 1$ corners. We now define the corner-point abstraction $\mathcal{C}_N(\mathcal{G})$ of a WTG $\mathcal{G}$ as the WTG obtained as a refinement of $\mathcal{R}_N(\mathcal{G})$ where guards on transitions are enforced to stay on one of the corners of the current $1/N$-region: the locations of $\mathcal{C}_N(\mathcal{G})$ are all corner states of $\mathcal{R}_N(\mathcal{G})$, associated to each player accordingly, and transitions are all $((\ell, r, v), g'', Y, (\ell', r', v'))$ such that there exists $t = ((\ell, r), g, Y, (\ell', r'))$ a transition of $\mathcal{R}_N(\mathcal{G})$ such that the model of guard g'' is a corner v'' satisfying the guard $\overline{g}$ (recall that $\overline{g}$ is the closed version of g), $v' = v''[Y := 0]$, and there exist two valuations $\nu \in r$, $\nu' \in r'$ such that $((\ell, r), \nu) \xrightarrow{d', t} ((\ell', r'), \nu')$ for some $d' \in \mathbb{R}_{\geqslant 0}$ (the latter condition ensures that the transition between corners is not spurious). Because of this closure operation, we must also define properly the final weight function: we simply define it over the only valuation v reachable in location (ℓ, r, v) (with $\ell \in L_T$) by $\mathsf{wt}_T((\ell, r, v), v) = \lim_{\nu \to v, \nu \in r} \mathsf{wt}_T(\ell, \nu)$ (the limit is well defined since wt_T is piecewise linear with a finite number of pieces on region r).

The WTG $\mathcal{C}_N(\mathcal{G})$ can be seen as a *weighted game* (with final weights), i.e. a WTG without clocks (which means that there are only weights on transitions), by removing guards, resets and rates of locations, and replacing the weights of transitions by the actual weight of jumping from one corner to another: a transition $(((\ell, r), v), g'', Y, ((\ell', r'), v'))$ becomes an edge from $((\ell, r), v)$ to $((\ell', r'), v')$ with weight $d \times \mathsf{wt}(\ell) + \mathsf{wt}(t)$ (for all possible values of d, which requires to allow for multi-edges[2]). Note that delay d is necessarily a rational of the form α/N with $\alpha \in \mathbb{N}$, since it must relate corners of $1/N$-regions. In particular, this proves that the cumulated weight $\mathsf{wt}_\Sigma(\rho)$ of a finite play ρ in $\mathcal{C}_N(\mathcal{G})$ is indeed a rational number with denominator N. We will call *corner play* a play ρ in the corner-point abstraction $\mathcal{C}_N(\mathcal{G})$: it can also be interpreted as a timed execution in $\mathcal{G}$ where all guards are closed (as explained in the definition above). It straightforwardly projects on a finite path π in the region game $\mathcal{R}_N(\mathcal{G})$: in this case, we say again that ρ follows π. Figure 2 depicts a play, its projected path in the region game and one of its associated corner plays.
Corner plays allow one to obtain faithful information on the plays that follow the same path:

▶ **Lemma 2.** *If π is a finite path in $\mathcal{R}_N(\mathcal{G})$, the set $\{\mathsf{wt}_\Sigma(\rho) \mid \rho$ finite play following $\pi\}$ is an interval bounded by the minimum and the maximum values of the set $\{\mathsf{wt}_\Sigma(\rho) \mid \rho$ finite corner play of $\mathcal{C}_N(\mathcal{G})$ following $\pi\}$.*

[2] The only case where several edges could link two corners using the same transition is when all clocks are reset in Y, in which case there is a choice for delay d.

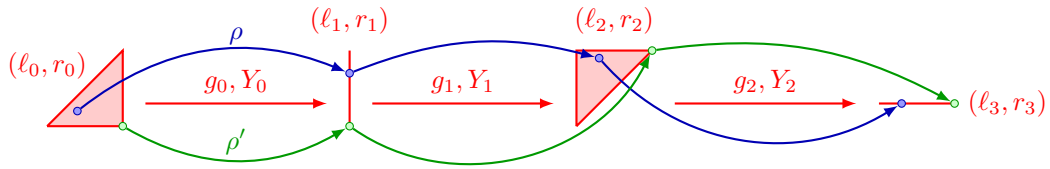


Figure 2 A play ρ (in blue), its projected path π in the region game (in red), and one of its associated corner plays ρ' (in green).

Value iteration. We will rely on the value iteration algorithm described in [1] for a WTG $\mathcal{G}$.

If V represents a value function—i.e. a mapping from configurations of $L \times \mathbb{R}_{\geqslant 0}^X$ to a value in $\mathbb{R}_\infty$—we denote by $V_{\ell,\nu}$ the image $V(\ell,\nu)$, for better readability, and by V_ℓ the function mapping each valuation ν to $V_{\ell,\nu}$. One step of the game is summarised in the following operator $\mathcal{F}$ mapping each value function V to a value function $V' = \mathcal{F}(V)$ defined by $V'_{\ell,\nu} = \mathsf{wt}_T(\ell,\nu)$ if $\ell \in L_T$, and otherwise

$$V'_{\ell,\nu} = \begin{cases} \sup_{(\ell,\nu) \xrightarrow{d,\delta} (\ell',\nu')} \left[d \times \mathsf{wt}(\ell) + \mathsf{wt}(\delta) + V_{\ell',\nu'} \right] & \text{if } \ell \in L_{\mathsf{Max}} \\ \inf_{(\ell,\nu) \xrightarrow{d,\delta} (\ell',\nu')} \left[d \times \mathsf{wt}(\ell) + \mathsf{wt}(\delta) + V_{\ell',\nu'} \right] & \text{if } \ell \in L_{\mathsf{Min}} \end{cases} \tag{1}$$

where $(\ell,\nu) \xrightarrow{d,\delta} (\ell',\nu')$ ranges over valid edges in $\mathcal{G}$. Then, starting from V^0 mapping every configuration to $+\infty$, except for the targets mapped to wt_T, we let $V^i = \mathcal{F}(V^{i-1})$ for all $i > 0$. The value function V^i represents the value $\mathsf{Val}^i_{\mathcal{G}}$, which is intuitively what Min can guarantee when forced to reach the target in at most i steps.

More formally, we define $\mathsf{wt}^i_{\mathcal{G}}(\rho)$ the weight of a maximal play ρ at horizon i, as $\mathsf{wt}_{\mathcal{G}}(\rho)$ if ρ reaches a target state in at most i steps, and $+\infty$ otherwise. Using this alternative definition of the weight of a play, we can obtain a new game value $\mathsf{Val}^i_{\mathcal{G}}(\ell,\nu) = \inf_{\sigma_{\mathsf{Min}}} \sup_{\sigma_{\mathsf{Max}}} \mathsf{wt}^i_{\mathcal{G}}(\mathsf{play}((\ell,\nu),\sigma_{\mathsf{Max}},\sigma_{\mathsf{Min}}))$. Then, if $\mathcal{G}$ is a tree of depth d, $V^i{=}\mathsf{Val}_{\mathcal{G}}$ if $i \geq d$.

The mappings V^0_ℓ are piecewise linear for all ℓ, and $\mathcal{F}$ preserves piecewise linearity over regions, so all iterates V^i_ℓ are piecewise linear with a finite number of pieces. In [1], it is proved that V^i_ℓ has a number of pieces (and can be computed within a complexity) exponential in i and in the size of $\mathcal{G}$ when $\mathsf{wt}_T = 0$. This result can be extended to handle negative weights in $\mathcal{G}$ and output weights $\mathsf{wt}_T \neq 0$.

3 Results

We consider the *value problem* that asks, given a WTG $\mathcal{G}$, a location ℓ and a threshold $\alpha \in \mathbb{Z} \cup \{-\infty, +\infty\}$, to decide whether $\mathsf{Val}_{\mathcal{G}}(\ell, \mathbf{0}) \leqslant \alpha$. In the context of timed games, optimal strategies may not exist. We generally focus on finding ε-optimal strategies, that guarantee the optimal value, up to a small error ε. Moreover, when the value problem is undecidable, we also consider the *approximation problem* that consists, given a precision $\varepsilon \in \mathbb{Q}_{>0}$, in computing an ε-approximation of $\mathsf{Val}_{\mathcal{G}}(\ell, \mathbf{0})$.

In the one-player case, computing the optimal value and an ε-optimal strategy for weighted timed automata is known to be PSPACE-complete [6]. In the two-player case, the value problem of WTGs (also called priced timed games in the literature) is undecidable with 3 clocks [12, 10], or even 2 clocks in the presence of negative weights [15] (for the existence problem asking if a strategy of player Min can guarantee a given threshold). To obtain decidability, one possibility is to limit the number of clocks to 1: then, there is

an exponential-time algorithm to compute the value as well as ε-optimal strategies in the presence of non-negative weights only [7, 19, 17], whereas the problem is only known to be PTIME-hard. A similar result can be lifted to arbitrary weights, under restrictions on the resets of the clock in cycles [13].

The other possibility to obtain a decidability result [9, 16] is to enforce a semantical property of divergence (originally called strictly non-Zeno cost): it asks that every play following a cycle in the region automaton has weight far from 0. It allows the authors to prove that playing for only a bounded number of steps is equivalent to the original game, which boils down to the problem of computing the value of a tree-shaped weighted timed game $\mathcal{G}$ using the value iteration algorithm.

Other objectives, not directly related to optimal reachability, have been considered in [11] for weighted timed games, like mean-payoff and parity objectives. In this work, the authors manage to solve these problems for the so-called class of δ-robust WTGs that they introduce. This class includes the class we consider, but is decidable in 2-EXPSPACE.

In [16], we generalised the strictly non-Zeno cost property of [9, 16] to weighted timed games with both positive and negative weights: we called them divergent weighted timed games. This article relaxes the divergence property, to introduce almost-divergent weighted timed games. We first define formally these classes of games. A cycle π of $\mathcal{R}(\mathcal{G})$ is said to be a positive cycle (resp. a 0-cycle, or a negative cycle) if every finite play ρ following π satisfies $\mathsf{wt}_\Sigma(\rho) \geqslant 1$ (resp. $\mathsf{wt}_\Sigma(\rho) = 0$, or $\mathsf{wt}_\Sigma(\rho) \leqslant -1$). A strongly connected component (SCC) S of $\mathcal{R}(\mathcal{G})$ is said to be positive (resp. negative) if every cycle $\pi \in S$ is positive (resp. negative). An SCC S of $\mathcal{R}(\mathcal{G})$ is said to be non-negative (resp. non-positive) if every play ρ following a cycle in S satisfies either $\mathsf{wt}_\Sigma(\rho) \geqslant 1$ or $\mathsf{wt}_\Sigma(\rho) = 0$ (resp. either $\mathsf{wt}_\Sigma(\rho) \leqslant -1$ or $\mathsf{wt}_\Sigma(\rho) = 0$).

▶ **Definition 3.** A WTG $\mathcal{G}$ is divergent if every SCC of $\mathcal{R}(\mathcal{G})$ is either positive or negative. As a generalisation, a WTG $\mathcal{G}$ is almost-divergent when every SCC of $\mathcal{R}(\mathcal{G})$ is either non-negative or non-positive.

In [16], we showed that we can decide in 2-EXPTIME the value problem for divergent WTGs. Unfortunately, it is shown in [10] that this problem is undecidable for almost-divergent WTGs (already with non-negative weights only, where almost-divergent WTGs are called *simple*). They propose a solution to the approximation problem, again with non-negative weights only. Our first result is the following extension of their result:

▶ **Theorem 4.** *Given an almost-divergent WTG $\mathcal{G}$, a location ℓ and $\varepsilon \in \mathbb{Q}_{>0}$, we can compute an ε-approximation of $\mathsf{Val}_{\mathcal{G}}(\ell, \mathbf{0})$ in time doubly-exponential in the size of $\mathcal{G}$ and polynomial in $1/\varepsilon$. Moreover, deciding if a WTG is almost-divergent is* PSPACE*-complete.*

To obtain this result, we follow an approximation schema that we now outline. First, we will always reason on the region game $\mathcal{R}(\mathcal{G})$ of the almost-divergent WTG $\mathcal{G}$. The goal is to compute an ε-approximation of $\mathsf{Val}_{\mathcal{R}(\mathcal{G})}(s_0, \mathbf{0})$ for some state $s_0 = (\ell_0, r_0)$, with r_0 the region where every clock value is 0. As already recalled, techniques of [1] allow one to compute the (exact) values of a WTG played on a finite tree, using operator $\mathcal{F}$. The idea is thus to decompose as much as possible the game $\mathcal{R}(\mathcal{G})$ in a WTG over a tree. First, we decompose the region game into SCCs (left of Figure 3).

During the approximation process, we must think about the final weight functions as the previously computed approximations of the values of SCCs below the current one. We will keep as an invariant that final weight functions are piecewise linear functions with a finite number of pieces, and are continuous on each region.

For an SCC of $\mathcal{R}(\mathcal{G})$ and an initial state s_0 of $\mathcal{R}(\mathcal{G})$ provided by the SCC decomposition, we show that the game on the SCC is equivalent to a game on a tree built from a semi-unfolding (see middle of Figure 3) of $\mathcal{R}(\mathcal{G})$ from s_0 of finite depth, with certain nodes of the

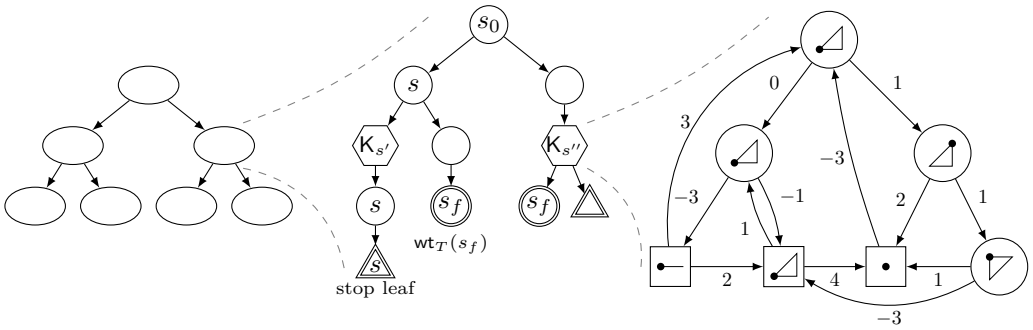


Figure 3 Static approximation schema: SCC decomposition of $\mathcal{R}(\mathcal{G})$, semi-unfolding of an SCC, corner-point abstraction for the kernels.

tree being *kernels*. These kernels are some parts of $\mathcal{R}(\mathcal{G})$ that contain all cycles of weight 0. The semi-unfolding is stopped either when reaching a final location, or when some location (or kernel) has been visited for a certain fixed number of times: such locations deep enough are called *stop leaves*.

Our second result is a more symbolic approximation schema based on the value iteration only. It is more symbolic in the sense that it does not require the SCC decomposition, the computation of kernels nor the semi-unfolding of the game in a tree.

▶ **Theorem 5.** *Let $\mathcal{G}$ be an almost-divergent WTG such that $\mathsf{Val}_{\mathcal{G}} > -\infty$ for all configurations. Then the sequence $(\mathsf{Val}^k_{\mathcal{G}})_{k\geqslant 0}$ converges towards $\mathsf{Val}_{\mathcal{G}}$ and for every $\varepsilon \in \mathbb{Q}_{>0}$, we can compute an integer P such that $\mathsf{Val}^P_{\mathcal{G}}$ is an ε-approximation of $\mathsf{Val}_{\mathcal{G}}$ for all configurations.*

▶ Remark. In a weighted-timed game, it is easy to detect the set of states with value $+\infty$: these are all the states from which Min cannot ensure reachability of a target location $\ell \in L_T$ with $\mathsf{wt}_T(\ell) < +\infty$. It can therefore be computed by an attractor computation, and is indeed a property constant on each region. In particular, removing those states from $\mathcal{R}(\mathcal{G})$ does not affect the value of any other state and can be done in complexity linear in $|\mathcal{R}(\mathcal{G})|$. We will therefore assume that the considered WTG have no configurations with value $+\infty$.

4 Kernels of an almost-divergent WTG

The approximation procedure described before uses the so-called *kernels* in order to group together all cycles of weight 0. We study those kernels and give a characterisation allowing computability. Contrary to the non-negative case, the situation is more complex in our arbitrary case, since weights of both locations and transitions may differ from 0 in the kernel. Moreover, it is not trivial (and may not be true in a non almost-divergent WTG) to know whether it is sufficient to consider only simple cycles, i.e. cycles without repetitions.

To answer these questions, let us first analyse the cycles of $\mathcal{R}(\mathcal{G})$ that we will encounter. Since we are in an almost-divergent game, by Lemma 2, all cycles $\pi = t_1 \cdots t_n$ of $\mathcal{R}(\mathcal{G})$ (with $t_1, \ldots, t_n$ transitions of $\mathcal{R}(\mathcal{G})$) are either 0-cycles, positive cycles or negative cycles. Additionally, in an SCC S of $\mathcal{R}(\mathcal{G})$, we cannot find both positive and negative cycles by definition. Moreover, we can classify a cycle by looking only at the corner plays following it.

▶ **Lemma 6.** *A cycle π is a 0-cycle iff there exists a corner play ρ following π with $\mathsf{wt}_\Sigma(\rho)=0$.*

Proof. If π is a 0-cycle, every such corner play ρ will have weight 0, by Lemma 2. Reciprocally, if such a corner play exists, all corner plays following π have weight 0: otherwise the set $\{\mathsf{wt}_\Sigma(\rho) \mid \rho \text{ play following } \pi\}$ would have non-empty intersection with the set $(-1,1)\setminus\{0\}$ which would contradict the almost-divergence. ◀

An important result is that 0-cycles are stable by rotation. This is not trivial because plays following a cycle can start and end in different valuations, therefore changing the starting state of the cycle could a priori change the plays that follow it and their weights.

▶ **Lemma 7.** *Let π and π' be paths of $\mathcal{R}(\mathcal{G})$. Then, $\pi\pi'$ is a 0-cycle iff $\pi'\pi$ is a 0-cycle.*

Proof. Since $\pi_1 = \pi\pi'$ is a cycle, $\mathsf{first}(\pi) = \mathsf{last}(\pi')$ and $\mathsf{first}(\pi') = \mathsf{last}(\pi)$, so $\pi_2 = \pi'\pi$ is correctly defined.

First, since there are finitely many corners, by constructing a long enough play following an iterate of $\pi'\pi$, we can obtain a corner play that starts and ends in the same corner. Formally, we define two sequences of region corners $(v_i \in \mathsf{first}(\pi))_i$ and $(v'_i \in \mathsf{first}(\pi'))_i$. We start by choosing any $v_0 \in \mathsf{first}(\pi)$. Let v'_0 be a corner of $\mathsf{first}(\pi')$ such that v'_0 is accessible from v_0 by following π. For every $i > 0$, let v_i be a corner of $\mathsf{first}(\pi)$ such that v_i is accessible from v'_{i-1} by following π', and let v'_i be a corner of $\mathsf{first}(\pi')$ such that v'_i is accessible from v_i by following π. We stop the construction at the first l such that there exists $k < l$ with $v_k = v_l$. Additionally, we let $v'_l = v'_k$ and $v_{l+1} = v_{k+1}$. This process is bounded since $\mathsf{first}(\pi)$ has at most $|X| + 1$ corners.

For every $0 \leqslant i \leqslant l$, let w_i be the weight of a play ρ_i from v_i to v'_i along π, and let w'_i be the weight of a play ρ'_i from v'_i to v_{i+1} along π'. The concatenation of the two plays has weight $w_i + w'_i = 0$, since it follows the 0-cycle π_1. Therefore, all corner plays from v_i to v'_i following π have the same weight w_i, and the same applies for w'_i. For every $0 \leqslant i < l$, the concatenation of ρ'_i and ρ_{i+1} is a play from v'_i to v_{i+1}, of weight $w'_i + w_{i+1} = -w_i + w_{i+1}$, following π_2. Since π_2 is a cycle, and the game is almost-divergent, all possible values of $w_{i+1} - w_i$ have the same sign.

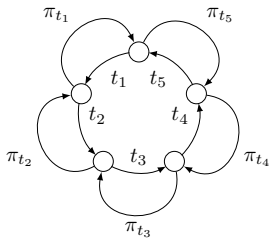
Finally, we can construct a corner play from v'_k to v'_l by concatenating the plays ρ'_k, ρ_{k+1}, $\rho'_{k+1}, \rho_{k+2}, \ldots, \rho'_{l-1}, \rho_l$. That play has weight $\sum_{i=k}^{l-1}(w_{i+1} - w_i) = w_l - w_k = 0$. This implies that the terms $w_{i+1} - w_i$, of constant sign, are all equal to 0. As a consequence, the concatenation of ρ'_k and ρ_{k+1} is a corner play following π_2 of weight 0. By Lemma 6, we deduce that π_2 is a 0-cycle. ◀

We will now construct the kernel K as the subgraph of $\mathcal{R}(\mathcal{G})$ containing all 0-cycles. Formally, let T_K be the set of transitions of $\mathcal{R}(\mathcal{G})$ belonging to a *simple* 0-cycle, and S_K be the set of states covered by T_K. We define the kernel K of $\mathcal{R}(\mathcal{G})$ as the subgraph of $\mathcal{R}(\mathcal{G})$ defined by S_K and T_K. Transitions in $T\backslash T_\mathsf{K}$ with starting state in S_K are called the output transitions of K. We define it using only simple 0-cycles in order to ensure its computability. However, we now show that this is of no harm, since the kernel contains exactly all the 0-cycles, which will be crucial in the approximation schema we present in Section 6.

▶ **Proposition 8.** *A cycle of $\mathcal{R}(\mathcal{G})$ is entirely in K if and only if it is a 0-cycle.*

Proof. We prove that every 0-cycle is in K by induction on the length of the cycles. The initialisation contains only cycles of length 1, that are in K by construction. If we consider a cycle π of length $n > 1$, it is either simple or it can be rotated and decomposed into $\pi'\pi''$, π' and π'' being smaller cycles. Let ρ be a corner play following $\pi'\pi''$. We denote by ρ' the

prefix of ρ following π' and ρ'' the suffix following π''. It holds that $\mathsf{wt}_\Sigma(\rho') = -\mathsf{wt}_\Sigma(\rho'')$, and in an almost-divergent SCC this implies $\mathsf{wt}_\Sigma(\rho') = \mathsf{wt}_\Sigma(\rho'') = 0$. Therefore, by Lemma 6 both π' and π'' are 0-cycles, and they must be in K by induction hypothesis. Note that this reasoning proves that every cycle contained in a longer 0-cycle is also a 0-cycle.



We now prove that every cycle in K is a 0-cycle. By construction, every transition $t \in T_\mathsf{K}$ is part of a simple 0-cycle. Thus, to every transition $t \in T_\mathsf{K}$, we can associate a path π_t such that $t\pi_t$ is a simple 0-cycle (rotate the simple cycle if necessary). We can prove (using both Lemmas 6 and 7) the following property by relying on another pumping argument on corners: If $t_1 \cdots t_n$ is a path in K, then $t_1 t_2 \cdots t_n \pi_{t_n} \cdots \pi_{t_2} \pi_{t_1}$ is a 0-cycle of $\mathcal{R}(\mathcal{G})$. Now, if π is a cycle of $\mathcal{R}(\mathcal{G})$ in K, there exists a cycle π' such that $\pi\pi'$ is a 0-cycle, therefore π is a 0-cycle. ◀

5 Semi-unfolding of almost-divergent WTGs

Given an almost-divergent WTG $\mathcal{G}$, we describe the construction of its *semi-unfolding* $\mathcal{T}(\mathcal{G})$ (as depicted in Figure 3). This crucially relies on the absence of states with value $-\infty$, so we explain how to deal with them first:

▶ **Lemma 9.** *In an SCC of $\mathcal{R}(\mathcal{G})$, the set of configurations with value $-\infty$ is a union of regions computable in time linear in the size of $\mathcal{R}(\mathcal{G})$.*

Sketch of proof. If the SCC is non-negative, the cumulated weight cannot decrease along a cycle, thus, the only way to obtain value $-\infty$ is to jump in a final state with final weight $-\infty$. We can therefore compute this set of states with an attractor for Min.

If the SCC is non-positive, we let $S_f^{\mathbb{R}}$ (resp. $S_f^{-\infty}$) be the set of target states where wt_T is bounded (resp. has value $-\infty$). We also define $T_f^{\mathbb{R}}$ (resp. $T_f^{-\infty}$), the set of transitions of $\mathcal{R}(\mathcal{G})$ whose end state belongs to $S_f^{\mathbb{R}}$ (resp. $S_f^{-\infty}$). Notice that the kernel cannot contain target states since they do not have outgoing transitions. We can prove that a configuration has value $-\infty$ iff it belongs to a state where player Min can ensure the LTL formula on transitions: $(\mathrm{G}\,\neg T_f^{\mathbb{R}} \wedge \neg \mathrm{FG}\, T_\mathsf{K}) \vee \mathrm{F}\, T_f^{-\infty}$. The procedure to detect $-\infty$ states thus consists of four attractor computations, which can be done in time linear in $|\mathcal{R}(\mathcal{G})|$. ◀

We can now assume that no states of $\mathcal{G}$ have value $-\infty$, and that the output weight function maps all configurations to $\mathbb{R}$. Since wt_T is piecewise linear with finitely many pieces, wt_T is bounded. Let $\sup|\mathsf{wt}_T|$ denote the bound of $|\mathsf{wt}_T|$, ranging over all target configurations.

We now explain how to build the semi-unfolding $\mathcal{T}(\mathcal{G})$. We only build the semi-unfolding $\mathcal{T}(\mathcal{G})$ of an SCC of $\mathcal{G}$ starting from some state $(\ell_0, r_0) \in S$ of the region game, since it is then easy to glue all the semi-unfoldings together to get the one of the full game. Since every configuration has finite value, we can prove that values of the game are bounded by $|\mathcal{R}(\mathcal{G})| w^e_{\max} + \sup|\mathsf{wt}_T|$. As a consequence, we can find a bound γ linear in $|\mathcal{R}(\mathcal{G})|$, $w^e_{\max}$

and $\sup|\mathsf{wt}_T|$ such that a play that visits some state outside the kernel more than γ times has weight strictly above $|\mathcal{R}(\mathcal{G})|w^e_{\max} + \sup|\mathsf{wt}_T|$, hence is useless for the value computation. This leads to considering the semi-unfolding $\mathcal{T}(\mathcal{G})$ of $\mathcal{G}$ (nodes in the kernel are not unfolded, see Figure 3) such that each node not in the kernel is encountered at most γ times along a branch: the end of each branch is called a *stop leaf* of the semi-unfolding. In particular, the depth of $\mathcal{T}(\mathcal{G})$ is bounded by $|\mathcal{R}(\mathcal{G})|\gamma$, and thus is polynomial in $|\mathcal{R}(\mathcal{G})|$, $w^e_{\max}$ and $\sup|\mathsf{wt}_T|$. Leaves of the semi-unfolding are thus of two types: target leaves that are copies of target locations of $\mathcal{G}$ for which we set the target weight as in $\mathcal{G}$, and stop leaves for which we set their target weight as being constant to $+\infty$ if the SCC $\mathcal{G}$ is non-negative, and $-\infty$ if the SCC is non-positive.

▶ **Proposition 10.** *Let $\mathcal{G}$ be an almost-divergent WTG, and let $(\ell_0, r_0) \in S$ be some state of the region game. The semi-unfolding $\mathcal{T}(\mathcal{G})$ with initial state $(\tilde{\ell}_0, r_0)$ (a copy of state (ℓ_0, r_0)) is equivalent to $\mathcal{G}$, i.e. for all $\nu_0 \in r_0$, $\mathsf{Val}_{\mathcal{G}}(\ell_0, \nu_0) = \mathsf{Val}_{\mathcal{T}(\mathcal{G})}((\tilde{\ell}_0, r_0), \nu_0)$.*

6 Approximation of almost-divergent WTGs

Approximation of kernels. We start by approximating a kernel $\mathcal{G}$ by extending the region-based approximation schema of [10]. In their setting, all runs in kernels had weight 0, allowing a simple reduction to a finite weighted game. In our setting, we have to approximate the timed dynamics of runs, and therefore resort to the corner-point abstraction (as shown to the right of Figure 3).

Since output weight functions are piecewise linear with a finite number of pieces and continuous on regions, they are Λ-Lipschitz-continuous[3], for a given constant $\Lambda \geqslant 0$. We let $\mathbf{B} = w^L_{\max}|L||\mathsf{Reg}(X, M)| + \Lambda$.

Let N be an integer. Consider the game $\mathcal{C}_N(\mathcal{G})$ described in the preliminary section, with locations of the form (ℓ, r, v) with v a corner of the $1/N$-region r. Two plays ρ of $\mathcal{G}$ and ρ' of $\mathcal{C}_N(\mathcal{G})$ are said to be $1/N$-*close* if they follow the same path π in $\mathcal{R}_N(\mathcal{G})$. In particular, at each step the configurations (ℓ, ν) in ρ and (ℓ', r', v') in ρ' (with v' a corner of the $1/N$-region r') satisfy $\ell = \ell'$ and $\nu \in r'$, and the transitions taken in both plays have the same discrete weights. Close plays have *close* weights, in the following sense:

▶ **Lemma 11.** *For all $1/N$-close plays ρ of $\mathcal{G}$ and ρ' of $\mathcal{C}_N(\mathcal{G})$, $|\mathsf{wt}_{\mathcal{G}}(\rho) - \mathsf{wt}_{\mathcal{C}_N(\mathcal{G})}(\rho')| \leqslant \mathbf{B}/N$.*

In particular, if we start in configurations (ℓ_0, ν_0) of $\mathcal{G}$, and $((\ell_0, r_0, v_0), v_0)$ of $\mathcal{C}_N(\mathcal{G})$, with $\nu_0 \in r_0$, since both players have the ability to stay $1/N$-close all along the plays, a bisimulation argument permits to obtain that the values of the two games are also close in (ℓ_0, ν_0) and $((\ell_0, r_0, v_0), v_0)$:

▶ **Lemma 12.** *For all locations $\ell \in L$, $1/N$-regions r, $\nu \in r$ and corners v of r, $|\mathsf{Val}_{\mathcal{G}}(\ell, \nu) - \mathsf{Val}_{\mathcal{C}_N(\mathcal{G})}((\ell, r, v), v)| \leqslant \mathbf{B}/N$.*

Using this result, picking N an integer larger than $\mathbf{B}/\varepsilon$, we can thus obtain $|\mathsf{Val}_{\mathcal{G}}(\ell, \nu) - \mathsf{Val}_{\mathcal{C}_N(\mathcal{G})}((\ell, r, v), v)| \leqslant \varepsilon$. Recall that $\mathcal{C}_N(\mathcal{G})$ can be considered as an untimed weighted game (with reachability objective). Thus we can apply the result of [14], where it is shown that the optimal values of such games can be computed in pseudo-polynomial time (i.e. polynomial

[3] The function wt_T is said to be Λ-Lipschitz-continuous when $|\mathsf{wt}_T(s, \nu) - \mathsf{wt}_T(s, \nu')| \leqslant \Lambda\|\nu - \nu'\|_\infty$ for all valuations ν, ν', where $\|v\|_\infty = \max_{x \in X}|v(x)|$ is the ∞-norm of vector $v \in \mathbb{R}^X$. The function wt_T is said to be Lipschitz-continuous if it is Λ-Lipschitz-continuous, for some Λ.

time with weights encoded in unary, instead of binary). We then define an ε-approximation of $\mathsf{Val}_{\mathcal{G}}$, named Val'_N, on each $1/N$-region by interpolating the values of its $1/N$-corners in $\mathcal{C}_N(\mathcal{G})$ with a piecewise linear function: therefore, we can control the Lipschitz constant of the approximated value for further use.

▶ **Lemma 13.** *Val'_N is an ε-approximation of $\mathsf{Val}_{\mathcal{G}}$, that is piecewise linear with a finite number of pieces and $2\mathbf{B}$-Lipschitz-continuous over regions.*

Approximation of almost-divergent WTGs. We now explain how to approximate the value of an almost-divergent WTG $\mathcal{G}$, thus proving Theorem 4. First, we compute a semi-unfolding $\mathcal{T}(\mathcal{G})$ as described in the previous section. Then we perform a bottom-up computation of the approximation. As already recalled, techniques of [1] allow us to compute exact values of a tree-shape WTG. In consequence, we know how to compute the value of a non-kernel node of $\mathcal{T}(\mathcal{G})$, depending of the values of its children. There is no approximation needed here, so that if all children are ε-approximation, we can compute an ε-approximation of the node. Therefore, the only approximation lies in the kernels, and we explained before how to compute arbitrarily close an approximation of a kernel's value. We crucially rely on the fact that the value function is 1-Lipschitz-continuous[4]. This entails that imprecisions will sum up along the bottom-up computations, as computing an ε-approximation of the value of a game whose output weights are ε'-approximations yields an $(\varepsilon+\varepsilon')$-approximation. Therefore we compute approximations with threshold $\varepsilon' = \varepsilon/\alpha$ for kernels in $\mathcal{T}(\mathcal{G})$, where α is the maximal number of kernels along a branch of $\mathcal{T}(\mathcal{G})$: α is smaller than the depth of $\mathcal{T}(\mathcal{G})$, which is bounded by Proposition 10.

The subregion granularity considered before for kernel approximation crucially depends on the Lipschitz constant of output weights. The growth of these constants is bounded for kernels in $\mathcal{T}(\mathcal{G})$ by Lemma 13. For non-kernel nodes of $\mathcal{T}(\mathcal{G})$, using a careful analysis of the algorithm of [1], we obtain the following bound:

▶ **Lemma 14.** *If all the output weights of a WTG $\mathcal{G}$ are Λ-Lipschitz-continuous over regions (and piecewise linear, with finitely many pieces), then $\mathsf{Val}^i_{\mathcal{G}}$ is $\Lambda\Lambda'$-Lipschitz-continuous over regions, with Λ' polynomial in $w^L_{\max}$ and $|X|$ and exponential in i.*

The overall time complexity of this method is doubly-exponential in the size of the input game and polynomial in $1/\varepsilon$.

7 Symbolic approximation algorithm

The previous approximation result suffers from several drawbacks. It relies on the SCC decomposition of the region automaton. Each of these SCCs have to be analysed in a sequential way, and their analysis requires an a priori refinement of the granularity of regions. This approach is thus not easily amenable to implementation. We instead prove in this section that the symbolic approach based on the value iteration paradigm, i.e. the computation of iterates of the operator $\mathcal{F}$ recalled in page 7, is an approximation schema. This is stated in Theorem 5, for which we now sketch a proof in this section.

Notice that configurations with value $+\infty$ are stable through value iteration, and do not affect its other computations. Since Theorem 5 assumes the absence of configurations of value $-\infty$, we will therefore consider in the following that all configurations have finite value in $\mathcal{G}$.

[4] Indeed, inf and sup are 1-Lipschitz-continuous functions, and with a fixed play ρ, the mapping $\mathsf{wt}_T \to \mathsf{wt}_\Sigma(\rho) + \mathsf{wt}_T(\mathsf{last}(\rho))$ is 1-Lipschitz-continuous.

Consider first a game $\mathcal{G}$ that is a kernel. By the results of Section 6, there exists an integer N such that solving the untimed weighted game $\mathcal{C}_N(\mathcal{G})$ computes an $\varepsilon/2$-approximation of the value of $1/N$ corners. Using the results of [14] for untimed weighted games, we know that those values are obtained after a finite number of steps of (the untimed version of) the value iteration operator. More precisely, if one considers a number of iterations $P = |L||\mathsf{Reg}_N(X,M)|(|X|+1)(2(|L||\mathsf{Reg}_N(X,M)|(|X|+1)-1)w^e_{\max}+1)$, then $\mathsf{Val}^P_{\mathcal{C}_N(\mathcal{G})}((\ell,r,v),v) = \mathsf{Val}_{\mathcal{C}_N(\mathcal{G})}((\ell,r,v),v)$. From this observation, we deduce the following property of P:

▶ **Lemma 15.** *If $\mathcal{G}$ is a kernel with no configurations of infinite value, then $|\mathsf{Val}_{\mathcal{G}}(\ell,\nu) - \mathsf{Val}^P_{\mathcal{G}}(\ell,\nu)| \leqslant \varepsilon$ for all configurations (ℓ,ν) of $\mathcal{G}$.*

Proof. We already know that $\mathsf{Val}^P_{\mathcal{C}_N(\mathcal{G})}((\ell,r,v),v) = \mathsf{Val}_{\mathcal{C}_N(\mathcal{G})}((\ell,r,v),v)$ for all configurations $((\ell,r,v),v)$ of $\mathcal{C}_N(\mathcal{G})$. Moreover, Section 6 ensures $|\mathsf{Val}_{\mathcal{G}}(\ell,\nu) - \mathsf{Val}_{\mathcal{C}_N(\mathcal{G})}((\ell,r,v),v)| \leqslant \varepsilon/2$ whenever ν is in the $1/N$-region r. Therefore, we only need to prove that $|\mathsf{Val}^P_{\mathcal{G}}(\ell,\nu) - \mathsf{Val}^P_{\mathcal{C}_N(\mathcal{G})}((\ell,r,v),v)| \leqslant \varepsilon/2$ to conclude. This is done as for Lemma 12, since Lemma 11 (that we need to prove Lemma 12) does not depend on the length of the plays ρ and ρ', and both runs reach the target state in the same step, i.e. both before or after the horizon of P steps. ◀

Once we know that value iteration converges on kernels, we can use the semi-unfolding of Section 5 to prove that it also converges on non-negative SCCs when all values are finite.

▶ **Lemma 16.** *If $\mathcal{G}$ is a non-negative SCC with no configurations of infinite value, we can compute P_+ such that $|\mathsf{Val}_{\mathcal{G}}(\ell,\nu) - \mathsf{Val}^{P_+}_{\mathcal{G}}(\ell,\nu)| \leqslant \varepsilon$ for all configurations (ℓ,ν) of $\mathcal{G}$.*

The idea is to unfold every kernel of the semi-unfolding game $\mathcal{T}(\mathcal{G})$ according to its bound in Lemma 15. More precisely, let α be the maximum number of kernels along one of the branches of $\mathcal{T}(\mathcal{G})$. In a bottom-up fashion, we can find for each kernel K in $\mathcal{T}(\mathcal{G})$ a bound P_{K} such that, for all configurations (ℓ,ν), $|\mathsf{Val}_{\mathsf{K}}(\ell,\nu) - \mathsf{Val}^{P_{\mathsf{K}}}_{\mathsf{K}}(\ell,\nu)| \leqslant \varepsilon/\alpha$. We thus unfold K in $\mathcal{T}(\mathcal{G})$ with depth up to P_{K}. After each kernel has been replaced this way, $\mathcal{T}(\mathcal{G})$ is no longer a semi-unfolding, it is instead a (complete) unfolding of $\mathcal{R}(\mathcal{G})$, of a certain bounded depth P_+. This new bound P_+ is bounded by the former depth of $\mathcal{T}(\mathcal{G})$ to which is added α times the biggest bound P_{K} we need for the kernels. Now, $\mathcal{T}(\mathcal{G})$ is a tree of depth P_+ whose value at its root is ε-close to the value of $\mathcal{G}$. Finally, the value computed by $\mathsf{Val}^{P_+}_{\mathcal{G}}$ is bounded between $\mathsf{Val}_{\mathcal{G}}$ and $\mathsf{Val}_{\mathcal{T}(\mathcal{G})}$, which allows us to conclude.

The bound P_{K} for a kernel K depends linearly in Λ, the Lipschitz constant of value functions on locations of $\mathcal{T}(\mathcal{G})$ reachable from K. Once K has been replaced by its unfolding of depth P_{K}, the Lipschitz constant of the value function at the root of $\mathcal{T}(\mathcal{G})$ are thus bounded exponentially in Λ. This means that we ensure a bound for P_+ that is at most polynomial in $1/\varepsilon$, and that is of the order of a tower of α exponentials.

Proving the same property on non-positive SCCs requires more work, because the semi-unfolding gives output weight $-\infty$ to stop leaves, which doesn't integrate well with value iteration (initialisation at $+\infty$ on non-target states). However, by unfolding those SCCs slightly more (at most $|\mathcal{R}(\mathcal{G})|$ more steps), we can obtain the desired property with a similar bound P_-.

▶ **Lemma 17.** *If $\mathcal{G}$ is a non-positive SCC with no configurations of infinite value, we can compute P_- such that $|\mathsf{Val}_{\mathcal{G}}(\ell,\nu) - \mathsf{Val}^{P_-}_{\mathcal{G}}(\ell,\nu)| \leqslant \varepsilon$ for all configurations (ℓ,ν) of $\mathcal{G}$.*

Now, if we are given an almost-divergent game $\mathcal{G}$ and a precision ε, we can add the bounds for value iteration obtained from each SCC by Lemmas 16 and 17, and obtain a final bound P such that for all $k \geqslant P$, $\mathsf{Val}^k_{\mathcal{G}}$ is an ε-approximation of $\mathsf{Val}_{\mathcal{G}}$.

Discussion. Overall, this leads to an upper bound complexity that is polynomial in $1/\varepsilon$ and of the order of a tower of n exponentials, with n polynomial in the size of the input WTG. However, we argue that this symbolic procedure is more amenable to implementation than the previous approximation schema. First, it avoids the three already mentioned drawbacks (SCC decomposition, sequential analysis of the SCCs, and refinement of the granularity of regions) of the previous approximation schema. Then, it allows one to directly launch the value iteration algorithm on the game $\mathcal{G}$, and we can stop the computation whenever we are satisfied enough by the approximation computed: however, there are no guarantees whatsoever on the quality of the approximation before the number of steps P given above. Finally, this schema allows one to easily obtain an almost-optimal strategy with respect to the computed value.

If $\mathcal{G}$ is not guaranteed to be free of configurations of value $-\infty$, then we must first perform the SCC decomposition of $\mathcal{R}(\mathcal{G})$, and, as $\mathcal{G}$ is almost-divergent, identify and remove regions whose value is $-\infty$, by Lemma 9. Then, we can apply the value iteration algorithm.

As a final remark, notice that our correctness proof strongly relies on Section 6, and thus would not hold with the approximation schema of [10] (which does not preserve the continuity on regions of the computed value functions, in turn needed to define output weights on $1/N$-corners).

8 Conclusion

We have given an approximation procedure for a large class of weighted timed games with unbounded number of clocks and arbitrary integer weights that can be executed in doubly-exponential time with respect to the size of the game. In addition, we proved the correctness of a symbolic approximation schema, that does not start by splitting exponentially every region, but only does so when necessary (as dictated by [1]). We argue that this paves the way towards an implementation of value approximation for weighted timed games.

Another perspective is to extend this work to the concurrent setting, where both players play simultaneously and the shortest delay is selected. We did not consider this setting in this work because concurrent WTGs are not determined, and several of our proofs rely on this property for symmetrical arguments (mainly to lift results of non-negative SCCs to non-positive ones). Another extension of this work is the exploration of the effect of almost-divergence in the case of multiple weight dimensions, and/or with mean-payoff objectives.

References

1 Rajeev Alur, Mikhail Bernadsky, and P. Madhusudan. Optimal Reachability for Weighted Timed Games. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP'04)*, volume 3142 of *LNCS*, pages 122–133. Springer, 2004.

2 Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

3 Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal Paths in Weighted Timed Automata. *Theoretical Computer Science*, 318(3):297–322, 2004.

4 Eugene Asarin and Oded Maler. As Soon as Possible: Time Optimal Control for Timed Automata. In *Hybrid Systems: Computation and Control*, volume 1569 of *LNCS*, pages 19–30. Springer, 1999.

5 Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim G. Larsen, Judi Romijn, and Frits W. Vaandrager. Minimum-cost Reachability for Priced Timed Automata. In *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *LNCS*, pages 147–161. Springer, 2001.

6 Patricia Bouyer, Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On the Optimal Reachability Problem of Weighted Timed Automata. *Formal Methods in System Design*, 31(2):135–175, 2007.

7 Patricia Bouyer, Thomas Brihaye, and Nicolas Markey. Improved Undecidability Results on Weighted Timed Automata. *Information Processing Letters*, 98(5):188–194, 2006.

8 Patricia Bouyer, Ed Brinksma, and Kim G. Larsen. Optimal Infinite Scheduling for Multi-Priced Timed Automata. *Formal Methods in System Design*, 32(1):3–23, 2008.

9 Patricia Bouyer, Franck Cassez, Emmanuel Fleury, and Kim G. Larsen. Optimal Strategies in Priced Timed Game Automata. In *Proceedings of the 24th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'04)*, volume 3328 of *LNCS*, pages 148–160. Springer, 2004.

10 Patricia Bouyer, Samy Jaziri, and Nicolas Markey. On the Value Problem in Weighted Timed Games. In *Proceedings of the 26th International Conference on Concurrency Theory (CONCUR'15)*, volume 42 of *Leibniz International Proceedings in Informatics*, pages 311–324. Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.CONCUR.2015.311.

11 Romain Brenguier, Franck Cassez, and Jean-François Raskin. Energy and mean-payoff timed games. In *17th International Conference on Hybrid Systems: Computation and Control (part of CPS Week), HSCC'14, Berlin, Germany, April 15-17, 2014*, pages 283–292. ACM, 2014.

12 Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On Optimal Timed Strategies. In *Proceedings of the Third international conference on Formal Modeling and Analysis of Timed Systems (FORMATS'05)*, volume 3829 of *LNCS*, pages 49–64. Springer, 2005.

13 Thomas Brihaye, Gilles Geeraerts, Axel Haddad, Engel Lefaucheux, and Benjamin Monmege. Simple Priced Timed Games Are Not That Simple. In *Proceedings of the 35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'15)*, volume 45 of *LIPIcs*, pages 278–292. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.278.

14 Thomas Brihaye, Gilles Geeraerts, Axel Haddad, and Benjamin Monmege. Pseudopolynomial Iterative Algorithm to Solve Total-Payoff Games and Min-Cost Reachability Games. *Acta Informatica*, 2016. doi:10.1007/s00236-016-0276-z.

15 Thomas Brihaye, Gilles Geeraerts, Shankara Narayanan Krishna, Lakshmi Manasa, Benjamin Monmege, and Ashutosh Trivedi. Adding Negative Prices to Priced Timed Games. In *Proceedings of the 25th International Conference on Concurrency Theory (CONCUR'14)*, volume 8704, pages 560–575. Springer, 2014. doi:10.1007/978-3-662-44584-6_38.

16 Damien Busatto-Gaston, Benjamin Monmege, and Pierre-Alain Reynier. Optimal Reachability in Divergent Weighted Timed Games. In Javier Esparza and Andrzej S. Murawski, editors, *Proceedings of the 20th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'17)*, volume 10203 of *Lecture Notes in Computer Science*, pages 162–178, Uppsala, Sweden, April 2017. Springer. doi:10.1007/978-3-662-54458-7_10.

17 Thomas Dueholm Hansen, Rasmus Ibsen-Jensen, and Peter Bro Miltersen. A Faster Algorithm for Solving One-Clock Priced Timed Games. In *Proceedings of the 24th International Conference on Concurrency Theory (CONCUR'13)*, volume 8052 of *LNCS*, pages 531–545. Springer, 2013.

18 Marcin Jurdziński and Ashutosh Trivedi. Reachability-Time Games on Timed Automata. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP'07)*, volume 4596 of *LNCS*, pages 838–849. Springer, 2007.

19 Michał Rutkowski. Two-Player Reachability-Price Games on Single-Clock Timed Automata. In *Proceedings of the Ninth Workshop on Quantitative Aspects of Programming Languages (QAPL'11)*, volume 57 of *EPTCS*, pages 31–46, 2011.

A Symbolic Framework to Analyse Physical Proximity in Security Protocols

Alexandre Debant
Université de Rennes, CNRS, IRISA, France

Stéphanie Delaune
Université de Rennes, CNRS, IRISA, France

Cyrille Wiedling
Université de Rennes, CNRS, IRISA, France

Abstract

For many modern applications like e.g., contactless payment, and keyless systems, ensuring physical proximity is a security goal of paramount importance. Formal methods have proved their usefulness when analysing standard security protocols. However, existing results and tools do not apply to e.g., distance bounding protocols that aims to ensure physical proximity between two entities. This is due in particular to the fact that existing models do not represent in a faithful way the locations of the participants, and the fact that transmission of messages takes time.

In this paper, we propose several reduction results: when looking for an attack, it is actually sufficient to consider a simple scenario involving at most four participants located at some specific locations. These reduction results allow one to use verification tools (e.g. ProVerif, Tamarin) developed for analysing more classical security properties. As an application, we analyse several distance bounding protocols, as well as a contactless payment protocol.

2012 ACM Subject Classification Theory of computation → Program verification

Keywords and phrases cryptographic protocols, verification, formal methods, process algebra, Dolev-Yao model

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.29

Related Version A full version of the paper is available at [19], `https://hal.archives-ouvertes.fr/hal-01708336`.

Funding This work has been partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 714955-POPSTAR)

1 Introduction

The shrinking size of microprocessors and the ubiquity of wireless communication have led to the proliferation of portable computing devices with novel security requirements. Whereas traditional security protocols achieve their security goals relying solely on cryptographic primitives like encryptions and hash functions, this is not the case anymore for many modern applications like e.g., contactless payment. Actually, a typical attack against these devices is the so-called relay attack, as demonstrated for EMV in [14]. Such an attack allows a malicious participant to relay communications between a victim's card (possibly inside a wallet) and a genuine terminal so that the victim's card, even if it is far away from the terminal, will pay the transaction. Due to the contactless nature of most of our communications, obtaining reliable

information regarding physical proximity is of paramount importance and specific protocols, namely distance bounding protocols, were proposed to achieve this specific goal [11, 26]. They typically take into account the round trip time of messages and the transmission velocity to infer an upper bound of the distance between two participants.

In the context of standard security protocols, such as key establishment protocols, formal methods have proved their usefulness for providing security guarantees or detecting attacks. The purpose of formal verification is to provide rigorous frameworks and techniques to analyse protocols and find their flaws. For example, a flaw has been discovered in the Single-Sign-On protocol used e.g., by Google Apps [2]. This flaw has been found when analysing the protocol using formal symbolic methods, abstracting messages by a term algebra and using the Avantssar validation platform [3]. The techniques used in symbolic models have become mature and several verification tools are nowadays available, e.g., ProVerif [8], Tamarin [29].

However, protocols whose security relies on constraints from the physical world fall outside the scope of traditional symbolic models that are based on the omniscient attacker who controls the entire network, and who can for instance relay messages without introducing any delay. Following [7, 24], and more recently [27], our aim is to bridge the gap between informal approaches currently used to analyse these protocols and the formal approaches already used for analysing traditional security protocols.

Our contributions. To model timed protocols as well as the notion of physical proximity, we first propose a calculus in which communications are subject to physical restrictions. These constraints apply to honest agents and attackers. An attacker can only intercept messages at his location, and attackers can *not* instantaneously exchange their knowledge: transmitting messages takes time. Moreover each agent has clocks to be able to perform time measurements. Then, our main contribution is to provide reduction results in the spirit of the one obtained in [15] for traditional protocols: if there is an attack, then there is one considering only few participants at some specific locations. As it is usually done in distance bounding protocols, we consider different types of attacks : mafia fraud and hijacking attack. A mafia fraud is an attack in which an attacker tries to convince the verifier that an honest prover is close to him whereas he is far away. The notion of distance hijacking attack has been introduced more recently [17]. In such a scenario, a dishonest prover located far away succeeds in convincing a verifier that they are actually close, and he may only exploit the presence of honest participants in the neighboorhood to achieve his goal. Our results slightly differ depending on the type of attacks we consider and allow one to reduce the number of topologies to be considered from infinitely many to only one (involving at most 4 participants including the malicious ones). They hold in a rather general setting: we consider arbitrary cryptographic primitives as soon as they can be expressed using rewriting rules modulo an equational theory.

An interesting consequence of our reduction results is that it allows one to use techniques and tools developed so far for traditional security protocols. As an application, we analyse several distance bounding protocols (relying on the automatic ProVerif tool), and a contactless payment protocol [14]. We confirmed some known vulnerabilities in a number of protocols, and discovered an unreported attack on the SPADE protocol [12]. All files related to our case studies as well as a full version of this paper are available [18, 19].

Related work. Recent efforts have been made on proving security of distance bounding protocols. For instance, in 2011, Avoine *et al.* [5] proposed a framework in which many protocols have been analysed and compared in a unified manner [4]. A rather general model

has been proposed by Boureanu *et al.* in [10]. This computational model captures all the classical types of attacks and generalises them enabling attackers to interact with many provers and verifiers. These models are very different from ours. Indeed, we consider here a *formal symbolic model* in which messages are no longer bitstrings but are abstracted away by terms. Some recent attempts have been made to design formal symbolic model suitable to analyse distance bounding protocols: e.g., a model based on multiset rewriting rules has been proposed in [7] and [27], another one based on strand spaces is available in [31]. Even if our model shares some similarities with those mentioned above, we design a new one based on the applied pi calculus [1] in order to connect our theoretical results with the ProVerif verification tool that we ultimately use to analyse protocols.

Our main reduction result follows the spirit of [16] where it is shown that it is sufficient to consider five specific topologies when analysing routing protocols. To our knowledge, the only work proposing a reduction result suitable for distance bounding protocols is [31]: they show that n attackers are sufficient when analysing a configuration involving at most n honest participants. However, an arbitrary number of participants is still needed when looking for an attack. Moreover, due to the way attackers are located (close to each honest participant), such a result can not be applied to analyse distance hijacking scenarios for which the presence of an attacker in the neighbourhood of the verifier is disallowed. In contrast, our result reduces to only one topology, even when considering an arbitrary number of honest participants, and it applies to the scenario mentioned above. In particular, this allows us to leverage existing verification tools such as ProVerif and Tamarin. To do that we get some inspiration from [14]. Our contributions improve upon their work by providing a strong theoretical foundation to their idea.

Recently, a methodology to analyse distance bounding protocols within Tamarin has been proposed [27]. They do not try to define the different class of attacks as usually considered in distance bounding protocols. Instead, they provide a generic definition of secure distance bounding: when an honest verifier successfully ends a session with a prover P, then he has correctly computed an upper bound on his distance to either P (if P is honest) or to some other dishonest participant P' (if P is dishonest). The security analysis is then performed w.r.t. such a generic security property. This prevents them to draw meaningful conclusions on protocols such as Paysafe which is *not* supposed to resist to some particular classes of attacks.

2 Modelling timed security protocols

As usual in symbolic models, we rely on a term algebra for modelling messages exchanged by the participants, and on a process algebra to represent the protocols themselves.

2.1 Messages as terms

We consider two infinite and disjoint sets of *names*: a set $\mathcal{N}$ of *basic names* used to represent keys, nonces, and a set $\mathcal{A}$ of *agent names* used to represent agents identities. We consider an infinite set Σ_0 of constant symbols that are used e.g., to represent nonces drawn by the attacker. We also consider two infinite and disjoint sets of *variables*, denoted $\mathcal{X}$ and $\mathcal{W}$. Variables in $\mathcal{X}$ refer to unknown parts of messages expected by participants while those in $\mathcal{W}$, namely *handles*, are used to store messages learnt by the attacker.

We assume a signature Σ, *i.e.* a set of function symbols together with their arity. The elements of Σ are split into *constructor* and *destructor* symbols, *i.e.* $\Sigma = \Sigma_c \uplus \Sigma_d$. We denote $\Sigma^+ = \Sigma \uplus \Sigma_0$, and $\Sigma_c^+ = \Sigma_c \uplus \Sigma_0$. Given a signature $\mathcal{F}$, and a set of atomic data A, we

denote by $\mathcal{T}(\mathcal{F}, \mathsf{A})$ the set of *terms* built from atomic data A by applying function symbols in $\mathcal{F}$. A *constructor term* is a term in $\mathcal{T}(\Sigma_c^+, \mathcal{N} \uplus \mathcal{A} \uplus \mathcal{X})$. We denote $vars(u)$ the set of variables that occur in a term u. A *message* is a constructor term u that is *ground*, *i.e.* such that $vars(u) = \emptyset$. The application of a substitution σ to a term u is written $u\sigma$. We denote $dom(\sigma)$ its *domain*, and $img(\sigma)$ its *image*. The positions of a term are defined as usual.

▶ **Example 1.** We consider the following signature $\Sigma_{\mathsf{ex}} = \Sigma_c \uplus \Sigma_d$:

$$\Sigma_c = \{\mathsf{commit}, \mathsf{sign}, \mathsf{sk}, \mathsf{vk}, \mathsf{ok}, \langle\rangle, \oplus, 0\}, \qquad \Sigma_d = \{\mathsf{open}, \mathsf{getmsg}, \mathsf{check}, \mathsf{proj}_1, \mathsf{proj}_2, \mathsf{eq}\}.$$

The symbols open and commit (arity 2) represent a commitment scheme, whereas the symbols sign, check (arity 2), getmsg, sk, and vk (arity 1) are used to model signature. Pairing and projections are modelled using $\langle\ \rangle$ (arity 2), and proj_i with $i \in \{1, 2\}$ (arity 1). The symbols $\oplus$ (arity 2) and the constant 0 model the exclusive-or operator. We consider the symbol eq to model equality test and ok a specific constant.

Following the approach developed in [9], constructor terms are subject to an *equational theory*. This allows one to model the algebraic properties of the primitives. It consists of a finite set of equations of the form $u = v$ where $u, v \in \mathcal{T}(\Sigma_c, \mathcal{X})$, and induces an equivalence relation $=_{\mathsf{E}}$ over constructor terms. Formally, $=_{\mathsf{E}}$ is the smallest congruence on constructor terms, which contains $u = v$ in E, and that is closed under substitutions of terms for variables.

▶ **Example 2.** To reflect the algebraic properties of the exclusive-or operator, we consider the equational theory $\mathsf{E}_{\mathsf{xor}}$ generated by the following equations:

$$(x \oplus y) \oplus z = x \oplus (y \oplus z) \qquad x \oplus y = y \oplus x \qquad x \oplus 0 = x \qquad x \oplus x = 0.$$

We also give a meaning to destructor symbols through a set of rewriting rules of the form $\mathsf{g}(t_1, \ldots, t_n) \to t$ where $\mathsf{g} \in \Sigma_d$, and $t, t_1, \ldots, t_n \in \mathcal{T}(\Sigma_c, \mathcal{X})$. A term u can be *rewritten* in v if there is a position p in u, and a rewriting rule $\mathsf{g}(t_1, \ldots, t_n) \to t$ such that $u|_p = \mathsf{g}(t_1, \ldots, t_n)\theta$ for some substitution θ. Moreover, we assume that $t_1\theta, \ldots, t_n\theta$ as well as $t\theta$ are messages. We only consider sets of rewriting rules that yield a *convergent* rewriting system, and we denote $u{\downarrow}$ the *normal form* of a term u. For modelling purposes, we split the signature Σ into two parts, Σ_{pub} and Σ_{priv}, and we denote $\Sigma_{\mathsf{pub}}^+ = \Sigma_{\mathsf{pub}} \cup \Sigma_0$. An attacker builds messages by applying public symbols to terms he knows and that are available through handles in $\mathcal{W}$. Formally, a computation done by the attacker is a *recipe*, *i.e.* a term in $\mathcal{T}(\Sigma_{\mathsf{pub}}^+, \mathcal{W})$.

▶ **Example 3.** Among symbols in Σ_{ex}, only sk is in Σ_{priv}. The properties of the symbols in Σ_d are reflected through the following rewriting rules:

$$\begin{array}{rlll} \mathsf{check}(\mathsf{sign}(x, \mathsf{sk}(y)), \mathsf{vk}(y)) \to \mathsf{ok} & \mathsf{getmsg}(\mathsf{sign}(x, \mathsf{sk}(y))) \to x & \mathsf{proj}_1(\langle x_1, x_2\rangle) \to x_1 \\ \mathsf{eq}(x, x) \to \mathsf{ok} & \mathsf{open}(\mathsf{commit}(x, y), y) \to x & \mathsf{proj}_2(\langle x_1, x_2\rangle) \to x_2. \end{array}$$

2.2 Protocols as processes

Protocols are modelled through processes using the following grammar:

$$\begin{array}{rcllllll} P, Q & := & 0 & | & \mathtt{in}(x).P & | & \mathtt{in}^{<t}(x).P & | & \mathtt{let}\ x = v\ \mathtt{in}\ P \\ & | & \mathtt{new}\ n.P & | & \mathtt{out}(u).P & | & \mathtt{reset}.P & & \end{array}$$

where $x \in \mathcal{X}$, $n \in \mathcal{N}$, $u \in \mathcal{T}(\Sigma_c^+, \mathcal{X} \uplus \mathcal{N} \uplus \mathcal{A})$, $v \in \mathcal{T}(\Sigma^+, \mathcal{X} \uplus \mathcal{N} \uplus \mathcal{A})$ and $t \in \mathbb{R}_+$.

Most of these constructions are rather standard. As usual, 0 denotes the empty process that does nothing, and the `new` instruction is used to model fresh name generation. Then, we have standard constructions to model inputs and outputs. We may note the special

construction $\texttt{in}^{<t}(x)$ that combines an input with a constraint on the local clock of the process executing this action. This construction is in contrast with the approach proposed in [27] where input actions are not subject to any timing constraint, and are therefore always possible provided that enough time has elapsed. From this point of view, our model represents the reality more faithfully since an agent will not proceed an input arriving later than expected. The `reset` instruction will reset the local clock of the process. Finally, the process $\texttt{let}\ x = v\ \texttt{in}\ P$ tries to evaluate v, the process P is executed in case of success, and the process is blocked otherwise. Note that the usual conditional operator can be modelled as follows: $\texttt{let}\ x = \mathsf{eq}(u, v)\ \texttt{in}\ P$.

We write $fv(P)$ (resp. $fn(P)$) for the set of *free* variables (resp. names) occurring in P, *i.e.* the set of variables (resp. names) that are not in the scope of an `in` or a `let` (resp. a `new`). We consider *parametrised processes*, denoted $P(z_0, \ldots, z_n)$, where $z_0, \ldots, z_n$ are variables from a special set $\mathcal{Z}$ (disjoint from $\mathcal{X}$ and $\mathcal{W}$). Intuitively, these variables will be instantiated by agent names, and z_0 corresponds to the name of the agent that executes the process. A *role* $R = P(z_0, \ldots, z_n)$ is a parametrised process that does not contain any agent name, and such that $fv(R) \subseteq \{z_0, \ldots, z_n\}$. A *protocol* is a set of roles.

▶ **Example 4.** As a running example, we consider the signature-based Brands and Chaum distance bounding protocol [11] that is informally described below.

1. $P \to V$: $\mathsf{commit}(m, k)$
2. $V \to P$: n
3. $P \to V$: $n \oplus m$
4. $P \to V$: k
5. $P \to V$: $\mathsf{sign}(\langle n, n \oplus m\rangle, \mathsf{sk}(P))$.

The prover P generates a nonce m and a key k, and sends a commitment to the verifier V. The verifier V generates his own nonce n and initiates the time measurement phase, also called the *rapid phase*. P has to provide an answer as quickly as possible since V will reject any answer arriving too late (a long response time does not give him any guarantee regarding its proximity with the prover). After this phase, P sends a means to open the commitment, as well as a signature on the values exchanged during the rapid phase. When a verifier ends the protocol, the prover with whom he is communicating should be located in his neighbourhood. In our setting, this 2-party protocol is modelled through the two following parametrised processes: $V(z_V, z_P)$ represents the role of the verifier played by agent z_V with agent z_P whereas $P(z'_P)$ represents the role of the prover played by agent z'_P.

$V(z_V, z_P) :=$
$\texttt{in}(y_c).\texttt{new}\ n.$
$\texttt{reset}.\texttt{out}(n).\texttt{in}^{<2\times t_0}(y_0).$
$\texttt{in}(y_k).\texttt{in}(y_{\mathsf{sign}}).$
$\texttt{let}\ y_m = \mathsf{open}(y_c, y_k)\ \texttt{in}$
$\texttt{let}\ y_{\mathsf{check}} = \mathsf{check}(y_{\mathsf{sign}}, \mathsf{vk}(z_P))\ \texttt{in}$
$\texttt{let}\ y_{\mathsf{eq}} = \mathsf{eq}(\langle n, n \oplus y_m\rangle, \mathsf{getmsg}(y_{\mathsf{sign}}))\ \texttt{in}\ 0.$

$P(z'_P) :=$
$\texttt{new}\ m.\texttt{new}\ k.$
$\texttt{out}(\mathsf{commit}(m, k)).$
$\texttt{in}(x_n).$
$\texttt{out}(x_n \oplus m).$
$\texttt{out}(k).$
$\texttt{out}(\mathsf{sign}(\langle x_n, x_n \oplus m\rangle, \mathsf{sk}(z'_P))).0$

2.3 Semantics

The operational semantics is defined using a relation over configurations, and is parametrised by a topology reflecting the fact that interactions between agents depend on their location.

▶ **Definition 5.** A *topology* is a tuple $\mathcal{T}_0 = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0)$ where:

- $\mathcal{A}_0 \subseteq \mathcal{A}$ is the finite set of agents composing the system;
- $\mathcal{M}_0 \subseteq \mathcal{A}_0$ is the subset of agents that are dishonest;
- $\mathsf{Loc}_0 : \mathcal{A}_0 \to \mathbb{R}^3$ is a mapping defining the position of each agent in space.
- p_0 and v_0 are two agents in $\mathcal{A}_0$ that represent respectively the prover and the verifier for which we analyse the security of the protocol.

In our model, the distance between two agents is expressed by the time it takes for a message to travel from one to another, $\mathsf{Dist}_{\mathcal{T}_0} : \mathcal{A}_0 \times \mathcal{A}_0 \to \mathbb{R}$, defined as follows:

$$\mathsf{Dist}_{\mathcal{T}_0}(a,b) = \frac{\|\mathsf{Loc}_0(a) - \mathsf{Loc}_0(b)\|}{c_0} \text{ for any } a, b \in \mathcal{A}_0$$

with $\|\cdot\| : \mathbb{R}^3 \to \mathbb{R}$ the euclidian norm and c_0 the transmission speed. We suppose, from now on, that c_0 is a constant for all agents, and thus an agent a can recover, at time t, any message emitted by any other agent b before $t - \mathsf{Dist}_{\mathcal{T}_0}(a,b)$.

Note that our model is not restricted to a single dishonest node. In particular, our results apply to the case of several compromised nodes that communicate (and therefore share their knowledge). However, communication is subject to physical constraints. This results in a distributed attacker with restricted, but more realistic, communication capabilities than those of the traditional omniscient Dolev-Yao attacker [21].

Our semantics is given by a transition system over configurations that manipulates *extended processes*, i.e. expressions of the form $\lfloor P_a \rfloor_a^{t_a}$ with $a \in \mathcal{A}$, P_a a process such that $fv(P_a) = \emptyset$, and $t_a \in \mathbb{R}_+$. Intuitively, P_a describes the actions of agent a, and t_a his local clock. Messages that have been outputted so far are stored into a *frame* (introduced in [1]) extended to keep track of the time at which the message has been outputted and by whom.

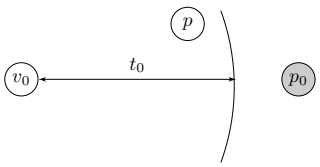
▶ **Definition 6.** Given a topology $\mathcal{T}_0 = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0)$, a *configuration* K over $\mathcal{T}_0$ is a tuple $(\mathcal{P}; \Phi; t)$, where:

- $\mathcal{P}$ is a multiset of *extended process* $\lfloor P \rfloor_a^{t_a}$ with $a \in \mathcal{A}_0$;
- $\Phi = \{\mathsf{w}_1 \xrightarrow{a_1,t_1} u_1, \ldots, \mathsf{w}_n \xrightarrow{a_n,t_n} u_n\}$ is an *extended frame*, i.e. a substitution such that $\mathsf{w}_i \in \mathcal{W}$, $u_i \in \mathcal{T}(\Sigma_c^+, \mathcal{N} \uplus \mathcal{A})$, $a_i \in \mathcal{A}_0$ and $t_i \in \mathbb{R}_+$ for $1 \leq i \leq n$;
- $t \in \mathbb{R}_+$ is the global time of the system.

We write $\lfloor \Phi \rfloor_a^t$ for the restriction of Φ to the agent a at time t, *i.e.* :

$$\lfloor \Phi \rfloor_a^t = \left\{ \mathsf{w}_i \xrightarrow{a_i,t_i} u_i \mid (\mathsf{w}_i \xrightarrow{a_i,t_i} u_i) \in \Phi \text{ and } a_i = a \text{ and } t_i \leq t \right\}.$$

▶ **Example 7.** Continuing Example 4, we may consider the topology $\mathcal{T}_0 = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0)$ depicted below where $\mathcal{A}_0 = \{p_0, v_0, p\}$, and $\mathcal{M}_0 = \{p_0\}$. The precise location of each agent is not relevant, only the distance between them matters. Here $\mathsf{Dist}_{\mathcal{T}_0}(p, v_0) < t_0$ whereas $\mathsf{Dist}_{\mathcal{T}_0}(p_0, v_0) \geq t_0$.



A typical initial configuration is:

$$K_0 = (\lfloor P(p) \rfloor_p^0 \uplus \lfloor V(v_0, p_0) \rfloor_{v_0}^0 ; \{\mathsf{w}_1 \xrightarrow{p_0,0} \mathsf{sk}(p_0)\}; 0)$$

$$\text{TIM} \quad (\mathcal{P};\Phi;t) \longrightarrow_{\mathcal{T}_0} (\mathsf{Shift}(\mathcal{P},\delta);\Phi;t+\delta) \qquad \text{with } \delta \geq 0$$

$$\text{OUT} \quad (\lfloor \mathtt{out}(u).P \rfloor_a^{t_a} \uplus \mathcal{P};\Phi;t) \xrightarrow{a,\mathtt{out}(u)}_{\mathcal{T}_0} (\lfloor P \rfloor_a^{t_a} \uplus \mathcal{P};\Phi \uplus \{\mathsf{w} \xrightarrow{a,t} u\};t) \quad \text{with } \mathsf{w} \in \mathcal{W} \text{ fresh}$$

$$\text{LET} \quad (\lfloor \mathtt{let}\ x = u\ \mathtt{in}\ P \rfloor_a^{t_a} \uplus \mathcal{P};\Phi;t) \xrightarrow{a,\tau}_{\mathcal{T}_0} (\lfloor P\{x \mapsto u{\downarrow}\} \rfloor_a^{t_a} \uplus \mathcal{P};\Phi;t) \quad \text{when } u{\downarrow} \in \mathcal{T}(\Sigma_c^+, \mathcal{N} \uplus \mathcal{A})$$

$$\text{NEW} \quad (\lfloor \mathtt{new}\ n.P \rfloor_a^{t_a} \uplus \mathcal{P};\Phi;t) \xrightarrow{a,\tau}_{\mathcal{T}_0} (\lfloor P\{n \mapsto n'\} \rfloor_a^{t_a} \uplus \mathcal{P};\Phi;t) \text{ with } n' \in \mathcal{N} \text{ fresh}$$

$$\text{RST} \quad (\lfloor \mathtt{reset}.P \rfloor_a^{t_a} \uplus \mathcal{P};\Phi;t) \xrightarrow{a,\tau}_{\mathcal{T}_0} (\lfloor P \rfloor_a^{0} \uplus \mathcal{P};\Phi;t)$$

$$\text{IN} \quad (\lfloor \mathtt{in}^\star(x).P \rfloor_a^{t_a} \uplus \mathcal{P};\Phi;t) \xrightarrow{a,\mathtt{in}^\star(u)}_{\mathcal{T}_0} (\lfloor P\{x \mapsto u\} \rfloor_a^{t_a} \uplus \mathcal{P};\Phi;t)$$

when there exist $b \in \mathcal{A}_0$ and $t_b \in \mathbb{R}_+$ such that $t_b \leq t - \mathsf{Dist}_{\mathcal{T}_0}(b,a)$ and:
- if $b \in \mathcal{A}_0 \setminus \mathcal{M}_0$ then $u \in img(\lfloor \Phi \rfloor_b^{t_b})$;
- if $b \in \mathcal{M}_0$ then $u = R\Phi{\downarrow}$ for some recipe R such that for all $\mathsf{w} \in vars(R)$ there exists $c \in \mathcal{A}_0$ such that $\mathsf{w} \in dom(\lfloor \Phi \rfloor_c^{t_b - \mathsf{Dist}_{\mathcal{T}}(c,b)})$.

Moreover, in case $\star$ is $< t_g$ for some t_g, we assume in addition that $t_a < t_g$.

Figure 1 Semantics of our calculus.

where p is playing the prover's role, and v_0 the verifier's role with a dishonest agent p_0. The signature key of this dishonest participant is given to the attacker through w_1. A more realistic configuration would include other instances of these two roles and will give more knowledge to the attacker. This simple configuration is sufficient to present an attack.

Given a topology $\mathcal{T}_0 = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0)$, the semantics of processes is formally defined by the rules given in Figure 1.

The TIM rule allows time to elapse, meaning that the global clock as well as the local clocks will be shifted by δ:

$$\mathsf{Shift}(\mathcal{P},\delta) = \biguplus_{\lfloor P \rfloor_a^{t_a} \in \mathcal{P}} \mathsf{Shift}(\lfloor P \rfloor_a^{t_a},\delta) \text{ and } \mathsf{Shift}(\lfloor P \rfloor_a^{t_a},\delta) = \lfloor P \rfloor_a^{t_a+\delta}.$$

The RST rule allows an agent to reset the local clock of the process. The other rules are rather standard. The IN rule allows an agent a to evolve when receiving a message: the received message has necessarily been forged and sent at time t_b by some agent b who was in possession of all the necessary information at that time.

We sometimes simply write $\to_{\mathcal{T}_0}$ instead of $\xrightarrow{a,\alpha}_{\mathcal{T}_0}$. The relation $\to^*_{\mathcal{T}_0}$ is the reflexive and transitive closure of $\to_{\mathcal{T}_0}$, and we often write $\xrightarrow{\mathsf{tr}}_{\mathcal{T}_0}$ to emphasise the sequence of labels tr that has been used during this execution.

▶ **Example 8.** Continuing Example 7, we may consider the following execution:

$$K_0 \xrightarrow{p,\tau}_{\mathcal{T}_0} \xrightarrow{p,\tau}_{\mathcal{T}_0} \xrightarrow{p,\mathtt{out}(\mathsf{commit}(m',k'))}_{\mathcal{T}_0} \to_{\mathcal{T}_0} \xrightarrow{v_0,\mathtt{in}(\mathsf{commit}(m',k'))}_{\mathcal{T}_0} \xrightarrow{v_0,\tau}_{\mathcal{T}_0} \xrightarrow{v_0,\tau}_{\mathcal{T}_0} K_1$$

where $K_1 = (\lfloor P_1 \rfloor_p^{\delta_0} \uplus \lfloor V_1 \rfloor_{v_0}^{0}; \{\mathsf{w}_1 \xrightarrow{p_0,0} \mathsf{sk}(p_0),\ \mathsf{w}_2 \xrightarrow{p,0} \mathsf{commit}(m',k')\}; \delta_0)$. The two first arrows correspond to applications of the rule NEW to generate m' and k', the one without label is an instance of the TIM rule, and the two last arrows correspond respectively to the rule NEW and the rule RST. We have that:
- $P_1 = \mathtt{in}(x_n).\mathtt{out}(x_n \oplus m').\mathtt{out}(k').\mathtt{out}(\mathsf{sign}(\langle x_n, x_n \oplus m' \rangle, \mathsf{sk}(p)))$; and
- $V_1 = \mathtt{out}(n').\mathtt{in}^{<2\times t_0}(y_0).\mathtt{in}(y_k).\mathtt{in}(y_{\mathsf{sign}}).\mathtt{let}\ y_m = \mathsf{open}(\mathsf{commit}(m',k'),y_k)\ \mathtt{in} \ldots$

This models the beginning of a normal execution between p and v_0. The message outputted at location p is received at location v_0. The instance of the rule TIM in between (here with $\delta_0 = \mathsf{Dist}_{\mathcal{T}_0}(p, v_0)$) allows the message to reach location v_0.

3 Modelling physical proximity

A distance bounding protocol is a protocol in which a party (the verifier) is assured of the identity of another party (the prover), as well as the fact that this prover is located in his neighbourhood. Before to consider the type of frauds we are interested in, we first introduce the notion of valid initial configuration that aims to represent the scenarios that have to be analysed once the topology is fixed.

3.1 Valid initial configurations

We consider a topology $\mathcal{T}_0 = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0)$, a protocol $\mathcal{P}_{\mathsf{prox}}$ i.e. a set of roles), and we assume that the initial knowledge of dishonest participants is given through a template $\mathcal{I}_0$, i.e. a set of terms in $\mathcal{T}(\Sigma_c^+, \mathcal{Z})$. Using this template $\mathcal{I}_0$, and considering a set of agents $\mathcal{A}_0$, we derive the initial knowledge of agent $a \in \mathcal{A}_0$ as follows:

$$\mathsf{Knows}(\mathcal{I}_0, a, \mathcal{A}_0) = \left\{ (u_0\{z_0 \mapsto a\})\sigma \text{ ground} \;\middle|\; \begin{array}{l} u_0 \in \mathcal{I}_0 \text{ and} \\ \sigma \text{ a substitution such that } img(\sigma) \subseteq \mathcal{A}_0 \end{array} \right\}$$

For the sake of simplicity, we extend our calculus with a special action of the form $\mathsf{end}(z_0, z_1)$ and we assume that configurations representing instances of distance bounding protocols contain a process (typically a session of the verifier) that ends with it. When analysing physical proximity, we consider any valid initial configuration as defined below:

▶ **Definition 9.** Let $\mathcal{P}_{\mathsf{prox}}$ be a protocol, $V_0(z_0, z_1)$ be a parametrised role containing the special action $\mathsf{end}(z_0, z_1)$, $\mathcal{I}_0$ be a template, and $\mathcal{T}_0 = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0)$ be a topology. A configuration $K = (\mathcal{P}; \Phi; t)$ is a *valid initial configuration* for the protocol $\mathcal{P}_{\mathsf{prox}}$ and V_0 w.r.t. $\mathcal{T}_0$ and $\mathcal{I}_0$ if:

1. $\mathcal{P} = \lfloor V_0(v_0, p_0) \rfloor_{v_0}^{t'} \uplus \mathcal{P}'$ for some t' and for each $\lfloor P' \rfloor_{a'}^{t_{a'}} \in \mathcal{P}'$ there exists $P(z_0, \ldots, z_k) \in \mathcal{P}_{\mathsf{prox}}$, and $a_1, \ldots, a_k \in \mathcal{A}_0$ such that $P' = P(a', a_1, \ldots, a_k)$.
2. $img(\lfloor \Phi \rfloor_a^t) = \mathsf{Knows}(\mathcal{I}_0, a, \mathcal{A}_0)$ when $a \in \mathcal{M}_0$, and $img(\lfloor \Phi \rfloor_a^t) = \emptyset$ otherwise.

The first condition says that we consider initial configurations made up of instances of the roles of the protocols, and we only consider roles executed by agents located at the right place, i.e. the agent a' who executes the role must be the first argument of the parametrised process. The second condition allows one to give some initial knowledge to each malicious node. We may note that we do not give any constraint regarding time. It is indeed important that all the possible initial configurations are analysed before declaring a protocol secure.

▶ **Example 10.** Going back to Example 7 and considering the template $\mathcal{I}_0 = \{\mathsf{sk}(z_0)\}$, we have that K_0 is a valid initial configuration.

3.2 Mafia fraud and distance hijacking

A *mafia fraud* [20] is an attack in which generally three agents are involved: a verifier, an honest prover located outside the neighbourhood of the verifier, and an attacker. We consider here its general version which may involve an arbitrary number of participants. The

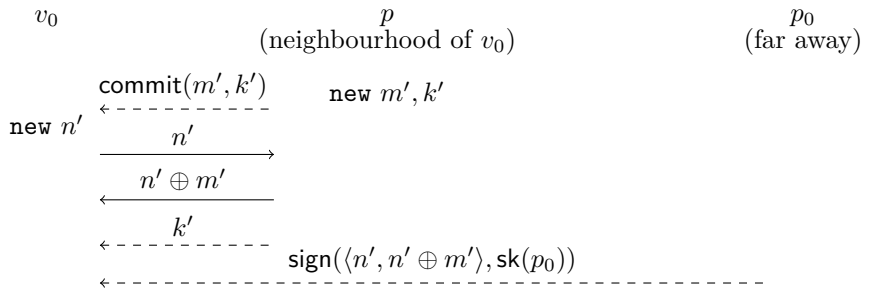


Figure 2 Distance hijacking attack on the Brands and Chaum's protocol.

aim of the attacker is to convince the verifier that the honest prover is actually close to it. We denote by $\mathcal{C}_{\mathsf{MF}}$ the set of all the mafia fraud topologies, i.e. any topology $\mathcal{T}$ such that $\mathcal{T} = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0)$ with $v_0, p_0 \in \mathcal{A}_0 \setminus \mathcal{M}_0$.

A *distance hijacking fraud* [17] is an attack in which a dishonest prover located far away succeeds in convincing a verifier that he is actually close to him. The dishonest prover may exploit honest entities located in the neighbourhood of the verifier. We denote by $\mathcal{C}_{\mathsf{DH}}$ the set of all the distance hijacking topologies, i.e. any topology $\mathcal{T}$ such that $\mathcal{T} = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0)$ with $p_0 \in \mathcal{M}_0$, $v_0 \in \mathcal{A}_0 \setminus \mathcal{M}_0$, and $\mathsf{Dist}_{\mathcal{T}_0}(v_0, a) \geq t_0$ for any $a \in \mathcal{M}_0$.

▶ **Definition 11.** Let $\mathcal{P}_{\mathsf{prox}}$ be a protocol, $V_0(z_0, z_1)$ be a parametrised role containing the special event $\mathsf{end}(z_0, z_1)$, and $\mathcal{I}_0$ be a template. We say that $\mathcal{P}_{\mathsf{prox}}$ admits a mafia fraud attack (resp. distance hijacking attack) w.r.t. t_0-proximity if there exist $\mathcal{T} \in \mathcal{C}_{\mathsf{MF}}$ (resp. $\mathcal{C}_{\mathsf{DH}}$), a valid initial configuration K for $\mathcal{P}_{\mathsf{prox}}$ and V_0 w.r.t. $\mathcal{T}$ and $\mathcal{I}_0$ such that:

$$K \to_{\mathcal{T}}^{*} (\lfloor \mathsf{end}(a_1, a_2) \rfloor_a^{t_a} \uplus \mathcal{P}; \Phi; t) \text{ with } \mathsf{Dist}_{\mathcal{T}}(a_1, a_2) \geq t_0.$$

We also say that K admits an attack w.r.t. t_0-proximity in $\mathcal{T}$.

In other words, there is an attack if starting from an initial valid configuration, the verifier a_1 successfully ends a session with an agent a_2 who is far away.

▶ **Example 12.** As reported in [17], the Brands and Chaum protocol is actually vulnerable to a *distance hijacking attack*. This attack is informally depicted in Figure 2. A honest prover who is in the neightbouhood of a legitimate verifier v_0 starts a session. At the end of the session, the dishonest prover p_0 who is far away hijacks the honest prover by sending a signature of the transcript of the rapid phase with his own signature key, namley $\mathsf{sk}(p_0)$. Upon reception of this signature, the verifier v_0 will believe that he played the session with p_0, and will wrongly conclude that p_0 is in his neighbourhood. Note that, the rapid phase (plain lines) can only be done by a prover who is in the neighbourhood of v_0 due to the guarded input that occurs in the verifer's role played by v_0.

We explain below how this attack is captured in our model. Continuing Example 7, we consider the configuration K_0' below:

$$K_0' = (\lfloor P(p) \rfloor_p^0 \uplus \lfloor V'(v_0, p_0) \rfloor_{v_0}^0; \{\mathsf{w}_1 \xrightarrow{p_0, 0} \mathsf{sk}(p_0)\}; 0)$$

where $V'(z_V, z_P)$ is $V(z_V, z_P)$ in which the null process has been replaced by $\mathsf{end}(z_V, z_P)$. The configuration K_0' can still follow the execution of Example 8:

$$K_1' = (\lfloor P_1 \rfloor_p^{\delta_0} \uplus \lfloor V_1' \rfloor_{v_0}^0; \{\mathsf{w}_1 \xrightarrow{p_0, 0} \mathsf{sk}(p_0),\ \mathsf{w}_2 \xrightarrow{p, 0} \mathsf{commit}(m', k')\}; \delta_0)$$

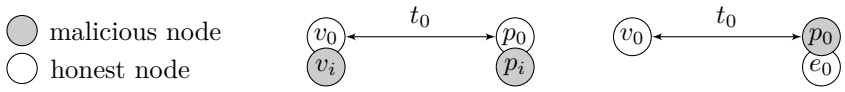


Figure 3 Topologies $\mathcal{T}_{\mathsf{MF}}^{t_0}$ (mafia fraud), and $\mathcal{T}_{\mathsf{DH}}^{t_0}$ (distance hijacking fraud).

where V_1' is V_1 in which the occurrence of the null process has been replaced by $\mathsf{end}(v_0, p_0)$. Now, we can pursue this execution as follows:

$$K_1' \xrightarrow{v_0,\mathtt{out}(n')} \mathcal{T}_0 \to \mathcal{T}_0 \xrightarrow{p,\mathtt{in}(n')} \mathcal{T}_0$$
$$\xrightarrow{p,\mathtt{out}(n'\oplus m')} \mathcal{T}_0 \xrightarrow{p,\mathtt{out}(k')} \mathcal{T}_0 \to \mathcal{T}_0 \xrightarrow{v_0,\mathtt{in}^{<2\times t_0}(n'\oplus m')} \mathcal{T}_0 \xrightarrow{v_0,\mathtt{in}(k')} \mathcal{T}_0$$
$$\to \mathcal{T}_0 \xrightarrow{v_0,\mathtt{in}(\mathsf{sign}(\langle n', n'\oplus m'\rangle,\mathsf{sk}(p_0)))} \mathcal{T}_0 \ (\lfloor P_2 \rfloor_p^{3\delta_0+2\delta_0'} \uplus \lfloor \mathsf{end}(v_0,p_0)\rfloor_{v_0}^{2\delta_0+2\delta_0'}; \Phi; 3\delta_0 + 2\delta_0').$$

The two first lines correspond to a normal execution of the protocol between v_0 and p. Note that, on each line, we need an instance of the TIM rule with $\delta_0 = \mathsf{Dist}_{\mathcal{T}_0}(v_0, p) = \mathsf{Dist}_{\mathcal{T}_0}(p, v_0)$ to allow the sent message to reach its destination and the guarded input passes because p is close to v_0. The last transition does not follow the normal execution of the protocol. Actually, the dishonest agent p_0 is responsible of this input. He built this message from the messages n' and $n' \oplus m'$ that have been sent on the network, and the key $\mathsf{sk}(p_0)$ that is part of his initial knowledge. Note that he has to wait the necessary amount of time to allow these messages to reach him (e.g. $\delta_0' = \mathsf{Dist}_{\mathcal{T}_0}(v_0, p_0)$), and some time is needed for the forged message to reach v_0 (actually $\delta_0' = \mathsf{Dist}_{\mathcal{T}_0}(v_0, p_0)$). Therefore, the first rule of the last line is an instance of the TIM rule during which a delay of $2\delta_0'$ has elapsed.

4 Reducing the topology

Our reduction results allow one to analyse the security of a protocol (w.r.t. t_0-proximity) considering only a specific and rather simple topology (see Figure 3) without missing any attacks.

4.1 Mafia fraud

A simple idea to reduce towards the topology $\mathcal{T}_{\mathsf{MF}}^{t_0}$ would be to move each node n in the neighbourhood of v_0 at the same location as v_0, and to keep the other ones at distance (i.e. location of p_0). However, such a reduction will lengthen the distance between n and p_0, and the resulting execution could not be feasible anymore. Since dishonest participants are allowed in the neighbourhood of v_0, getting inspiration from [31], we consider a dishonest participant right next to each honest participant. Such a dishonest participant is ideally located to forge and send messages that will be received by honest agents close to him.

However, contrary to the result provided in [31], our goal is not only to reduce the number of dishonest agents but also the number of honest agents that are involved in an attack trace. In order to ensure that moving (and reducing) the honest agents will lead to a feasible execution, we need an extra assumption. We require that each role of the protocol is executable. This is a reasonable assumption that only put weird protocols aside. This allows us to discard any role executed by a malicious participant. Intuitively, all these operations will be done directly by the attacker.

▶ **Definition 13.** Given a template $\mathcal{I}_0 = \{u_1, \ldots, u_k\}$, we say that a parametrised role $P(z_0, ..., z_n)$ is $\mathcal{I}_0$-*executable* if $fv(P) \subseteq \{z_0, ..., z_n\}$, $fn(P) = \emptyset$ and for any term u (resp. v) occurring in an `out` or a `let` construction, there exists a recipe $R \in \mathcal{T}(\Sigma_{\mathsf{pub}}^+, \{\mathsf{w}_1, \ldots, \mathsf{w}_k\} \uplus \mathcal{N} \uplus \mathcal{X})$ such that $u = R\sigma\downarrow$ (resp. $v\downarrow = R\sigma\downarrow$) where $\sigma = \{\mathsf{w}_1 \mapsto u_1, \ldots, \mathsf{w}_k \mapsto u_k\}$.

A protocol $\mathcal{P}$ is $\mathcal{I}_0$-*executable* if each role of $\mathcal{P}$ is $\mathcal{I}_0$-executable.

▶ **Example 14.** Going back to our running example given in Example 4. We have that $V(z_0, z_1)$ is $\{z_1\}$-executable, and $P(z_0)$ is $\{\mathsf{sk}(z_0)\}$-executable. Thus, the protocol made of these two roles is $\{\mathsf{sk}(z_0), z_1\}$-executable.

We are now able to state our main reduction result regarding mafia fraud.

▶ **Theorem 15.** *Let $\mathcal{I}_0$ be a template, $\mathcal{P}_{\mathsf{prox}}$ be a protocol $\mathcal{I}_0$-executable, and $V_0(z_0, z_1)$ be a parametrised role containing the special event $\mathsf{end}(z_0, z_1)$. We have that $\mathcal{P}_{\mathsf{prox}}$ admits a mafia fraud attack w.r.t. t_0-proximity, if and only if, there is an attack against t_0-proximity in the topology $\mathcal{T}^{t_0}_{\mathsf{MF}}$.*

Proof. Since $\mathcal{T}^{t_0}_{\mathsf{MF}} \in \mathcal{C}_{\mathsf{MF}}$, we have that the existence of an attack in $\mathcal{T}^{t_0}_{\mathsf{MF}}$ is a mafia fraud. Regarding the other direction, we present the main steps of the proof below. A detailed proof is available in [19].

We consider an attack trace in $\mathcal{T} = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0) \in \mathcal{C}_{\mathsf{MF}}$.

$K_0 \to^*_{\mathcal{T}} (\lfloor \mathsf{end}(v_0, p_0) \rfloor^{t_v}_{v_0} \uplus \mathcal{P}; \Phi; t)$ with $\mathsf{Dist}_{\mathcal{T}}(v_0, p_0) \geq t_0$.

We proceed in three main steps:

1. We reduce the number of active agents (those that are actually executing a process) - we do this for honest and malicious agents. We transform honest agent (but v_0 and p_0) into malicious ones. This intuitively gives more power to the attacker, and malicious agents in the neighborhood of v_0 are allowed in a mafia fraud scenario. Then, relying on our executability condition, we discard processes executed by malicious agents. These actions can actually be mimicked by an attacker located at the same place.
2. In the spirit of [31], we reduce the number of attackers by placing them ideally (one close to each honest agent). Since we have removed all honest agents but two, we obtain a topology with only two dishonest agents.
3. To conclude, we reduce the knowledge showing that we can project all the dishonest agents that are located in p_0 on p_i and all the dishonest agents that are located in v_0 on v_i. ◀

4.2 Distance hijacking attack

First, we may note that the reduction we did in case of mafia fraud is not possible anymore. There is no hope to reduce the number of attackers by placing them close to each honest participant since the addition of a malicious node in the neighbourhood of v_0 is not authorised when considering distance hijacking. Actually, adding such a dishonest node in the neighbourhood of v_0 will always introduce a false attack since in our model dishonest participants share their knowledge. Therefore, this dishonest participant would be able to impersonate the dishonest prover p_0 (who is actually far away).

Given a process P, we denote $\overline{P}$ the process obtained from P by removing `reset` instructions, and replacing all the occurrences of $\mathtt{in}^{<t}(x)$ by $\mathtt{in}(x)$. This transformation will be applied on the protocol but not on the role V_0 for which these instructions play a crucial role. Our reduction result ensures that no distance hijacking attack will be missed if we just analyse the transformed protocol in topology $\mathcal{T}^{t_0}_{\mathsf{DH}}$.

▶ **Theorem 16.** *Let $\mathcal{I}_0$ be a template, $\mathcal{P}_{\mathsf{prox}}$ be a protocol, $t_0 \in \mathbb{R}_+$, and $V_0(z_0, z_1)$ be a parametrised role obtained using the following grammar:*

$$\begin{array}{rcl} P, Q & := & \mathsf{end}(z_0, z_1) \mid \mathtt{in}(x).P \mid \mathtt{let}\ x = v\ \mathtt{in}\ P \\ & \mid & \mathtt{new}\ n.P \mid \mathtt{out}(u).P \mid \mathtt{reset}.\mathtt{out}(u').\mathtt{in}^{<t}(x).P \end{array}$$

where $x \in \mathcal{X}$, $n \in \mathcal{N}$, $u, u' \in \mathcal{T}(\Sigma_c^+, \mathcal{X} \cup \mathcal{N} \cup \{z_0, z_1\})$, $v \in \mathcal{T}(\Sigma^+, \mathcal{X} \cup \mathcal{N} \cup \{z_0, z_1\})$ and $t \leq 2 \times t_0$. If $\mathcal{P}_{\mathsf{prox}}$ admits a distance hijacking attack w.r.t. t_0-proximity, then $\overline{\mathcal{P}_{\mathsf{prox}}}$ admits an attack against t_0-proximity in the topology $\mathcal{T}_{\mathsf{DH}}^{t_0}$.

In order to establish this result, we will first transform the initial attack trace into an "attack" trace in an untimed model. This model (with no timing constraints to fullfill) is more suitable to reorder some actions in the trace. We will show in a second step how to come back in the original timed model. We consider the untimed configuration associated to a configuration $K = (\mathcal{P}; \Phi; t)$. Formally, we have $\mathsf{untimed}(K) = (\mathcal{P}'; \Phi')$ with:

$$\mathcal{P}' = \{ \lfloor P \rfloor_a \mid \lfloor P \rfloor_a^t \in \mathcal{P} \}, \text{ and } \Phi' = \{ w \xrightarrow{a} u \mid \mathsf{w} \xrightarrow{a,t} u \in \Phi \}.$$

Then, we consider a *relaxed semantics* over untimed configurations: $K \overset{a,\alpha}{\rightsquigarrow}_{\mathcal{T}} K'$ if there exist K_0 and K_0' such that $K_0 \xrightarrow{a,\alpha}_{\mathcal{T}} K_0'$ (for some rule other than the TIM rule), and for which $K = \mathsf{untimed}(K_0)$ (resp. $K' = \mathsf{untimed}(K_0')$).

Under the same hypotheses as those stated in Theorem 16, we establish a result that allows one to "clean" an attack trace by pushing instructions (before or after) outside the rapid phase delimited by a `reset` and its following guarded input `in`. In the resulting trace, the only remaining actions in the rapid phase are those performed by agents who are close to v_0.

▶ **Proposition 17.** *Let K_0 be a valid initial configuration for $\overline{\mathcal{P}_{\mathsf{prox}}}$ and V_0 w.r.t. a topology $\mathcal{T} = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}, v_0, p_0)$ and $\mathcal{I}_0$. If $K_0 \xrightarrow{\mathsf{tr}}_{\mathcal{T}} K_1$ then there exists an execution $K_0' \overset{\mathsf{tr}'}{\rightsquigarrow} K_1'$ such that $K_i' = \mathsf{untimed}(K_i)$ for $i \in \{0, 1\}$.*

Moreover, for any sub-execution of $K_0' \overset{\mathsf{tr}'}{\rightsquigarrow} K_1'$ of the form

$$(\lfloor \mathtt{reset}.P \rfloor_{v_0} \uplus \mathcal{P}; \Phi_{\mathtt{reset}}) \overset{v_0,\tau}{\rightsquigarrow} (\lfloor P \rfloor_{v_0} \uplus \mathcal{P}; \Phi_{\mathtt{reset}}) \overset{\mathsf{tr}_0'}{\rightsquigarrow} K_{\mathtt{in}}^- \overset{v_0,\mathtt{in}^{<t}(u)}{\rightsquigarrow} K_{\mathtt{in}}'$$

where tr_0' only contains actions (a, α) with $\alpha \in \{\tau, \mathtt{out}(u), \mathtt{in}(u)\}$, we have that:

- *$2 \times \mathit{Dist}_{\mathcal{T}}(v_0, a) < t$ for any $(a, \alpha) \in \mathsf{tr}_0'$;*
- *for any $(a, \mathtt{in}^\star(v))$ occurring in $\mathsf{tr}_0'.(v_0, \mathtt{in}^{<t}(u))$, the agent b responsible of the output and the recipe R (as defined in Figure 1) are such that either $2 \times \mathit{Dist}_{\mathcal{T}}(v_0, b) < t$, or $\mathit{vars}(R) \subseteq \mathit{dom}(\Phi_{\mathtt{reset}})$.*

Relying on Proposition 17, we are then able to provide a sketch of proof for Theorem 16 (the full and detailed proof is available in [19]).

1. We start by removing `reset` instructions and by transforming any guarded input $\mathtt{in}^<$ (but those in V_0) into simple inputs. The resulting trace is still an attack trace w.r.t. $\overline{\mathcal{P}_{\mathsf{prox}}}$.
2. Then, we apply Proposition 17 in order to obtain an attack trace in the relaxed semantics. We will exploit the extra conditions given by Proposition 17 in order to lift the trace in the timed model at step 4.

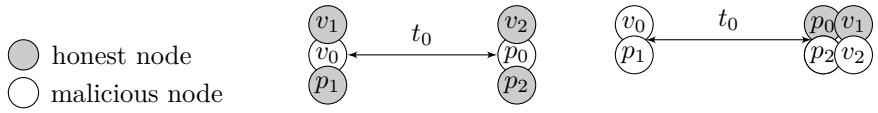


Figure 4 Reduced topologies for mafia fraud and distance hijacking when agents are *not* allowed to execute both roles.

3. We now consider another topology $\mathcal{T}'$ with two locations (as $\mathcal{T}_{\mathsf{DH}}^{t_0}$) and such that agents close to v_0 are now located with v_0, and those that are far away from v_0 in $\mathcal{T}$ are now located with p_0. This execution is still a valid trace in $\mathcal{T}'$ since we consider the relaxed semantics.
4. Then, to lift this execution trace into our timed model, the basic idea is to wait enough time before a `reset` instruction to allow messages to be received by all the participants before starting the rapid phase.
5. To conclude, as in the previous attack scenarios, we reduce the initial knowledge and the number of agents by applying a renaming on agent names.

▶ **Example 18.** The hijacking attack briefly described in Example 12 on the topology $\mathcal{T}_0 \in \mathcal{C}_{\mathsf{DH}}$ can be retrieved on the topology $\mathcal{T}_{\mathsf{DH}}^{t_0}$ starting with the valid initial configuration:

$$K_{\mathsf{init}} = (\lfloor P(v_0) \rfloor_{v_0}^0 \uplus \lfloor V(v_0, p_0) \rfloor_{v_0}^0 ; \{\mathsf{w}_1 \xrightarrow{p_0,0} \mathsf{sk}(p_0)\}; 0)$$

We may note that in the reduced topology the role of the prover has to be played by v_0 who is the only agent in the neighbourhood of himself. Such a configuration is indeed a valid initial configuration according to our definition. Actually, our reduction results still apply considering a model in which a same agent is *not* allowed to execute both roles. The resulting reduced topologies are depicted in Figure 4. Basically, we have to duplicate agents to ensure that both kinds of roles (verifier and prover) are available at each location.

5 Case studies using ProVerif

We have reduced the topology but we still have to take it into account when analysing the protocol preventing us from using automatic verification tools dedicated to traditional security protocols. In this section, we will explain how to get rid of the resulting topology and obtain interesting results on timed protocols relying on the ProVerif verification tool.

5.1 ProVerif in a nutshell

We consider a subset of the ProVerif calculus defined as follows:

$$P := 0 \mid \mathtt{in}(x).P \mid \mathtt{let}\ x = v\ \mathtt{in}\ P \mid \mathtt{new}\ n.P \mid \mathtt{out}(u).P \mid i : P \mid !P$$

where $x \in \mathcal{X}$, $n \in \mathcal{N}$, $u \in \mathcal{T}(\Sigma_c^+, \mathcal{X} \uplus \mathcal{N} \uplus \mathcal{A})$, $v \in \mathcal{T}(\Sigma^+, \mathcal{X} \uplus \mathcal{N} \uplus \mathcal{A})$ and $i \in \mathbb{N}$.

The semantics is similar to the one introduced earlier, and formally defined through a relation, denoted $\Rightarrow$, over configurations (only partially described below) . A configuration is a tuple $(\mathcal{P}; \phi; i)$ where $\mathcal{P}$ is a multiset of processes (as given by the grammar), ϕ is a frame as usual (with no decoration on the arrow), and $i \in \mathbb{N}$ is an integer that indicates the current

phase. Intuitively, the process $!P$ executes P an arbitrary number of times (in parallel), and only processes in the current phase are allowed to evolve. We often write P instead of $0 : P$.

$$\begin{array}{rcl}(i : \mathtt{in}(x).P \uplus \mathcal{P}; \phi; i) & \xrightarrow{\mathtt{in}(R\phi\downarrow)} & (i : P\{x \mapsto R\phi\downarrow\} \uplus \mathcal{P}; \phi; i) \text{ for some recipe } R \\ (i : !P \uplus \mathcal{P}; \phi; i) & \xRightarrow{\tau} & (i : P \uplus (i : !P) \uplus \mathcal{P}; \phi; i) \\ (\mathcal{P}; \phi; i) & \xrightarrow{\mathsf{phase}\ i'} & (\mathcal{P}; \phi; i') \quad \text{with } i' > i.\end{array}$$

5.2 Our transformation

Given a topology $\mathcal{T}$ (typically one in Figure 3), a protocol $\mathcal{P}_{\mathsf{prox}}$, a role V_0, and a template $\mathcal{I}_0$, we build a configuration $(\mathcal{P}; \phi; 0)$ on which the security analysis could be done using ProVerif. In such a configuration, $\mathcal{P}$ is a multiset of (non-extended) processes with phases, and ϕ is a (non extended) frame. From now on, we assume that $V_0(v_0, p_0)$ only contains one block of the form $\mathtt{reset}.\mathtt{out}(n).\mathtt{in}^{<t}(x)$, i.e. it is of the form:

$$\mathsf{block}_1 \,.\, \mathtt{reset} \,.\, \mathtt{out}(n) \,.\, \mathtt{in}^{<t}(x) \,.\, \mathsf{block}_2 \,.\, \mathsf{end}(v_0, p_0)$$

where $\mathsf{block_i}$ is a sequence of actions (only simple inputs, outputs, let, and new instructions are allowed). The main idea is to use phase 1 to represent the rapid phase. Such a phase starts when V_0 performs its `reset` instruction, and ends when V_0 performs its $\mathtt{in}^{<t}(x)$ instruction. During this rapid phase, only participants that are close enough to V_0 can manipulate messages outputted in this rapid phase. The other ones are intuitively too far. Therefore, we mainly consider two transformations, namely $\mathcal{F}^{<}$ and $\mathcal{F}^{\geq}$, whose purposes are to transform a parametrised role of our process algebra given in Section 2.2 (with no `reset` instruction and no guarded input) into a process in the ProVerif calculus.

- *Transformation $\mathcal{F}^{<}$:* this transformation introduces the phase instructions with $i = 0$, 1 and 2 considering all the possible ways of splitting the role into three phases (0, 1, and 2). Each phase instruction is placed before an `in` instruction. Such a slicing is actually sufficient for our purposes.
- *Transformation $\mathcal{F}^{\geq}$:* this transformation does the same but we forbid the use of the instruction phase 1, jumping directly from phase 0 to phase 2.

The configuration, denoted $\mathcal{F}(\mathcal{T}, \overline{\mathcal{P}_{\mathsf{prox}}}, V_0, \mathcal{I}_0, t_0)$, is the tuple $(\mathcal{P}; \phi; 0)$ where ϕ is such that $img(\phi) = \bigcup_{a \in \mathcal{M}_0} \mathsf{Knows}(\mathcal{I}_0, a, \mathcal{A}_0)$, and $\mathcal{P}$ contains:

- $\mathsf{block}_1 \,.\, 1 : \mathtt{out}(m) \,.\, \mathtt{in}(x) \,.\, 2 : \mathsf{block}_2 \,.\, \mathsf{end}(v_0, p_0)$;
- $!R(a_0, .., a_n)$ when $R(z_0, .., z_n) \in \mathcal{F}^{<}(\overline{\mathcal{P}_{\mathsf{prox}}})$, $a_0, .., a_n \in \mathcal{A}_0$, $\mathsf{Dist}_{\mathcal{T}}(v_0, a_0) < t_0$;
- $!R(a_0, .., a_n)$ when $R(z_0, .., z_n) \in \mathcal{F}^{\geq}(\overline{\mathcal{P}_{\mathsf{prox}}})$, $a_0, .., a_n \in \mathcal{A}_0$, $\mathsf{Dist}_{\mathcal{T}}(v_0, a_0) \geq t_0$.

We then establish the following result that justifies the transformation presented above.

▶ **Proposition 19.** *Let $\mathcal{T} = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0)$ be a topology, $\mathcal{P}_{\mathsf{prox}}$ a protocol, $t_0 \in \mathbb{R}_+$, $\mathcal{I}_0$ a template, and $V_0(z_0, z_1)$ a parametrised process of the form:*

$$\mathsf{block}_1 \,.\, \mathtt{reset} \,.\, \mathtt{out}(n) \,.\, \mathtt{in}^{<t}(x) \,.\, \mathsf{block}_2 \,.\, \mathsf{end}(z_0, z_1) \text{ with } t \leq 2 \times t_0$$

Let K_0 be a valid initial configuration for the protocol $\mathcal{P}_{\mathsf{prox}}$ and V_0 w.r.t. $\mathcal{T}$ and $\mathcal{I}_0$. If K_0 admits an attack w.r.t. t_0-proximity in $\mathcal{T}$, then we have that:

$$\mathcal{F}(\mathcal{T}_0, \overline{\mathcal{P}_{\mathsf{prox}}}, V_0, \mathcal{I}_0, t_0) \xRightarrow{\mathsf{tr}} (\{2 : \mathsf{end}(v_0, p_0)\} \uplus \mathcal{P}; \phi; 2).$$

Moreover, in case there is no $a \in \mathcal{M}_0$ such that $\mathsf{Dist}_{\mathcal{T}_0}(a, v_0) < t_0$, we have that for any $\mathtt{in}(u)$ occurring in tr during phase 1, the underlying recipe R is either of the form w, or only uses handles ouputted in phase 0.

This result allows us to turn any mafia attack attack or distance hijacking attack into an attack trace regarding the reachability of the event end.

Table 1 Results on our case studies (×: attack found, ✓: proved secure, *n.a.*: not applicable).

Protocols	MF	DH	Protocols	MF	DH
Brands and Chaum [11]	✓	×	Munilla *et al.* [30]	✓	✓
Meadows *et al.* $(n_V \oplus n_P, P)$ [28]	✓	✓	CRCS [32]	✓	×
Meadows *et al.* $(n_V, n_P \oplus P)$ [28]	✓	×	Hancke and Kuhn ⋆ [23]	✓	✓
TREAD-Asymmetric [6]	×	×	Eff-pkDB [25]	✓	✓
TREAD-Symmetric [6]	✓	×			
SPADE [12]	×	×	PaySafe [14]	✓	*n.a.*
MAD (One-Way) [13]	✓	×	PaySafe-v2 [14]	×	*n.a.*
Swiss-Knife [26]	✓	✓	PaySafe-v3 [14]	×	*n.a.*

⋆ the protocols Tree-based, Poulidor, and Uniform are actully equivalent to this one.

5.3 Case studies

Regardless of the type of the considered attack, we only need to consider a single topology (depicted in Figure 3). Once down to this single topology, we can apply Proposition 19, and analyse the configuration $(\mathcal{P};\phi;0) = \mathcal{F}(\mathcal{T}_{\mathsf{XX}}^{t_0}, \overline{\mathcal{P}_{\mathsf{prox}}}, V_0, \mathcal{I}_0, t_0)$ with $\mathsf{XX} \in \{\mathsf{MF}, \mathsf{DH}\}$ in ProVerif. If the protocol is proved secure, then $\mathcal{P}_{\mathsf{prox}}$ is resistant to the class of attacks we have considered. Otherwise, we have to check whether the trace returned by ProVerif can be turned into a real attack in our timed model. In principle, it may happen that ProVerif returns an attack trace that is only suitable in the untimed model.

Actually, to obtain meaningful results regarding scenarios that only involved honest participants in the neighbourhood of v_0, we have to go one step further. Indeed, the attacker model behind ProVerif allows him to interact with any participant (even those that are far away) with no delay. To avoid these behaviours that are not possible in the rapid phase, we slightly modify the ProVerif code taking advantage of the extra condition stated in Proposition 19. During phase 1, we consider an attacker who is only able to forward messages previously sent, and forged new messages using his knowledge obtained in phase 0.

On all our case studies, ProVerif answered in less than one second (on a standard laptop) and all the traces returned by ProVerif are actually real attack traces (no false positive).

Distance bounding protocols. We apply our methodology to a number of well-known distance bounding protocols. In symbolic models, it is not possible to reason at the bit-level, and therefore we replace the bit-sized exchanges by a single challenge-response exchange using a fresh nonce (as done in Example 4). Sometimes, we also abstract the answer from the prover relying on an uninterpreted function symbol with relevant arguments. Finally, in order to rely on ProVerif, the xor operator has been abstracted as follows (even if our theoretical development is generic enough to deal with such an operator):

$$x \oplus y) \oplus x \to y \quad (x \oplus y) \oplus y \to x \quad x \oplus (x \oplus y) \to y \quad y \oplus (x \oplus y) \to x.$$

For instance, we succeed in proving resistance against mafia fraud for the first version of the Meadows *et al.* protocol which could not be proved using the framework proposed in [28]. The results are consistent with the ones obtained in [27, 17]. In addition, we discovered a new attack on the SPADE protocol [12] which has been designed to be mafia fraud resistant.

As described in Figure 5, the prover generates a nonce n_P, signs it and encrypts the resulting message with the public key of the verifier. Receiving such a message, the verifier answers by sending fresh nonces. Once these two messages are exchanged, the rapid

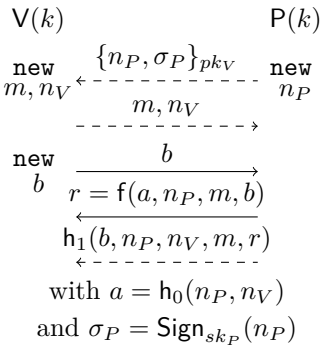


Figure 5 SPADE protocol.

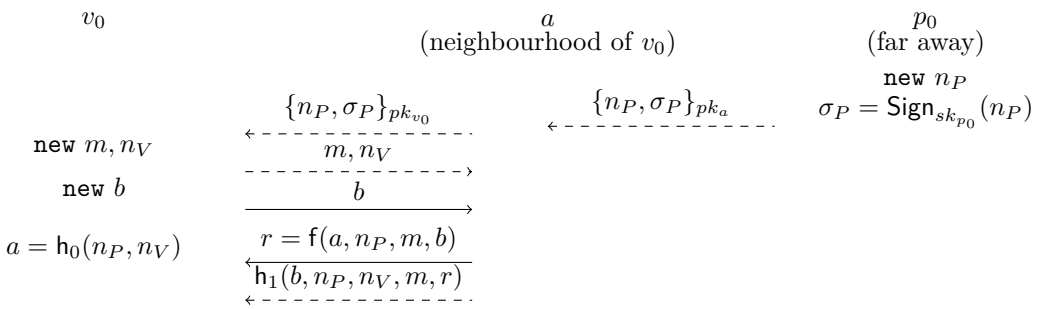


Figure 6 Mafia fraud on the SPADE protocol.

phase begins: the prover has to answer to the challenge b relying on some hash functions applied on various nonces. Finally the protocol ends when the prover sends the final hash $\mathsf{h}_1(b, n_P, n_V, m, r)$. The attack we discovered (see the description given in Figure 6) is similar to the one obtained on TREAD-Asymmetric by [27]. Roughly, once a dishonest verifier a received a message $\{n_P, \sigma_P\}_{pk_a}$, he can use it to forge $\{n_P, \sigma_P\}_{pk_{p_0}}$ and acts as if the prover p_0 was in the neighboorhood of v_0. The attacker model presented in [12] does not allow them to capture this attack since they not consider dishonest verifers.

PaySafe protocol. We studied the PaySafe protocol [14] designed to be resistant against mafia fraud attacks. More generally, contactless payment protocols need to prevent relay attacks where malicious agents would abuse from an honest agent to perform a payment, which corresponds to the mafia fraud scenario.

The PaySafe protocol is schematised in Figure 7 where plain arrows represents the rapid exchange phase. During the initialisation phase, the reader and the card exchange some identifiers, while during the authentication exchange, the reader ensures that the card is legitimate using signatures and certificates verifications. The main idea is to send nonces and constants during the rapid phase and to perform all the necessary checks later on. The aim is to increase the accuracy on the proximity property needed to ensure the security of the protocol. We also considered two other versions of PaySafe, also described in [14]: in PaySafe-v2 we just remove the reader nonce n_R and in PaySafe-v3 we remove n_R and n_C.

Our results confirmed those presented in [14]. Their methodology and ours, especially when it comes down to the use of ProVerif, are quite similar but we would like to emphasise the fact that our use of ProVerif is a consequence of our formal development. Actually our ProVerif models differ from those given in [14]. We typically define richer scenario by

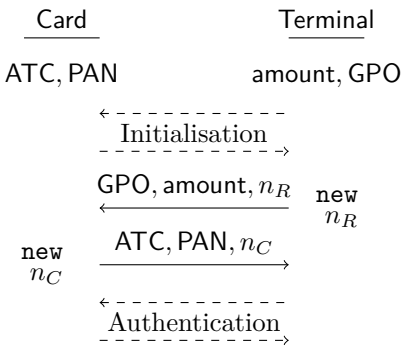


Figure 7 PaySafe (simplified).

giving additional knowledge to the attacker and allowing prover roles and verifier roles to be executed in phase 1. The authors of [27] reported a distance fraud attack which is not relevant in the context of EMV contactless payment protocols. Their result does not allow them to isolate each class of attacks, and therefore, they can not prove whether PaySafe is mafia fraud resistant or not.

6 Conclusion

Regarding physical proximity, we have shown two main reduction results: if there is an attack on an arbitrary topology then there is an attack on a simple one having at most four nodes. Relying on these reduction results, we have shown how to use ProVerif to analyse several protocols. Our methodology is flexible enough to draw meaningful conclusions on each class of attacks: hijacking attack, and mafia fraud. The interested reader may also find some additional results regarding distance fraud in our companion report [19].

As future work, we would like to extend our result to consider the notion of terrorist fraud. This would require to consider dishonest participants who only share a part of their knowledge. Our work should also benefit from the recent advances that have been made to integrate the exclusive-or operator in existing verification tool such as Tamarin [22]. Even if our formal development allows us to rely on the Tamarin prover (e.g. we obtain meaningful results on the Hancke and Kuhn protocol with Tamarin), it happens that Tamarin (automatic mode) behaves poorly on some of our case studies (e.g. Brands and Chaum) that uses the xor operator. This deserves further investigations.

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 A. Armando, R. Carbone, L. Compagna, J. Cuéllar, and M. L. Tobarra. Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for Google apps. In *Proc. 6th ACM Workshop on Formal Methods in Security Engineering (FMSE'08)*, pages 1–10. ACM, 2008.

3 A. Armando et al. The AVANTSSAR Platform for the Automated Validation of Trust and Security of Service-Oriented Architectures. In *Proc. 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'12)*, volume 7214, pages 267–282. Springer, 2012.

4 G Avoine, MA Bingol, Ioana Boureanu, S Capkun, G Hancke, S Kardas, CH Kim, C Lauradoux, B Martin, J Munilla, et al. Security of Distance-Bounding: A Survey. *ACM Computing Surveys*, 2017.

5 Gildas Avoine, Muhammed Ali Bingöl, Süleyman Kardaş, Cédric Lauradoux, and Benjamin Martin. A framework for analyzing RFID distance bounding protocols. *Journal of Computer Security*, 19(2):289–317, 2011.

6 Gildas Avoine, Xavier Bultel, Sébastien Gambs, David Gerault, Pascal Lafourcade, Cristina Onete, and Jean-Marc Robert. A terrorist-fraud resistant and extractor-free anonymous distance-bounding protocol. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 800–814. ACM, 2017.

7 David Basin, Srdjan Capkun, Patrick Schaller, and Benedikt Schmidt. Formal reasoning about physical properties of security protocols. *ACM Transactions on Information and System Security (TISSEC)*, 14(2):16, 2011.

8 B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proc. 14th Computer Security Foundations Workshop (CSFW'01)*, pages 82–96. IEEE Computer Society Press, 2001.

9 Bruno Blanchet. Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif. *Foundations and Trends in Privacy and Security*, 1(1-2):1–135, 2016. doi:10.1561/3300000004.

10 Ioana Boureanu, Aikaterini Mitrokotsa, and Serge Vaudenay. Practical and provably secure distance-bounding. *Journal of Computer Security*, 23(2):229–257, 2015.

11 Stefan Brands and David Chaum. Distance-bounding protocols. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 344–359. Springer, 1993.

12 Xavier Bultel, Sébastien Gambs, David Gerault, Pascal Lafourcade, Cristina Onete, and Jean-Marc Robert. A Prover-Anonymous and Terrorist-Fraud Resistant Distance-Bounding Protocol. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks, WISEC 2016, Darmstadt, Germany, July 18-22, 2016*, pages 121–133. ACM, 2016.

13 Srdjan Čapkun, Levente Buttyán, and Jean-Pierre Hubaux. SECTOR: secure tracking of node encounters in multi-hop wireless networks. In *Proc. 1st ACM workshop on Security of ad hoc and sensor networks*, pages 21–32. ACM, 2003.

14 Tom Chothia, Flavio D. Garcia, Joeri de Ruiter, Jordi van den Breekel, and Matthew Thompson. Relay Cost Bounding for Contactless EMV Payments. In *Proc. 19th International Conference on Financial Cryptography and Data Security (FC'15)*, volume 8975 of *Lecture Notes in Computer Science*, pages 189–206. Springer, 2015.

15 Hubert Comon-Lundh and Véronique Cortier. Security properties: two agents are sufficient. *Programming Languages and Systems*, pages 99–113, 2003.

16 Véronique Cortier, Jan Degrieck, and Stéphanie Delaune. Analysing routing protocols: four nodes topologies are sufficient. In *International Conference on Principles of Security and Trust*, pages 30–50. Springer, 2012.

17 Cas Cremers, Kasper B Rasmussen, Benedikt Schmidt, and Srdjan Capkun. Distance hijacking attacks on distance bounding protocols. In *Proc. IEEE Symposium on Security and Privacy (S&P'12)*, pages 113–127. IEEE, 2012.

18 Alexandre Debant, Stéphanie Delaune, and Cyrille Wiedling. http://people.irisa.fr/Alexandre.Debant/proving-physical-proximity-using-symbolic-methods.html.

19 Alexandre Debant, Stéphanie Delaune, and Cyrille Wiedling. Proving physical proximity using symbolic models. Research report, Univ Rennes, CNRS, IRISA, France, February 2018. URL: https://hal.archives-ouvertes.fr/hal-01708336.

20 Yvo Desmedt, Claude Goutier, and Samy Bengio. Special uses and abuses of the Fiat-Shamir passport protocol. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 21–39. Springer, 1987.

21 D. Dolev and A. C. Yao. On the Security of Public Key Protocols. In *Proc. 22nd Symposium on Foundations of Computer Science (FCS'81)*, pages 350–357. IEEE Computer Society Press, 1981.

22 Jannik Dreier, Lucca Hirschi, Sasa Radomirovic, and Ralf Sasse. Automated Unbounded Verification of Stateful Cryptographic Protocols with Exclusive OR. In *CSF'2018 - 31st IEEE Computer Security Foundations Symposium*, Oxford, United Kingdom, July 2018. URL: `https://hal.archives-ouvertes.fr/hal-01780603`.

23 Gerhard P Hancke and Markus G Kuhn. An RFID distance bounding protocol. In *Proc. 1st International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM'05)*, pages 67–73. IEEE, 2005.

24 Max Kanovich, Tajana Ban Kirigin, Vivek Nigam, Andre Scedrov, and Carolyn Talcott. Towards timed models for cyber-physical security protocols. In *Joint Workshop on Foundations of Computer Security and Formal and Computational Cryptography*, 2014.

25 Handan Kilinç and Serge Vaudenay. Efficient Public-Key Distance Bounding Protocol. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 873–901, 2016.

26 Chong Hee Kim, Gildas Avoine, François Koeune, François-Xavier Standaert, and Olivier Pereira. The swiss-knife RFID distance bounding protocol. In *International Conference on Information Security and Cryptology*, pages 98–115. Springer, 2008.

27 S. Mauw, Z. Smith, J. Toro-Pozo, and R. Trujillo-Rasua. Distance-Bounding Protocols: Verification without Time and Location. In *2018 IEEE Symposium on Security and Privacy (S&P)*, volume 00, pages 152–169, 2018. `doi:10.1109/SP.2018.00001`.

28 Catherine Meadows, Radha Poovendran, Dusko Pavlovic, LiWu Chang, and Paul Syverson. Distance bounding protocols: Authentication logic analysis and collusion attacks. In *Secure localization and time synchronization for wireless sensor and ad hoc networks*, pages 279–298. Springer, 2007.

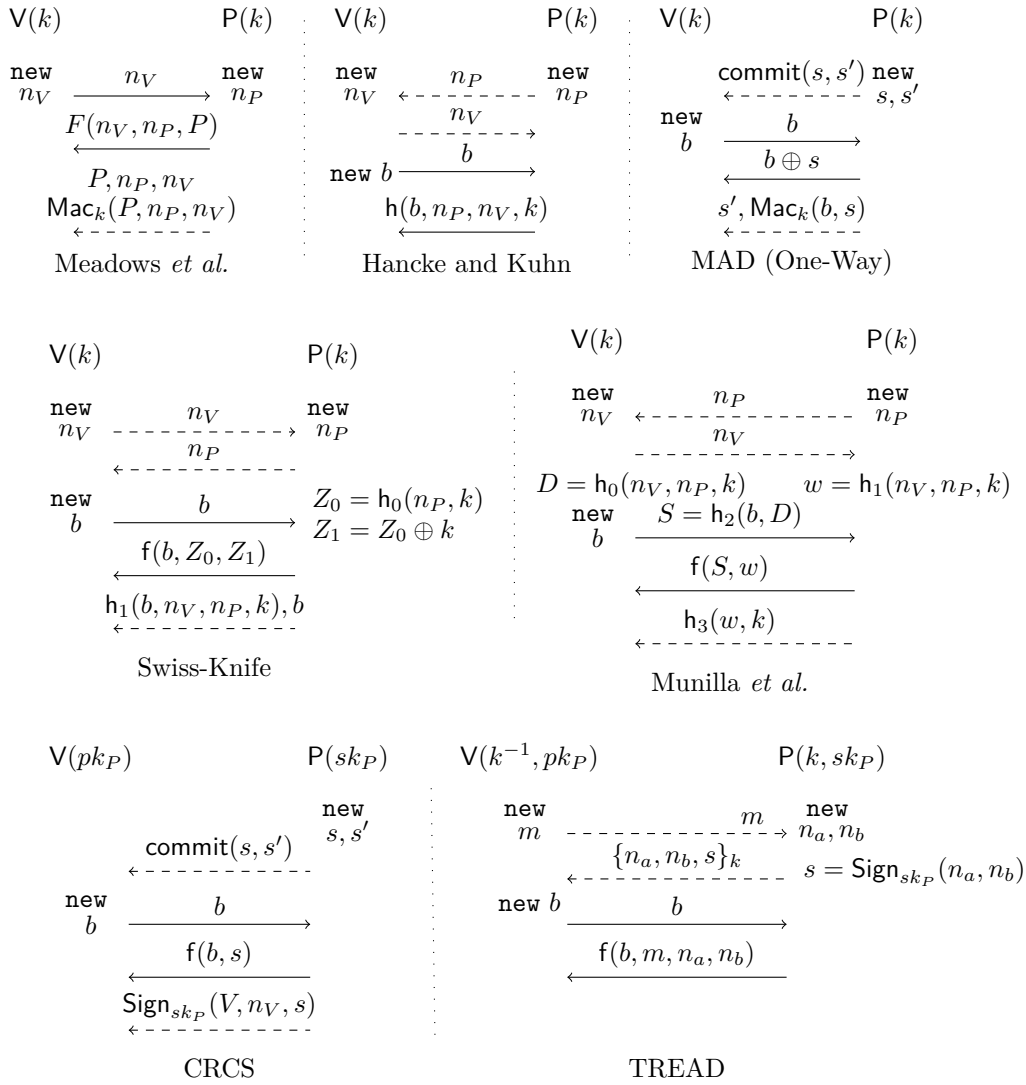
29 S. Meier, B. Schmidt, C. Cremers, and D. Basin. The Tamarin Prover for the Symbolic Analysis of Security Protocols. In *Proc. 25th International Conference on Computer Aided Verification (CAV'13)*, volume 8044 of *LNCS*, pages 696–701. Springer, 2013.

30 Jorge Munilla and Alberto Peinado. Distance bounding protocols for RFID enhanced by using void-challenges and analysis in noisy channels. *Wireless communications and mobile computing*, 8(9):1227–1232, 2008.

31 Vivek Nigam, Carolyn Talcott, and Abraão Aires Urquiza. Towards the Automated Verification of Cyber-Physical Security Protocols: Bounding the Number of Timed Intruders. In *Proc. 21st European Symposium on Research in Computer Security (ESORICS'16)*, pages 450–470. Springer, 2016.

32 Kasper Bonne Rasmussen and Srdjan Capkun. Realization of RF Distance Bounding. In *USENIX Security Symposium*, pages 389–402, 2010.

A Brief description of our case studies



On Canonical Models for Rational Functions over Infinite Words

Emmanuel Filiot
Université Libre de Bruxelles, Belgium
efiliot@ulb.ac.be

Olivier Gauwin
LaBRI, Université de Bordeaux, France
olivier.gauwin@labri.fr

Nathan Lhote
LaBRI, Université de Bordeaux, France and Université Libre de Bruxelles, Belgium
nlhote@labri.fr

Anca Muscholl
LaBRI, Université de Bordeaux, France
anca@labri.fr

Abstract

This paper investigates canonical transducers for rational functions over infinite words, i.e., functions of infinite words defined by finite transducers. We first consider sequential functions, defined by finite transducers with a deterministic underlying automaton. We provide a Myhill-Nerode-like characterization, in the vein of Choffrut's result over finite words, from which we derive an algorithm that computes a transducer realizing the function which is minimal and unique (up to the automaton for the domain).

The main contribution of the paper is the notion of a canonical transducer for rational functions over infinite words, extending the notion of canonical bimachine due to Reutenauer and Schützenberger from finite to infinite words. As an application, we show that the canonical transducer is aperiodic whenever the function is definable by some aperiodic transducer, or equivalently, by a first-order transduction. This allows to decide whether a rational function of infinite words is first-order definable.

2012 ACM Subject Classification Theory of computation → Transducers

Keywords and phrases transducers, infinite words, minimization, aperiodicty, first-order logic

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.30

Related Version A full version of the paper is available at `https://hal.archives-ouvertes.fr/hal-01889429`.

Funding This work was supported by the French ANR projects *ExStream* (ANR-13-JS02-0010) and *DeLTA* (ANR-16-CE40-0007), the Belgian FNRS CDR project *Flare* (J013116) and the ARC project *Transform* (Fédération Wallonie Bruxelles).

Introduction

Machine models, such as automata and their extensions, describe mathematical objects in a finite way. Finite automata, for instance, describe languages (of words, trees, etc). A canonization function $\mathcal{C}$ is a function from and to machine models (not necessarily of the same type) such that, whenever two machines M_1, M_2 describe the same object, then

38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018).
Editors: Sumit Ganguly and Paritosh Pandya; Article No. 30; pp. 30:1–30:17

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

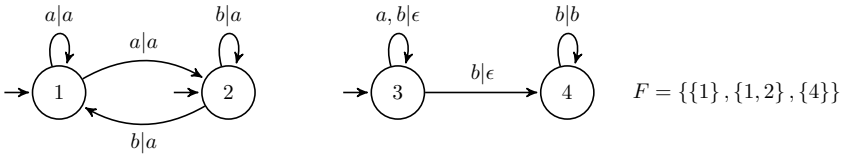


Figure 1 Transducer with Muller sets F realizing the function $f_{\#a}$ mapping any word with infinitely many a to a^ω, otherwise to b^ω.

$\mathcal{C}(M_1) = \mathcal{C}(M_2)$. Accordingly, $\mathcal{C}(M_1)$ is called the *canonical model* of the object described by M_1, and it does not depend on the initial representation of the object. A classical example of canonization is the function which associates with a finite automaton its equivalent minimal deterministic automaton. A canonization function becomes interesting when it satisfies additional constraints like being computable, preserving some algebraic properties, and enjoying minimal models. Canonical models not only shed light on the intrinsic characteristics of the class of objects they describe, but can also serve to decide *definability* problems. For instance, it is well-known that the minimal DFA of a word language L is aperiodic if and only if L is definable in first-order logic [17, 21]. Hence, this allows to decide whether a monadic second-order formula has an equivalent first-order one over words. This result has been extended to infinite words [23, 24, 1, 18], although there is no unique minimal automaton for languages of infinite words (see also [10] for a survey).

Rational functions are functions defined by word transducers. A canonical model for rational functions over *finite* words has been introduced in [20]. This result, which can be considered as one of the jewels of transducer theory, states the existence of a procedure that computes from a given transducer a canonical input-deterministic transducer with look-ahead, called bimachine. For the subclass of functions realized by input-deterministic transducers, called sequential functions, it is even possible to compute a unique and minimal transducer realizing the function [8]. For rational functions, the procedure of [20], though it preserves aperiodicity of the transition congruence of the transducer, does not preserve other congruence varieties, in general. In [14, 15] it was shown how to adapt [20] to obtain a canonization procedure which overcomes this issue. Later it was shown that the first-order definability problem for rational functions is PSpace-c [13]. In a different setting, functions *with origin information* realizable by two-way transducers were shown to have decidable first-order definability [4]. In this paper, we extend the results of [20] and the decidability of first-order definability of [13] to rational functions of infinite words.

Rational functions of infinite words. We consider rational functions of infinite words, *i.e.* functions defined by transducers with Muller acceptance condition. Such machines map any ω-word for which there exists an accepting run to either a finite or an ω-word. Take as example the function $f_{\#a}$ over alphabet $\{a, b\}$ mapping any word containing an infinite number of a to a^ω, and to b^ω otherwise. This function is realized by the transducer of Fig. 1.

The class of *sequential functions* is of particular interest: they are realized by transducers whose underlying input Muller automaton is deterministic. Note that the function $f_{\#a}$ is not sequential, unlike the function f_{ab} of Fig. 2. Sequential functions over infinite words have been studied *e.g.* in [2]. One difference between our setting and [2] is that in the latter paper infinite words are mapped to infinite words, whereas we need also functions that map infinite words to finite words. Deciding whether a rational function is sequential can be done in

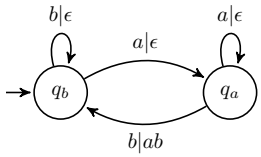


Figure 2 Sequential transducer with Muller condition $F = \{\{q_b\}\}$ realizing the function f_{ab} which maps any word containing a finite number of a's to the subsequence of ab factors, and is undefined otherwise.

PTIME, as shown in [2]. Bimachines for infinite words were introduced in [25] to define the particular class of total letter-to-letter rational functions, and in their counter-free versions, a connection with linear temporal logic was established.

To the best of our knowledge, nothing is known about canonical models for sequential and rational functions over infinite words, and their applications to definability problems in logics.

Contributions.

(1) We provide a characterization of sequential functions by means of the finiteness of a congruence. We give a PTIME procedure which, for any sequential function f given as a transducer whose domain is topologically closed, produces *the* minimal (and hence canonical) sequential transducer $\mathcal{T}_f$ realizing f. When the domain of f is not topologically closed, we extend f to a domain-closed sequential function $\overline{f}$ which coincides with f on its domain. By intersecting $\mathcal{T}_{\overline{f}}$ with some automaton $\mathcal{D}$ recognizing the domain of f, one obtains a canonical transducer for f, as long as $\mathcal{D}$ can be obtained in a canonical way (such a procedure exists, see *e.g.* [7]).

(2) Our main contribution (Theorem 29) is a notion of canonical sequential transducer with look-ahead for any rational function. This canonical transducer is an effectively computable bimachine. Hence we lift results of Reutenauer and Schützenberger [20] on rational transductions from finite to infinite words.

(3) As a side result we lift a result by Elgot and Mezei [11] from finite to infinite words, stating that a function f is rational if and only if $f = g_1 \circ h_1$ (resp. $f = g_2 \circ h_2$) such that h_1, h_2 are letter-to-letter, g_1, h_2 are sequential and h_1, g_2 are right-sequential (*i.e.*, realized by a transducer whose underlying input automaton is prophetic [6]). The existence of such g_1, h_1 was already shown in [5], but the one of g_2, h_2 was left open.

(4) Finally, we show that our procedure which computes a canonical bimachine for any rational function given by a transducer, preserves aperiodicity. As an application, after showing some correspondences between logics and transducers, we obtain the decidability of FO-transductions in MSO-transductions over infinite words.

Overview of the canonization procedure for rational functions. The main idea to get a canonical object for a rational function, inspired by [20], is to add a canonical look-ahead information to the input word, so that the function can be evaluated in a sequential (equivalently, deterministic) manner. We say that the look-ahead "makes the function sequential". By doing so, we can reduce the problem to computing canonical machines for sequential functions. The main difficulty is to define a canonical (and computable) notion of look-ahead which makes the function sequential. Over finite words, the look-ahead information is computed by a co-deterministic automaton, or equivalently, a deterministic automaton reading the input word from right to left (called a right automaton). On infinite words we

need something different, so we use prophetic automata [6] to define look-aheads (called right automata in this paper). Prophetic automata are a special form of co-deterministic automata over infinite words. In Section 3, sequential transducers with look-ahead are formalized via the notion of bimachines, consisting of a left automaton and a right automaton. We show that bimachines over infinite words capture exactly the class of rational transductions. Our goal is to obtain a canonical bimachine, fine enough to realize the function, but coarse enough to preserve algebraic properties like aperiodicity.

Unlike the setting of finite words, some difficulties arise when prefix-independent properties matter (such as for instance that a suffix contains an infinite number of a's). We overcome this issue by defining two kinds of look-ahead information which we combine later on. This decomposition simplifies the overall proof.

The first look-ahead information we define allows one to make any rational function almost sequential, in the sense that it can be implemented by a transducer model which can additionally output some infinite word after processing the whole input, depending on the run (similar to so-called *subsequential* transducers in the case of finite words). We call *quasi-sequential* functions realized by such transducers. They constitute a class with interesting properties. We show that they correspond precisely to transducers satisfying the weak twinning property, a syntactic condition defined in [2]. On the algebraic side, we exhibit a congruence having finite index exactly for quasi-sequential functions.

We then define another kind of canonical look-ahead which makes any quasi-sequential function sequential. Combined together, these two look-aheads turn any rational function into a sequential one: the first one from rational to quasi-sequential, and the second one from quasi-sequential to sequential.

The whole procedure does not yield a minimal bimachine in general. While the minimality question is an important and interesting (open) question, our procedure still has the strong advantages of being canonical, effective, and of preserving aperiodicity. This allows one to answer positively the important question of the decidability of first-order definability for rational functions of infinite words. Detailed proofs are provided in a long version of this paper, available online.

1 Regular languages and rational functions

Finite words, infinite words and languages. An *alphabet* A is a finite set of symbols called *letters*. A *finite word* is a finite sequence of letters, the empty sequence is called the *empty word* and is denoted by ϵ. The set of (resp. non-empty) finite words over A is denoted by A^* (resp. A^+). An infinite sequence of letters is called an *ω-word* (or just an infinite word), we denote by A^ω the set of ω-words and we write $A^\infty = A^* \cup A^\omega$. For a word $x \in A^\omega$ we denote by $\mathrm{Inf}(x)$ the set of letters of x which appear an infinite number of times. The *length* of a word w is written $|w|$, with $|w| = \infty$ if $w \in A^\omega$. Throughout the paper, we often denote finite words by $u, v, \ldots$ and infinite words by $x, y \ldots$

For a non-empty word w and two integers $1 \leq i \leq j \leq |w|$ we denote by $w(i)$ the ith letter of w, by $w(i{:})$ the suffix of w starting at the ith position, by $w({:}i)$ the prefix of w ending at the ith position and by $w(i{:}j)$ the infix of w starting at the ith position and ending at the jth, both included. For two words $u \in A^*$ and $v \in A^\infty$, we write $u \prec v$ if u is a strict prefix of v, *i.e.* there exists a non-empty word $w \in A^\infty$ such that $uw = v$, and we write $u^{-1}v$ for w. For $u, v \in A^\infty$, we write $u \preceq v$ if either $u \prec v$, or $u = v$. We denote by $u \wedge v$ the longest common prefix of u and v. The *delay* $\mathsf{del}(u, v)$ between two words $u, v \in A^\infty$ is the unique pair (u', v') such that $u = (u \wedge v)u'$ and $v = (u \wedge v)v'$. For example, $\mathsf{del}(aab, ab) = (ab, b)$ and $\mathsf{del}(a^\omega, a^\omega) = (\epsilon, \epsilon)$.

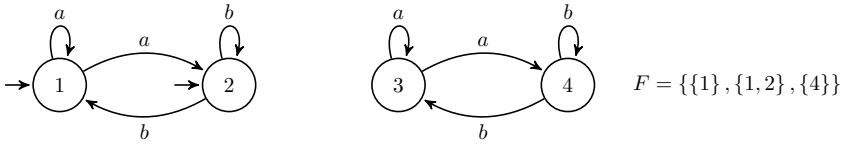


Figure 3 A right automaton (with Muller condition) recognizing $(b^*a)^\omega$. Words with finitely many b's have final run with $\{1\}$, words with finitely many a's have final run with $\{4\}$, and those with infinitely many a's and infinitely many b's have final run with $\{1,2\}$.

A *language* is a set of words $L \subseteq A^\infty$, and by $\bigwedge L$ we denote the longest common prefix of all words in L (if $L \neq \varnothing$). The closure $\overline{L}$ of L is $\{u \in A^\infty \,|\, \forall i \in \mathbb{N},\ i \leq |u|,\ \exists w \text{ s.t. } u(:i)w \in L\}$, *i.e.* the set of words for which any finite prefix has a continuation in L. For instance $\overline{a^*b^\omega} = a^*b^\omega \cup a^\omega$. A word is called *regular* if it is of the form uv^ω with $u, v \in A^*$. In particular any finite word is regular (since $\epsilon^\omega = \epsilon$) and regular ω-words are also called *ultimately periodic*. We say that a regular word uv^ω is in *normal form* if v has minimal length and is minimal in the lexicographic order among all possible decompositions of uv^ω, and v is not a suffix of u (if $v \neq \epsilon$). *E.g.* the normal form of $(ba)^\omega$ is $b(ab)^\omega$. In the sequel we often assume regular words are in normal form.

Automata. A Muller[1] automaton over an alphabet A is a tuple $\mathcal{A} = (Q, \Delta, I, F)$ where Q is a finite set of *states*, $\Delta \subseteq Q \times A \times Q$ is the set of *transitions*, $I \subseteq Q$ is the set of *initial* states, and $F \subseteq \mathcal{P}(Q)$ is called the *final condition*. When there is no final condition, so $F = \mathcal{P}(Q)$, we will omit it. A *run* of $\mathcal{A}$ over a word $w \in A^\infty$ is itself a word $r \in Q^\infty$ of length $|w| + 1$, (with the convention that $\infty + 1 = \infty$) such that for any $1 \leq i < |r|$, we have $(r(i), w(i), r(i+1)) \in \Delta$. A run r is called *initial* if $r(1) \in I$, *final* if $r \in Q^\omega$ and $\mathrm{Inf}(r) \in F$, and *accepting* if it is both initial and final. For a finite word u and two states p, q, we write $p \xrightarrow{u}_{\mathcal{A}} q$ to denote that there is a run r of $\mathcal{A}$ over u such that $r(1) = p$ and $r(|r|) = q$. For an ω-word x, a state p and a subset of states $P \subseteq Q$, we write $p \xrightarrow{x}_{\mathcal{A}} P$ to denote that there is a run r of $\mathcal{A}$ over x such that $r(1) = p$ and $\mathrm{Inf}(r) = P$. A word is *accepted* by $\mathcal{A}$ if there exists an accepting run over it, and the language *recognized* by $\mathcal{A}$ is the set of words it accepts, denoted by $[\![\mathcal{A}]\!] \subseteq A^\omega$. A state p is *accessible* (resp. *co-accessible*) if there exists a finite initial (resp. infinite final) run r such that $r(|r|) = p$ (resp. $r(1) = p$), and an automaton $\mathcal{A}$ is called *accessible* (resp. *co-accessible*) if all its states are. An automaton which is both accessible and co-accessible is called *trim*.

An automaton is called *deterministic* if its set of initial states is a singleton, and any word has at most one initial run. We define a *left automaton* as a deterministic automaton $\mathcal{L} = (Q, \Delta, I)$ with no acceptance condition. We call a *right automaton* an automaton for which any ω-word has exactly one final run[2]. A language is called *ω-regular* if it is recognized by an automaton. It is well-known that every ω-regular language can be recognized by a deterministic (Muller) automaton. Moreover, [6] shows that every ω-regular language can be recognized by a right automaton (even with Büchi condition). Figure 3 shows a right automaton accepting the words with infinitely many a's. Throughout the paper, all automata – except for right automata – are assumed trim, without loss of generality.

[1] We consider the Muller condition since it is more general than Büchi or parity for instance, but most of our results hold for other conditions as well.

[2] Such automata are called prophetic and were introduced in [6].

Transductions. Given two alphabets A, B, we call *transduction* a relation $R \subseteq A^\omega \times B^\infty$ whose domain is denoted by $\mathrm{dom}(R)$. A transducer over A, B is a tuple $\mathcal{T} = (\mathcal{A}, i, o)$ with $\mathcal{A} = (Q, \Delta, I, F)$ the *underlying automaton*, $i : I \to B^*$ the *initial function* and $o : \Delta \to B^*$ the *output function*.

Let u be a finite word of length n, let r be a run of $\mathcal{A}$ over u with $r(1) = p, r(n+1) = q$, and let v be the word $o(p, u(1), r(2)) \cdots o(r(n), u(n), q)$ then we write $p \xrightarrow{u|v}_{\mathcal{T}} q$ to denote that fact. Similarly, for $p \in Q$ and $P \subseteq Q$ we write $p \xrightarrow{x|v}_{\mathcal{T}} P$ to denote that there is a run r of $\mathcal{A}$ over the ω-word x such that $r(1) = p$, $\mathrm{Inf}(r) = P$ and $v = o(p, u(1), r(2))o(r(2), u(2), r(3)) \cdots$. In that case, if $p \in I$ and $P \in F$, let $w = i(p) \cdot v$, then we say that the pair (x, w) is *realized* by $\mathcal{T}$. We denote by $[\![\mathcal{T}]\!]$ the set of pairs realized by $\mathcal{T}$, which we call the *transduction realized* by $\mathcal{T}$. A transducer is called *functional* if it realizes a (partial) function, and in that case we write $[\![\mathcal{T}]\!](x) = w$ rather than $(x, w) \in [\![\mathcal{T}]\!]$. Functionality is a decidable property, see *e.g.* [16], and it can be checked in PTime (see *e.g.* [19]). *In the following all the transductions we consider are functional, and when we speak about functions, we tacitly assume that they are partial.* A transduction is *rational* if it is realized by a transducer. A transducer with a deterministic underlying automaton is called *sequential*, as well as the function it realizes. A transducer with a left (resp. right) underlying automaton is called *left-sequential* (resp. *right-sequential*), and again we extend this terminology to the function it realizes.

Congruences. Given an equivalence relation $\sim$ over a set L, we denote by $[w]^\sim$ (or simply $[w]$) the equivalence class of an element $w \in L$. We say that $\sim$ has *finite index* if the set $L/_\sim = \{[w] \mid w \in L\}$ is finite. Given two equivalence relations $\sim_1, \sim_2$ over the same set we say that $\sim_1$ is *finer* than $\sim_2$ (or that $\sim_2$ is *coarser* than $\sim_1$) if for any u, v we have $u \sim_1 v \Rightarrow u \sim_2 v$. Equivalently we could say that the equivalence classes of $\sim_2$ are unions of equivalence classes of $\sim_1$ or that $\sim_1$ is included (as a set of pairs) in $\sim_2$, which we denote by $\sim_1 \sqsubseteq \sim_2$. A *right congruence* over A^* is an equivalence relation $\sim$ such that for any letter a and any words u, v we have $u \sim v \Rightarrow ua \sim va$. A *left congruence* over A^* (resp. A^ω) is an equivalence relation $\approx$ such that for any letter a and any words u, v we have $u \approx v \Rightarrow au \approx av$. We say that a left congruence is *regular* if it has finite index and any equivalence class is an ω-regular language. In the following all the left congruences will be regular. A *congruence* over A^* is a left and right congruence. A congruence $\equiv$ is *aperiodic* if there exists an integer n such that $\forall u \in A^*$, $u^n \equiv u^{n+1}$.

Given an automaton $\mathcal{A}$ with state space Q, the *right congruence associated with* $\mathcal{A}$ is defined for $u, v \in A^*$ by $u \sim_{\mathcal{A}} v$ if $\forall q \in Q$, there is an initial run of $\mathcal{A}$ over u reaching q if and only if there is one over v. Note that for a trim deterministic automaton, there is a bijection (up to adding a sink state) between Q and the equivalence classes of $\mathcal{A}$. Similarly, the *left congruence associated with* $\mathcal{A}$ is defined for $x, y \in A^\omega$ by $x \approx_{\mathcal{A}} y$ if $\forall q \in Q$ there is a final run of $\mathcal{A}$ over x from q if and only if there is one over y. Given a right automaton there is a bijection between Q and the equivalence classes of $\approx_{\mathcal{A}}$. Finally, the *transition congruence of* $\mathcal{A}$ is defined for $u, v \in A^*$ by $u \equiv_{\mathcal{A}} v$ if $\forall p, q \in Q$, there is a run over u from p to q if and only if there is one over v. An automaton is called *aperiodic* if its transition congruence is aperiodic. A language is called *aperiodic* if there exists an aperiodic automaton recognizing it. A transducer is *aperiodic* if its underlying automaton is aperiodic and in that case the transduction it realizes is called *aperiodic*.

Given a right congruence $\sim$, the *left automaton associated with* $\sim$ is $\mathcal{A}_\sim = (Q_\sim, \Delta_\sim, I_\sim)$: $Q_\sim = A^*/_\sim$, $\Delta_\sim = \{([u], a, [ua]) \mid u \in A^*\}$, $I_\sim = \{[\epsilon]\}$. Given a left congruence $\approx$ and a right automaton $\mathcal{R}$, if $\approx_{\mathcal{R}} \sqsubseteq \approx$ then we say that $\mathcal{R}$ *recognizes* $\approx$. The existence of a canonical automaton for a left congruence is less obvious. From [6] we know that every ω-regular

language can be recognized by a right automaton. We rely on the construction of [6] and, abusing language, we denote the right automaton obtained in the next proposition as the *canonical right automaton recognizing a left congruence*:

▶ **Proposition 1.** Given a (regular) left congruence, we can compute in 2-ExpTime a right automaton recognizing it. Furthermore, this automaton is aperiodic if the congruence is aperiodic.

2 Sequential and quasi-sequential transductions

We define the syntactic congruence associated with any functional transduction over infinite words. Sequential functions are exactly the rational functions having a syntactic congruence of finite index, and being continuous over their domain. When removing this last condition on continuity, we obtain the class of quasi-sequential transductions. These transductions are also characterized by the weak twinning property [2].

We will show that for any sequential function, like in the case of finite words [8], we can define a canonical transducer, with a minimal underlying automaton. This minimal transducer extends the domain of the function to its closure.

▶ **Definition 2** ($\widehat{f}$ and $\overline{f}$)**.** Let $f : A^\omega \to B^\infty$ be a function, we define $\widehat{f} : A^* \to B^\infty$ by $\widehat{f}(u) = \bigwedge \{f(ux) \mid ux \in \mathrm{dom}(f)\}$. In other words, $\widehat{f}$ outputs the longest possible output that f could produce on any word that begins with u. We also define $\overline{f} : A^\omega \to B^\infty$ by setting $\overline{f}(x) = \lim_n \widehat{f}(x(:n))$, for $x \in \overline{\mathrm{dom}(f)}$.

We refer to $\overline{f}$ as the *sequential extension* of f. Note that if f is sequential, then $\overline{f}$ extends f over the closure $\overline{\mathrm{dom}(f)}$ of the domain of f.

▶ **Example 3.** We illustrate these definitions on three rational transductions, described in Table 1.

▶ **Definition 4** (syntactic congruence $\sim_f$)**.** The *syntactic congruence* associated with a transduction f is defined over A^* by $u \sim_f v$ if:

1. $\forall x \in A^\omega$, $ux \in \overline{\mathrm{dom}(f)} \Leftrightarrow vx \in \overline{\mathrm{dom}(f)}$, and
2. either $\widehat{f}(u)$ and $\widehat{f}(v)$ are both ultimately periodic with the same period (in normal form) or they are both finite and $\forall x \in A^\omega$ such that $ux, vx \in \mathrm{dom}(f)$, $\widehat{f}(u)^{-1} f(ux) = \widehat{f}(v)^{-1} f(vx)$.

▶ **Example 5.** Let us illustrate the definition of $\sim_f$ on f_{ab}, as defined in Table 1. The syntactic congruence $\sim_{f_{ab}}$ has only two classes: $[\epsilon]$ and $[a]$. Indeed, if we consider two finite words u and v, condition (1) on the domain is always true, and $\widehat{f_{ab}}(u)$ and $\widehat{f_{ab}}(v)$ are finite (ab-factors in u and v, respectively). Hence $u \sim_{f_{ab}} v$ if and only if $\forall x \in A^\omega$, $\widehat{f_{ab}}(u)^{-1} f_{ab}(ux) = \widehat{f_{ab}}(v)^{-1} f_{ab}(vx)$.

Let us analyze $\widehat{f_{ab}}(u)^{-1} f_{ab}(ux)$. If u does not end with an a, then $\widehat{f_{ab}}(u)^{-1} f_{ab}(ux) = ((ab)^n)^{-1}((ab)^{n+k}) = (ab)^k$ where n and k are the number of ab-factors in u and x, respectively. Now, if u ends with an a and x starts with a b, then a new ab-factor appears in ux and we get $\widehat{f_{ab}}(u)^{-1} f_{ab}(ux) = ((ab)^n)^{-1}((ab)^{n+k+1}) = (ab)^{k+1}$. This means that $\sim_{f_{ab}}$ contains exactly two classes: one for the words ending with an a, and one for the others.

The resulting transducer $\mathcal{T}_{f_{ab}}$ is depicted in Figure 4. Let us check for instance the transition from $[a]$ to $[\epsilon]$ when reading b. We have $[ab] = [\epsilon]$, so $([a], b, [\epsilon]) \in \Delta_{f_{ab}}$. From the definition, $o_{f_{ab}}([a], b, [\epsilon]) = \widehat{f_{ab}}(a)^{-1} \widehat{f_{ab}}(ab) = \epsilon^{-1}.ab = ab$.

▶ **Proposition 6.** Let f be a functional transduction, then $\sim_f$ is a right congruence.

Table 1 Examples of rational transductions, and their associated $\widehat{f}$ and $\overline{f}$ functions.

	f_{ab}	$f_{\#a}$	f_{blocks}
definition	maps a word over $\{a,b\}$ with a finite number of a's to the subsequence of ab-factors.	maps a word x over $\{a,b\}$ to a^ω if x contains an infinite number of a's, and to b^ω otherwise.	maps $u_1\#\dots\#u_n\#v$ where v does not contain $\#$, to $a_1^{\lvert u_1\rvert}\#\dots\#a_n^{\lvert u_n\rvert}\#w$ where $u_i \in \{a,b\}^*$, a_i is the last letter of u_i (if any), $w = a^\omega$ if v has an infinite number of a's, and $w = b^\omega$ otherwise.
A and B	$A = B = \{a,b\}$	$A = B = \{a,b\}$	$A = B = \{a,b,\#\}$
$\text{dom}(f)$	words over $\{a,b\}$ with a finite number of a's	$\{a,b\}^\omega$	words over $\{a,b,\#\}$ with a finite (non-zero) number of $\#$'s
examples	$f_{ab}(abbab^\omega) = abab$, $f_{ab}(b^\omega) = \epsilon$	$f_{\#a}(ab^\omega) = b^\omega$, $f_{\#a}((ab)^\omega) = a^\omega$	$f_{\text{blocks}}((ab\#)^n b^\omega) = (bb\#)^n b^\omega$, $f_{\text{blocks}}(\#(ab)^\omega) = a^\omega$.
$\widehat{f}$	$\widehat{f_{ab}}$ extracts the ab-factors, for instance $\widehat{f_{ab}}(abbabb) = abab$.	reading a finite prefix u does not give any insight on the output, thus $\widehat{f_{\#a}}(u) = \epsilon$	$\widehat{f_{\text{blocks}}}(u_1\#\dots\#u_n\#v) = a_1^{\lvert u_1\rvert}\#\dots\#a_n^{\lvert u_n\rvert}\#$ whenever v does not contain $\#$.
$\overline{f}$	$\overline{f_{ab}}$ is defined over $\text{dom}(\overline{f_{ab}}) = \{a,b\}^\omega$ and $\overline{f_{ab}}((ba)^\omega) = \lim_n \widehat{f_{ab}}((ba)^n) = \lim_n (ab)^{n-1} = (ab)^\omega$.	$\overline{f_{\#a}}(x) = \epsilon$ for every $x \in \{a,b\}^\omega$ as it is based on $\widehat{f_{\#a}}$	$\overline{f_{\text{blocks}}}(u_1\#\dots\#u_n\#v) = a_1^{\lvert u_1\rvert}\#\dots\#a_n^{\lvert u_n\rvert}\#$ whenever v does not contain $\#$.
class	sequential	quasi-sequential	not quasi-sequential

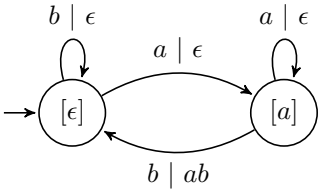


Figure 4 Transducer $\mathcal{T}_{f_{ab}}$.

From $\sim_f$ we define[3] the transducer $\mathcal{T}_f = (\mathcal{A}_f, i_f, o_f)$ with $\mathcal{A}_f = (Q_f, \Delta_f, I_f)$ and:

- $Q_f = A^*/_{\sim_f}$ and $I_f = \{[\epsilon]\}$
- $\Delta_f = \{([u], a, [ua]) \mid u \in A^*, a \in A, \exists x \text{ s.t. } uax \in \text{dom}(f)\}$
- $o_f([u], a, [ua]) = \begin{cases} \widehat{f}(u)^{-1}\widehat{f}(ua) & \text{if } \widehat{f}(ua) \text{ is finite} \\ \beta & \text{if } \widehat{f}(u) = \alpha\beta^\omega,\ \beta \neq \epsilon \\ \alpha & \text{if } \widehat{f}(u) \text{ is finite and } \widehat{f}(u)^{-1}\widehat{f}(ua) = \alpha\beta^\omega,\ \beta \neq \epsilon \end{cases}$
- $i_f([\epsilon]) = \begin{cases} \widehat{f}(\epsilon) & \text{if } \widehat{f}(\epsilon) \text{ is finite} \\ \alpha & \text{if } \widehat{f}(\epsilon) = \alpha\beta^\omega,\ \beta \neq \epsilon \end{cases}$

▶ Remark. Note that, in general, $\sim_f$ may have an infinite index, thus $\mathcal{T}_f$ may be infinite. This is the case for f_{blocks}: for two words $u = u_0\#w$ and $v = u_0\#w'$ with u_0ww' not containing $\#$, $u \sim_{f_{\text{blocks}}} v$ if and only if $|w| = |w'|$ and they end with the same letter. We will define below a subclass of rational transductions, which captures exactly finite $\sim_f$ (Theorem 12).

[3] We check in the long version that $\mathcal{T}_f$ is well-defined.

As shown below, the sequential transducer $\mathcal{T}_f$ computes the sequential extension $\overline{f}$ of f. If f is sequential then f and $\overline{f}$ coincide on $\mathrm{dom}(f)$.

▶ **Proposition 7.** Given a function f, the transducer $\mathcal{T}_f$ realizes $\overline{f}$.

We now focus on sequential transductions, and show first that $\mathcal{T}_f$ can be built in PTime.

▶ **Proposition 8.** There is a PTime algorithm that, for a given sequential transducer $\mathcal{T}$ realizing the function f, computes the transducer $\mathcal{T}_f$.

For sequential transductions we get a characterization, as stated in the next theorem. We will see that the first condition is equivalent to the weak twinning property. Thus, the next theorem adapts a result from [2] to the case where transducers may output finite words.

▶ **Theorem 9.** *A rational function f is sequential if and only if the following conditions hold:*

- $\sim_f$ *has finite index*
- $\overline{f}_{|dom(f)} = f$

If we remove the last restriction $\overline{f}_{|\mathrm{dom}(f)} = f$ in Theorem 9, we obtain a class of transductions where the output can be still generated deterministically (as for sequential transductions), although not necessarily in a progressive manner:

▶ **Definition 10.** A function f is called *quasi-sequential* if it is rational and $\sim_f$ has finite index.

Intuitively, quasi-sequential functions generalize the so-called *subsequential* functions on finite words to infinite words. For subsequential functions there is a final output associated with final states. Quasi-sequential functions can be shown to correspond to sequential transducers where final sets may have an associated word in A^∞. The output of an accepting run with such a final set is obtained by appending the associated word to the output word obtained through the transitions (if finite). Since we do not use this model in the present paper, we do not provide more details in the following. The following property and construction are now taken directly from [2]. As in the latter article, a state is called *constant* if the set of words produced by final runs from this state is a singleton.

▶ **Definition 11** (weak twinning property)**.** A transducer $\mathcal{T}$ is said to satisfy the *weak twinning property* (WTP) if for any initial runs $p_1 \xrightarrow{u|\alpha_1} q_1 \xrightarrow{v|\beta_1} q_1$ and $p_2 \xrightarrow{u|\alpha_2} q_2 \xrightarrow{v|\beta_2} q_2$ the following property holds:

- If q_1, q_2 are not constant then $\mathsf{del}(i(p_1)\alpha_1, i(p_2)\alpha_2) = \mathsf{del}(i(p_1)\alpha_1\beta_1, i(p_2)\alpha_2\beta_2)$
- If q_1 is not constant, q_2 is constant and produces the regular word γ, then either $\beta_1 = \epsilon$ or $i(p_1)\alpha_1\beta_1^\omega = i(p_2)\alpha_2\beta_2\gamma$

Note that if q_2 is constant and $\beta_2 \neq \epsilon$ then $\gamma = \beta_2^\omega$.

The authors of [2] provide a determinization procedure – which we call *subset construction with delays*– which terminates if and only if the transducer satisfies the WTP. We show that actually the procedure gives a transducer realizing the sequential extension of the function and we use this fact in Sec. 4 in order to compute a canonical look-ahead.

▶ **Theorem 12.** *Let $\mathcal{T}$ be a transducer realizing a function f, let $\mathcal{S}$ be the transducer obtained by subset construction with delays. The following statements are equivalent:*

1. *The transducer $\mathcal{T}$ satisfies the WTP*
2. *The transducer $\mathcal{S}$ is finite*
3. *f is quasi-sequential*

Furthermore, if $\mathcal{T}$ is aperiodic then $\mathcal{S}$ is aperiodic as well.

3 Rational transductions

Bimachines over infinite words. A *bimachine* over alphabets A, B is a tuple $\mathcal{B} = (\mathcal{L}, \mathcal{R}, i, o)$ where $\mathcal{L} = (Q_\mathcal{L}, \Delta_\mathcal{L}, \{l_0\})$ is a left automaton, $\mathcal{R} = (Q_\mathcal{R}, \Delta_\mathcal{R}, I, F)$ is a right automaton, $i : I \to B^*$ is the *initial function* and $o : Q_\mathcal{L} \times A \times Q_\mathcal{R} \to B^*$ is the *output function*. We have a semantic restriction that $[\![\mathcal{L}]\!] = \overline{[\![\mathcal{R}]\!]}$. The output produced on an infinite word $w \in [\![\mathcal{R}]\!]$ at position $i \geq 1$ is $\alpha_i = o(l, a, r)$, where l is the state reached in $\mathcal{L}$ after reading the prefix $w(:i-1)$ of w up to position $i-1$ (if defined), r is the state of the unique final run of $\mathcal{R}$ on w (if defined) reached by the suffix $w(i+1:)$ of w from position $i+1$ on, and $a = w(i)$. In other words, the output at position i is determined by the left context up to position $i-1$, the right context from position $i+1$ onwards, and the letter at position i. The output produced on w is $i(r_0)\alpha_1\alpha_2\cdots$, with $r_0 \in I$ the state from which there is a final run of $\mathcal{R}$ on w (if defined). Thus, the right automaton $\mathcal{R}$ provides a look-ahead and the output depends both on the state of $\mathcal{L}$ and the unique final run of $\mathcal{R}$ on the given word. The transduction *realized* by $\mathcal{B}$ is denoted by $[\![\mathcal{B}]\!]$. Note that $[\![\mathcal{B}]\!]$ is defined over $[\![\mathcal{R}]\!]$. A bimachine is called *aperiodic* if both its automata are aperiodic.

▶ **Example 13.** Let us define a bimachine for f_{ab}, the function that outputs ab-factors of the input over $\{a, b\}$, if this input has a finite number of a's. We use as left automaton the underlying automaton of the transducer in Figure 2, without its Muller acceptance condition. This automaton will only be used to store the last letter read. The domain has to be checked by the right automaton, and we choose the one in Figure 3. As output functions, we let $i(q) = \epsilon$ for the initial states of the right automaton, and let $o(q_a, b, r) = ab$ for $r \in \{1, 2\}$, and $o(l, c, r) = \epsilon$ for all other states l, r of the left and right automata, and letter $c \in \{a, b\}$.

Left minimization. We show how to minimize the left automaton of a bimachine with respect to a right automaton $\mathcal{R}$. The procedure is very similar to the minimization for sequential transducers. The objects we use are the same as in Section 2, but relativized to the right context defined by the look-ahead provided by the right automaton $\mathcal{R}$. The bimachine with minimal left automaton with respect to the right automaton $\mathcal{R}$ is the bimachine $\mathcal{B}_f^\mathcal{R}$ defined below.

Recall that the left congruence $\approx_\mathcal{R}$ of a right automaton $\mathcal{R}$ sets $x \approx_\mathcal{R} y$ if the unique state from which there is a final run on x is the same as for y. Let $f : A^\omega \to B^\infty$ be a function and let $\mathcal{R} = (Q_\mathcal{R}, \Delta_\mathcal{R}, I, F)$ be a right automaton recognizing $\mathrm{dom}(f)$. We write $[x]^\mathcal{R}$ for the class of a word x with respect to $\approx_\mathcal{R}$, and, abusing notations, for the state of $Q_\mathcal{R}$ from which words of $[x]^\mathcal{R}$ have a final run. We define $\widehat{f}_x : A^* \to B^\infty$ by setting $\widehat{f}_x(u) = \bigwedge \{f(uy) \mid y \approx_\mathcal{R} x\}$. Note that there are finitely many functions $\widehat{f}_x$, one for each equivalence class of $\approx_\mathcal{R}$. We also define $\overline{f}^\mathcal{R} : A^\omega \to B^\infty$, by setting $\overline{f}^\mathcal{R}(x) = \lim_n \widehat{f}_{x(n+1:)}(x(:n))$. The transduction $\overline{f}^\mathcal{R}$ is defined over $\mathrm{dom}(f)$.

▶ **Definition 14** ($\mathcal{R}$-syntactic congruence)**.** The *$\mathcal{R}$-syntactic congruence of f* is defined over A^* by letting $u \sim_f^\mathcal{R} v$ if:

1. $\forall x \in A^\omega$, $ux \in \mathrm{dom}(f) \Leftrightarrow vx \in \mathrm{dom}(f)$, and
2. for any $x \in A^\omega$, either $\widehat{f}_x(u)$ and $\widehat{f}_x(v)$ are both infinite with the same ultimate period (in normal form) or they are both finite and $\widehat{f}_x(u)^{-1} f(ux) = \widehat{f}_x(v)^{-1} f(vx)$.

Similarly to the sequential case, we define from $\sim_f^\mathcal{R}$ a bimachine $\mathcal{B}_f^\mathcal{R} = \left(\mathcal{L}_f^\mathcal{R}, \mathcal{R}, i_f^\mathcal{R}, o_f^\mathcal{R}\right)$ with right automaton $\mathcal{R}$, and left automaton $\mathcal{L}_f^\mathcal{R} = \left(Q_f^\mathcal{R}, \Delta_f^\mathcal{R}, I_f^\mathcal{R}\right)$ corresponding to $\sim_f^\mathcal{R}$. To simplify notations we denote the congruence class of a word u with respect to $\sim_f^\mathcal{R}$ by $[u]$. Abusing notations we also write $[x]^\mathcal{R}$ for the state of $\mathcal{R}$ from which x has an accepting run.

- $Q_f^{\mathcal{R}} = A^*/_{\sim_f^{\mathcal{R}}}$ and $I_f^{\mathcal{R}} = \{[\epsilon]\}$
- $\Delta_f^{\mathcal{R}} = \{([u], a, [ua]) \mid u \in A^*, a \in A, uax \in \mathrm{dom}(f) \text{ for some } x \in A^\omega\}$
- $o_f([u], a, [x]^{\mathcal{R}}) = \begin{cases} \widehat{f}_{ax}(u)^{-1}\widehat{f}_x(ua) & \text{if } \widehat{f}_x(ua) \text{ is finite} \\ \beta & \text{if } \widehat{f}_{ax}(u) = \alpha\beta^\omega,\ \beta \neq \epsilon \\ \alpha & \text{if } \widehat{f}_{ax}(u) \text{ is finite, } \widehat{f}_{ax}(u)^{-1}\widehat{f}(ua) = \alpha\beta^\omega \\ & \quad \text{and } \beta \neq \epsilon \end{cases}$
- $i_f([x]^{\mathcal{R}}) = \begin{cases} \widehat{f}_x(\epsilon) & \text{if } \widehat{f}_x(\epsilon) \text{ is finite} \\ \alpha & \text{if } \widehat{f}_x(\epsilon) = \alpha\beta^\omega, \beta \neq \epsilon \end{cases}$

We show in the long version of this paper that $\mathcal{B}_f^{\mathcal{R}}$ is well-defined, and exhibit some of its properties. We also describe in the long version a polynomial time algorithm that computes $\mathcal{B}_f^{\mathcal{R}}$ from a bimachine with right automaton $\mathcal{R}$, with a technique similar to the sequential case.

From transducers to bimachines. For the theorem below, recall that $\sim_{\mathcal{A}}$ denotes the right congruence of an automaton $\mathcal{A}$. The left congruence $\approx_{\mathcal{A}}$ of an automaton $\mathcal{A}$ sets $x \approx_{\mathcal{A}} y$ if for every state q of $\mathcal{A}$, there is some final run on x from q if and only if there is one on y.

▶ **Theorem 15.** *Given a transducer with underlying automaton $\mathcal{A}$ and a right automaton $\mathcal{R}$ with $\approx_{\mathcal{R}} \sqsubseteq \approx_{\mathcal{A}}$. Then $\sim_{\mathcal{A}} \sqsubseteq \sim_f^{\mathcal{R}}$ and the bimachine $\mathcal{B}_f^{\mathcal{R}}$ realizes f.*

In particular any aperiodic transduction can be realized by an aperiodic bimachine.

The other direction also holds: from a bimachine we can build an equivalent (unambiguous) transducer, by taking the product of the left and right automata of the bimachine. The construction is not hard but given in the long version. By Theorem 15 and Proposition 1 we obtain:

▶ **Theorem 16.** *A function is rational (resp. rational and aperiodic) if and only if it can be realized by a bimachine (resp. aperiodic bimachine).*

Labelings and bimachines. We define the *labeling function* associated with a right automaton $\mathcal{R} = (Q, \Delta, I, F)$ by the right transducer $\ell(\mathcal{R}) = (\mathcal{R}, i, o)$, with $i(q) = \epsilon$ and $o(p, a, q) = (a, q)$. Intuitively, the labeling function labels each position with the look-ahead information about the suffix provided by $\mathcal{R}$. For a transduction f we define $f_{\mathcal{R}} = f \circ [\![\ell(\mathcal{R})]\!]^{-1}$. Note that $f_{\mathcal{R}}$ is a function, since the labeling is injective (because $\mathcal{R}$ is unambiguous). Thus, $f_{\mathcal{R}}$ corresponds to f defined over words enriched by the look-ahead information of $\mathcal{R}$.

▶ **Proposition 17.** Let f be a transduction and let $\mathcal{R}$ be a right automaton. There exists a bimachine $\mathcal{B}$ realizing f with $\mathcal{R}$ as a right automaton if and only if $f_{\mathcal{R}}$ is left-sequential. Furthermore, assuming that $\mathcal{R}$ is aperiodic, then $\sim_f^{\mathcal{R}}$ is aperiodic if and only if $f_{\mathcal{R}}$ is aperiodic.

We say that a transducer is *letter-to-letter* if its initial output function always outputs the empty word and its output function always outputs a single letter. The following corollary states the classical result of [11] but over infinite words, and generalizes a result of [5].

▶ **Corollary 18.** *For any rational function f, there exists a left-sequential (right-seq. resp.) function g and a letter-to-letter right-sequential (left-seq. resp.) function h such that $f = g \circ h$.*

4 Canonical machines

The goal of this section is to define a canonical bimachine for any rational function. By canonicity we mean that it should be machine-independent. Our ultimate goal is to show that the canonical bimachine suffices to decide the algebraic properties we are interested in. To get a canonical bimachine, we need a right automaton for the look-ahead that is 1) canonical, 2) coarse-grained enough to preserve algebraic properties, and 3) fine-grained enough to obtain a deterministic left automaton (and hence a bimachine).

We define the delay congruence and show that it is the coarsest left congruence such that any automaton $\mathcal{R}$ recognizing it satisfies that $f_{\mathcal{R}}$ is quasi-sequential (Proposition 21). However, this congruence is, in general, too coarse to make $f_{\mathcal{R}}$ sequential. We then introduce the ultimate congruence, and show how to combine these two congruences to build a canonical bimachine.

Let f be a transduction. We define the *delay* between $x, y \in A^\omega$ with respect to f by: $\mathsf{del}_f(x, y) = \{\mathsf{del}(f(ux), f(uy)) \mid ux, uy \in \mathrm{dom}(f)\}$. The following definition is taken from [20, 3].

▶ **Definition 19** (delay congruence)**.** The *delay congruence* of f is defined by setting $x \overset{\Delta}{\approx}_f y$ for $x, y \in A^\omega$ if (1) for all $u \in A^*$, $ux \in \mathrm{dom}(f) \Leftrightarrow uy \in \mathrm{dom}(f)$, and (2) $|\mathsf{del}_f(x, y)| < \infty$.

▶ **Example 20.** Let us illustrate the above definition on f_{blocks} (recall Example 3). We consider $x = u_1\#\dots\#u_n\#v$ and $y = u'_1\#\dots\#u'_n\#v'$ where v and v' are infinite words not containing $\#$. Note that $x \overset{\Delta}{\approx}_{f_{\mathrm{blocks}}} y$ if and only if u_1, u'_1 are either both empty, or end with the same letter. Indeed, if the latter holds then $\mathsf{del}(f(ux), f(uy)) = \mathsf{del}(f(x), f(y))$. Conversely, if both u_1, u'_1 are non-empty but end with different letters, then for any u without $\#$, $\mathsf{del}(f_{\mathrm{blocks}}(ux), f_{\mathrm{blocks}}(uy)) = (f(ux), f(uy))$. If $u_1 = \epsilon$ and u, u'_1 end with different letters, then again, $\mathsf{del}(f_{\mathrm{blocks}}(ux), f_{\mathrm{blocks}}(uy)) = (f(ux), f(uy))$. There are two more classes with respect to $\overset{\Delta}{\approx}_{f_{\mathrm{blocks}}}$, one for infinitely many $\#$, and one for no $\#$.

The look-ahead $\overset{\Delta}{\approx}_{f_{\mathrm{blocks}}}$ provides enough information to transform the blocks deterministically (we only need the last letter before the next $\#$), but not enough information to produce the output after the last $\#$ deterministically.

The following proposition shows that the delay congruence, when used as a look-ahead (see the definition of $f_{\mathcal{R}}$ page 11), transforms any rational function into a quasi-sequential one.

▶ **Proposition 21.** Let f be a transduction and let $\mathcal{R}$ be a right automaton recognizing $\mathrm{dom}(f)$. Then $f_{\mathcal{R}}$ is quasi-sequential iff $\approx_{\mathcal{R}} \sqsubseteq \overset{\Delta}{\approx}_f$. In particular, if f is aperiodic then $\overset{\Delta}{\approx}_f$ is aperiodic.

The delay congruence is minimal, *i.e.* coarsest, among right congruences of bimachines realizing a function, and we show in the long version that it can be computed in PTime from a bimachine.

▶ **Proposition 22.** Given a transducer $\mathcal{T}$ (resp. a bimachine $\mathcal{B}$) with underlying automaton $\mathcal{A}$ (resp. right automaton $\mathcal{R}$) realizing a function f, we have that $\approx_{\mathcal{A}}$ (resp. $\approx_{\mathcal{R}}$) is finer than $\overset{\Delta}{\approx}_f$.

Canonical machine for quasi-sequential functions. As noted in [2], the class of quasi-sequential functions, or equivalently, the class of functions satisfying the WTP, is strictly larger than the class of sequential functions. The last left congruence that we define now will be fine enough to make a quasi-sequential function sequential. By taking the intersection between

this congruence and the left delay congruence we will obtain a congruence that is fine enough to make any rational function sequential. However, it should be noted that this look-ahead is not minimal, in the sense that it is not necessarily coarser than any look-ahead that is fine enough to realize the function.

▶ **Definition 23** (Ultimate congruence)**.** We define the *ultimate congruence* of a rational function f by setting $x \overset{\cup}{\approx}_f y$ for $x, y \in A^\omega$ if the following conditions hold:

- For all $u \in A^*$, $ux \in \mathrm{dom}(f) \Leftrightarrow uy \in \mathrm{dom}(f)$
- If $ux \in \mathrm{dom}(f)$ then $\widehat{f}(u) = \overline{f}(ux) \Leftrightarrow \widehat{f}(u) = \overline{f}(uy)$ Moreover, if $\widehat{f}(u) = \overline{f}(ux)$ then $f(ux) = f(uy)$.

Observe that $\widehat{f}(u) \preceq \overline{f}(ux)$ for every $ux \in \mathrm{dom}(f)$. So the intuition behind $\widehat{f}(u) = \overline{f}(ux)$ is that no finite look-ahead on x can help to output $f(ux)$ deterministically after u. And the intuition behind $f(ux) = f(uy)$ is that the missing outputs $\widehat{f}(u)^{-1} f(ux)$ and $\widehat{f}(u)^{-1} f(uy)$ have to be equal, which is equivalent to $f(ux) = f(uy)$. Now, for a given class of $\overset{\cup}{\approx}_f$ as look-ahead, a left automaton would know the missing output and start producing it. We show in the long version that $\overset{\cup}{\approx}_f$ is a left congruence.

▶ **Example 24.** Recall the function f_{blocks} defined in Example 3. $\widehat{f_{\mathrm{blocks}}}$ maps every block to its output and stops at the last #. Hence $\widehat{f_{\mathrm{blocks}}}(u) = \overline{f_{\mathrm{blocks}}}(ux)$ if and only if x does not contain #. When $\widehat{f_{\mathrm{blocks}}}(u) = \overline{f_{\mathrm{blocks}}}(ux)$, we have $f(ux) = f(uy)$ if and only if x and y both contain an infinite number of a's, or none of them does. The congruence classes of $\overset{\cup}{\approx}_{f_{\mathrm{blocks}}}$ are thus: a) words x with an infinite number of # (yielding ux outside the domain), b) words x with a finite (non-zero) number of #, c) words without #, with an infinite number of a's, d) words without #, with a finite number of a's. This is precisely the information lacking in the look-ahead provided by $\overset{\triangle}{\approx}_{f_{\mathrm{blocks}}}$ (see Example 20) to obtain a look-ahead allowing a sequential processing of the input.

▶ **Proposition 25.** For a quasi-sequential transduction f, the ultimate congruence $\overset{\cup}{\approx}_f$ has finite index. If f is given as a bimachine, $\overset{\cup}{\approx}_f$ can be computed in 2-ExpTime. Furthermore, if f is aperiodic then $\overset{\cup}{\approx}_f$ is aperiodic.

Let $\mathcal{R}$ be a right automaton recognizing $\overset{\cup}{\approx}_f$. We define the bimachine $\mathcal{U}_f^{\mathcal{R}} = (\mathcal{A}_f, \mathcal{R}, i_f, o_{\mathcal{R}})$ with $\mathcal{A}_f$ and i_f (as in Section 2), and for $o_{\mathcal{R}}$ we take:

$$o_{\mathcal{R}}([u], a, [x]_{\mathcal{R}}) = \begin{cases} \widehat{f}(u)^{-1}\widehat{f}(ua) & \text{if } \widehat{f}(ua) \prec \overline{f}(uax) \\ \beta & \text{if } \widehat{f}(u) = \overline{f}(uax) \text{ and } \widehat{f}(u)^{-1} f(uax) = \alpha\beta^\omega \\ \alpha & \text{if } \widehat{f}(u) \prec \widehat{f}(ua) = \overline{f}(uax) \text{ and } \widehat{f}(u)^{-1} f(uax) = \alpha\beta^\omega \end{cases}$$

The following lemma states that $\mathcal{U}_f^{\mathcal{R}}$ realizes f.

▶ **Lemma 26.** *Let f be a quasi-sequential transduction, and let $\mathcal{R}$ be a right automaton recognizing the ultimate congruence $\overset{\cup}{\approx}_f$, then $\mathcal{U}_f^{\mathcal{R}}$ realizes f.*

Let $\mathcal{R}$ be the canonical right automaton of $\overset{\cup}{\approx}_f$. By the previous lemma, there exists a bimachine with $\mathcal{R}$ as right automaton realizing f. By minimizing its left automaton with respect to $\mathcal{R}$, we obtain a canonical bimachine for f.

▶ **Corollary 27.** *Let f be a quasi-sequential transduction, and let $\mathcal{R}$ be the canonical right automaton of the ultimate congruence $\overset{\cup}{\approx}_f$, then $\mathcal{B}_f^{\mathcal{R}}$ realizes f (and is finite).*

Canonical bimachine. We finally show that by composing the information given by the delay and the ultimate congruences, we obtain a canonical bimachine for any rational function. Let us make clear what we mean by composition. Let $\mathcal{R}_1 = (Q_1, \Delta_1, I_1, F_1)$ be a right automaton and let $\mathcal{R}_2 = (Q_1, \Delta_2, I_2, F_2)$ be a right automaton over $A \times Q_1$. The automaton $\mathcal{R}_1 \bowtie \mathcal{R}_2$ is defined as $(Q_1 \times Q_2, \Delta_{\{1,2\}}, I_1 \times I_2, F_1 \times F_2)$ with $F_1 \times F_2 = \{P_1 \times P_2 \mid\ P_1 \in F_1,\ P_2 \in F_2\}$ and $\Delta_{\{1,2\}} = \{((s_1, s_2), a, (r_1, r_2) \mid (s_1, a, r_1) \in \Delta_1,\ (s_2, (a, r_1), r_2) \in \Delta_2\}$, which is a right automaton.

▶ **Lemma 28.** *Let $\mathcal{R}_1 = (Q_1, \Delta, I, F)$ be a right automaton and let $\mathcal{R}_2$ be a right automaton over $A \times Q_1$. Then $[\![\ell(\mathcal{R}_2)]\!] \circ [\![\ell(\mathcal{R}_1)]\!] = [\![\ell(\mathcal{R}_1 \bowtie \mathcal{R}_2)]\!]$ (up to the isomorphism between $(A \times Q_1) \times Q_2$ and $A \times (Q_1 \times Q_2)$).*

We can now state our main result. In our construction we focused on clarity and compositionality and we obtain a several-fold exponential complexity. At the cost of greater technicality, one should obtain a tighter result.

▶ **Theorem 29** (Canonical Bimachine)**.** *Let f be a transduction given by a bimachine, let $\mathcal{R}_1$ be the canonical automaton of the delay congruence $\overset{\Delta}{\approx}_f$, and let $\mathcal{R}_2$ be the canonical automaton of the ultimate congruence $\overset{\cup}{\approx}_{(f_{\mathcal{R}_1})}$. Then the bimachine $\mathcal{B}_f^{\mathcal{R}_1 \bowtie \mathcal{R}_2}$ realizes f. Furthermore if f is aperiodic then $\mathcal{B}_f^{\mathcal{R}_1 \bowtie \mathcal{R}_2}$ is aperiodic.*

Proof. Let f be a transduction, let $\mathcal{R}_1$ be the canonical automaton of the delay congruence $\overset{\Delta}{\approx}_f$ and let $\mathcal{R}_2$ be the canonical automaton of the ultimate congruence $\overset{\cup}{\approx}_{(f_{\mathcal{R}_1})}$. Since $\mathcal{R}_1$ recognizes $\overset{\Delta}{\approx}_f$, we know according to Proposition 22 that $f_{\mathcal{R}_1}$ is quasi-sequential. Hence since $\mathcal{R}_2$ is finer than $\overset{\cup}{\approx}_{(f_{\mathcal{R}_1})}$, we know from Cor. 27 that the bimachine $\mathcal{B}_{f_{\mathcal{R}_1}}^{\mathcal{R}_2}$ realizes f. From Proposition 17 we obtain that $(f_{\mathcal{R}_1})_{\mathcal{R}_2}$, the function obtained by composing the labelings $\ell(\mathcal{R}_2)$ and $\ell(\mathcal{R}_1)$, is left-sequential. We use Lemma 28 to obtain that $f_{\mathcal{R}_1 \bowtie \mathcal{R}_2}$ is left-sequential and thus, again by Proposition 17 we know there is a bimachine with $\mathcal{R}_1 \bowtie \mathcal{R}_2$ as right automaton which realizes f. In particular, $\mathcal{B}_f^{\mathcal{R}_1 \bowtie \mathcal{R}_2}$ realizes f.

If we assume that f is aperiodic, we obtain from Proposition 22 that $\mathcal{R}_1$ is aperiodic and from Proposition 17 that $f_{\mathcal{R}_1}$ is aperiodic. Hence from Proposition 25 we have that $\mathcal{R}_2$ is aperiodic. Again from Proposition 17, we have that $(f_{\mathcal{R}_1})_{\mathcal{R}_2} = f_{\mathcal{R}_1 \bowtie \mathcal{R}_2}$ is aperiodic. A third time from Proposition 17 we have that $\mathcal{B}_f^{\mathcal{R}_1 \bowtie \mathcal{R}_2}$ is aperiodic. ◀

Note that the right automaton constructed in Proposition 1 is actually a right *Büchi* automaton. So our result would still hold for bimachines with Büchi right automata.

5 First-Order Definability Problem

In this section, we show that given a transducer $\mathcal{T}$ realizing a transduction $[\![\mathcal{T}]\!] : A^\omega \to B^\infty$, one can decide whether $[\![\mathcal{T}]\!]$ is first-order definable (FO-definable). First, let us recall the notion of FO-definability for word languages. Any word $w \in A^\infty$ is seen as a structure of domain $\{1, \dots, |w|\}$ linearly ordered by $\preceq$ and with unary predicates $a(x)$, for all $a \in A$. By FO we denote the first-order logic over these predicates, and by MSO the extension of FO with quantification over sets and membership tests $x \in X$ (see for instance [22] for a detailed definition). We write $w \models \phi$ if some word w satisfies a formula ϕ, and $\phi(x_1, \dots, x_n)$ any formula ϕ with n free first-order variables $x_1, \dots, x_n$. Interpreted over words in A^ω (resp. A^∞), any sentence ϕ defines a language $[\![\phi]\!] \subseteq A^\omega$ (resp. $[\![\phi]\!] \subseteq A^\infty$) defined as the set of words satisfying ϕ. *E.g.* the sentence $\phi_0 = \forall x, y \cdot a(x) \wedge b(y) \to x \preceq y$, interpreted on A^ω, defines the language $a^\omega \cup a^*b^\omega$. Interpreted on A^∞, it defines the language $a^\omega \cup a^*b^\omega \cup a^*b^*$. A language L is said to be FO-definable (resp. MSO-definable) if $L = [\![\phi]\!]$ for some sentence $\phi \in$ FO (resp. $\phi \in$ MSO).

Definability of transductions. An MSO-transducer is a tuple $\mathcal{F} = (A, B, \phi_{dom}, V, \mu)$ where ϕ_{dom} is an MSO-sentence, V is a finite subset of B^* and μ a function mapping any word $v \in V$ to some MSO-formula (over alphabet A) denoted $\phi_v(x)$, with one-free variable. An *FO-transducer* is an MSO-transducer which uses only FO-formulas. Any MSO-transducer defines a transduction denoted $[\![\mathcal{F}]\!] \subseteq A^\omega \times B^\infty$ such that $(u, v) \in [\![\mathcal{F}]\!]$ if $u \models \phi_{dom}$ and there exists $(v_i)_{i\geq 1}$ such that $v = v_1 v_2 v_3 \dots$ and for all $i \geq 1$, $v_i \in V$ and $u \models \phi_{v_i}(i)$. We say that $f : A^\omega \to B^\infty$ is MSO- (resp. FO-) definable if there exists some MSO- (resp. FO-) transducer $\mathcal{F}$ such that $[\![\mathcal{F}]\!] = f$.

For example the functional transduction which erases all a's of any input ω-word over $\{a, b\}$ is defined by $\phi_{dom} = \top$ and the two formulas $\phi_\epsilon(x) = a(x)$ and $\phi_b(x) = b(x)$. The functional transduction mapping any word of the form $a^n b^\omega$ to $a^{\lfloor n/2 \rfloor} b^\omega$ is not FO-definable, even though its domain is. Intuitively, the formula $\phi_a(x)$ would have to decide whether x is an odd or even position, which is a typical non FO-definable property. It is one of the goal of this paper to automatically verify that such a property is indeed not FO. It is however MSO-definable with $\phi_{dom} = \phi_0 \wedge \exists x \cdot b(x)$, where ϕ_0 has been defined before, and the three formulas $\phi_\epsilon(x) = a(x) \wedge odd(x)$, $\phi_a(x) = a(x) \wedge even(x)$ (properties which are MSO-definable) and $\phi_b(x) = b(x)$.

As a remark, Courcelle has defined in the context of graph transductions the notion of MSO-transducers [9], which can also be restricted to FO-transducers. Cast to infinite words, Courcelle's formalism is strictly more expressive than rational functions, as they allow to mirror factors of the input word for instance. Restricted to the so called *order-preserving* Courcelle transducers [4, 12], they are equivalent to our MSO- and FO-transducers, however with a more complicated definition. This equivalence can be seen, for finite words, in the proof of Theorem 4 in [12]. The same proof works for infinite words as well.

We first exhibit a correspondence between logics and transducers, the proof of which is similar to the finite case [12], but requires some additional results on aperiodic automata on ω-words.

▶ **Theorem 30** (Logic-transducer correspondences). *Let $f : A^\omega \to B^\infty$. Then:*

- *f is MSO-definable if and only if it is realizable by some transducer.*
- *f is FO-definable if and only if it is realizable by some aperiodic transducer.*

We obtain the following decidability result (in elementary complexity if the input is a transducer).

▶ **Theorem 31.** *It is decidable whether a rational function $f : A^\omega \to B^\infty$, given as a transducer or equivalently as an MSO-transducer, is definable in FO.*

Proof. By Theorem 30, it suffices to show that f is aperiodic, *i.e.* definable by some aperiodic transducer. By Theorem 16, one can construct a bimachine which is aperiodic if and only if f is. So, it suffices to construct this bimachine and to test its aperiodicity, *i.e.*, whether its left and right automata are both aperiodic, a property which is decidable [10]. ◀

References

1 André Arnold. A Syntactic Congruence for Rational ω-Languages. *Theoretical Computer Science*, 39(2–3):333–335, August 1985. Note.

2 Marie-Pierre Béal and Olivier Carton. Determinization of Transducers over Infinite Words: The General Case. *Theory Comput. Syst.*, 37(4):483–502, 2004.

3 Adrien Boiret, Aurélien Lemay, and Joachim Niehren. Learning Rational Functions. In *Developments in Language Theory - 16th International Conference, DLT 2012, Taipei, Taiwan, August 14-17, 2012. Proceedings*, pages 273–283, 2012.

4 Mikolaj Bojanczyk. Transducers with Origin Information. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 26–37. Springer, 2014.

5 Olivier Carton. Right-Sequential Functions on Infinite Words. In *Computer Science - Theory and Applications, 5th International Computer Science Symposium in Russia, CSR 2010, Kazan, Russia, June 16-20, 2010. Proceedings*, pages 96–106, 2010.

6 Olivier Carton and Max Michel. Unambiguous Büchi automata. *Theor. Comput. Sci.*, 297(1-3):37–81, 2003.

7 Olivier Carton, Dominique Perrin, and Jean-Eric Pin. Automata and semigroups recognizing infinite words. In *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*., pages 133–168, 2008.

8 Christian Choffrut. Minimizing subsequential transducers: a survey. *Theor. Comput. Sci.*, 292(1):131–143, 2003.

9 Bruno Courcelle. Monadic Second-Order Definable Graph Transductions: A Survey. *Theor. Comput. Sci.*, 126(1):53–75, 1994.

10 Volker Diekert and Paul Gastin. First-order definable languages. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata*, volume 2 of *Texts in Logic and Games*, pages 261–306. Amsterdam University Press, 2008.

11 Calvin C. Elgot and Jorge E. Mezei. On Relations Defined by Generalized Finite Automata. *IBM Journal of Research and Development*, 9(1):47–68, 1965.

12 Emmanuel Filiot. Logic-Automata Connections for Transformations. In *Indian Conference on Logic and Its Applications (ICLA)*, volume 8923 of *Lecture Notes in Computer Science*, pages 30–57. Springer, 2015.

13 Emmanuel Filiot, Olivier Gauwin, and Nathan Lhote. Aperiodicity of Rational Functions Is PSPACE-Complete. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016, December 13-15, 2016, Chennai, India*, pages 13:1–13:15, 2016.

14 Emmanuel Filiot, Olivier Gauwin, and Nathan Lhote. First-order definability of rational transductions: An algebraic approach. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, pages 387–396. ACM, 2016.

15 Emmanuel Filiot, Olivier Gauwin, and Nathan Lhote. Logical and Algebraic Characterizations of Rational Transductions. *CoRR*, abs/1705.03726, 2017. Available from: `http://arxiv.org/abs/1705.03726`.

16 Françoise Gire. Two Decidability Problems for Infinite Words. *Inf. Process. Lett.*, 22(3):135–140, 1986.

17 Robert McNaughton and Seymour Papert. *Counter-free automata.* M.I.T. Press, 1971.

18 D. Perrin. Recent results on automata and infinite words. In M. P. Chytil and V. Koubek, editors, *Proceedings of the 11th Symposium on Mathematical Foundations of Computer Science*, volume 176 of *LNCS*, pages 134–148, Praha, Czechoslovakia, September 1984. Springer.

19 Christophe Prieur. How to decide continuity of rational functions on infinite words. *Theor. Comput. Sci.*, 276(1-2):445–447, 2002.

20 Christophe Reutenauer and Marcel Paul Schützenberger. Minimization of Rational Word Functions. *SIAM J. Comput.*, 20(4):669–685, 1991.

21 Marcel-Paul Schützenberger. On Finite Monoids Having Only Trivial Subgroups. *Information and Control*, 8(2):190–194, 1965.

22 W. Thomas. Languages, Automata and Logic. In A. Salomaa and G. Rozenberg, editors, *Handbook of Formal Languages*, volume 3, Beyond Words. Springer, Berlin, 1997.

23 Wolfgang Thomas. Star-Free Regular Sets of omega-Sequences. *Information and Control*, 42(2):148–156, August 1979.

24 Wolfgang Thomas. A Combinatorial Approach to the Theory of omega-Automata. *Information and Control*, 48(3):261–283, 1981.

25 Thomas Wilke. Past, Present, and Infinite Future. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 95:1–95:14, 2016.

Reachability for Two-Counter Machines with One Test and One Reset

Alain Finkel
LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France
alain.finkel@lsv.ens-paris-saclay.fr

Jérôme Leroux
LaBRI, Université de Bordeaux, CNRS, Bordeaux-INP, Talence, France
jerome.leroux@labri.fr

Grégoire Sutre
LaBRI, Université de Bordeaux, CNRS, Bordeaux-INP, Talence, France
gregoire.sutre@labri.fr

Abstract

We prove that the reachability relation of two-counter machines with one zero-test and one reset is Presburger-definable and effectively computable. Our proof is based on the introduction of two classes of Presburger-definable relations effectively stable by transitive closure. This approach generalizes and simplifies the existing different proofs and it solves an open problem introduced by Finkel and Sutre in 2000.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Counter machine, Vector addition system, Reachability problem, Formal verification, Presburger arithmetic, Infinite-state system

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.31

Funding This work was supported by the grant ANR-17-CE40-0028 of the French National Research Agency ANR (project BRAVAS).

Acknowledgements The work reported was carried out in the framework of ReLaX, UMI2000 (ENS Paris-Saclay, CNRS, Univ. Bordeaux, CMI, IMSc).

1 Introduction

Context. Vector addition systems with states (VASS) are equivalent to Petri nets and to counter machines without the ability to test counters for zero. Although VASS have been studied since the 1970's, they remain fascinating since there are still some important open problems like the complexity of reachability (known between ExpSpace and cubic-Ackermannian) or even an efficient (in practice) algorithm to solve reachability. In 1979, Hopcroft and Pansiot [13] gave an algorithm that computes the Presburger-definable reachability set of a 2-dim VASS, hence VASS in dimension 2 are more easy to verify and they enjoy interesting properties like reachability and equivalence of reachability sets, for instance, are both decidable. Unfortunately, these results do not extend in dimension 3 or for 2-dim VASS with zero-tests on the two counters: the reachability set (hence also the reachability relation) is not Presburger-definable for 3-dim VASS [13]; reachability, and all non-trivial problems, are undecidable for 2-dim VASS extended with zero-tests on the two counters.

38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018).
Editors: Sumit Ganguly and Paritosh Pandya; Article No. 31; pp. 31:1–31:14

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

Table 1 Reachability sets ($post^*$ and pre^*) and reachability relation ($\xrightarrow{*}$) for extensions of 2-dimensional VASS. We let $\simeq$ denote the existence of mutual reductions between two classes of machines that preserve the effective Preburger-definability of the reachability sets and relation. The contributions of this paper are indicated in boldface.

Class	$Post^*$	Pre^*	$\xrightarrow{*}$
$\mathbf{T}_1\mathbf{Tr}_2 \simeq \mathbf{T}_{1,2} \simeq \mathbf{T}_{1,2}\mathbf{R}_{1,2}\mathbf{Tr}_{1,2}$	Not Recursive	Not Recursive	Not Recursive
$\mathbf{T}_1\mathbf{R}_2 \simeq \mathbf{T}_1\mathbf{R}_{1,2}\mathbf{Tr}_1$	**Eff.** Presburger	Eff. Presburger	**Eff. Presburger**
$\mathbf{R}_{1,2}\mathbf{Tr}_1 \simeq \mathbf{R}_{1,2}\mathbf{Tr}_{1,2}$	Eff. Presburger	Eff. Presburger	**Eff. Presburger**
$\mathbf{T}_1 \simeq \mathbf{T}_1\mathbf{R}_1\mathbf{Tr}_1$	Eff. Presburger	Eff. Presburger	**Eff. Presburger**
2-dim VASS	Eff. Presburger	Eff. Presburger	Eff. Presburger

In 2004, Leroux and Sutre proved that the reachability relation of a 2-dim VASS is also effectively Presburger-definable [17] and this is not a consequence of the Presburger-definability of the reachability set. As a matter of fact, there exist counter machines (even 3-dim VASS) with a Presburger-definable reachability set but with a non Presburger-definable reachability relation [13, 17]. But, for all recursive 2-dim extended VASS, the reachability sets are Presburger-definable [11, 10]. More precisely, let us denote by $\mathbf{T}_I\mathbf{R}_J\mathbf{Tr}_K$, with $I, J, K \subseteq \{1, 2\}$, the class of 2-dim VASS extended with zero-tests on the I-counters, resets on the J-counters and transfers from the K-counters. For instance, $\mathbf{T}_{\{1\}}\mathbf{R}_{\{1,2\}}\mathbf{Tr}_\emptyset$, also written $\mathbf{T}_1\mathbf{R}_{1,2}$ for short, is the class of 2-dim VASS extended with zero-tests on the first counter, resets on both counters, and no transfer. The relations between classes from [11] are recalled in Figure 1 and the class $\mathbf{T}_1\mathbf{R}_2$ has been shown to be the "maximal" class having Presburger-definable $post^*$ and pre^* reachability sets [11]. However, it was unknown whether the Presburger-definable reachability set $post^*$ can be effectively computed or not. In fact, even the boundedness problem (is the reachability set $post^*$ finite?) was open for this class.

Contributions. Our main contribution is a proof that the reachability relation of counter machines in $\mathbf{T}_1\mathbf{R}_2$ is effectively Presburger-definable. Our proof relies on the effective Presburger-definability of the reachability relation for 2-dim VASS [17]. The impact of our result is threefold.

- We solve the main *open problem* in [11] which was the question of the existence of an algorithm that computes the Presburger-definable reachability set for two-counter machines in $\mathbf{T}_1\mathbf{R}_2$.
- In fact, we prove a stronger result, namely that the *reachability relation* of counter machines in $\mathbf{T}_1\mathbf{R}_2$ is Presburger-definable and computable. This completes the decidability picture of 2-dim extended VASS.
- We provide a *simple* proof of the effective Presburger-definability of the reachability relation in $\mathbf{T}_1\mathbf{R}_2$. As an immediate consequence, one may deduce all existing results [11, 10] for 2-dim extended VASS and our proof unifies all different existing proofs on 2-dim extended VASS, including the proof in [6] that the boundedness problem is decidable for the class $\mathbf{R}_{1,2}$ of 2-dim VASS extended with resets on both counters.

Related work. VASS have been extended with resets, transfers and zero-tests. Extended VASS with resets and transfers are well structured transition systems [9] hence termination and coverability are decidable; but reachability and boundedness are undecidable (except boundedness which is decidable for extended VASS with transfers) [5, 6]. The reachability and place-boundedness problems are decidable for extended VASS with *one* zero-test [19, 3, 8, 4].

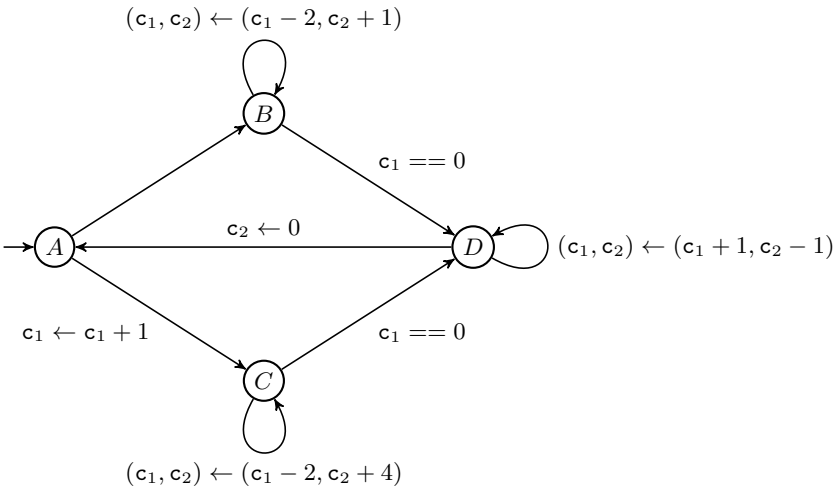


Figure 1 A 2-dimensional VASS extended with zero-tests on the first counter and resets on the second counter (shortly called TRVASS).

Recently, Akshay et al. studied extended Petri nets with a hierarchy on places and with resets, transfers and zero-tests [1]. As a counter is a particular case of a stack, it is natural to study counter machines with one stack. Termination and boundedness are decidable for VASS with one stack [16] but surprisingly, the decidability status of the reachability problem is open for VASS with one stack, both in arbitrary dimension and in dimension 1. We only know that reachability and coverability for VASS with one stack are TOWER-hard [14, 15].

Outline. We present in Section 2 an example of 2-dim extended VASS in $\mathbf{T}_1\mathbf{R}_2$. This example motivates the study of two classes of binary relations on natural numbers, namely diagonal relations in Section 3 and horizontal relations in Section 4. These two classes of relations are combined in Section 5 into a new class of one counter automata with effectively Presburger-definable reachability relations. These automata are used in Section 6 to compute the reachability relations of 2-dim extended VASS in $\mathbf{T}_1\mathbf{R}_2$.

For the remainder of the paper, 2-dim extended VASS in $\mathbf{T}_1\mathbf{R}_2$ are shortly called TRVASS.

2 Motivating Example

Figure 1 depicts an example of a TRVASS. There are four *states* A, B, C and D, and two *counters* c_1 and c_2. Following the standard semantics of vector addition systems, these counters range over natural numbers. The operations labeling the three loops and the edge from A to C are classical addition instructions of vector addition systems. In dimension 2, these addition instructions are always of the form $(c_1, c_2) \leftarrow (c_1 + a_1, c_2 + a_2)$ where a_1 and a_2 are integer constants. For instance, the instruction $(c_1, c_2) \leftarrow (c_1 - 2, c_2 + 1)$ labeling the loop on B means that c_1 is decremented by 2 and at the same time c_2 is incremented by 1. As the counters must remain nonnegative, this instruction may be executed (i.e., the loop on B may be taken) only if $c_1 \geq 2$. In addition to classical addition instructions, TRVASS may test the first counter for zero, written $c_1 == 0$, and reset the second counter to zero, written $c_2 \leftarrow 0$.

The operational semantics of a TRVASS is given, as for vector addition systems, by an infinite directed graph whose nodes are called *configurations* and whose edges are called *steps*. Formal definitions will be given in Section 6. For the TRVASS of Figure 1, configurations are triples $q(x_1, x_2)$ where $q \in \{A, B, C, D\}$ is a state and $x_1, x_2 \in \mathbb{N}$ are values of the counters c_1 and c_2, respectively. It is understood that $\mathbb{N}$ denotes the set of natural numbers $\{0, 1, 2, \ldots\}$. There is a step from a configuration $p(x_1, x_2)$ to a configuration $q(y_1, y_2)$, written $p(x_1, x_2) \to q(y_1, y_2)$, if there is an edge from p to q labeled by an operation (1) that can be executed from the counter values (x_1, x_2) and (2) whose execution changes the counter values from (x_1, x_2) to (y_1, y_2). Here, we have the steps $B(5, 1) \to B(3, 2)$, $C(0, 2) \to D(0, 2)$ and $D(7, 3) \to A(7, 0)$. But there is no step from $C(1, 2)$ and there is no step to $A(7, 1)$.

The *reachability relation* of a TRVASS, written $\xrightarrow{*}$, is the reflexive-transitive closure of the step relation $\to$. The reachability relation is one of the main objects of interest for verification purposes. Coming back to our example of Figure 1, we have $A(1, 0) \xrightarrow{*} A(2, 0)$ since we have the following contiguous sequence of steps:

$$A(1, 0) \to C(2, 0) \to C(0, 4) \to D(0, 4) \to D(1, 3) \to D(2, 2) \to A(2, 0)$$

By removing the steps $\to D(1, 3) \to D(2, 2)$, we also get that $A(1, 0) \xrightarrow{*} A(0, 0)$. In fact, it can be shown that $A(1, 0) \xrightarrow{*} A(y, 0)$ for every $y \in \mathbb{N}$, thanks to the following pattern, where k denotes an odd natural number and $i \in \{1, 2\}$:

$$A(k, 0) \to C(k + 1, 0) \xrightarrow{*} D(2k + 2, 0) \xrightarrow{*} D(k + i, k + 2 - i) \xrightarrow{*} A(k + i, 0)$$

One may wonder whether it also holds that $A(x, 0) \xrightarrow{*} A(y, 0)$ for every $x, y \in \mathbb{N}$. A consequence of our main result (see Theorem 14) is that we can do even better: we can compute the set of pairs $(x, y) \in \mathbb{N} \times \mathbb{N}$ such that $A(x, 0) \xrightarrow{*} A(y, 0)$, as a formula in Presburger arithmetic[1].

▶ Remark. It is well-known that zero-tests are more expressive than resets. Indeed, a reset $\mathsf{c}_1 \leftarrow 0$ can be simulated by a loop $\mathsf{c}_1 \leftarrow \mathsf{c}_1 - 1$ followed by a zero-test $\mathsf{c}_1 == 0$. A crucial difference between resets and zero-tests is *monotony*. In a 2-dimensional VASS extended with resets on both counters (shortly called RRVASS), larger counter values are always better, in the sense that every behavior from a configuration $q(x_1, x_2)$ can be reproduced from a configuration $q(x_1', x_2')$ with $x_1' \geq x_1$ and $x_2' \geq x_2$. This is not true anymore in presence of zero-tests. This difference makes the analysis of TRVASS more complex than that of RRVASS, as illustrated in the following example.

▶ **Example 1.** Consider the RRVASS obtained from the TRVASS of Figure 1 by replacing the two zero-tests (from B to D and from C to D) with resets $\mathsf{c}_1 \leftarrow 0$. Suppose that we want to show that c_1 is unbounded in state A from $A(1, 0)$, i.e., $A(1, 0) \xrightarrow{*} A(y, 0)$ for infinitely many $y \in \mathbb{N}$. A natural strategy is, starting from $A(x, 0)$ with $x \geq 1$, to reach $D(0, y)$ with y as large as possible (without visiting A on the way), and then to reach $A(y, 0)$ by taking the "transfer" loop on D as much as possible. By iterating this strategy, we get

$$A(1, 0) \xrightarrow{*} D(0, 4) \xrightarrow{*} A(4, 0) \xrightarrow{*} D(0, 8) \xrightarrow{*} A(8, 0) \xrightarrow{*} D(0, 16) \xrightarrow{*} A(16, 0) \cdots$$

This witnesses that c_1 is unbounded in state A from $A(1, 0)$. In comparison, this strategy does not work for the original TRVASS of Figure 1. Indeed, we get

$$A(1, 0) \xrightarrow{*} D(0, 4) \xrightarrow{*} A(4, 0) \xrightarrow{*} D(0, 2) \xrightarrow{*} A(2, 0) \xrightarrow{*} D(0, 1) \xrightarrow{*} A(1, 0)$$

by following this strategy. This is because the only way to reach D from a configuration $A(x, 0)$ with x even is via B.

[1] Recall that Presburger arithmetic [18] is the first-order theory of the natural numbers with addition.

The rest of the paper is devoted to the proof that the reachability relation of a TRVASS is *effectively Presburger-definable*, i.e., there is an algorithm that, given a TRVASS and two states p and q, computes a formula $\varphi(x_1, x_2, y_1, y_2)$ in Presburger arithmetic whose models are precisely the quadruples (x_1, x_2, y_1, y_2) of natural numbers such that $p(x_1, x_2) \xrightarrow{*} q(y_1, y_2)$. It is already known that the reachability relation is effectively Presburger-definable in the absence of zero-tests and resets [17]. Obviously, the counter c_1 is zero after a zero-test $\mathsf{c}_1 == 0$ and, similarly, the counter c_2 is zero after a reset $\mathsf{c}_2 \leftarrow 0$. So we focus on the reachability subrelations between configurations where at least one of the counters is zero, for instance, $\{(x, 0, 0, y) \mid p(x, 0) \xrightarrow{*} q(0, y)\}$. Such a subrelation can be seen as a (binary) relation on $\mathbb{N}$. This motivates our study in Sections 3 and 4 of two classes of relations on $\mathbb{N}$ that naturally stem from the operational semantics of TRVASS.

3 Diagonal Relations

We call a relation $R \subseteq \mathbb{N} \times \mathbb{N}$ *diagonal* when $(x, y) \in R$ implies $(x + c, y + c) \in R$ for every $c \in \mathbb{N}$. For instance, the *identity relation* on $\mathbb{N}$, namely $\{(x, x) \mid x \in \mathbb{N}\}$, is a diagonal relation. The usual order $\leq$ on natural numbers is also a diagonal relation. It is readily seen that the class of diagonal relations is closed under union, intersection, composition, and transitive closure. In this section, we show that the transitive closure of a diagonal Presburger-definable relation is effectively Presburger-definable. Our study of diagonal relations is motivated by the following observation.

▶ Remark. The reachability subrelations $\{(x, y) \mid p(0, x) \xrightarrow{*} q(0, y)\}$, where p and q are states, are diagonal in a TRVASS with no reset. Analogously, the reachability subrelations $\{(x, y) \mid p(x, 0) \xrightarrow{*} q(y, 0)\}$ are diagonal in a TRVASS with no zero-test.

▶ **Example 2.** Let us consider the diagonal relation $R \subseteq \mathbb{N} \times \mathbb{N}$ defined by $(x, y) \in R$ if, and only if, the Presburger formula $x \leq y \wedge y \leq 2x$ holds. It is routinely checked that the transitive closure R^+ of R satisfies $(x, y) \in R^+$ if, and only if, the Presburger formula $(x = 0 \Leftrightarrow y = 0) \wedge x \leq y$ holds.

We fix, for the remainder of this section, a diagonal relation $R \subseteq \mathbb{N} \times \mathbb{N}$. Consider the subsets I_R and D_R of $\mathbb{N}$ defined by

$$I_R \stackrel{\text{def}}{=} \{x \mid \exists y : (x, y) \in R \wedge x < y\} \qquad D_R \stackrel{\text{def}}{=} \{y \mid \exists x : (x, y) \in R \wedge x > y\}$$

Since R is diagonal, the sets I_R and D_R are upward-closed, meaning that $x \in I_R$ implies $x' \in I_R$ for every $x' \geq x$ (and similarly for D_R). If $x \in I_R$ then $(x, x + \delta) \in R$ for some positive integer $\delta > 0$. Since R is diagonal, $(x', x' + \delta) \in R$ for every $x' \geq x$. So the pair $(x, x + \delta)$ can be viewed as an "increasing loop" that applies to every $x' \geq x$. Similarly, if $y \in D_R$ then there is a "decreasing loop" $(y + \delta, y) \in R$ that applies to every $y' \geq y$. We are mostly interested in increasing and decreasing loops that apply to every element of I_R and D_R, respectively. This leads us to the following definitions:

$$\alpha \stackrel{\text{def}}{=} \begin{cases} \min\{\delta > 0 \mid \forall x \in I_R : (x, x + \delta) \in R\} & \text{if } I_R \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$\beta \stackrel{\text{def}}{=} \begin{cases} \min\{\delta > 0 \mid \forall y \in D_R : (y + \delta, y) \in R\} & \text{if } D_R \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Let us explain why the natural numbers α and β are well-defined. If $I_R \neq \emptyset$ then there exists $\delta > 0$ such that $(m, m + \delta) \in R$ where $m = \min I_R$. It follows from diagonality

of R that $(x, x+\delta) \in R$ for every $x \geq m$, hence, for every $x \in I_R$. Therefore the set $\{\delta > 0 \mid \forall x \in I_R : (x, x+\delta) \in R\}$ is non-empty, and so it has a minimum. A similar argument shows that $\{\delta > 0 \mid \forall y \in D_R : (y+\delta, y) \in R\}$ is non-empty when $D_R \neq \emptyset$.

We are now almost ready to provide a characterization of the transitive closure of R^+. To do so, we introduce the relations $\mathtt{Inc}_R$ and $\mathtt{Dec}_R$ on $\mathbb{N}$ defined by

$$\mathtt{Inc}_R(x,y) \stackrel{\text{def}}{=} (x = y) \vee (x \in I_R \wedge \exists h \in \mathbb{N} : y = x + h\alpha)$$
$$\mathtt{Dec}_R(x,y) \stackrel{\text{def}}{=} (x = y) \vee (y \in D_R \wedge \exists k \in \mathbb{N} : x = y + k\beta)$$

We let $\fatsemi$ denote *relational composition* ($S \fatsemi R \stackrel{\text{def}}{=} \{(x,z) \mid \exists y : x\, S\, y\, R\, z\}$). The *powers* of a relation R are inductively defined by $R^1 \stackrel{\text{def}}{=} R$ and $R^{n+1} \stackrel{\text{def}}{=} R \fatsemi R^n$.

▶ **Lemma 3.** *It holds that* $R^+ = \mathtt{Inc}_R \fatsemi (R \cup \cdots \cup R^{\alpha+\beta+1}) \fatsemi \mathtt{Dec}_R$.

Proof. We introduce the relation $C = \mathtt{Inc}_R \fatsemi (R \cup \cdots \cup R^{\alpha+\beta+1}) \fatsemi \mathtt{Dec}_R$, so as to reduce clutter. To prove that $C \subseteq R^+$, we show that $\mathtt{Inc}_R$ and $\mathtt{Dec}_R$ are both contained in R^*. Let $(x,y) \in \mathtt{Inc}_R$. If $x = y$ then $(x,y) \in R^*$. Otherwise, $x \in I_R$ and there exists $h \in \mathbb{N}$ such that $y = x + h\alpha$. Moreover, h and α are positive as $x \neq y$. It follows from $x \in I_R$ and $\alpha > 0$ that $(x, x+\alpha) \in R$. Since R is diagonal, we derive that $(x, x+\alpha), \ldots, (x+(h-1)\alpha, x+h\alpha)$ are all in R. Hence, $(x,y) \in R^+$. We have shown that $\mathtt{Inc}_R \subseteq R^*$. Now let $(x,y) \in \mathtt{Dec}_R$. If $x = y$ then $(x,y) \in R^*$. Otherwise, $y \in D_R$ and there exists $k \in \mathbb{N}$ such that $x = y + k\beta$. Moreover, k and β are positive as $x \neq y$. It follows from $y \in D_R$ and $\beta > 0$ that $(y+\beta, y) \in R$. Since R is diagonal, we derive that $(y+k\beta, y+(k-1)\beta), \ldots, (y+\beta, y)$ are all in R. Hence, $(x,y) \in R^+$. We have shown that $\mathtt{Dec}_R \subseteq R^*$. We derive from $\mathtt{Inc}_R \subseteq R^*$ and $\mathtt{Dec}_R \subseteq R^*$ that $C \subseteq R^+$.

Let us now prove the converse inclusion $R^+ \subseteq C$. We first observe that $\mathtt{Inc}_R = \mathtt{Inc}_R^*$ and $\mathtt{Dec}_R = \mathtt{Dec}_R^*$. These equalities easily follow from the definitions of $\mathtt{Inc}_R$ and $\mathtt{Dec}_R$. As a consequence, we get that

$$C \;=\; \mathtt{Inc}_R^* \fatsemi (R \cup \cdots \cup R^{\alpha+\beta+1}) \fatsemi \mathtt{Dec}_R^* \tag{3}$$

Let us prove by induction on n that $R^n \subseteq C$ for all $n \geq 1$. The base cases $n = 1, \ldots, \alpha+\beta+1$ are trivial. Assume that $R^m \subseteq C$ for all $1 \leq m < n$, where $n \geq \alpha+\beta+2$, and let us show that this inclusion also holds for $m = n$. Let $(x,y) \in R^n$. There exists $x_0, \ldots, x_n$ such that $x = x_0\, R\, x_1\, R \cdots R\, x_n = y$. We start by showing the two following properties, as they will be crucial for the rest of the proof.

$$x \notin I_R \implies x_0 \geq x_1 \geq \cdots \geq x_n \qquad \text{and} \qquad y \notin D_R \implies x_0 \leq x_1 \leq \cdots \leq x_n$$

We prove these properties by contraposition. If $x_i < x_{i+1}$ for some $0 \leq i < n$, then we may, w.l.o.g., choose the first such i. This entails that $x_0 \geq \cdots \geq x_i$. Moreover, $x_i \in I_R$ since $x_i < x_{i+1}$ and $x_i\, R\, x_{i+1}$. It follows that $x = x_0 \in I_R$ as I_R is upward-closed. Similarly, if $x_{i-1} > x_i$ for some $0 < i \leq n$, then we may, w.l.o.g., choose the last such i. This entails that $x_i \leq \cdots \leq x_n$. Moreover, $x_i \in D_R$ since $x_{i-1} > x_i$ and $x_{i-1}\, R\, x_i$. It follows that $y = x_n \in D_R$ as D_R is upward-closed.

To prove that $(x,y) \in C$, we consider four cases, depending on the membership of x in I_R and on the membership of y in D_R.

If $x \notin I_R$ and $y \notin D_R$ then $x_0 = x_1 = \cdots = x_n$. This means in particular that $x_0\, R\, x_n$, hence, $x = x_0\, C\, x_n = y$.

If $x \notin I_R$ and $y \in D_R$ then $x_0 \geq x_1 \geq \cdots \geq x_n$. Note that $\beta > 0$ since D_R is non-empty. Since $n \geq \beta$, there exists $0 \leq i < j \leq n$ such that $x_i \equiv x_j \pmod{\beta}$, hence, $x_i = x_j + k\beta$ for some $k \in \mathbb{N}$. Recall that $x = x_0\, R^i\, x_i$ and $x_j\, R^{n-j}\, x_n = y$. As R is diagonal, we derive that $x_i\, R^{n-j}\, y'$ where $y' = y + k\beta$. We obtain that $x\, R^{n+i-j}\, y'$. It follows from the induction hypothesis that $x\, C\, y'$. Moreover, we have $(y', y) \in \mathtt{Dec}_R$ since $y \in D_R$ and $y' = y + k\beta$. Hence, $x\, (C \fatsemi \mathtt{Dec}_R)\, y$ and we derive from Equation 3 that $x\, C\, y$.

If $x \in I_R$ and $y \notin D_R$ then $x_0 \leq x_1 \leq \cdots \leq x_n$. Note that $\alpha > 0$ since I_R is non-empty. Since $n \geq \alpha$, there exists $0 \leq i < j \leq n$ such that $x_i \equiv x_j \pmod{\alpha}$, hence, $x_j = x_i + h\alpha$ for some $h \in \mathbb{N}$. Recall that $x = x_0\, R^i\, x_i$ and $x_j\, R^{n-j}\, x_n = y$. As R is diagonal, we derive that $x'\, R^i\, x_j$ where $x' = x + h\alpha$. We obtain that $x'\, R^{n+i-j}\, y$. It follows from the induction hypothesis that $x'\, C\, y$. Moreover, we have $(x, x') \in \mathtt{Inc}_R$ since $x \in I_R$ and $x' = x + h\alpha$. Hence, $x\, (\mathtt{Inc}_R \fatsemi C)\, y$ and we derive from Equation 3 that $x\, C\, y$.

If $x \in I_R$ and $y \in D_R$ then both α and β are positive. Since $n \geq \alpha$, there exists $0 \leq i < j \leq n$ such that $x_i \equiv x_j \pmod{\alpha}$. If $x_i \leq x_j$ then $x_j = x_i + h\alpha$ for some $h \in \mathbb{N}$ and we may proceed as in the case $x \in I_R \wedge y \notin D_R$ to show that $x\, C\, y$. Otherwise, $x_i = x_j + k\alpha$ for some $k \in \mathbb{N}$. Recall that $x = x_0\, R^i\, x_i$ and $x_j\, R^{n-j}\, x_n = y$. As R is diagonal, we derive that $x'\, R^i\, z'\, R^{n-j}\, y'$ where $x' = x + k\alpha(\beta - 1)$, $z' = x_i + k\alpha(\beta - 1) = x_j + k\alpha\beta$ and $y' = y + k\alpha\beta$. We obtain that $x'\, R^{n+i-j}\, y'$. It follows from the induction hypothesis that $x'\, C\, y'$. Moreover, we have $(x, x') \in \mathtt{Inc}_R$ since $x \in I_R$ and $x' = x + k\alpha(\beta - 1)$, and we also have $(y', y) \in \mathtt{Dec}_R$ since $y \in D_R$ and $y' = y + k\alpha\beta$. Hence, $x\, (\mathtt{Inc}_R \fatsemi C \fatsemi \mathtt{Dec}_R)\, y$ and we derive from Equation 3 that $x\, C\, y$. ◀

We derive the following theorem.

▶ **Theorem 4.** *The transitive closure of a diagonal Presburger-definable relation is effectively Presburger-definable.*

Proof. Assume that $\varphi_R(x, y)$ is a Presburger formula denoting a diagonal relation R. The sets I_R and D_R are defined by the Presburger formulas $\exists y : \varphi_R(x, y) \wedge x < y$ and $\exists x : \varphi_R(x, y) \wedge x > y$, respectively. The natural numbers α and β defined in Equations 1 and 2 are obviously computable from φ_R. So the characterization given in Lemma 3 immediately provides a computable Presburger formula denoting R^+. ◀

4 Horizontal Relations

A relation $R \subseteq \mathbb{N} \times \mathbb{N}$ is said to be *horizontal* if $(x, y) \in R$ implies $(x + c, y) \in R$ for every $c \in \mathbb{N}$. The class of horizontal relations is clearly stable by union, intersection, composition, and transitive closure. In this section we prove that the transitive closure of a horizontal Presburger-definable relation is effectively Presburger-definable. Our study of horizontal relations is motivated by the following observation.

▶ Remark. The reachability subrelations $\{(x, y) \mid p(0, x) \xrightarrow{*} \xrightarrow{c_2 \leftarrow 0} \xrightarrow{*} q(y, 0)\}$, where p and q are states, are horizontal in a TRVASS.

▶ **Example 5.** Let us consider the following horizontal relation R:

$$R \stackrel{\text{def}}{=} \{(x, y) \mid 2y \leq x \ \vee \ (y \in 4\mathbb{N} \wedge y \leq 2x + 2)\}$$

We prove that R^+ is equal to $C \stackrel{\text{def}}{=} \{(x, y) \mid x = 0 \Rightarrow y = 0\}$ as follows. Since $R \subseteq C$ and C is transitive, we get $R^+ \subseteq C$. Conversely, let $(x, y) \in C$. If $x = 0$ then $y = 0$ and from $(0, 0) \in R$ we derive $(x, y) \in R^+$. So, we can assume that $x \geq 1$. In that case $(x, 4) \in R$ and

$(4z, 4(z+1)) \in R$ for every $z > 0$. It follows that $(x, n) \in R^+$ for every $n \in 4 + 4\mathbb{N}$. Moreover, there exists such an n satisfying $2y \leq n$. For such an n, we have $(x, n) \in R^+$ and $(n, y) \in R$. We deduce that $(x, y) \in R^+$. It follows that $R^+ = C$.

The effective Presburger-definability of the transitive closure comes from the following characterization.

▶ **Lemma 6.** *For every horizontal relation R we have:*

$$R^+ = \{(x, y) \mid \exists z : (z, y) \in R \wedge \forall u : x \leq u < z \Rightarrow \exists v : (u, v) \in R \wedge u < v\} \quad (4)$$

Proof. Assume first that $(x, y) \in R^+$. There exists a sequence $x_0, \ldots x_k$ such that $x = x_0\, R\, x_1 \ldots R\, x_k = y$ with $k \geq 1$. Let $z = x_{k-1}$ and let us prove that for every $u \in \{x, \ldots, z-1\}$ there exists $v > u$ such that $(u, v) \in R$. If $z \leq x$ we are done. So we can assume that $z > x$. Since $x_0 \leq u$, there exists a maximal $j \in \{1, \ldots, k\}$ such that $x_{j-1} \leq u$. Let $v = x_j$ and observe that $(u, v) \in R$. Since $x_{k-1} = z > u$, it follows that $j < k$ and by maximality of j we deduce that $x_j > u$. Therefore $v > u$.

Conversely, let us consider $(x, y) \in \mathbb{N} \times \mathbb{N}$ such that there exists z satisfying $(z, y) \in R$ and such that for every $u \in \{x, \ldots, z-1\}$ there exists $v > u$ such that $(u, v) \in R$. Notice that there exists a sequence $x_0 < \cdots < x_k$ with $k \geq 0$ such that $x = x_0\, R\, x_1 \ldots R\, x_k \geq z$. It follows that $(x, x_k) \in R^*$. Moreover, since $(z, y) \in R$, $z \leq x_k$, and R is horizontal we deduce that $(x_k, y) \in R$. It follows that $(x, y) \in R^+$. ◀

The previous lemma shows that the transitive closure of a horizontal relation R denoted by a Presburger formula φ_R is denoted by the Presburger formula obtained from (4) by replacing $(z, y) \in R$ and $(u, v) \in R$ by $\varphi_R(z, y)$ and $\varphi_R(u, v)$ respectively. We have proved the following theorem.

▶ **Theorem 7.** *The transitive closure of a horizontal Presburger-definable relation is effectively Presburger-definable.*

5 Presburger Automata

We exhibit in this section a general class of one counter automata with effectively Presburger-definable reachability relations. These automata will be used in the next section to compute the reachability relations of TRVASS.

A *Presburger automaton* is a pair $\mathcal{P} = (Q, \Delta)$ where Q is a finite set of *states*, and Δ is a finite set of *transitions* (p, R, q) where $p, q \in Q$ and $R \subseteq \mathbb{N} \times \mathbb{N}$ is a relation denoted by a Presburger formula (which is left implicit). A *configuration* is a pair $(q, x) \in Q \times \mathbb{N}$, also written as $q(x)$ in the sequel. The *one-step* relation $\rightarrow_{\mathcal{P}}$ is the binary relation over configurations defined by $p(x) \rightarrow_{\mathcal{P}} q(y)$ if there exists $(p, R, q) \in \Delta$ such that $(x, y) \in R$. The *reachability relation* $\stackrel{*}{\rightarrow}_{\mathcal{P}}$ is defined as the reflexive-transitive closure of $\rightarrow_{\mathcal{P}}$.

▶ Remark. The reflexive-transitive closure R^* of a Presburger-definable relation $R \subseteq \mathbb{N} \times \mathbb{N}$ need not be Presburger-definable, in general. For instance, if $R = \{(x, y) \in \mathbb{N} \times \mathbb{N} \mid y = 2x\}$ then R^* is the relation $\{(x, y) \in \mathbb{N} \times \mathbb{N} \mid \exists k \in \mathbb{N} : y = 2^k x\}$, which is not definable in Presburger arithmetic. Worse, a simple reduction from the halting problem for Minsky machines shows that membership of a pair (x, y) in R^* is undecidable (where R is a Presburger-definable relation given as input along with x and y).

A consequence of the above remark is that the reachability problem for Presburger automata is undecidable, even if we restrict ourselves to Presburger automata with a single state and a single transition. This comes from the fact that transitions can use arbitrary Presburger-definable relations. We will exhibit a subclass of Presburger automata with effectively Presburger-definable reachability relations (hence, with a decidable reachability problem) by limiting the expressive power of the transitions occurring on cycles. We say that a transition (p, R, q) is *diagonal* if R is diagonal, *horizontal* if R is horizontal, and *ordinary* if it is neither diagonal nor horizontal. Note that a relation on $\mathbb{N}$ may be both diagonal and horizontal, for instance $\{(x, y) \in \mathbb{N} \times \mathbb{N} \mid y \leq 2x\}$. A *cycle* is non-empty sequence of transitions $(p_1, R_1, q_1), \ldots, (p_n, R_n, q_n)$ such that $q_n = p_1$ and $q_i = p_{i+1}$ for all $1 \leq i < n$.

▶ **Lemma 8.** *Let $\mathcal{P}$ be a Presburger automaton. If every cycle of $\mathcal{P}$ contains only diagonal transitions then $\xrightarrow{*}_{\mathcal{P}}$ is effectively Presburger-definable.*

Proof. We first observe that $\xrightarrow{*}_{\mathcal{P}}$ is effectively Presburger-definable when $\mathcal{P} = (Q, \Delta)$ is a Presburger automaton whose transitions are all diagonal. Indeed, we may view $\mathcal{P}$ as a finite-state automaton over the finite alphabet $\{R \mid (p, R, q) \in \Delta\}$. For every states p and q, we may compute a regular expression denoting the language accepted by $\mathcal{P}$ with initial state p and final state q. The obvious evaluation of this regular expression (concatenation $\cdot$ becomes relational composition $\fatsemi$, sum $+$ becomes union $\cup$, and star $\star$ becomes reflexive-transitive closure $*$) yields the relation $\{(x, y) \mid p(x) \xrightarrow{*}_{\mathcal{P}} q(y)\}$. This evaluation is computable because Presburger-definable diagonal relations are effectively closed under union, composition and reflexive-transitive closure (as an immediate consequence of Theorem 4). We have shown that $\xrightarrow{*}_{\mathcal{P}}$ is effectively Presburger-definable when all transitions of $\mathcal{P}$ are diagonal.

We now prove the lemma. Let $\mathcal{P} = (Q, \Delta)$ be a Presburger automaton such that every cycle of $\mathcal{P}$ contains only diagonal transitions. Let $\mathcal{N}$ be the Presburger automaton obtained from $\mathcal{P}$ by keeping only diagonal transitions. Consider two configurations $p(x)$ and $q(y)$. It is readily seen that $p(x) \xrightarrow{*}_{\mathcal{P}} q(y)$ if, and only if, there exists $1 \leq k \leq |Q|$, $s_1, \ldots, s_k \in Q$ and $x_1, y_1, \ldots, x_k, y_k \in \mathbb{N}$ such that $p(x) = s_1(x_1)$, $s_k(y_k) = q(y)$ and

$$s_1(x_1) \xrightarrow{*}_{\mathcal{P}} s_1(y_1) \rightarrow_{\mathcal{P}} s_2(x_2) \xrightarrow{*}_{\mathcal{P}} s_2(y_2) \cdots s_{k-1}(y_{k-1}) \rightarrow_{\mathcal{P}} s_k(x_k) \xrightarrow{*}_{\mathcal{P}} s_k(y_k)$$

Observe that for every state $s \in Q$ and for every $x, y \in \mathbb{N}$, $s(x) \xrightarrow{*}_{\mathcal{P}} s(y)$ if, and only if, $s(x) \xrightarrow{*}_{\mathcal{N}} s(y)$. Moreover, $\xrightarrow{*}_{\mathcal{N}}$ is effectively Presburger-definable since all transitions of $\mathcal{N}$ are diagonal. We derive from the above characterization of $\xrightarrow{*}_{\mathcal{P}}$ that $\xrightarrow{*}_{\mathcal{P}}$ is also effectively Presburger-definable. ◀

We say that a Presburger automaton $\mathcal{P}$ is *shallow* if every cycle that contains an ordinary transition also contains a horizontal transition. Shallowness of Presburger automata is decidable. This follows from two easy observations. Firstly, diagonality and horizontality of Presburger-definable relations on $\mathbb{N}$ are decidable, since these properties can be expressed in Presburger arithmetic. Secondly, a Presburger automaton is shallow if, and only if, every *simple* cycle containing an ordinary transition also contains a horizontal transition. We now show the main result of this section.

▶ **Theorem 9.** *The reachability relation of a shallow Presburger automaton is effectively Presburger-definable.*

Proof. By induction on the number of horizontal transitions. The base case follows from Lemma 8. Indeed, if $\mathcal{P}$ is a shallow Presburger automaton with no horizontal transition then every cycle of $\mathcal{P}$ contains only diagonal transitions. Assume that the theorem holds for every

shallow Presburger automaton with n horizontal transitions, where $n \in \mathbb{N}$. Let $\mathcal{P} = (Q, \Delta)$ be a Presburger automaton with $n+1$ horizontal transitions. Pick a horizontal transition $(p, R, q) \in \Delta$ and let $\mathcal{N}$ be the Presburger automaton obtained from $\mathcal{P}$ by removing the transition (p, R, q). Let S denote the reachability relation from q to p in $\mathcal{N}$, namely the relation $S = \{(y, x) \mid q(y) \xrightarrow{*}_{\mathcal{N}} p(x)\}$. It is readily seen that, for every configurations $s(x)$ and $t(y)$ of $\mathcal{P}$, $s(x) \xrightarrow{*}_{\mathcal{P}} t(y)$ if, and only if, $s(x) \xrightarrow{*}_{\mathcal{N}} t(y)$ or there exists $x', y' \in \mathbb{N}$ such that

$$s(x) \xrightarrow{*}_{\mathcal{N}} p(x') \quad \wedge \quad (x', y') \in ((R \fatsemi S)^* \fatsemi R) \quad \wedge \quad q(y') \xrightarrow{*}_{\mathcal{N}} t(y)$$

By induction hypothesis, the relation $\xrightarrow{*}_{\mathcal{N}}$ is effectively Presburger-definable, and so is $R \fatsemi S$. Moreover, $R \fatsemi S$ is horizontal since R is horizontal. It follows from Theorem 7 that $(R \fatsemi S)^*$ is effectively Presburger-definable. We derive from the above characterization of $\xrightarrow{*}_{\mathcal{P}}$ that $\xrightarrow{*}_{\mathcal{P}}$ is also effectively Presburger-definable. ◀

▶ Remark. The notions of diagonal relations, horizontal relations and Presburger automata are extended to larger dimensions in the obvious way. A relation $R \subseteq \mathbb{N}^d \times \mathbb{N}^d$ is *diagonal* (resp. *horizontal*) if $(\mathbf{x}, \mathbf{y}) \in R$ implies $(\mathbf{x} + \mathbf{c}, \mathbf{y} + \mathbf{c}) \in R$ (resp. $(\mathbf{x} + \mathbf{c}, \mathbf{y}) \in R$) for every $\mathbf{c} \in \mathbb{N}^d$. But Theorem 9 does not extend to larger dimensions, even if we restrict ourselves to Presburger automata with a single state and a single transition. In fact, the reflexive-transitive closure of a Presburger-definable relation that is diagonal (resp. horizontal) need not be Presburger-definable. Consider the relation $R \subseteq \mathbb{N}^2 \times \mathbb{N}^2$ defined by $(x_1, x_2)\, R\, (y_1, y_2)$ if, and only if, the Presburger formula $y_1 \leq 2x_1 \wedge y_2 < x_2$ holds. The relation R is both diagonal and horizontal. It is routinely checked that the reflexive-transitive closure R^* is the set of pairs $((x_1, x_2), (y_1, y_2)) \in \mathbb{N}^2 \times \mathbb{N}^2$ such that $y_1 \leq 2^{x_2 - y_2} x_1$ and $y_2 \leq x_2$, which is not definable in Presburger arithmetic.

6 Reachability Relations of TRVASS

A TRVASS is a 2-dimensional vector addition system with states (2-dim VASS) such that the first counter can be tested for zero and the second one can be reset to zero. Formally, a *TRVASS* is a triple $\mathcal{V} = (Q, \Sigma, \Delta)$ where Q is a finite set of *states*, $\Sigma \subseteq \mathbb{Z}^2 \cup \{T, R\}$ is a finite set of *actions*, and $\Delta \subseteq Q \times \Sigma \times Q$ is a finite set of *transitions*. A *configuration* of $\mathcal{V}$ is a triple $(q, x_1, x_2) \in Q \times \mathbb{N} \times \mathbb{N}$ written as $q(x_1, x_2)$ in the sequel. The operational semantics of $\mathcal{V}$ is given by the binary relations $\xrightarrow{a}_{\mathcal{V}}$ over configurations, with $a \in \Sigma$, defined by $p(x_1, x_2) \xrightarrow{a}_{\mathcal{V}} q(y_1, y_2)$ if $(p, a, q) \in \Delta$ and

$$\begin{array}{lcll} (y_1, y_2) & = & (x_1 + a_1, x_2 + a_2) & \text{if } a = (a_1, a_2) \in \mathbb{Z}^2 \\ (y_1, y_2) & = & (0, x_2) \wedge x_1 = 0 & \text{if } a = T \\ (y_1, y_2) & = & (x_1, 0) & \text{if } a = R \end{array}$$

Given a word $w = a_1 \dots a_k$ of actions $a_j \in \Sigma$, we denote by $\xrightarrow{w}_{\mathcal{V}}$ the binary relation over configurations defined as the relational composition $\xrightarrow{a_1}_{\mathcal{V}} \fatsemi \cdots \fatsemi \xrightarrow{a_k}_{\mathcal{V}}$. The relation $\xrightarrow{\varepsilon}_{\mathcal{V}}$ denotes the identity relation on configurations. Given a subset $W \subseteq \Sigma^*$, we let $\xrightarrow{W}_{\mathcal{V}}$ denote the union $\bigcup_{w \in W} \xrightarrow{w}_{\mathcal{V}}$. The relation $\xrightarrow{\Sigma^*}_{\mathcal{V}}$, also written $\xrightarrow{*}_{\mathcal{V}}$, is called the *reachability relation* of $\mathcal{V}$. Observe that $\xrightarrow{*}_{\mathcal{V}}$ is the reflexive-transitive closure of the *step* relation $\rightarrow_{\mathcal{V}} \stackrel{\text{def}}{=} \bigcup_{a \in \Sigma} \xrightarrow{a}_{\mathcal{V}}$.

The remainder of this section is devoted to the proof that TRVASS have effectively Presburger-definable reachability relations. Let us fix a TRVASS $\mathcal{V} = (Q, \Sigma, \Delta)$. We let A denote the set $\Sigma \cap \mathbb{Z}^2$ of *addition vectors*.

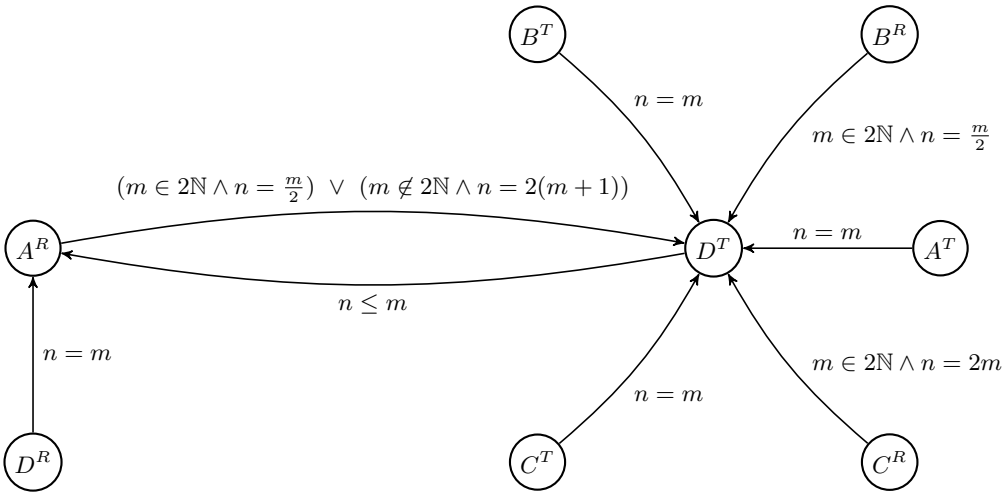


Figure 2 The Presburger automaton $\mathcal{P}$ associated to the TRVASS of Figure 1.

The reachability relation of $\mathcal{V}$ can be expressed in terms of the reachability relation of a Presburger automaton by observing that configurations reachable just after a zero-test T or a reset R are restricted to $q(0,n)$ or $q(n,0)$, respectively, where $q \in Q$ and $n \in \mathbb{N}$. Those configurations are parametrized by introducing the set $S = \{q^T, q^R \mid q \in Q\}$ obtained as two disjoint copies of Q. Elements in $\{q^T \mid q \in Q\}$ are called *test states*, and those in $\{q^R \mid q \in Q\}$ are called *reset states*. Given $s \in S$ and $n \in \mathbb{N}$, we introduce the configuration $[\![s,n]\!]$ in $Q \times \mathbb{N}^2$ defined as follows:

$$[\![s,n]\!] \stackrel{\text{def}}{=} \begin{cases} q(0,n) & \text{if } s = q^T \\ q(n,0) & \text{if } s = q^R \end{cases}$$

We also introduce, for each pair $(s,t) \in S \times S$, the binary relation $R_{s,t}$ defined by

$$R_{s,t} \stackrel{\text{def}}{=} \{(m,n) \in \mathbb{N} \times \mathbb{N} \mid [\![s,m]\!] \xrightarrow{A^*X}_{\mathcal{V}} [\![t,n]\!]\}$$

where $X = T$ if t is a test state and $X = R$ if t is a reset state. It is known that the reachability relation of a 2-dim VASS is effectively Presburger-definable [17, 2]. This entails that the relation $\xrightarrow{A^*}_{\mathcal{V}}$ is effectively Presburger-definable, and it follows that the relations $R_{s,t}$ are also effectively Presburger-definable. We introduce the Presburger automaton $\mathcal{P}$ with set of states S and set of transitions $\{(s, R_{s,t}, t) \mid (s,t) \in S \times S\}$. Note that $\mathcal{P}$ is computable from $\mathcal{V}$.

▶ **Example 10.** Let us come back to the TRVASS of Figure 1. The relations $R_{s,t}$ are all empty except for R_{D^T,A^R}, R_{D^R,A^R} and R_{s,D^T} with $s \in \{A^T, A^R, B^T, B^R, C^T, C^R\}$. The corresponding automaton $\mathcal{P}$ is depicted in Figure 2. Each transition $(s, R_{s,t}, t)$ is depicted by an edge from s to t labeled by a Presburger formula $\varphi_{s,t}(m,n)$ denoting the relation $R_{s,t}$. The empty relations (which are both diagonal and horizontal) are not depicted. Notice that the transition from A^R to D^T is ordinary and the one from D^T to A^R is horizontal. It follows that $\mathcal{P}$ is shallow. We observe that the horizontal relation R defined as the composition $R_{D^T,A^R} \fatsemi R_{A^R,D^T}$ is the one introduced in Example 5.

We first show that the Presburger automaton $\mathcal{P}$ is shallow. By Theorem 9, this will entail that its reachability relation $\xrightarrow{*}_{\mathcal{P}}$ is effectively Presburger-definable.

▶ **Lemma 11.** *The Presburger automaton $\mathcal{P}$ is shallow.*

Proof. It is readily seen that $\mathcal{P}$ satisfies the following properties:
- Transitions from reset states to reset states are diagonal,
- Transitions from test states to reset states are horizontal,
- Transitions from test states to test states are diagonal.

It follows that an ordinary transition of $\mathcal{P}$ is a transition from a reset state to a test state. If a cycle contains such a transition then it must contain a transition from a test state to a reset state as well. Since such a transition is horizontal, we obtain that $\mathcal{P}$ is shallow. ◀

The two following lemmas show how to decompose the reachability relation of $\mathcal{V}$ in terms of the reachability relation of $\mathcal{P}$.

▶ **Lemma 12.** *For every $s, t \in S$ and $m, n \in \mathbb{N}$, if $s(m) \xrightarrow{*}_{\mathcal{P}} t(n)$ then $[\![s, m]\!] \xrightarrow{*}_{\mathcal{V}} [\![t, n]\!]$.*

Proof. It is easily seen that $s(m) \to_{\mathcal{P}} t(n)$ implies $[\![s, m]\!] \xrightarrow{*}_{\mathcal{V}} [\![t, n]\!]$, for every $s, t \in S$ and $m, n \in \mathbb{N}$. We derive, by an immediate induction on $k \geq 1$, that $s(m)\ (\to_{\mathcal{P}})^k\ t(n)$ implies $[\![s, m]\!] \xrightarrow{*}_{\mathcal{V}} [\![t, n]\!]$, for every $s, t \in S$ and $m, n \in \mathbb{N}$. The lemma follows. ◀

▶ **Lemma 13.** *Consider two configurations $p(x_1, x_2)$ and $q(y_1, y_2)$ of $\mathcal{V}$. It holds that $p(x_1, x_2) \xrightarrow{\Sigma^* \setminus A^*}_{\mathcal{V}} q(y_1, y_2)$ if, and only if, there exist $s, t \in S$ and $m, n \in \mathbb{N}$ such that:*

$$p(x_1, x_2) \xrightarrow{A^*\{T,R\}}_{\mathcal{V}} [\![s, m]\!] \quad \wedge \quad s(m) \xrightarrow{*}_{\mathcal{P}} t(n) \quad \wedge \quad [\![t, n]\!] \xrightarrow{A^*}_{\mathcal{V}} q(y_1, y_2)$$

Proof. Lemma 12 shows the "if" direction of the equivalence. For the other direction, let $w \in \Sigma^* \setminus A^*$ such that $p(x_1, x_2) \xrightarrow{w}_{\mathcal{V}} q(y_1, y_2)$. By splitting w after each occurrence of an action in $\{T, R\}$, we deduce that $w = w_0 X_1 \dots w_{k-1} X_k w_k$ where $k \geq 1$, and $w_0, \dots, w_k \in A^*$. Let us introduce the configurations $c_1, \dots, c_k$ satisfying the following relations:

$$p(x_1, x_2) \xrightarrow{w_0 X_1}_{\mathcal{V}} c_1 \cdots \xrightarrow{w_{k-1} X_k}_{\mathcal{V}} c_k \xrightarrow{w_k}_{\mathcal{V}} q(y_1, y_2)$$

Notice that $c_j = [\![q_j^{X_j}, n_j]\!]$ for some $q_j \in Q$ and some $n_j \in \mathbb{N}$. By definition of $\mathcal{P}$, we get $q_{j-1}^{X_{j-1}}(n_{j-1}) \to_{\mathcal{P}} q_j^{X_j}(n_j)$ for every $j \in \{1, \dots, k\}$. We have proved the lemma. ◀

We deduce our main result.

▶ **Theorem 14.** *The reachability relation of a TRVASS is effectively Presburger-definable.*

Proof. Lemma 13 shows that $p(x_1, x_2) \xrightarrow{*}_{\mathcal{V}} q(y_1, y_2)$ if, and only if, $p(x_1, x_2) \xrightarrow{A^*}_{\mathcal{V}} q(y_1, y_2)$ or there exists $s, t \in S$ and $m, n \in \mathbb{N}$ such that:

$$p(x_1, x_2) \xrightarrow{A^*\{T,R\}}_{\mathcal{V}} [\![s, m]\!] \quad \wedge \quad s(m) \xrightarrow{*}_{\mathcal{P}} t(n) \quad \wedge \quad [\![t, n]\!] \xrightarrow{A^*}_{\mathcal{V}} q(y_1, y_2)$$

From [17, 2], the relation $\xrightarrow{A^*}_{\mathcal{V}}$ is effectively Presburger-definable. From Lemma 11 and Theorem 9, the relation $\xrightarrow{*}_{\mathcal{P}}$ is effectively Presburger-definable as well. ◀

Coming back to the classes of 2-dim extended VASS discussed in the introduction (see Table 1), Theorem 14 means that the reachability relation is effectively Presburger-definable for the "maximal" class $\mathbf{T}_1\mathbf{R}_2$. This result also applies to 2-dim VASS extended with resets and transfers on both counters (i.e., the class $\mathbf{R}_{1,2}\mathbf{Tr}_{1,2}$), since they can be simulated by machines in $\mathbf{T}_1\mathbf{R}_2$.

7 Conclusion and Open Problems

We have shown that the reachability relation of 2-dim VASS extended with tests on the first counter and resets on the second counter, is effectively Presburger-definable. This completes the decidability picture of 2-dim extended VASS initiated in [11]. Our proof techniques may also be used for other classes of counter machines where shallow Presburger automata would naturally appear. Many other problems on extensions of VASS are still interesting to solve.

- The reachability problem is NP-complete [12] for 1-dim VASS, PSPACE-complete [2] for 2-dim VASS, and NL-complete [7] for unary 2-dim VASS. But we do not know what are the complexities for the reachability problem, for the construction of the reachability set and for the reachability relation for all 2-dim extended VASS.
- The boundedness problem is undecidable for 3-dim VASS extended with resets on all counters [5] and it is decidable for arbitrary dimension VASS extended with resets on two counters [6]. Is boundedness decidable for arbitrary dimension TRVASS?

References

1 S. Akshay, Supratik Chakraborty, Ankush Das, Vishal Jagannath, and Sai Sandeep. On Petri Nets with Hierarchical Special Arcs. In *CONCUR*, volume 85 of *LIPIcs*, pages 40:1–40:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

2 Michael Blondin, Alain Finkel, Stefan Göller, Christoph Haase, and Pierre McKenzie. Reachability in Two-Dimensional Vector Addition Systems with States Is PSPACE-Complete. In *LICS*, pages 32–43. IEEE Computer Society, 2015.

3 Rémi Bonnet. The reachability problem for Vector Addition Systems with one zero-test. In Filip Murlak and Piotr Sankowski, editors, *Proceedings of the 36th International Symposium on Mathematical Foundations of Computer Science (MFCS'11)*, volume 6907 of *Lecture Notes in Computer Science*, pages 145–157, Warsaw, Poland, August 2011. Springer.

4 Rémi Bonnet, Alain Finkel, Jérôme Leroux, and Marc Zeitoun. Place-Boundedness for Vector Addition Systems with one zero-test. In Kamal Lodaya and Meena Mahajan, editors, *Proceedings of the 30th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'10)*, volume 8 of *Leibniz International Proceedings in Informatics*, pages 192–203, Chennai, India, December 2010. Leibniz-Zentrum für Informatik.

5 Catherine Dufourd, Alain Finkel, and Philippe Schnoebelen. Reset Nets Between Decidability and Undecidability. In *ICALP*, volume 1443 of *Lecture Notes in Computer Science*, pages 103–115. Springer, 1998.

6 Catherine Dufourd, Petr Jancar, and Philippe Schnoebelen. Boundedness of Reset P/T Nets. In *ICALP*, volume 1644 of *Lecture Notes in Computer Science*, pages 301–310. Springer, 1999.

7 Matthias Englert, Ranko Lazic, and Patrick Totzke. Reachability in Two-Dimensional Unary Vector Addition Systems with States is NL-Complete. In *LICS*, pages 477–484. ACM, 2016.

8 Alain Finkel and Arnaud Sangnier. Mixing coverability and reachability to analyze VASS with one zero-test. In David Peleg and Anca Muscholl, editors, *Proceedings of the 36th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'10)*, volume 5901 of *Lecture Notes in Computer Science*, pages 394–406, Špindlerův Mlýn, Czech Republic, January 2010. Springer.

9 Alain Finkel and Philippe Schnoebelen. Well-Structured Transition Systems Everywhere! *Theoretical Computer Science*, 256(1-2):63–92, April 2001.

10 Alain Finkel and Grégoire Sutre. An Algorithm Constructing the Semilinear Post* for 2-Dim Reset/Transfer VASS. In *MFCS*, volume 1893 of *Lecture Notes in Computer Science*, pages 353–362. Springer, 2000.

11 Alain Finkel and Grégoire Sutre. Decidability of Reachability Problems for Classes of Two Counters Automata. In *STACS*, volume 1770 of *Lecture Notes in Computer Science*, pages 346–357. Springer, 2000.

12 Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. Reachability in Succinct and Parametric One-Counter Automata. In *CONCUR*, volume 5710 of *Lecture Notes in Computer Science*, pages 369–383. Springer, 2009.

13 John Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8(2):135–159, 1979.

14 Ranko Lazic. The reachability problem for vector addition systems with a stack is not elementary. *CoRR*, abs/1310.1767, 2013.

15 Ranko Lazic and Patrick Totzke. What Makes Petri Nets Harder to Verify: Stack or Data? In *Concurrency, Security, and Puzzles*, volume 10160 of *Lecture Notes in Computer Science*, pages 144–161. Springer, 2017.

16 Jérôme Leroux, M. Praveen, and Grégoire Sutre. Hyper-Ackermannian bounds for pushdown vector addition systems. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, page 63. ACM, 2014.

17 Jérôme Leroux and Grégoire Sutre. On Flatness for 2-Dimensional Vector Addition Systems with States. In *CONCUR 2004 - Concurrency Theory, 15th International Conference, London, UK, August 31 - September 3, 2004, Proceedings*, volume 3170 of *Lecture Notes in Computer Science*, pages 402–416. Springer, 2004.

18 M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. *Comptes Rendus du premier congrès de mathématiciens des Pays Slaves, Warszawa*, pages 92–101, 1929.

19 Klaus Reinhardt. Reachability in Petri Nets with Inhibitor Arcs. *Electr. Notes Theor. Comput. Sci.*, 223:239–264, 2008.

The Parikh Property for Weighted Context-Free Grammars

Pierre Ganty[1]
IMDEA Software Institute, Madrid, Spain
pierre.ganty@imdea.org
https://orcid.org/0000-0002-3625-6003

Elena Gutiérrez[2]
IMDEA Software Institute, Madrid, Spain
Universidad Politécnica de Madrid, Spain
elena.gutierrez@imdea.org

Abstract

Parikh's Theorem states that every context-free grammar (CFG) is equivalent to some regular CFG when the ordering of symbols in the words is ignored. The same is not true for the so-called weighted CFGs, which additionally assign a weight to each grammar rule. If the result holds for a given weighted CFG G, we say that G satisfies the Parikh property. We prove constructively that the Parikh property holds for every weighted nonexpansive CFG. We also give a decision procedure for the property when the weights are over the rationals.

2012 ACM Subject Classification Theory of computation → Grammars and context-free languages

Keywords and phrases Weighted Context-Free Grammars, Algebraic Language Theory, Parikh Image

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.32

Related Version A full version of this paper is available at [7], `https://arxiv.org/abs/1810.01351`.

Acknowledgements We thank Miguel Ambrona for pointing us to the theory of Groebner bases.

1 Introduction

The celebrated Parikh's Theorem [17] establishes that every context-free language is *Parikh-equivalent* to some regular language. Two words w, w' over an alphabet of symbols are Parikh-equivalent if the number of occurrences of each symbol in w coincides with that of w'. For instance, the words $aabb$ and $abab$ over the alphabet $\{a, b\}$ are Parikh-equivalent as both have 2 a's and 2 b's. Two languages L and L' are Parikh-equivalent if for each word in L there is a Parikh-equivalent word in L', and viceversa, e.g., the language $\{ab, aabb\}$ is Parikh-equivalent to the language $\{ba, abab, abba\}$. Consider, for instance, the context-free language $L = \{a^n b^n \mid n \geq 0\}$. Then, a regular language that satisfies Parikh's Theorem is $(ab)^*$. In fact, given a context-free grammar, one can construct a finite-state automaton that

[1] Supported by the Spanish Ministry of Economy and Competitiveness project No. TIN2015-71819-P, RISCO - RIgorous analysis of Sophisticated COncurrent and distributed systems, and by a Ramón y Cajal fellowship RYC-2016-20281.
[2] Supported by BES-2016-077136 grant from the Spanish Ministry of Economy, Industry and Competitiveness, and by RISCO - RIgorous analysis of Sophisticated COncurrent and distributed systems.

recognizes a Parikh-equivalent language [5]. Parikh's Theorem has been applied in automata theory for decision problems concerning Parikh-equivalence such as membership, universality, equivalence and disjointness [4, 11, 12, 13], to establish complexity bounds on verification problems for counter machines [9], equational Horn clauses [21], among many others. It has also found application in the analysis of asynchronous programs with procedures [8, 20] where the Parikh-equivalent finite-state automaton is used to compute another asynchronous program without procedures that preserves safety bugs.

Weighted finite-state automata are a generalization of the classical nondeterministic finite-state automata in which each transition carries a weight. This weight can be defined, for instance, as a nonnegative number representing the cost of its execution. Then, the weight of a path in the weighted automaton can be computed by adding the weights of its transitions. If we are interested in the minimal cost of execution of a given word, we can compute its weight as the minimum of the weights of the paths accepting that word. In general, the algebraic structure underlying the computation of the weights is that of a *semiring*, an algebraic structure with two operations $\cdot$ (product) and $+$ (sum) used to compute the weight of a path and the weight of a word, respectively. In the same way, it is possible to add weights to the transitions of a pushdown automaton. The later model, so-called weighted pushdown automata, has been used to perform data-flow analysis of programs with procedures [19].

In this paper we study the question of whether Parikh's Theorem can be extended to the weighted case. Roughly speaking, for a given weighted pushdown automaton $\mathcal{P}$, we ask whether there is a weighted finite-state automaton $\mathcal{F}$ that accepts a Parikh-equivalent language and such that for every word w, the sum of the weights of all words Parikh-equivalent to w in $\mathcal{P}$ coincides with that of all Parikh-equivalent words to w in $\mathcal{F}$. Extending Parikh's Theorem to the weighted case has the potential of reaching new applications, for instance, the analysis of event-driven asynchronous programs with procedures where each transition is augmented with the probability of the event associated to it. Finding a weighted finite-state automaton that is Parikh-equivalent to the original program and preserves the probabilities can be used to perform probabilistic analysis of programs following this paradigm.

We will present our results using the grammar model (as opposed to the automata model). It is well-known that both models are equivalent, in the sense that both representations generate the same family of languages of weighted words. Using weighted context-free grammars (WCFGs for short) allows us to exploit their connection with algebraic systems of equations to give more simple and convincing proofs of our results. In a WCFG, a weight is assigned to each rule of the grammar. The notion of weight is extended from rules to parse trees by multiplying the weights of the rules used along a tree, and from parse trees to words by adding the weights of all the possible parse trees that yield to a word. We say that two WCFGs G_1 and G_2 are Parikh-equivalent if for each Parikh-equivalence class $\mathcal{E}$, the sum of the weights of every word in $\mathcal{E}$ under G_1 and G_2 coincide.

We consider the following problem: given a WCFG G, does there exist a Parikh-equivalent WCFG G' that is regular? If the answer is positive we say that G satisfies the Parikh property. It follows from a known counterexample by Petre [18] that the property is not true in general. Recently, Bhattiprolu et al. [2] further investigated this question. They show a class of WCFGs over the unary alphabet that always satisfy the Parikh property. Now, we show that every *nonexpansive* WCFG (over an arbitrary alphabet and arbitrary semiring) satisfies the Parikh property. A WCFG is nonexpansive if no grammar derivation is of the form $X \Rightarrow^* w_0\, X\, w_1\, X\, w_2$. Note that nonexpansiveness is decidable as it reduces to computing predecessors of a regular set [6]. We can show that in the unary case the class of nonexpansive grammars strictly contains the class defined by Bhattiprolu et al. [2] (see Appendix D in the

extended version of the paper [7]). However, nonexpansiveness is a sufficient condition for the Parikh property, but not necessary. In particular, we give an example of an expansive WCFG for which there exists a Parikh-equivalent regular WCFG. This shows that a conjecture formulated by Baron and Kuich [1] in 1981 is false[3]. Furthermore, we can show that nonexpansiveness is not necessary for the property even when the alphabet is unary by means of a similar example.

In the second part of our work, we study the question of whether the Parikh property is decidable. As far as we can tell, this question is open. However, it implicitly follows from a result by Kuich et al. [15] that, when we equivalently formulate the property in terms of formal power series, it is decidable over the semiring of rational numbers. Their proof relies on an ad-hoc elimination procedure which is hard to perform even on small examples. Now we give a decision procedure that sidesteps this problem by using a different technique that allows to illustrate the algorithm on small examples with the support of mainstream open-source computer algebra systems.

The document is organized as follows. After preliminaries in Section 2, we show in Section 3 that every nonexpansive WCFG is Parikh-equivalent to a regular WCFG. In Section 4, we give a decision procedure for the property when the weight domain is over the rational numbers and we illustrate its use with several examples. Finally, we give further details of the related work in Section 5, and conclusions and further work in Section 6. Missing proofs can be found in the Appendix. For space reasons, some proofs are deferred to a full version of this paper [7].

2 Preliminaries

We denote by Σ^* ($\Sigma^\oplus$) the free (commutative) monoid generated by Σ. The elements of Σ^* are written as words over the alphabet Σ, typically denoted by w, w' and w_i ($i \in \mathbb{N}$), while the elements of $\Sigma^\oplus$ are written as monomials in the variables Σ and they are typically denoted by v, v' and v_i. For instance, if $\Sigma = \{a, b\}$ then all the elements in Σ^* of length two containing 1 a and 1 b are the words ab and ba while the only element with that property in $\Sigma^\oplus$ is the monomial ab.

We denote a *context-free grammar* (CFG for short) as a tuple (V, Σ, S, R) where V is a finite set of *variables* including S, the *start* variable, Σ is the set of *terminals* and $R \subseteq V \times (\Sigma \cup V)^*$ is a finite set of *rules*. Rules are conveniently denoted $X \to \gamma$. We will always assume that CFGs are *cycle-free*, i.e., there is no derivation of the form $X \Rightarrow^+ X$ with $X \in V$. This guarantees that the number of parse trees for one given word is finite and thus the weight of a word is a well-defined function. W.l.o.g., we assume that every *regular* CFG is *right-regular* , i.e., $\gamma \in \Sigma^+(\varepsilon \cup V)$ for each γ. A CFG is *nonexpansive* if no derivation is of the form $X \Rightarrow^* w_0\, X\, w_1\, X\, w_2$ with $X \in V$ and $w_i \in (\Sigma \cup V)^*$. Otherwise, it is *expansive*.

A *semiring* is a structure $(A, +, \cdot, 0_A, 1_A)$ where $(A, +, 0_A)$ is a commutative monoid with identity 0_A, $(A, \cdot, 1_A)$ is a monoid with identity 1_A, $\cdot$ distributes over $+$ and 0_A satisfies that $a \cdot 0_A = 0_A \cdot a = 0_A$, for all $a \in A$. A semiring is called *commutative* iff $a \cdot b = b \cdot a$ for every $a, b \in A$. In the sequel, we assume that A is always a commutative semiring. An *idempotent* semiring is one that satisfies $a + a = a$, for all $a \in A$, . A (commutative) *ring* is a (commutative) semiring where $(A, +, 0_A)$ is a commutative *group* (i.e., every element in A has an additive inverse). Finally, a *field* is a ring where $(A \setminus \{0_A\}, \cdot, 1_A)$ is a *commutative*

[3] Essentially, they conjectured that every unambiguous WCFG G is nonexpansive iff G has the Parikh property [1, Conjecture C].

group (i.e., every element in A except 0_A has a multiplicative inverse). We sometimes use A for both the structure and the underlying set when the meaning is clear from the context. We abuse notation and use $+$ and $\cdot$ to denote the ordinary sum and product in $\mathbb{N}$ and $\mathbb{Q}$. Classical examples of a commutative semirings are $(\mathbb{N}, +, \cdot, 0, 1)$ and $(\mathbb{Q}, +, \cdot, 0, 1)$. The later is also a *field* and we will refer to it as the *rational semiring*. Another classical example of a commutative semiring is the *tropical semiring*, defined as $(\mathbb{N} \cup \{\infty\}, min, +, \infty, 0)$. Note that this semiring is also idempotent as $min(a, a) = a$, for all $a \in \mathbb{N} \cup \{\infty\}$.

A *weighted context-free grammar* (WCFG for short) is a pair (G, W) where G is a CFG as defined above and W is a mapping with the signature $W : R \to A$ that assigns a weight from A to each production in R, for some (commutative) semiring A. Note that W may assign 0_A to some rules in R. The mapping W is usually referred to as the *weight function* of the WCFG. We extend the definition of W from rules to *derivation sequences*[4] by assigning to each derivation sequence ψ a weight value which is the product of the weights of the rules applied in ψ. We assume that, the derivation policy for G, i.e., the derivation strategy that determines the next variable to rewrite along a derivation, defines one unique derivation sequence for each parse tree. We also assume that the $\cdot$ operation is commutative, i.e., we will always consider *commutative semirings*. Then, the weight of a derivation sequence does not depend on the choice of the derivation policy. Under these assumptions we can extend the definition from rules to *parse trees* (instead of derivation sequences). Before, we recall some definitions. We define a *labeled tree* $c(\tau_1, \ldots, \tau_n)$ (with $n \geq 0$) as a finite tree whose nodes are labeled, where c is the label of the root and $\tau_1, \ldots, \tau_n$ are labeled trees, the children of the roots. When $n = 0$ we prefer to write c instead of $c()$. We simply write $\tau = c(\ldots)$ when the children nodes $\tau_1, \ldots, \tau_n$ are not important. We will write parse trees as labeled trees of the form $\tau = \pi(\tau_1, \ldots, \tau_n)$ to denote that the topmost level of τ is induced by the grammar rule π and has exactly n children nodes which root (from left to right) the parse trees $\tau_1, \ldots, \tau_n$, i.e., the right-hand side of π contains n grammar variables where the i-th (from the left) is derived according to τ_i. We thus define the *yield* of a parse tree $\tau = \pi(\tau_1, \ldots, \tau_n)$, denoted as $\mathcal{Y}(\tau)$ inductively as follows. If $n = 0$, then $\mathcal{Y}(\tau) = \gamma$ where π is of the form $X \to \gamma$ and $\gamma \in \Sigma^* \cup \{\varepsilon\}$. Otherwise, $\mathcal{Y}(\tau) = \alpha_1 \mathcal{Y}(\tau_1) \ldots \alpha_n \mathcal{Y}(\tau_n) \alpha_{n+1}$ where π is of the form $X \to \alpha_1 X_1 \ldots \alpha_n X_n \alpha_{n+1}$ with $\alpha_i \in \Sigma^* \cup \{\varepsilon\}$, and each X_i corresponds to the left-hand side of the rule in the root of τ_i. Define the *weight of a parse tree* $\tau = \pi(\tau_1, \ldots, \tau_n)$ inductively as:

$$W(\tau) \stackrel{\text{def}}{=} W(\pi) \prod_{i=1}^{n} W(\tau_i) \ .$$

Note that $W(\tau)$ does not depend on the order in which we consider the rules in τ as we assume that $\cdot$ is commutative. Denote by $\mathcal{T}_G$ the set of all parse trees of a CFG G. Then, define the *weight of a word* $w \in \Sigma^*$ as follows:

$$W(w) \stackrel{\text{def}}{=} \sum_{\substack{\mathcal{Y}(\tau) = w \\ \tau \in \mathcal{T}_G}} W(\tau) \ .$$

If for some $w \in \Sigma^*$, the set $\{\tau \mid \mathcal{Y}(\tau) = w, \tau \in \mathcal{T}_G\} = \emptyset$ then $W(w) \stackrel{\text{def}}{=} 0_A$. Define the *semantics* of a WCFG (G, W), denoted by $[\![G]\!]_W$, as the mapping $[\![G]\!]_W : \Sigma^* \to A$ such that $[\![G]\!]_W(w) \stackrel{\text{def}}{=} W(w)$. Define the *Parikh image* of a word $w \in \Sigma^*$ with $\Sigma =$

[4] For a definition of *derivation sequence* go to the beginning of Appendix A.

$\{a_1, \ldots, a_n\}$, denoted by $\wr w \int$ as the monomial $a_1^{\alpha_1} a_2^{\alpha_2} \ldots a_n^{\alpha_n} \in \Sigma^{\oplus}$ such that α_i is the number of occurrences of a_i in w. Define the *Parikh image* of a weighted context-free grammar (G, W), denoted by $Pk[\![G]\!]_W$, as the mapping $Pk[\![G]\!]_W : \Sigma^{\oplus} \to A$ such that:

$$Pk[\![G]\!]_W(v) \stackrel{\text{def}}{=} \sum_{\substack{v = \wr w \int \\ w \in \Sigma^*}} [\![G]\!]_W(w) \ .$$

We write $[\![G]\!]_W$ and $Pk[\![G]\!]_W$ as the formal sums $\sum_{w \in \Sigma^*} [\![G]\!]_W(w)\, w$ and $\sum_{v \in \Sigma^{\oplus}} Pk[\![G]\!]_W(v)\, v$, respectively. Two WCFGs (G, W) and (G', W') are *language-equivalent* iff $[\![G]\!]_W = [\![G']\!]_{W'}$, while (G, W) and (G', W') are *Parikh-equivalent* iff $Pk[\![G]\!]_W = Pk[\![G']\!]_{W'}$. Finally, a WCFG (G, W) is *regular/nonexpansive/cycle-free* iff G is regular/nonexpansive/cycle-free, respectively.

▶ **Definition 1** (Parikh property)**.** A WCFG (G, W) satisfies the *Parikh property* iff there exists a WCFG (G_ℓ, W_ℓ) such that:

1. (G_ℓ, W_ℓ) is regular, and
2. $Pk[\![G]\!]_W = Pk[\![G_\ell]\!]_{W_\ell}$.

3 Sufficient condition for the Parikh property

Petre [18] shows that the Parikh property is not true in general. In the following example we show a well-known WCFG (for instance, see [2, 18]) for which no regular Parikh-equivalent WCFG exists.

▶ **Example 2.** Consider the WCFG (G, W) with $G = (\{X\}, \{a\}, X, \{X \to aXX,\ X \to a\})$ and the weight function W over $(\mathbb{N}, +, \cdot, 0, 1)$ that assigns 1 to each production in the grammar. Note that, because the alphabet is unary, we have that $Pk[\![G]\!]_W = [\![G]\!]_W$. As W assigns 1 to each grammar rule, the weight of each word can be interpreted as its ambiguity according to G. Then, the reader can check that:

$$[\![G]\!]_W = \sum_{n \geq 0} C_n\, a^{2n+1} = 1a + 1a^3 + 2a^5 + 5a^7 + 14a^9 + 42a^{11} + 132a^{13} + 429a^{15} + \ldots$$

with $C_n = \frac{1}{n+1}\binom{2n}{n}$ the n-th Catalan number. We will see in Example 10 that this formal power series cannot be generated by a regular WCFG.

Now we show that every nonexpansive WCFG over an arbitrary commutative weight domain satisfies the Parikh property.

▶ **Theorem 3.** *Let (G, W) be an arbitrary WCFG. If G is nonexpansive then (G, W) satisfies the Parikh property.*

Proof. The proof is constructive. Here we give the main intuition of the construction. For a complete proof go to Appendix A. For every nonexpansive WCFG (G, W), we give a 2-step construction that results in a Parikh-equivalent regular WCFG (G_ℓ, W_ℓ). The steps are:

1. construct a new WCFG $\left(G^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$, with $k \in \mathbb{N}$, language-equivalent to (G, W); and
2. construct a regular WCFG (G_ℓ, W_ℓ) Parikh-equivalent to $\left(G^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$.

The general idea behind the first step is to build a WCFG $\left(G^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$ that contains all the information needed to define a "strategic" derivation policy. This derivation policy is strategic in the sense that the total number of grammar variables in all *derivation sentences*[5] produced along a derivation sequence is bounded.

[5] For a definition of *derivation sentence* go to the beginning of Appendix A.

In the second step of the construction, we use $\left(G^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$ to build a regular WCFG (G_ℓ, W_ℓ) that is Parikh-equivalent. Each grammar variable of (G_ℓ, W_ℓ) represents each possible sentence (without the terminals) along a derivation sequence of $\left(G^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$, and each rule simulates a derivation step of $\left(G^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$. Because the number of variables in the sentences is bounded, the number of variables and rules of (G_ℓ, W_ℓ) is necessarily finite. ◀

The converse of Theorem 3 is not true. The next counterexample illustrates this fact by defining an expansive WCFG G_2 for which a Parikh-equivalent regular WCFG G_1 exists. Thus, nonexpansiveness does not provide an exact characterization of the Parikh property.

▶ **Example 4.** Consider the WCFG (G_1, W_1) where $G_1 = (\{X_1\}, \{a, \overline{a}\}, X_1, R_1 = \{X_1 \to aX_1,\ X_1 \to \overline{a}X_1,\ X_1 \to \varepsilon\})$ and W_1 is defined over $(\mathbb{N}, +, \cdot, 0, 1)$ and assigns 1 to each rule in R_1. First, note that (G_1, W_1) is regular and the weight of each word can be interpreted as its ambiguity according to G_1. Because G_1 is unambiguous, the weight of each word in the language of G_1 is 1. It is easy to see that $[\![G_1]\!]_{W_1}$ is:

$$[\![G_1]\!]_{W_1} = (a + \overline{a})^* = \sum_{n \geq 0} (a + \overline{a})^n = 1\varepsilon + 1a + 1\overline{a} + 1a\overline{a} + 1\overline{a}a + 1aaa + 1aa\overline{a} + 1a\overline{a}a + 1a\overline{a}\overline{a} + \dots$$

Now consider the expansive WCFG (G_D, W_D) where $G_D = (\{D\}, \{a, \overline{a}\}, D, R_D = \{D \to aD\overline{a}D,\ D \to \varepsilon\})$ and W_D is defined over $\mathbb{N}$ and assigns 1 to each rule in R_D. The grammar G_D generates the Dyck language L_D over the alphabet $\{a, \overline{a}\}$ and it is also unambiguous (i.e., the weight of each $w \in L_D$ is 1). It is well-known that L_D is a deterministic context-free language (DCFL). Then the complement of L_D, namely $\{a, \overline{a}\}^* \setminus L_D$, is also a DCFL and thus admits an unambiguous CFG. Let $G_{\overline{D}} = (V_{\overline{D}}, \{a, \overline{a}\}, \overline{D}, R_{\overline{D}})$ be the unambiguous CFG that generates $\{a, \overline{a}\}^* \setminus L_D$, and define $(G_{\overline{D}}, W_{\overline{D}})$ where $W_{\overline{D}}$ is defined over $\mathbb{N}$ and assigns 1 to each rule in $R_{\overline{D}}$.

W.l.o.g., assume $V_D \cap V_{\overline{D}} = \emptyset$ and consider a new variable $X_2 \notin V_D \cup V_{\overline{D}}$. Define the WCFG (G_2, W_2) where $G_2 = (\{X_2\} \cup V_D \cup V_{\overline{D}}, \{a, \overline{a}\}, X_2, R_2)$, R_2 is defined as $R_2 = \{X_2 \to D,\ X_2 \to \overline{D}\} \cup R_D \cup R_{\overline{D}}$ where W_2 is defined over $\mathbb{N}$ and assigns 1 to each rule in R_2. First, G_2 is expansive because G_D is expansive. Furthermore, D and $\overline{D}$ generate unambiguously languages that are complementary over $\{a, \overline{a}\}$. As the weight of each word in (G_2, W_2) corresponds to its ambiguity, we have that $[\![G_2]\!]_{W_2} = (a + \overline{a})^*$. Hence $[\![G_1]\!]_{W_1} = [\![G_2]\!]_{W_2}$ and thus $Pk[\![G_1]\!]_{W_1} = Pk[\![G_2]\!]_{W_2}$. Recall that (G_1, W_1) is regular. We conclude that (G_2, W_2) is expansive and satisfies the Parikh property.

We can give a similar counterexample over a unary alphabet (see Appendix B). This shows that nonexpansiveness is not necessary for the Parikh property even in the unary case.

4 A decision procedure for the Parikh property over the rationals

In this section we give a decision procedure that tells whether or not a given WCFG with weights over the rational semiring satisfies the Parikh property. Our procedure relies on a decidability result by Kuich and Salomaa [15, Theorem 16.13]. It implicitly follows from this result that the Parikh property is decidable over the rational semiring. However, their decision procedure is hard to follow as it relies on algebraic methods beyond the scope of this field. This makes its implementation rather involved even for small instances. We propose an alternative method to sidestep this problem using Groebner basis theory.

First, we give some preliminaries. In what follows, A will denote a partially ordered commutative semiring. Given A and an alphabet Σ, a *formal power series in commuting variables* is a mapping of $\Sigma^\oplus$ into A. $A\langle\langle \Sigma^\oplus \rangle\rangle$ denotes the set of all formal power series

in commuting variables Σ and coefficients in A. The values of a formal power series r are denoted by (r, v) where $v \in \Sigma^{\oplus}$. As r is a mapping of $\Sigma^{\oplus}$ into A, it can be written as a formal sum as $r = \sum_{v \in \Sigma^{\oplus}} (r, v)\, v$. When $v = \varepsilon$ we will write the term $(r, \varepsilon)\varepsilon$ of r simply as (r, ε). We define the *support* of a formal power series as $supp(r) \stackrel{\text{def}}{=} \{v \mid (r, v) \neq 0_A\}$. The subset of $A\langle\langle \Sigma^{\oplus} \rangle\rangle$ consisting of all series with a finite support is denoted by $A\langle \Sigma^{\oplus} \rangle$ and its elements are called *polynomials*. Finally, define, for $k \geq 0$, the operator R_k by $R_k(r) \stackrel{\text{def}}{=} \sum_{|v| \leq k} (r, v)v$ where $r \in A\langle\langle \Sigma^{\oplus} \rangle\rangle$.

Now we establish the connection between WCFGs and algebraic systems in commuting variables. Let (G, W) be a WCFG with $G = (V, \Sigma, X_1, R)$, $V = \{X_1, \ldots, X_n\}$, and W defined over the semiring A. We associate to (G, W) the algebraic system in commuting variables defined as follows. For each $X_i \in V$:

$$X_i = \sum_{\substack{\pi \in R \\ \pi = (X_i \to \gamma)}} W(\pi) \wr \gamma \int \quad . \tag{1}$$

We refer to this system as the *algebraic system (in commuting variables) corresponding to* (G, W). Sometimes, we write $A\langle \Sigma^{\oplus} \rangle$-algebraic system to indicate that the coefficients of the system lie in $A\langle \Sigma^{\oplus} \rangle$. Note that (1) can be written as follows. For each $X_i \in V$:

$$X_i = p_i \ , \text{ with } p_i \in A\langle (\Sigma \cup V)^{\oplus} \rangle \ . \tag{2}$$

A *solution* to (2) is defined as an n-tuple $r = (r_1, \ldots, r_n)$ of elements of $A\langle\langle \Sigma^{\oplus} \rangle\rangle$ such that $r_i = r(p_i)$, for $i = 1, \ldots, n$, where $r(p_i)$ denotes the series obtained from p_i by replacing, for $j = 1, \ldots, n$, simultaneously each occurrence of X_j by r_j. Note that, r_1, the first component of r, always corresponds to the solution for X_1, the initial variable of G. The *approximation sequence* $\sigma^0, \sigma^1, \ldots, \sigma^j, \ldots$ where each σ^j is an n-tuple of elements of $A\langle \Sigma^{\oplus} \rangle$ associated to an algebraic system as (2) is defined as $\sigma^0 = (0_A, \ldots, 0_A)$ and $\sigma^{j+1} = (\sigma^j(p_1), \ldots, \sigma^j(p_n))$ for all $j \geq 0$. We have that $\lim_{j \to \infty} = \sigma^j$ iff for all $k \geq 0$ there exists an $m(k)$ such that $R_k(\sigma^{m(k)+j}) = R_k(\sigma^{m(k)}) = R_k(\sigma)$ for all $j \geq 0$. If $\lim_{j \to \infty} \sigma^j = \sigma$, then σ is a solution of (2) (from Theorem 14.1 in [15]) and is referred to as the *strong solution*. Note that, by definition, the strong solution is unique whenever it exits. Finally, if (G, W) is a regular WCFG then each p_i in its corresponding algebraic system written as in (2) is a polynomial in $A\langle \mathcal{M} \rangle$, where $\mathcal{M}$ denotes the set of monomials of the form $a_1^{\alpha_1} \ldots a_m^{\alpha_m} X_1^{\beta_1} \ldots X_n^{\beta_n}$ with $a_i \in \Sigma$, $\alpha_i, \beta_j \in \mathbb{N}$ for all i and j, and $\sum_{i=1}^{n} \beta_i \leq 1$. We call a system of this form a *regular algebraic system*. Conversely, we associate to each $A\langle \Sigma^{\oplus} \rangle$-algebraic system S in commuting variables of the form (2) a WCFG (G, W) over the semiring A as follows. Define $G = (\{X_1, \ldots, X_n\}, \Sigma, X_1, R)$ and such that $\pi = (X_i \to \gamma) \in R$ iff $(p_i, \gamma) \neq 0_A$. If $\pi \in R$ then $W(\pi) = (p_i, \gamma)$. We will refer to (G, W) as the *WCFG corresponding to the algebraic system* S. Note that if we begin with an algebraic system in commuting variables, then go to the corresponding WCFG and back again to an algebraic system, then the latter coincides with the original. However, if we begin with the WCFG, form the corresponding algebraic system and then again the corresponding WCFG, then the latter grammar may differ from the original.

Next theorem shows that the Parikh image of a cycle-free WCFG corresponds to the solution for the initial variable in the corresponding algebraic system.

▶ **Theorem 5.** *Let (G, W) be a cycle-free WCFG and let S be the algebraic system in commuting variables corresponding to (G, W). Then, the strong solution r of S exists and the first component of r corresponds to $Pk[\![G]\!]_W$.*

Proof. See Appendix C in the extended version [7]. ◀

Now we introduce the class of *rational* power series in commuting variables Σ with coefficients in the semiring A, denoted by $A^{rat}\langle\langle\Sigma^{\oplus}\rangle\rangle$.

▶ **Definition 6.** $r \in A^{rat}\langle\langle\Sigma^{\oplus}\rangle\rangle$ iff r is the first component of the solution of a regular algebraic system in commuting variables.

From the previous definition and Theorem 5 we can characterize the WCFGs that satisfy the Parikh property as follows.

▶ **Lemma 7.** *Let (G, W) be a cycle-free WCFG. Then (G, W) satisfies the Parikh property iff $Pk[\![G]\!]_W \in A^{rat}\langle\langle\Sigma^{\oplus}\rangle\rangle$.*

Proof. See Appendix C in the extended version [7]. ◀

Next we observe that every WCFG (G, W) defined over a commutative *ring* with the Parikh property satisfies a linear equation of a special kind. This result directly follows from Theorem 16.4 in [15].

▶ **Theorem 8.** *Let (G, W) be a cycle-free WCFG with W defined over a commutative ring A. Then (G, W) satisfies the Parikh property iff $Pk[\![G]\!]_W$ satisfies a linear equation of the form: $X = sX + t$, for some $s, t \in A\langle\Sigma^{\oplus}\rangle$ with $(s, \varepsilon) = 0$.*

Proof. The result is a consequence of Theorem 16.4 in [15] and Lemma 7. ◀

We conclude from the previous theorem that, given a WCFG (G, W) with W defined over a commutative ring, if such a linear equation exists then (G, W) satisfies the Parikh property; otherwise it does not. Now we will use a result by Kuich et al. [15] to conclude that, if (G, W) is defined over $\mathbb{Q}$ then there exists an irreducible polynomial $q(X)$ such that q evaluates to 0 when $X = Pk[\![G]\!]_W$, denoted by $q(Pk[\![G]\!]_W) \equiv 0$. Intuitively, this polynomial contains all the information needed to decide whether or not (G, W) has the Parikh property.

▶ **Theorem 9** (from Theorem 16.9 in [15])**.** *Let S be the $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$-algebraic system in commuting variables corresponding to a cycle-free WCFG. Let r_1 be the first component of its strong solution. Then there exists an irreducible polynomial $q(X_1)$ with coefficients in $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$, and unique up to a factor in $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$, such that $q(r_1) \equiv 0$.*

Kuich et al. [15] show that the polynomial q is effectively computable by means of a procedure based on the classical elimination theory. Now we develop an alternative method using Groebner bases. Before introducing this technique, we give some intuition on the ideas presented above by revisiting the examples of the previous section.

▶ **Example 10.** Consider the cycle-free WCFG (G, W) defined in Example 2 where the weight function W is now defined over $(\mathbb{Q}, +, \cdot, 0, 1)$ and assigns 1 to each production in the grammar. The algebraic system S corresponding to (G, W) is given by the equation $X = a\,X^2 + a$. Let r_1 be its strong solution. Assume for now that the irreducible polynomial $q(X) \in \mathbb{Q}\langle\{a\}^{\oplus}\rangle\langle X\rangle$ from Theorem 9 is $q(X) = aX^2 - X + a$ (later we will give its construction using Groebner bases). We will see later that the fact that $q(X)$ is not linear is enough to conclude that (G, W) does not satisfy the Parikh property (as we expected). Note that the solution of S is $r_1 = \frac{1-\sqrt{1-4a^2}}{2a}$, which written as a series corresponds to $r_1 = \sum_{n\geq 0} C_n\, a^{2n+1}$, with $C_n = \frac{1}{n+1}\binom{2n}{n}$ the n-th Catalan number. It is known that this formal power series cannot be written as the solution of a linear equation with coefficients in $\mathbb{Q}\langle\{a\}^{\oplus}\rangle$ [2].

▶ **Example 11.** Now we will consider the WCFG given in Example 4. This time we will give a complete definition of its grammar rules and, as in the previous example, we will extend its weight domain from $\mathbb{N}$ to $\mathbb{Q}$. Define the WCFG (G_2, W_2) where $G_2 = (\{X_2, \overline{D}, D, Y, Z\}, \{a, \overline{a}\}, X_2, R_2)$, R_2 is given by:

$$X_2 \to D \mid \overline{D} \qquad \overline{D} \to D\,\overline{a}\,Y \mid D\,a\,Z \qquad Z \to D\,a\,Z \mid D \ .$$
$$D \to a\,D\,\overline{a}\,D \mid \varepsilon \qquad Y \to a\,Y \mid \overline{a}\,Y \mid \varepsilon$$

and the weight function W_2 is defined over $(\mathbb{Q}, +, \cdot, 0, 1)$ and assigns 1 to each production in the grammar. Note that (G_2, W_2) is cycle-free. The grammar variable D generates all the words in the Dyck language L_D over the alphabet $\{a, \overline{a}\}$, while the variable $\overline{D}$ generates $\{a, \overline{a}\}^* \setminus L_D$. The system S corresponding to (G_2, W_2) consists of the following equations:

$$X_2 = D + \overline{D} \qquad \overline{D} = D\,\overline{a}\,Y + D\,a\,Z \qquad Z = D\,a\,Z + D \ .$$
$$D = a\,D\,\overline{a}\,D + 1 \qquad Y = a\,Y + \overline{a}\,Y + 1$$

Let $\sigma = (r_1, r_2, r_3, r_4, r_5)$ be its strong solution where r_1 corresponds to the solution for the initial variable X_2. Assume for now that the irreducible polynomial $q(X_2) \in \mathbb{Q}\langle\{a, \overline{a}\}^{\oplus}\rangle\langle X_2\rangle$ described by Theorem 9 is:

$$q(X_2) = (1 - (a + \overline{a}))X_2 - 1 \ .$$

We observe that q is linear in X_2 and can be written as:

$$q(X_2) = (1 - s)X_2 - t = (1 - (a + \overline{a}))X_2 - 1 \ ,$$

with $(s, \varepsilon) = 0$. Thus, by Theorem 8, we conclude that (G_2, W_2) satisfies the Parikh property as we expected.

Now we develop the technique we will use to construct the irreducible polynomial of Theorem 9: Groebner bases. A Groebner basis is a set of polynomials in one or more variables enjoying certain properties. Given a set of polynomials F with coefficients in a field, one can compute a Groebner basis G of F with the property that G has the same solutions as F when interpreted as a polynomial system of equations. Then, problems such as finding the solutions for the system induced by F, or looking for alternative representations of polynomials in terms of other polynomials become easier using G instead of F. One of the main insights for using Groebner bases is that they are *effectively constructable* using standard computer algebra systems, for any set of polynomials with coefficients in a field.

We are interested in computing Groebner bases of algebraic systems in commuting variables corresponding to weighted CFGs. Given a WCFG and its corresponding algebraic system, our goal is to obtain a system with the same solution as the original, and such that one equation in the new system depends only on the initial grammar variable X_1. This equation will contain all the information needed to decide whether or not the given WCFG satisfies the Parikh property. We will not enter into the technical details of how Groebner bases are constructed and their properties as these lie beyond the scope of this document (however, an explicit reference will be given in connection with each result applied). Instead, we will give a result that encapsulates all the preconditions and postconditions we need for our purpose (Theorem 13). We first introduce the definitions that will appear in the theorem.

In what follows, K will always denote a field. First we need to introduce the notion of *ideal*. Let $K\langle V^{\oplus}\rangle$ denote the ring of polynomials in variables V and with coefficients in K. A subset $I \subset K\langle V^{\oplus}\rangle$ is an *ideal* iff (i) $0_K \in I$, (ii) if $f, g \in I$ then $f + g \in I$, and (iii) if

$f \in I$ and $h \in K\langle V^{\oplus}\rangle$ then $h \cdot f \in I$. Given a set of polynomials $F = \{f_1, \ldots, f_n\}$, we define $\langle F\rangle$ as $\langle F\rangle \stackrel{\text{def}}{=} \{\sum_{i=1}^{n} h_i \cdot f_i \mid h_i \in K\langle V^{\oplus}\rangle, f_i \in F\}$. It can be shown that $\langle F\rangle$ is an ideal [3] and we call it the *ideal generated by* F. When an ideal is generated by a finite number of polynomials $g_1, \ldots, g_n \in K\langle V^{\oplus}\rangle$, we say that $g_1, \ldots, g_n$ is a *basis* of the ideal. It is known that every ideal in $K\langle V^{\oplus}\rangle$ has a basis (actually many, but the ones we are particularly interested in are the so-called Groebner bases) [3]. If one considers the set of polynomial equations $\{f = 0 \mid f \in F\}$, denoted by $F = \mathbf{0}$, then the set of all solutions of $F = \mathbf{0}$ is defined as $\{(r_1, r_2, \ldots, r_n) \in K^n \mid f(r_1, \ldots, r_n) \equiv 0, \text{ for all } f \in F\}$. Then, given two sets of polynomials F and G, if $\langle F\rangle = \langle G\rangle$ then the set of solutions of $F = \mathbf{0}$ coincides with the set of solutions of $G = \mathbf{0}$ [3]. To construct a Groebner basis of an ideal I, one needs to impose first a total ordering on the monomials of variables occuring in I. This choice is significant as different orderings lead to different Groebner bases with different properties. We are interested in computing Groebner bases with the *elimination property* for the initial variable X_1, i.e., bases where at least one polynomial depends only on X_1. Hence, we will always impose the *reverse lexicographic ordering* to construct Groebner bases.

▶ **Definition 12.** Let $V = \{X_1, \ldots, X_n\}$ be a set of variables. Let α and β be two monomials in $V^{\oplus}$ and let $\overline{\alpha}$ (resp. $\overline{\beta}$) be the vector in $\mathbb{N}^n$ such that its i-th component corresponds to the number of occurrences of the variable X_i in α (resp. β). Then we say that α *is greater than* β *w.r.t. the reverse lexicographic ordering*, denoted by $\alpha \succ_{revlex} \beta$, iff the first non-zero component of the vector $\overline{\alpha} - \overline{\beta}$ is negative.

Notice that Definition 12 implies an ordering of the variables: $X_n \succ_{revlex} X_{n-1} \succ_{revlex} \ldots \succ_{revlex} X_1$. The reason for choosing the *reverse* lexicographic ordering is that, in order to compute a Groebner basis with the elimination property for the initial variable X_1, we need X_1 to be the least monomial (with one or more variable). In what follows, the phrase "w.r.t. the reverse lexicographic ordering" (for some given $V = \{X_1, \ldots, X_n\}$) will refer to the one described in Definition 12 with variables V, unless stated otherwise. Fixed a total monomial ordering, we define the *leading monomial* of a polynomial p as the greatest monomial in p, and we denote it by $LM(p)$. We define the *leading term* of p as the leading monomial of p together with its coefficient, and we denote it by $LT(p)$. Finally, we introduce the notion of a *reduced* Groebner basis which allows to define uniquely a Groebner basis of an ideal of polynomials. Let F be a set of polynomials and G a Groebner basis of $\langle F\rangle$. We say that G is a *reduced* Groebner basis of $\langle F\rangle$ iff for each $g_i \in G$ (i) the coefficient of $LT(g_i) = 1$; and (ii) $LM(g_i)$ does not divide any term of any g_j with $i \neq j$. For a given set of polynomials F and monomial ordering $\succ$, there exists exactly one reduced Groebner basis of $\langle F\rangle$ w.r.t. $\succ$ [3]. We abuse notation and write $K\langle X\rangle$ instead of $K\langle\{X\}^{\oplus}\rangle$ to refer to the ring of polynomials in the variable X with coefficients in K. Now we are ready to give the theorem.

▶ **Theorem 13.** *Let K be a field and $V = \{X_1, \ldots, X_n\}$ a set of variables. Let $F \subseteq K\langle V^{\oplus}\rangle$ be a set of polynomials such that the strong solution of the system $F = \mathbf{0}$ is $(r_1, \ldots, r_n)$ where r_i corresponds to the solution for X_i. Let G be the reduced Groebner basis of $\langle F\rangle$ w.r.t. the reverse lexicographic ordering. Then the following properties are satisfied:*

1. *$(r_1, \ldots, r_n)$ is also the strong solution of the system $G = \mathbf{0}$ and,*
2. *there is exactly one polynomial $g \in G$ s.t. $g \in K\langle X_1\rangle$, and for that g we have $g(r_1) \equiv 0$.*

Proof. Property 1. follows from the fact that G is a basis of $\langle F\rangle$. Now we prove property 2. G is a Groebner basis of $\langle F\rangle$ w.r.t. the reverse lexicographic ordering. Then, as a result of the Elimination Theorem [3, Theorem 3.1.2], $G \cap K\langle X_1\rangle$ is a Groebner basis of $\langle F\rangle \cap K\langle X_1\rangle$. Assume first that $G \cap K\langle X_1\rangle$ contains only the zero polynomial (the constant polynomial

whose coefficients are equal to 0). Then the ideal $\langle F\rangle \cap K\langle X_1\rangle$ also contains only the zero polynomial. But this contradicts Theorem 9. Then $G \cap K\langle X_1\rangle$ contains at least one nonzero polynomial g. Assume now that $G \cap K\langle X_1\rangle$ contains two different elements g_1 and g_2 in $K\langle X_1\rangle$. W.l.o.g., let g_1 be such that $LM(g_1) \preceq_{lex} LM(g_2)$. Thus, $LM(g_1)$ divides (at least) the leading term of g_2. Then G is not in reduced form (contradiction). We conclude that there is exactly one (nonzero) polynomial $g \in G$ such that $g \in K\langle X_1\rangle$. Finally, $g(r_1) \equiv 0$ follows from 1. and the fact that $g \in (G \cap K\langle X_1\rangle)$. ◀

Now we show in Theorem 14 how to construct q using Groebner bases. Finally, we give in Theorem 15 the main result of this section.

▶ **Theorem 14.** *Let S be a $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$-algebraic system in commuting variables corresponding to a cycle-free WCFG and r_1 be the first component of its strong solution. Then an irreducible polynomial $q(X_1)$ with coefficients in $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$ such that $q(r_1) \equiv 0$ can be effectively constructed.*

Proof. We begin with the first part of the algorithm. Let K be the fraction field of $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$, i.e., the smallest field (w.r.t. inclusion) containing $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$. Consider S as defined in (2) (page 7) where now each polynomial p_i has its coefficients in K and its variables in V, and let $F \subseteq K\langle V^{\oplus}\rangle$ be the set of polynomials $\{p_i \mid 1 \leq i \leq n\}$. Construct the reduced Groebner basis G of F w.r.t. the reverse lexicographic ordering. Let $G = \{g_1, \ldots, g_s\}$ with $s \geq 1$. By Theorem 13, there is exactly one $g \in G$ such that $g \in K\langle X_1\rangle$, and g satisfies $g(r_1) \equiv 0$.

We cannot conclude yet that $g(X_1)$ is the polynomial $q(X_1)$ we are looking for since $g(X_1)$ might not be irreducible in the fraction field of $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$. This constitutes the second part of the algorithm which follows the method given in [15] to obtain from $g(X_1)$ an irreducible polynomial $q(X_1)$ such that $q(r_1) \equiv 0$. Compute the factorization[6] of g in the fraction field of $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$ and let $\{q_1(X_1), \ldots, q_m(X_1)\}$ with $m \geq 1$ be the set of all irreducible polynomials obtained thus as factors. Because $g(r_1) \equiv 0$, there exists an index j_0 with $1 \leq j_0 \leq m$ such that $q_{j_0}(r_1) \equiv 0$ and $q_j(r_1) \not\equiv 0$ for $j \neq j_0$ and $1 \leq j \leq m$. Now we show how to find j_0. Using the operator R_k introduced in the beginning of Section 4, we have that $R_k(q_{j_0}(R_k(r_1))) \equiv 0$ for all $k \geq 0$, while for each $j \neq j_0$ there is always an index k_j such that $R_{k_j}(q_j(R_{k_j}(r_1))) \not\equiv 0$. Then, eventually an index j_0 is always found. Let $q_{j_0}(X_1) = \frac{n_k}{d_k}X_1^k + \frac{n_{k-1}}{d_{k-1}}X_1^{k-1} + \ldots + \frac{n_0}{d_0}$ with $k \geq 0, n_i, d_i \in \mathbb{Q}\langle\Sigma^{\oplus}\rangle$ and $d_i \neq 0$ for all i. Let $lcm(d_0, \ldots, d_k)$ denote the least common multiple of $d_0, \ldots, d_k$ and define $q(X_1) = lcm(d_0, \ldots, d_k) \cdot q_{j_0}(X_1)$. Now $q(X_1) \in \mathbb{Q}\langle\Sigma^{\oplus}\rangle\langle X_1\rangle$ and this completes the algorithm. ◀

▶ Remark. It is worth noting that, even though $q(X_1)$ is an irreducible polynomial over K, the fraction field of $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$, it might not be irreducible over $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$ since it might have a factorization consisting of a polynomial $\widetilde{q}(X_1) \in \mathbb{Q}\langle\Sigma^{\oplus}\rangle\langle X_1\rangle$ of the same degree and one or more constant polynomials over $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$, i.e., polynomials of degree zero, that are not units in $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$. However, since constant factors are not relevant for the result, we say that a polynomial over $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$ is *irreducible* iff either no factorization exists, or, if there is one, then it is of the aforementioned form.

▶ **Theorem 15.** *Let (G, W) be a cycle-free WCFG with W defined over $\mathbb{Q}$. Then, it is decidable whether or not (G, W) verifies the Parikh property.*

Proof. Let S be the $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$-algebraic system corresponding to G and let r_1 be the first component of its strong solution. Construct the irreducible polynomial $q(X_1)$ with coefficients in $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$ as in Theorem 14. By Theorem 8, we only need to check whether or not the

[6] Polynomial factorizations are performed w.r.t. polynomials with coefficients in the fraction field of $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$ which is a computable field.

equation $q(X_1) = 0$ can be written as a linear equation of the form: $(1-s)X_1 - t = 0$, with $s, t \in \mathbb{Q}\langle\Sigma^{\oplus}\rangle$ and $(s, \varepsilon) = 0$. Observe that the procedure given in Theorem 14 is complete, i.e., if the polynomial q obtained is not linear in X_1 then there cannot exist a polynomial $q_\ell(X_1)$ with coefficients in $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$ and linear in X_1 such that $q_\ell(r_1) \equiv 0$. If it were the case, then q_ℓ would be necessarily a factor of q, and this contradicts the fact that q is irreducible over $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$. Then, if q is not linear in X_1, we conclude that (G, W) does not satisfy the Parikh property. Otherwise, $q(X_1)$ can be rewritten as $q(X_1) = (1-s)X_1 - t$ with $s, t \in \mathbb{Q}\langle\Sigma^{\oplus}\rangle$ and $(s, \varepsilon) = 0$, and we conclude that (G, W) satisfies the Parikh property. ◀

Consider a WCFG (G, W) with r_1 the first component of the solution of its corresponding algebraic system. Observe that, if the decision procedure returns a positive answer for (G, W) then the polynomial $q(X_1)$ constructed as in Theorem 14 is of the form:

$$q(X_1) = (s_0 - s_1)X_1 - t = 0 \ ,$$

with $s_0 \in \mathbb{Q}, s_0 \neq 0$ and $s_1, t \in \mathbb{Q}\langle\Sigma^{\oplus}\rangle$ with $(s_1, \varepsilon) = (t, \varepsilon) = 0$. It follows that the algebraic system consisting of the equation:

$$X_1 = \frac{1}{s_0} s_1 X_1 + \frac{1}{s_0} t \ , \tag{3}$$

has also r_1 as solution. Then a regular WCFG Parikh-equivalent to (G, W) is the one corresponding to the regular algebraic system (3).

Now we complete Examples 10 and 11 by following the decision procedure given in Theorem 15 and giving the construction of a Parikh-equivalent regular WCFG (if exists). Additionally, we give a third example.

▶ **Example 16.** Consider the WCFG (G, W) given in Example 10. Recall that its corresponding algebraic system S is given by the equation $X = a\,X^2 + a$. Let r be its strong solution. Now we construct the irreducible polynomial $q(X) \in \mathbb{Q}\langle\{a\}^{\oplus}\rangle\langle X\rangle$ following the procedure given in Theorem 14. Let $F = \{a\,X^2 - X + a\}$. The reduced Groebner basis G of F w.r.t. reverse lexicographic ordering is (trivially) $G = \{X^2 - \frac{1}{a}X + 1\}$. Then the polynomial $g \in G$ such that $g \in K\langle X\rangle$ where K is the fraction field of $\mathbb{Q}\langle\{a\}^{\oplus}\rangle$, and $g(r_1) \equiv 0$ is:

$$g(X) = X^2 - \frac{1}{a}X + 1 \ .$$

Note that this polynomial cannot be reduced into factors in the fraction field of $\mathbb{Q}\langle\{a\}^{\oplus}\rangle$. Multiplying g by a, we get $q(X) = aX^2 - X + a \in \mathbb{Q}\langle\{a\}^{\oplus}\rangle\langle X\rangle$ and we conclude that $q(X)$ is the irreducible polynomial described by Theorem 9. As $q(X)$ is not linear we conclude that (G, W) does not satisfy the Parikh property.

▶ **Example 17.** Now consider the WCFG given in Example 4 and its corresponding algebraic system S. We construct the irreducible polynomial $q(X_2) \in \mathbb{Q}\langle\{a, \overline{a}\}^{\oplus}\rangle\langle X_2\rangle$ following the procedure given in Theorem 14. Given F, the set of polynomials in the left-hand sides of the equations of S after moving all monomials from right to left, we construct the reduced Groebner basis G of F w.r.t. reverse lexicographic ordering. For clarity, we just show the polynomial $g \in G$ such that $g \in K\langle X_2\rangle$ where K is the fraction field of $\mathbb{Q}\langle\{a, \overline{a}\}^{\oplus}\rangle$, and verifies $g(r_1) \equiv 0$:

$$g(X_2) = X_2 - \frac{1}{1 - (a + \overline{a})} \ .$$

This polynomial is linear so it is irreducible over the fraction field of $\mathbb{Q}\langle\{a,\overline{a}\}^{\oplus}\rangle$. Now we multiply g by $(1-(a+\overline{a}))$ and thus obtain $q(X_2) = (1-(a+\overline{a}))X_2 - 1 \in \mathbb{Q}\langle\{a,\overline{a}\}^{\oplus}\rangle\langle X_2\rangle$ which is the irreducible polynomial described by Theorem 9. Now we apply the decision procedure described in Theorem 15. We observe that q can be written as follows:

$$q(X_2) = (1-s)X_2 - t = (1-(a+\overline{a}))X_2 - 1 \ ,$$

with $(s,\varepsilon) = 0$. Thus, we conclude that (G,W) satisfies the Parikh property. Finally, we give a regular Parikh-equivalent WCFG (G_ℓ, W_ℓ). The regular algebraic system:

$$(1-(a+\overline{a}))X_2 - 1 = 0 \iff X_2 = (a+\overline{a})X_2 + 1 \tag{4}$$

has r_1 as solution. Then, the WCFG (G_ℓ, W_ℓ) corresponding to (4) is given by $G_\ell = (\{X_2\}, \{a,\overline{a}\}, R_\ell, X_2)$ with R_ℓ defined as:

$$\pi_1 = X_2 \to aX_2$$
$$\pi_2 = X_2 \to \overline{a}X_2$$
$$\pi_3 = X_2 \to \varepsilon$$

and W_ℓ defined over $(\mathbb{Q},+,\cdot,0,1)$ as $W_\ell(\pi_i) = 1$, for all i. Notice that (G_ℓ, W_ℓ) coincides with (G_1, W_1) in Example 4.

▶ **Example 18.** Consider the cycle-free WCFG (G,W) given by $G = (\{X_1,X_2\},\{a,b\},R,X_1)$ with R defined as follows:

$$X_1 \to aX_2X_2$$
$$X_2 \to bX_2 \mid a \ ,$$

and the weight function W over $(\mathbb{Q},+,\cdot,0,1)$ that assigns 1 to each production in the grammar. The algebraic system S corresponding to (G,W) is defined as follows:

$$\begin{cases} X_1 = a\,X_2^2 \\ X_2 = b\,X_2 + a \ . \end{cases}$$

Let $\sigma = (r_1, r_2)$ be its strong solution. Now we construct the irreducible polynomial $q(X_1) \in \mathbb{Q}\langle\{a,b\}^{\oplus}\rangle\langle X_1\rangle$ following the procedure given in Theorem 14. Let $F = \{X_1 - a\,X_2^2, X_2 - b\,X_2 - a\}$. The reduced Groebner basis[7] G of F w.r.t. lexicographic ordering is:

$$G = \left\{ X_1 - \frac{a^3}{b^2-2b+1}, X_2 + \frac{a}{b-1} \right\} \ .$$

Clearly, the polynomial $g \in G$ such that $g \in K\langle X_1\rangle$ where K is the fraction field of $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$, and $g(r_1) \equiv 0$ is:

$$g(X_1) = X_1 - \frac{a^3}{b^2-2b+1} \ .$$

This polynomial cannot be reduced into factors in the fraction field of $\mathbb{Q}\langle\Sigma^{\oplus}\rangle$. Now we multiply g by (b^2-2b+1) and thus obtain $q(X_1) = (b^2-2b+1)X_1 - a^3$ in $\mathbb{Q}\langle\Sigma^{\oplus}\rangle\langle X_1\rangle$

[7] The Groebner basis G was computed using the `groebner_basis` method of the open-source mathematics software system SageMath.

which is the irreducible polynomial described by Theorem 9. Now we apply the decision procedure described in Theorem 15. We observe that q is linear in X_1 and can be written as:

$$q(X_1) = (1-s)X_1 - t = (1-(2b-b^2))X_1 - a^3 \ ,$$

with $(s, \varepsilon) = 0$. Then we conclude that (G, W) satisfies the Parikh property. Note that this is the result expected as (G, W) is nonexpansive. Finally, we give a regular Parikh-equivalent WCFG (G_ℓ, W_ℓ). We know that the algebraic system:

$$X_1 = (2b - b^2)X_1 + a^3 \tag{5}$$

has r_1 as solution. Then the WCFG (G_ℓ, W_ℓ) corresponding to the regular system (5) is given by $G_\ell = (\{X_1\}, \{a, b\}, R_\ell, X_1)$ with R_ℓ defined as:

$$\pi_1 = X_1 \to bX_1$$
$$\pi_2 = X_1 \to b^2X_1$$
$$\pi_3 = X_1 \to a^3$$

and W_ℓ defined over $(\mathbb{Q}, +, \cdot, 0, 1)$ as:

$$W_\ell(\pi) = \begin{cases} 2 & \text{if } \pi = \pi_1 \\ -1 & \text{if } \pi = \pi_2 \\ 1 & \text{if } \pi = \pi_3 \end{cases} .$$

5 Related Work

The problem of extending Parikh's Theorem to the weighted case has been significantly considered in the literature [2, 14, 16, 18]. Petre [18] establishes that the family of power series in commuting variables that can be generated by regular WCFGs is *strictly* contained in that of the series generated by arbitrary WCFGs. In this way, he shows that Parikh's Theorem does not hold in the weighted case. It is well-known that the Parikh property holds in a commutative and idempotent semiring [2, 14, 16]. Luttenberger et al. [16] deal with WCFGs where the weight of a word corresponds to its ambiguity (or commutative ambiguity when considering monomials instead of words) and they show that if a CFG is nonexpansive then its commutative ambiguity can be expressed by a weighted rational expression relying on the fact that all the parse trees of a nonexpansive CFG are of bounded dimension. We used this fact to give a Parikh-equivalent regular WCFG construction, for a given nonexpansive WCFG defined over *any* commutative semiring. Baron and Kuich [1] gave a similar characterization of nonexpansive grammars using rational power series to that of Luttenberger et al. They also conjectured that an unambiguous WCFG is nonexpansive iff it has the Parikh property. This conjecture appears to be false as evidenced by Example 4. Bhattiprolu et al. [2] also show that the class of *polynomially ambiguous* WCFGs over the unary alphabet satisfies the property. In the unary case, this class is strictly contained in the class of nonexpansive grammars (see Appendix D in the extended version of this paper [7]). Finally, our decision procedure relies on a result by Kuich and Saloma [15] that decides if an *algebraic* series in commuting variables with coefficients in $\mathbb{Q}$ is rational. To the best of our knowledge, the connection of this result to a decidability result for the Parikh property was only implicit.

6 Conclusions and Further Work

Note that from the theoretical point of view, our decision procedure can be applied to WCFGs over any arbitrary *field*. For arbitrary semirings, the decidability of the Parikh property remains open. It would be interesting to tackle the question first in the unary case. Finally, Theorem 3 shows an equivalent characterization of the Parikh property. Namely, the Parikh property holds for a WCFG (G, W) iff there exists a Parikh-equivalent nonexpansive WCFG, i.e., iff (G, W) is not *inherently* expansive. It is known that inherent expansiveness is undecidable in the noncommutative and unweighted case [10], but the question remains unsolved in the commutative case when weights are considered.

References

1 Gerd Baron and Werner Kuich. The Characterization of Nonexpansive Grammars by Rational Power Series. *Information and Control*, 48(2):109–118, 1981. doi:10.1016/S0019-9958(81)90634-3.

2 Vijay Bhattiprolu, Spencer Gordon, and Mahesh Viswanathan. Extending Parikh's Theorem to Weighted and Probabilistic Context-Free Grammars. In *QEST 2017*, pages 3–19, 2017. doi:10.1007/978-3-319-66335-7_1.

3 David A. Cox, John Little, and Donal O'Shea. *Ideals, varieties, and algorithms - an introduction to computational algebraic geometry and commutative algebra (2. ed.)*. Undergraduate texts in mathematics. Springer, 1997.

4 Javier Esparza. Petri Nets, Commutative Context-Free Grammars, and Basic Parallel Processes. *Fundam. Inform.*, 31(1):13–25, 1997. doi:10.3233/FI-1997-3112.

5 Javier Esparza, Pierre Ganty, Stefan Kiefer, and Michael Luttenberger. Parikh's theorem: A simple and direct automaton construction. *Inf. Process. Lett.*, 111(12):614–619, 2011. doi:10.1016/j.ipl.2011.03.019.

6 Javier Esparza, Peter Rossmanith, and Stefan Schwoon. A Uniform Framework for Problems on Context-Free Grammars. *Bulletin of the EATCS*, 72:169–177, 2000.

7 Pierre Ganty and Elena Gutiérrez. The Parikh Property for Weighted Context-Free Grammars (extended version). *arXiv*, 2018. arXiv:1810.01351.

8 Pierre Ganty and Rupak Majumdar. Algorithmic Verification of Asynchronous Programs. *CoRR*, abs/1011.0551, 2010. arXiv:1011.0551.

9 Stefan Göller, Richard Mayr, and Anthony Widjaja To. On the Computational Complexity of Verifying One-Counter Processes. In *LICS 2009*, pages 235–244, 2009. doi:10.1109/LICS.2009.37.

10 Jozef Gruska. A Few Remarks on the Index of Context-Free Grammars and Languages. *Information and Control*, 19(3):216–223, 1971. doi:10.1016/S0019-9958(71)90095-7.

11 Dung T. Huynh. The Complexity of Equivalence Problems for Commutative Grammars. *Information and Control*, 66(1/2):103–121, 1985. doi:10.1016/S0019-9958(85)80015-2.

12 Thiet-Dung Huynh. The Complexity of Semilinear Sets. In *ICALP 1980*, pages 324–337, 1980. doi:10.1007/3-540-10003-2_81.

13 Thiet-Dung Huynh. Deciding the Inequivalence of Context-Free Grammars with 1-Letter Terminal Alphabet is Σ_2^P-complete. In *FOCS 1982*, pages 21–31, 1982. doi:10.1109/SFCS.1982.65.

14 Werner Kuich. The Kleene and the Parikh Theorem in Complete Semirings. In *ICALP 1987*, pages 212–225, 1987. doi:10.1007/3-540-18088-5_17.

15 Werner Kuich and Arto Salomaa. *Semirings, Automata, Languages*, volume 5 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1986. doi:10.1007/978-3-642-69959-7.

16 Michael Luttenberger and Maximilian Schlund. Convergence of Newton's Method over Commutative Semirings. *Inf. Comput.*, 246:43–61, 2016. doi:10.1016/j.ic.2015.11.008.

17 Rohit Parikh. On Context-Free Languages. *J. ACM*, 13(4):570–581, 1966. doi:10.1145/321356.321364.

18 Ion Petre. Parikh's theorem does not hold for multiplicities. *Journal of Automata, Languages and Combinatorics*, 4(1):17–30, 1999.

19 Thomas W. Reps, Stefan Schwoon, Somesh Jha, and David Melski. Weighted pushdown systems and their application to interprocedural dataflow analysis. *Sci. Comput. Program.*, 58(1-2):206–263, 2005. doi:10.1016/j.scico.2005.02.009.

20 Koushik Sen and Mahesh Viswanathan. Model Checking Multithreaded Programs with Asynchronous Atomic Methods. In *CAV 2006*, pages 300–314, 2006. doi:10.1007/11817963_29.

21 Kumar Neeraj Verma, Helmut Seidl, and Thomas Schwentick. On the Complexity of Equational Horn Clauses. In *CADE 2005*, pages 337–352, 2005. doi:10.1007/11532231_25.

A Proof of Theorem 3

First, we give the definitions we will use in this section. Given a CFG $G = (V, \Sigma, S, R)$, define the *degree* of G as $max\{|\gamma\downarrow_V| : (X \to \gamma) \in R\} - 1$, where $\gamma\downarrow_V$ denotes the projection of γ onto the variables V. Given a production $\pi = (X \longrightarrow \gamma) \in R$ and a position $1 \leq i \leq |\alpha|$, we define a *derivation step* $\alpha \overset{\pi/i}{\Longrightarrow} \beta$ with $\alpha, \beta \in (\Sigma \cup V)^*$ iff $(\alpha)_i = X$ and $\beta = (\alpha)_1 \dots (\alpha)_{i-1} \gamma (\alpha)_{i+1} \dots (\alpha)_{|\alpha|}$. We omit the position i when it is not important. We say that α and β in $(\Sigma \cup V)^*$ are *derivation sentences* of G. We define a *derivation sequence* $\alpha_0 \overset{\pi_1}{\Longrightarrow} \alpha_1 \overset{\pi_2}{\Longrightarrow} \dots \overset{\pi_n}{\Longrightarrow} \alpha_n$ iff for every $i \in \{1, \dots, n\}$, $\alpha_{i-1} \overset{\pi_i}{\Longrightarrow} \alpha_i$ is a derivation step. We call the derivation step $\alpha_{i-1} \overset{\pi_i}{\Longrightarrow} \alpha_i$ the *i-step* of the derivation sequence. A derivation sequence $\psi = \alpha_0 \Rightarrow \dots \Rightarrow \alpha_n$ of G has *index* j, denoted by $idx(\psi)$, if for every $i \in \{0, \dots, n\}$, no word $(\alpha_i)\downarrow_V$ is longer than j. Now we define the dimension of a labeled tree as follows.

▶ **Definition 19.** Given a labeled tree $\tau = c(\tau_1, \dots, \tau_n)$ $(n \geq 0)$, the *dimension* of τ represented as $dim(\tau)$ is defined as follows:

$$dim(c(\tau_1, \dots, \tau_n)) \overset{\text{def}}{=} \begin{cases} 0 & \text{if } n = 0 \\ dim(\tau_i) & \text{if } n > 0 \wedge |\{i \mid \forall j \colon dim(\tau_j) \leq dim(\tau_i)\}| = 1 \\ dim(\tau_i) + 1 & \text{if } n > 0 \wedge |\{i \mid \forall j \colon dim(\tau_j) \leq dim(\tau_i)\}| > 1 \end{cases}$$

Now we present the proof of Theorem 3. All the definitions, lemmas and theorems referred there can be found below the proof.

▶ **Theorem 3.** *Let (G, W) be an arbitrary WCFG. If G is nonexpansive then (G, W) satisfies the Parikh property.*

Proof. The proof is constructive. For every nonexpansive WCFG (G, W), we give a 2-step construction that results in a Parikh-equivalent regular WCFG (G_ℓ, W_ℓ). The steps are:

1. construct a new WCFG $\left(G^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$, with $k \in \mathbb{N}$, language-equivalent to (G, W); and
2. construct a regular WCFG (G_ℓ, W_ℓ) Parikh-equivalent to $\left(G^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$.

The first part of the construction consists of building a new WCFG $\left(G^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$ (Definition 20 below), so-called *at-most-k-dimension* WCFG of (G, W), which is language-equivalent to the original and where grammar variables are annotated with information about the dimension of the parse trees that can be obtained from these variables. Let us give an intuition on its construction.

For a given CFG G and $k \in \mathbb{N}$ (the choice of $k \in \mathbb{N}$ will be described later on), we define $G^{\lceil k \rceil}$ using the same construction as Luttenberger et al. [16]. They show how to construct, for a given CFG G, a new grammar $G^{\lceil k \rceil}$ with the property that $\mathcal{T}_{G^{\lceil k \rceil}}$ corresponds to the subset of $\mathcal{T}_G$ of trees of dimension at most k. They annotate each grammar variable with the superscript $[d]$ (resp. $\lceil d \rceil$) to denote that only parse trees of dimension exactly d (resp. at most d), where $d \leq k$, can be obtained from these variables. When constructing the grammar, they also consider those rules containing two or more variables in its right-hand side and distinguish which cases yield an increase of dimension. We recall the construction of $G^{\lceil k \rceil}$ in Definition 20.

To define the weight function $W^{\lceil k \rceil}$, we assign to each rule in $G^{\lceil k \rceil}$ the same weight as its corresponding version in G (note that for those rules in $G^{\lceil k \rceil}$ with no corresponding version in G, i.e. the so-called e-rules, we assign the identity 1_A with respect to $\cdot$, where A denotes the weight domain). Let us discuss the choice of k in $\left(G^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$. Luttenberger et al. [16] also show that if G is a nonexpansive CFG then the dimension of every parse tree in $\mathcal{T}_G$ is bounded (Theorem 21). Moreover, the bound is at most the number of grammar variables of G. Then, for a given nonexpansive WCFG (G, W), define k as this bound. Because k is at most equal to the number of variables of G, such a value is always found and consequently, the first part of the construction always terminates. Finally, we show that the WCFG $\left(G^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$ is language-equivalent to (G, W) (Lemma 22).

In the second part of the construction, we build a regular WCFG (G_ℓ, W_ℓ) that is Parikh-equivalent to $\left(G^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$. Esparza et al. [5] show that if the dimension of a parse tree is bounded by k then there exists a derivation sequence for the yield of the tree whose index is bounded by some affine function of k (Lemma 23). We rely on this result to define a special derivation policy over at-most-k-dimension WCFGs, for which we know the dimension of every parse tree is bounded by k. They are called *lowest-dimension-first (LDF) derivations*. We prove that, for every WCFG $\left(G^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$, the index of an LDF derivation sequence is always bounded by an affine function of k (Lemma 25). Then, each grammar variable of (G_ℓ, W_ℓ) represents each possible sentence (without the terminals) along an LDF derivation sequence of $\left(G^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$, and each grammar rule is intended to simulate an LDF derivation step of $\left(G^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$. Because the number of variables in these sentences is bounded, the sets of variables and rules of (G_ℓ, W_ℓ) are necessarily finite. A formal definition of the weighted regular (G_ℓ, W_ℓ) is given in Definition 26 . Finally we show that (G_ℓ, W_ℓ) is Parikh-equivalent to $\left(G^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$ (Lemma 27) and this concludes the proof. ◀

Now we give the construction of the at-most-k-dimension WCFG $(G^{\lceil k \rceil}, W^{\lceil k \rceil})$ for a given WCFG (G, W) and $k \in \mathbb{N}$. For the construction of $G^{\lceil k \rceil}$, we rely on the one given by Luttenberger et al. [16].

▶ **Definition 20** (The at-most-k-dimension WCFG)**.** Let (G, W) be a WCFG with $G = (V, \Sigma, S, R)$ and W defined over the commutative semiring A, and let $k \in \mathbb{N}$. Define the *at-most-k-dimension WCFG* $\left(G^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$ with $G^{\lceil k \rceil} = (V^{\lceil k \rceil}, \Sigma, S^{\lceil k \rceil}, R^{\lceil k \rceil})$ of (G, W) (with $u_0, \ldots, u_n \in \Sigma^*$) as follows:

- The set $V^{\lceil k \rceil}$ of variables is given by

$$\{X^{[d]}, X^{\lceil d \rceil} \mid X \in V, 0 \leq d \leq k\} \ .$$

- The set $R^{\lceil k \rceil}$ of production rules is given by
  1. Linear rules:
     - $r_0(\pi) = \{X^{[0]} \to u_0\}$ for each rule $\pi = (X \to u_0) \in R$.
     - $r_1(\pi) = \{X^{[d]} \to u_0\, X_1^{[d]}\, u_1 \mid 0 \leq d \leq k\}$ for each rule $\pi = (X \to u_0\, X_1\, u_1) \in R$.

2. Non-linear rules:
   For each rule $\pi = (X \to u_0\, X_1\, u_1 \dots u_{n-1}\, X_n\, u_n) \in R$
   - $r_2(\pi) = \{X^{[d]} \to u_0\, Z_1\, u_1 \dots u_{n-1}\, Z_n\, u_n \mid 1 \le d \le k, J \subseteq \{1,\dots,n\} \text{ with } |J| = 1 : Z_i = X_i^{[d]} \text{ if } i \in J, \text{ and } Z_i = X_i^{\lceil d-1 \rceil} \text{ for all } i \in \{1,\dots,n\} \setminus J\}$ and
   - $r_3(\pi) = \{X^{[d]} \to u_0\, Z_1\, u_1 \dots u_{n-1}\, Z_n\, u_n \mid 1 \le d \le k, J \subseteq \{1,\dots,n\} \text{ with } |J| \ge 2 : Z_i = X_i^{[d-1]} \text{ for all } i \in J \text{ and } Z_i = X_i^{\lceil d-1 \rceil} \text{ for all } i \in \{1,\dots,n\} \setminus J\}$.
3. e-rules:
   - $r_4 = \{X^{\lceil d \rceil} \to X^{[e]} \mid 0 \le e \le d \le k\}$.

- The weight function $W^{\lceil k \rceil}$ is given by

$$W^{\lceil k \rceil}(\varphi) = \begin{cases} W(\pi) & \text{if } \varphi \in r_0(\pi) \text{ for some } \pi = (X \to u_0) \in R \\ W(\pi) & \text{if } \varphi \in r_1(\pi) \text{ for some } \pi = (X \to u_0\, X_1\, u_1) \in R \\ W(\pi) & \text{if } \varphi \in r_2(\pi) \cup r_3(\pi) \text{ for some } \pi = (X \to u_0\, Z_1\, u_1, \dots, u_{n-1}\, Z_n\, u_n) \in R \\ 1_A & \text{if } \varphi \in r_4 \end{cases}$$

We say that a variable $Z \in V^{\lceil k \rceil}$ is *of dimension* d iff either $Z = X^{\lceil d \rceil}$, or $Z = X^{[d]}$, with $X \in V$, and we denote it by $dim(Z) = d$. Define $V^{(d)} \stackrel{\text{def}}{=} \{Z \in V^{\lceil k \rceil} \mid dim(Z) = d\}$, for each $0 \le d \le k$.

▶ **Theorem 21** (from Theorem 3.3 in [16]). *Let G be a nonexpansive CFG with n variables. Then there exists $k \in \mathbb{N}$ with $k \le n$ such that every parse tree in $\mathcal{T}_G$ has dimension at most k.*

▶ **Lemma 22.** $[\![G]\!]_W = [\![G^{\lceil k \rceil}]\!]_{W^{\lceil k \rceil}}$.

Proof. See Appendix A in the extended version [7]. ◀

▶ **Lemma 23** (from Lemma 2.2 in [5]). *Let G be a CFG of degree m and let $\tau \in \mathcal{T}_G$ with $dim(\tau) \le k$ and $k \in \mathbb{N}$. Then there is a derivation sequence for $\mathcal{Y}(\tau)$ of index at most $km+1$.*

Now we define a derivation policy over at-most-k-dimension WCFGs. We will prove that this derivation policy satisfies Lemma 23 and thus the index of every derivation is bounded. We call these derivations *lowest-dimension-first (LDF) derivations.*

Intuitively, given a parse tree τ of an at-most-k-dimension WCFG, we define the LDF derivation sequence of τ by performing a depth-first traversal of τ where nodes in the same level of the tree are visited from lower to greater dimension and, if more than one node has the same dimension, then from left to right. Recall that the dimension of a node corresponds to the dimension of the parse tree that it roots.

Before giving a formal definition, we introduce the following notation. Given a derivation sequence $\psi = \alpha_0 \Rightarrow \dots \Rightarrow \alpha_n$ and β_0, β_1 (possibly empty) sequences of symbols and/or variables, we will denote by $\beta_0\, \psi\, \beta_1$ the derivation sequence $\beta_0\, \alpha_0\, \beta_1 \Rightarrow \dots \Rightarrow \beta_0\, \alpha_n\, \beta_1$.

▶ **Definition 24.** Let $G^{\lceil k \rceil}$ be an at-most-k-dimension CFG as in Definition 20. Let $\tau = \pi(\tau_1, \dots, \tau_n)$ be a parse tree of $G^{\lceil k \rceil}$. Define the *lowest-dimension-first (LDF) derivation sequence ψ of τ* inductively as follows:

- If $n = 0$, then π is of the form $\pi = X^{[0]} \to u_0$, and $\tau = \pi$. Then, the LDF derivation sequence of τ is:

$$\psi = X^{[0]} \Rightarrow_{ldf}^{\pi} u_0 \ .$$

- If $n \ge 1$, we distinguish the following cases:

1. If $\pi \in r_1$, i.e., π is of the form $\pi = X^{[d]} \to u_0 X_1^{[d]} u_1$ with $0 \leq d \leq k$, and $\tau = \pi(\tau_1)$. Then, the LDF derivation sequence of τ is:

$$\psi = X^{[d]} \Rightarrow^{\pi}_{ldf} u_0 X_1^{[d]} u_1 \Rightarrow_{ldf} u_0 \psi_1 u_1 \ ,$$

where ψ_1 is the LDF derivation sequence of τ_1.

2. If $\pi \in r_4$, i.e., π is of the form $\pi = X^{\lceil d \rceil} \to X^{[e]}$ with $0 \leq e \leq d \leq k$, and $\tau = \pi(\tau_1)$. Then, the LDF derivation sequence of τ is:

$$\psi = X^{\lceil d \rceil} \Rightarrow^{\pi}_{ldf} X^{[e]} \Rightarrow_{ldf} \psi_1 \ ,$$

where ψ_1 is the LDF derivation sequence of τ_1.

3. If $\pi \in r_2$, w.l.o.g., we assume that π is of the form:

$$\pi = X^{[d]} \to u_0 X_1^{[d]} u_1 X_2^{\lceil d-1 \rceil} u_2 \dots u_{n-2} X_{n-1}^{\lceil d-1 \rceil} u_{n-1} X_n^{\lceil d-1 \rceil} u_n \ ,$$

with $1 \leq d \leq k$, and $\tau = \pi(\tau_1, \dots, \tau_n)$. Define, for each $i \in \{2, \dots, n\}$, the derivation sequence $\tilde{\psi}_i$ as follows:

$$\begin{aligned} \tilde{\psi}_i &\stackrel{\text{def}}{=} u_0\, X_1^{[d]}\, u_1\, \mathcal{Y}(\tau_2)\, u_2\, \dots\, \mathcal{Y}(\tau_{i-1})\, u_{i-1}\, X_i^{\lceil d-1 \rceil}\, u_i\, \dots\, u_{n-1}\, X_n^{\lceil d-1 \rceil} u_n \\ &\Rightarrow^{*}_{ldf} u_0\, X_1^{[d]}\, u_1\, \mathcal{Y}(\tau_2)\, u_2\, \dots\, \mathcal{Y}(\tau_{i-1})\, u_{i-1}\, \psi_i\, u_i\, X_{i+1}^{\lceil d-1 \rceil}\, u_{i+1}\, \dots\, u_{n-1}\, X_n^{\lceil d-1 \rceil} u_n \ , \end{aligned}$$

where ψ_i is the LDF derivation sequence of τ_i. And define:

$$\begin{aligned} \tilde{\psi}_1 &\stackrel{\text{def}}{=} u_0\, X_1^{[d]}\, u_1\, \mathcal{Y}(\tau_2)\, u_2\, \dots\, u_{n-1}\, \mathcal{Y}(\tau_n)\, u_n \\ &\Rightarrow^{*}_{ldf} u_0\, \psi_1\, u_1\, \mathcal{Y}(\tau_2)\, u_2\, \dots\, u_{n-1}\, \mathcal{Y}(\tau_n)\, u_n \ , \end{aligned}$$

where ψ_1 is the LDF derivation sequence of τ_1. Then the LDF derivation ψ of τ is:

$$\psi = X^{[d]} \Rightarrow^{\pi}_{ldf} \tilde{\psi}_2 \Rightarrow_{ldf} \dots \Rightarrow_{ldf} \tilde{\psi}_n \Rightarrow_{ldf} \tilde{\psi}_1 \ .$$

4. If $\pi \in r_3$, w.l.o.g., we assume that π is of the form:

$$\pi = X^{[d]} \to u_0 X_1^{\lceil d-1 \rceil} u_1 X_2^{\lceil d-1 \rceil} u_2 \dots u_{n-2} X_{n-1}^{[d-1]} u_{n-1} X_n^{[d-1]} u_n \ ,$$

with $1 \leq d \leq k$, and $\tau = \pi(\tau_1, \dots, \tau_n)$. Define, for each $i \in \{1, \dots, n\}$, the derivation sequence $\tilde{\psi}_i$ as follows:

$$\begin{aligned} \tilde{\psi}_i &\stackrel{\text{def}}{=} u_0\, \mathcal{Y}(\tau_1)\, u_1\, \mathcal{Y}(\tau_2)\, u_2\, \dots\, \mathcal{Y}(\tau_{i-1})\, u_{i-1}\, X_i^{\lceil d-1 \rceil}\, u_i\, \dots\, u_{n-1}\, X_n^{[d-1]}\, u_n \\ &\Rightarrow^{*}_{ldf} u_0\, \mathcal{Y}(\tau_1)\, u_1\, \mathcal{Y}(\tau_2)\, u_2\, \dots\, \mathcal{Y}(\tau_{i-1})\, u_{i-1}\, \psi_i\, u_i\, X_{i+1}^{\lceil d-1 \rceil}\, u_{i+1}\, \dots\, u_{n-1}\, X_n^{[d-1]}\, u_n \ , \end{aligned}$$

where ψ_i is the LDF derivation sequence of τ_i. The the LDF derivation ψ of τ is:

$$\psi = X^{[d]} \Rightarrow_{ldf} \tilde{\psi}_1 \Rightarrow_{ldf} \dots \Rightarrow_{ldf} \tilde{\psi}_n \ .$$

Note that, given a parse tree τ of $G^{\lceil k \rceil}$, the LDF derivation sequence of τ is uniquely defined.

▶ **Lemma 25.** *Let $G^{\lceil k \rceil}$ be an at-most-k-dimension CFG of degree m and $\tau \in \mathcal{T}_{G^{\lceil k \rceil}}$ such that $dim(\tau) \leq k$. Then, the LDF derivation sequence of τ verifies $idx(\psi) \leq km + 1$.*

Proof. See Appendix A in the extended version [7]. ◀

Given a derivation sentence $\alpha \in \left(\Sigma \cup V^{\lceil k \rceil}\right)^*$ of an at-most-k-dimension CFG, define $\mathcal{LDF}(\alpha) \stackrel{\text{def}}{=} \alpha\downarrow_{\Sigma} \alpha\downarrow_{V^{(0)}} \alpha\downarrow_{V^{(1)}} \dots \alpha\downarrow_{V^{(k)}}$ and $\mathcal{LDF}_{V^{\lceil k \rceil}}(\alpha) \stackrel{\text{def}}{=} (\mathcal{LDF}(\alpha))\downarrow_{V^{\lceil k \rceil}}$. Now we show how to construct a regular (G_ℓ, W_ℓ) that is Parikh-equivalent to $\left(G^{\lceil k \rceil}, W^{\lceil k \rceil}\right)$.

▶ **Definition 26** (Regular WCFG for $\left(G^{\lceil k\rceil}, W^{\lceil k\rceil}\right)$)**.** Let $\left(G^{\lceil k\rceil}, W^{\lceil k\rceil}\right)$ be an at-most-k-dimension WCFG with $G^{\lceil k\rceil} = (V^{\lceil k\rceil}, \Sigma, S^{\lceil k\rceil}, R^{\lceil k\rceil})$ and degree m, and $W^{\lceil k\rceil}$ defined over the commutative semiring A. Define the WCFG (G_ℓ, W_ℓ) with $G_\ell = (V_\ell, \Sigma, S_\ell, R_\ell)$ as follows:

- Each variable in V_ℓ corresponds to a sequence $\alpha \in \left(V^{\lceil k\rceil}\right)^{km+1}$ where $\left(V^{\lceil k\rceil}\right)^{km+1}$ denotes the set $\{w \mid w \in \left(V^{\lceil k\rceil}\right)^*, |w| \leq km+1\}$, and we denote it by $\langle\alpha\rangle$. Formally,

$$V_\ell \stackrel{\text{def}}{=} \{\langle\alpha\rangle \mid \alpha \in \left(V^{\lceil k\rceil}\right)^{km+1}\} \ .$$

- The initial variable is defined as $S_\ell \stackrel{\text{def}}{=} S^{\lceil k\rceil}$.
- For each rule $\pi = (X \longrightarrow \gamma) \in R^{\lceil k\rceil}$ define $\pi^\alpha \stackrel{\text{def}}{=} (\langle X\,\alpha\rangle \longrightarrow \gamma{\downarrow_\Sigma}\,\langle\mathcal{LDF}_{V^{\lceil k\rceil}}(\gamma)\,\alpha\rangle)$. The set R_ℓ of rules is given by

$$\{\pi^\alpha \mid \pi = (X \longrightarrow \gamma) \in R^{\lceil k\rceil} \text{ and } \langle X\alpha\rangle, \langle\mathcal{LDF}_{V^{\lceil k\rceil}}(\gamma)\,\alpha\rangle \in V_\ell\} \ .$$

- The weight function W_ℓ is given by

$$W_\ell(\pi^\alpha) \stackrel{\text{def}}{=} W^{\lceil k\rceil}(\pi) \text{ for all } \pi^\alpha \in R_\ell \ .$$

▶ **Lemma 27.** $Pk[\![G^{\lceil k\rceil}]\!]_{W^{\lceil k\rceil}} = Pk[\![G_\ell]\!]_{W_\ell}$.

Proof. See Appendix A in the extended version [7]. ◀

B Counterexample in the unary case

▶ **Example 28.** The idea behind this example is to use the definition of (G_2, W_2) from Example 4 (a complete definition is given in Example 11) and replace each occurrence of the alphabet symbol $\overline{a}$ in the rules of (G_2, W_2) by a. Thus, define the WCFG (G, W) where $G = (\{X, \overline{D}, D, Y, Z\}, \{a\}, X, R)$, R is given by:

$$X \to D \mid \overline{D} \qquad \overline{D} \to D\,a\,Y \mid D\,a\,Z \qquad Z \to D\,a\,Z \mid D \ ,$$
$$D \to a\,D\,a\,D \mid \varepsilon \qquad Y \to a\,Y \mid \varepsilon$$

and the weight function W is defined over $(\mathbb{N}, +, \cdot, 0, 1)$ and assigns 1 to each production in the grammar except from the rule $Y \to a\,Y$ which is assigned weight 2. Notice that we preferred to assign weight 2 to the later rule instead of adding two copies each of weight 1. Recall that $Pk[\![G_2]\!]_{W_2} = (a + \overline{a})^*$. Now, relying on our construction of (G, W), we have that $Pk[\![G]\!]_W$ is the formal power series that results from replacing each $\overline{a}$ by a in the series $Pk[\![G_2]\!]_{W_2}$. Thus, we obtain that $Pk[\![G]\!]_W = (a + a)^* = (2a)^*$. The reader can check that the formal power series $(2a)^*$ corresponds to the Parikh image of the regular WCFG (G_ℓ, W_ℓ) where G_ℓ is defined as $G_\ell = (\{X\}, \{a\}, X, \{X \to aX,\ X \to \varepsilon\})$ and the weight function W_ℓ is defined over $(\mathbb{N}, +, \cdot, 0, 1)$ and assigns 2 to the rule $X \to aX$ and 1 to the rule $X \to \varepsilon$.

We refer to the extended version of this document [7] for an alternative proof that (G, W) satisfies the Parikh property using the decision procedure presented in Section 4.

Characterizing Demand Graphs for (Fixed-Parameter) Shallow-Light Steiner Network

Amy Babay
Johns Hopkins University, Baltimore, Maryland, USA
babay@cs.jhu.edu

Michael Dinitz
Johns Hopkins University, Baltimore, Maryland, USA
mdinitz@cs.jhu.edu

Zeyu Zhang
Johns Hopkins University, Baltimore, Maryland, USA
zyzhang92@gmail.com

Abstract

We consider the Shallow-Light Steiner Network problem from a fixed-parameter perspective. Given a graph G, a distance bound L, and p pairs of vertices $(s_1, t_1), \ldots, (s_p, t_p)$, the objective is to find a minimum-cost subgraph G' such that s_i and t_i have distance at most L in G' (for every $i \in [p]$). Our main result is on the fixed-parameter tractability of this problem for parameter p. We exactly characterize the demand structures that make the problem "easy", and give FPT algorithms for those cases. In all other cases, we show that the problem is W[1]-hard. We also extend our results to handle general edge lengths and costs, precisely characterizing which demands allow for good FPT approximation algorithms and which demands remain W[1]-hard even to approximate.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases fixed-parameter tractable, network design, shallow-light steiner network, demand graphs

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.33

Related Version A full version of the paper is available at [2], `https://arxiv.org/abs/1802.10566`.

Funding Supported in part by NSF award 1535887.

1 Introduction

In many network design problems we are given a graph $G = (V, E)$ and some demand pairs $(s_1, t_1), (s_2, t_2), \ldots, (s_p, t_p) \subseteq V \times V$, and are asked to find the "best" (usually minimum-cost) subgraph in which every demand pair satisfies some type of connectivity requirement. In the simplest case, if the demands are all pairs and the connectivity requirement is just to be connected, then this is the classical Minimum Spanning Tree problem. If we consider other classes of demands, then we get more difficult but still classical problems. Most notably, if the demands form a star (or any connected graph on V), then we have the famous Steiner Tree problem. If the demands are completely arbitrary, then we have the Steiner Forest problem. Both problems are known to be in FPT parameterized by the number of demands [9] (i.e., they can be solved in $f(p) \cdot poly(n)$ time for some function f).

There are many obvious generalizations of Steiner Tree and Steiner Forest of the general network design flavor given above. We will be particularly concerned with *length-bounded* variants, which are related to (but still quite different from) *directed* variants. In Directed Steiner Tree (DST) the input graph is directed and the demands are a directed star (either into or out of the root), while in Directed Steiner Network (DSN) the input graph and demands are both directed, but the demands are an arbitrary subset of $V \times V$. Both problems have been well-studied (e.g., [5, 19, 6, 8, 1]), and in particular it is known that the same basic dynamic programming algorithm used for Steiner Tree will also give an FPT algorithm for DST. However, DSN is known to be W[1]-hard, so it is not believed to be in FPT [11].

In the length-bounded setting, we typically assume that the input graph and demands are undirected but each demand has a distance bound, and a solution is only feasible if every demand is connected within distance at most the given bound (rather than just being connected). One of the most basic problems of this form is the Shallow-Light Steiner Tree problem (SLST), where the demands form a star with root $r = s_1 = s_2 = \cdots = s_p$ and there is a global length bound L (so in any feasible solution the distance from r to t_i is at most L for all $i \in [p]$). As with DST and DSN, SLST has been studied extensively [16, 18, 14, 13]. If we generalize this problem to arbitrary demands, we get the Shallow-Light Steiner Network problem, which is the main problem we study in this paper. Surprisingly, it has not received nearly the same amount of study (to the best of our knowledge, this paper is the first to consider it explicitly). It is formally defined as follows (note that we focus on the special case of unit lengths, and will consider general lengths in Section 5:

▶ **Definition 1** (Shallow-Light Steiner Network)**.** Given a graph $G = (V, E)$, a cost function $c : E \to \mathbb{R}^+$, a length function $l : E \to \mathbb{R}^+$, a distance bound L, and p pairs of vertices $\{s_1, t_1\}, \ldots, \{s_p, t_p\}$. The objective of SLSN is to find a minimum cost subgraph $G' = (V, S)$, such that for every $i \in [p]$, there is a path between s_i and t_i in G' with length less or equal to L.

Let H be the graph with $\{s_1, \ldots, s_p, t_1, \ldots, t_p\}$ as its vertex set and $\{\{s_1, t_1\}, \ldots, \{s_p, t_p\}\}$ as its edge set. We call H the *demand graph* of the problem. We use $|H|$ to represent the number of edges in H.

Both the directed and the length-bounded settings share a dichotomy between considering either star demands (DST/SLST) or totally general demands (DSN/SLSN). But this gives an obvious set of questions: what demand graphs make the problem "easy" (in FPT) and what demand graphs make the problem "hard" (W[1]-hard)? Recently, Feldmann and Marx [11] gave a complete characterization for this for DSN. Informally, they proved that if the demand graph is transitively equivalent to an "almost-caterpillar" (the union of a constant number of stars where their centers form a path, as well as a constant number of extra edges), then the problem is in FPT, and otherwise the problem is W[1]-hard.

While *a priori* there might not seem to be much of a relationship between the directed and the length-bounded problems, there are multiple folklore results that relate them, usually by means of some sort of layered graph. For example, any FPT algorithm for the DST problem can be turned into an FPT algorithm for SLST (with unit edge lengths) and vice versa through such a reduction (though this is a known result, to the best of our knowledge it has not been written down before, so we include it for completeness in Section 3.2). Such a relationship is not known for more general demands, though.

In light of these relationships between the directed and the length-bounded settings and the recent results of [11], it is natural to attempt to characterize the demand graphs that make SLSN easy or hard. We solve this problem, giving (as in [11]) a complete characterization of

easy and hard demand graphs. Our formal results are given in Section 2, but informally we show that SLSN is significantly harder than DSN: the only "easy" demand graphs are stars (in which case the problem is just SLST) and constant-size graphs. Even tiny modifications, like a star with a single independent edge, become W[1]-hard (despite being in FPT for DSN).

1.1 Connection to Overlay Routing

SLSN is particularly interesting due to its connection to overlay routing protocols that use *dissemination graphs* to support next-generation Internet services. In fact, our motivation for studying the fixed-parameter complexity of SLSN is from our use of heuristics for SLSN in a recent system [3] in which the number of demands was relatively small.

Many emerging applications (such as remote surgery) require extremely low-latency yet highly reliable communication, which the Internet does not natively support. Babay et al. [3] recently showed that such applications can be supported by using overlay networks to enable routing schemes based on *subgraphs* (dissemination graphs) rather than paths. Their extensive analysis of real-world data shows that two node-disjoint overlay paths effectively overcome any one fault in the middle of the network, but specialized dissemination graphs are needed to address problems at a flow's source or destination. Because problems affecting a source typically involve probabilistic loss on that source's outgoing links, a natural approach to increase the probability of a packet being successfully transmitted is to increase the number of outgoing links on which it is sent. In [3], when a problem is detected at a particular flow's source, that source switches to use a dissemination graph that floods its packets to all of its overlay neighbors and then forwards them from these neighbors to the destination. The paths from the source's neighbors to the destination must meet the application's strict latency requirement, but since the bandwidth used on every edge a packet traverses must be paid for, the total number of edges used should be minimized. Thus, constructing the optimal dissemination graph in this setting is precisely the Shallow-Light Steiner Tree problem, where the root of the demands is the destination and the other endpoints are the neighbors of the source.

However, in order to achieve the desired reliability, it was shown in [3] that *simultaneous* failures at both the source *and* the destination of a flow must also be addressed. Since it is not known in advance which neighbors of the source or destination will be reachable during a failure, the most resilient approach is to require a latency-bounded path from *every* neighbor of the source to *every* neighbor of the destination. This is precisely SLSN with a complete bipartite demand graph. Since no FPT algorithm for SLSN with complete bipartite demands was known, [3] relied on a heuristic that worked well in practice. The search for an FPT algorithm for SLSN with complete bipartite demands was the main motivation for this work.

In the context of dissemination-graph-construction problems, our results provide a good solution for problems affecting either a source or a destination: the FPT algorithm for the SLST problem is quite practical, since overlay topologies typically have bounded degree (and thus a bounded total number of demands). Unfortunately a trivial corollary of our main result implies that the other case which was particularly important in this setting, SLSN with complete bipartite demands, is W[1]-hard. This has important applications to future system design, since (like all hardness results) it will allow system designers to focus on issues other than perfect algorithms, even for dissemination graphs that provide only slightly more resiliency than SLST.

2 Our Results and Techniques

In order to distinguish the easy from the hard cases of the SLSN problem with respect to the demand graph, we should first define the problem with respect to a class (set) of demand graphs.

▶ **Definition 2.** Given a class $\mathcal{C}$ of graphs. The problem of Shallow-Light Steiner Network with restricted demand graph class $\mathcal{C}$ ($\text{SLSN}_{\mathcal{C}}$) is the SLSN problem with the additional restriction that the demand graph H of the problem must be isomorphic to some graph in $\mathcal{C}$.

We define $\mathcal{C}_\lambda$ as the class of all demand graphs with at most λ edges, and $\mathcal{C}^*$ as the class of all star demand graphs (there is a central vertex called the root, and every other vertex in the demand graph is adjacent to the root and only the root). Our main result is that these are *precisely* the easy classes: We first prove that SLSN is in XP parameterized by the number of demands (i.e. solvable in $n^{f(p)}$ time for some function f), which immediately implies that $\text{SLSN}_{\mathcal{C}_\lambda}$ can be solved in polynomial time if λ is a constant. Note that $\text{SLSN}_{\mathcal{C}^*}$ is precisely the SLST problem, for which a folklore FPT algorithm exists, thus $\text{SLSN}_{\mathcal{C}^*}$ (while NP-hard) is in FPT for parameter p. We also show that, for any other class $\mathcal{C}$ (i.e., any class which is not just a subset of $\mathcal{C}^* \cup \mathcal{C}_\lambda$ for some constant λ), the problem $\text{SLSN}_{\mathcal{C}}$ is W[1]-hard for parameter p. In other words, if the class of demand graphs includes arbitrarily large non-stars, then the problem is W[1]-hard parameterized by the number of demands.

More formally, we prove the following theorems.

▶ **Theorem 3.** *The unit-length arbitrary-cost* SLSN *problem with parameter p is in* XP*, and it can be solved in $n^{O(p^4)}$ time.*

By "unit-length arbitrary-cost" we mean that the length $l(e) = 1$ for all edges $e \in E$, while the cost c is arbitrary. To prove this theorem, we first prove a structural lemma which shows that the optimal solution must be the union of several lowest cost paths with restricted length (these paths may be between steiner nodes, but we show that there cannot be too many). Then we just need to guess all the endpoints of these paths, as well as all the lengths of these paths. We prove that there are only $n^{O(p^4)}$ possibilities, and the running time is also $n^{O(p^4)}$. The algorithm and proof is in Section 3.1.

▶ **Theorem 4.** *The unit-length arbitrary-cost* $\text{SLSN}_{\mathcal{C}^*}$ *problem is* FPT *for parameter p.*

As mentioned, $\text{SLSN}_{\mathcal{C}^*}$ is exactly the same as SLST, so we use a folklore reduction between SLST and DST in Section 3.2 to prove this theorem.

▶ **Theorem 5.** *If $\mathcal{C}$ is a recursively enumerable class, and $\mathcal{C} \nsubseteq \mathcal{C}_\lambda \cup \mathcal{C}^*$ for any constant λ, then* $\text{SLSN}_{\mathcal{C}}$ *is* W[1]*-hard for parameter p, even in the unit-length and unit-cost case.*

Many W[1]-hardness results for network design problems reduce from the Multi-Colored Clique (MCC) problem, and ours are no exception. We reduce from MCC to $\text{SLSN}_{\mathcal{C}'}$, where $\mathcal{C}'$ is a specific subset of $\mathcal{C}$ which has some particularly useful properties, and which we show must exist for any such $\mathcal{C}$. Since $\mathcal{C}' \subseteq \mathcal{C}$, this will imply the theorem. The reduction is in Section 4.2.

All of the above results are in the unit-length setting. We extend both our upper bounds and hardness results to handle arbitrary lengths, but with some extra complications. If $p = 1$ (there is only one demand), then with arbitrary lengths and arbitrary costs the SLSN problem is equivalent to the Restricted Shortest Path problem, which is known to be

NP-hard [15]. Therefore we can no longer hope for a polynomial time exact solution when $p = 1$, and thus cannot hope for an FPT algorithm (with parameter p). So we change our notion of "easy" from "solvable in FPT" to "arbitrarily approximable in FPT": we show $(1 + \varepsilon)$-approximation algorithms for the easy cases, and prove that there is no $\left(\frac{5}{4} - \varepsilon\right)$-approximation algorithm for the hard cases in $f(p) \cdot poly(n)$ time for any function f. We discuss these results in Section 5.

▶ **Theorem 6.** *For any constant $\lambda > 0$, there is a fully polynomial time approximation scheme (*FPTAS*) for the arbitrary-length arbitrary-cost* $\text{SLSN}_{\mathcal{C}_\lambda}$ *problem.*

▶ **Theorem 7.** *There is a $(1 + \epsilon)$-approximation algorithm in $O(4^p \cdot poly(\frac{n}{\varepsilon}))$ time for the arbitrary-length arbitrary-cost* $\text{SLSN}_{\mathcal{C}^*}$ *problem.*

For both upper bounds, we use basically the same algorithm as in the unit-length arbitrary-cost case, with some changes inspired by the $(1 + \varepsilon)$-approximation algorithm for the Restricted Shortest Path problem [17].

Our next theorem is analogous to Theorem 5, but since costs are allowed to be arbitrary we can prove stronger hardness of approximation (under stronger assumptions).

▶ **Theorem 8.** *Assume that (randomized) Gap-Exponential Time Hypothesis (Gap-ETH, see [4]) holds. Let $\varepsilon > 0$ be a small constant, and $\mathcal{C}$ be a recursively enumerable class where $\mathcal{C} \nsubseteq \mathcal{C}_\lambda \cup \mathcal{C}^*$ for any constant λ. Then, there is no $\left(\frac{5}{4} - \varepsilon\right)$-approximation algorithm in $f(p) \cdot n^{O(1)}$ time for* $\text{SLSN}_{\mathcal{C}}$ *for any function f, even in the unit-length and polynomial-cost case.*

Note that this theorem uses a much stronger assumption (Gap-ETH rather than W[1] $\neq$ FPT), which assumes that there is no (possibly randomized) algorithm running in $2^{o(n)}$ time that can distinguish whether a 3SAT formula is satisfiable or at most a $(1 - \varepsilon)$-fraction of its clauses can be satisfied. This enables us to utilize the hardness result for a generalized version of the MCC problem from [7], which will allow us to modify our reduction from Theorem 5 to get hardness of approximation.

2.1 Relationship to [11]

As mentioned, our results and techniques are strongly motivated and influenced by the work of Feldmann and Marx [11], who proved similar results in the directed setting. Informally, they showed that Directed Steiner Network is in FPT if the demand graph is transitively equivalent to an "almost-caterpillar", and otherwise it is W[1]-hard. Since "transitively equivalent to an almost-caterpillar" is a complex and subtle class, this showed that the tractability of DSN exhibits interesting behavior. Our results, on the other hand, show that SLSN is extraordinarily hard: there simply are not any algorithms possible for demand graphs that are even a little bit complex, despite the folklore relationships between directed settings and length-bounded settings. Thus our hardness proof is significantly more complicated than the reduction in [11], despite sharing some ideas.

The main case of the hardness reduction of [11] (which, like our reduction, is from MCC) is when the demand graph is a 2-by-k complete bipartite graph (i.e., two stars with the same leaf set). For this case, their reduction from MCC uses one star to control the choice of edges in the clique and another star to control the choice of vertices in the clique. They set this up so that if there is a clique of the right size then the "edge demands" and the "vertex demands" can be satisfied with low cost by making choices corresponding to the clique, while if no such clique exists then any way of satisfying the two types of demands simultaneously must have larger cost.

The 2-by-k complete bipartite graph is also a hard demand graph in our setting, and the same reduction from [11] can be straightforwardly modified to prove this (this appears as one of our cases). However, we prove that far simpler demand graphs are also hard. Most notably, the "main" case of our proof is when the demand graph is a single star together with one extra edge. Since we have only a single star in our demand graph, we cannot have two "types" of demands (vertex demands and edge demands) in our reduction. So we instead use the star to correspond to "edge demands" and use the single extra edge to simultaneously simulate all of the "vertex demands". This makes our reduction significantly more complicated.

With respect to upper bounds, the algorithm of [11] is quite complex in part due to the complexity of the demand graphs that it must solve. Our hardness results for SLSN imply that we need only concern ourselves with demand graphs that are star or have constant size. The star setting is relatively simple due to a reduction to DST, but it is not obvious how to use any adaptation of [11] (or the earlier [10]) to handle a constant number of demands for SLSN. Our algorithm ends up being relatively simple, but requires a structural lemma which was not necessary in the DSN setting.

3 Algorithms for Unit-Length Arbitrary-Cost SLSN

In this section we discuss the "easy" cases of SLSN. We first present an XP algorithm for SLSN in Section 3.1. In Section 3.2, we describe a reduction from SLSN with star demand graphs to DST, which gives an FPT algorithm.

3.1 The XP algorithm

The XP algorithm for Theorem 3 relies on the following structural lemma, which allows us to limit the structure of the optimal solution and finally find it out. Note that this lemma works not only for the unit-length case, but also for the arbitrary-length case.

▶ **Lemma 9.** *In any feasible solution $S \subseteq E$ of the* SLSN *problem, there exists a way to assign a path P_i between s_i and t_i in S for each demand $\{s_i, t_i\} \in H$ such that:*

- *For each $i \in [p]$, the total length of P_i is at most L and there is no cycle in P_i.*
- *For each $i, j \in [p]$ and $u, v \in P_i \cap P_j$, the paths between u and v in P_i and P_j are the same.*

Proof. We give a constructive proof. Let $m = |S|$ and $S = \{e_1, \ldots, e_m\}$. We first want to modify the lengths to ensure that there is always a unique shortest path. Let Δ denote the minimum length difference between any two subsets of S with different total length, i.e.,

$$\Delta = \min_{A,B \subseteq S, \sum_{e\in A} l(e) \neq \sum_{e\in B} l(e)} \left| \sum_{e\in A} l(e) - \sum_{e\in B} l(e) \right|.$$

We create a new length function g where $g(e_i) = l(e_i) + \Delta \cdot 2^{-i}$. Note that Δ is always non-zero for any S which has at least 2 edges, and the problem is trivial when $|S| = 1$.

We now show that any two paths have different lengths under g. Consider any two different paths P_x and P_y. If $\sum_{e\in P_x} l(e) \neq \sum_{e\in P_y} l(e)$, then without loss of generality we assume $\sum_{e\in P_x} l(e) < \sum_{e\in P_y} l(e)$. Then

$$\sum_{e\in P_x} g(e) \leq \sum_{e\in P_x} l(e) + \sum_{i=1}^{m} \Delta \cdot 2^{-i} < \sum_{e\in P_x} l(e) + \Delta \leq \sum_{e\in P_y} l(e) < \sum_{e\in P_y} g(e). \tag{1}$$

Algorithm 1 Unit-length arbitrary-cost SLSN.

```
Let M ← Σ_{e∈E} c(e) and S ← E
for Q ⊆ V where |Q| ≤ p(p − 1) do
    Q' ← Q ∪ (⋃_{i=1}^{p} {s_i, t_i})
    for E' ⊆ {{u, v} | u, v ∈ Q', u ≠ v} and l' : E' → [L] do
        T ← ∅
        for {u, v} ∈ E' do
            T ← T ∪ {the lowest cost path between u and v with length at most l'({u, v})}
              // if such path does not exist, T remains the same
        end for
        if T is a feasible solution and Σ_{e∈T} l'(e) < M then
            M ← Σ_{e∈T} c(e) and S ← T
        end if
    end for
end for
return S
```

Otherwise, if $\sum_{e\in P_x} l(e) = \sum_{e\in P_y} l(e)$, then

$$\sum_{e\in P_x} g(e) - \sum_{e\in P_y} g(e) = \sum_{i:e_i\in P_x} \Delta\cdot 2^{-i} - \sum_{i:e_i\in P_y} \Delta\cdot 2^{-i} \neq 0.$$

Therefore in both cases P_x and P_y have different lengths under g.

For each demand $\{s_i, t_i\} \in H$, we let P_i be the shortest path between s_i and t_i in S under the new length function g. Because any two paths under g have different length, the shortest path between each $\{s_i, t_i\} \in H$ is unique. In addition, because these are shortest paths and edge lengths are positive, they do not contain any cycles.

For each $i \in [p]$, we can see that P_i is also one of the shortest paths between s_i and t_i under original length function l. This is because in equation (1) we proved that a shorter path under length function l is still a shorter path under length function g. Since S is a feasible solution, the shortest path between s_i and t_i in S must have length at most L. Thus for each $i \in [p]$, we have $\sum_{e\in P_i} l(e) \leq L$.

For any two different paths P_i and P_j, let $u, v \in P_i \cap P_j$. If the subpath of P_i between u and v is different from the subpath of P_j between u and v, then by the uniqueness of shortest paths under g we know that either P_i or P_j is not a shortest path (since one of them could be improved by changing the subpath between u and v). This contradicts our definition of P_i and P_j, and hence they must use the same subpath between u and v. ◀

Lemma 9 implies that any two paths P_i, P_j in the optimal solution are either disjoint, or share exactly one (maximal) subpath. Since there are only p demands, the total number of shared subpaths is at most $\binom{p}{2}$. Therefore we can solve the unit-length arbitrary-cost $\text{SLSN}_{\mathcal{C}_\lambda}$ by guessing these subpaths.

Informally, we guess the set of endpoints of all the "maximal overlapping subpaths" (Q), guess how these endpoints are paired up to create the distinct subpaths (E'), guess the length of each subpath, and then find the lowest cost path that connects the endpoints of each guessed subpath and is within the guessed length. The full algorithm is given as Algorithm 1.

▶ **Claim 10.** *The running time of Algorithm 1 is* $n^{O(p^4)}$.

Proof. Clearly there are at most $n^{p(p-1)}$ possibilities for Q, and for each Q there are at most $2^{(p(p-1)+2p)^2}$ possible sets E' and at most $L^{(p(p-1)+2p)^2}$ possible l'. Since we assume unit edge lengths, we can use the Bellman-Ford algorithm to find the lowest cost path within a given length bound in polynomial time. Checking feasibility also takes polynomial time using standard shortest path algorithms. Thus, the running time is at most $n^{p(p-1)} \cdot 2^{(p(p+1))^2} \cdot n^{(p(p+1))^2} \cdot poly(n)$. ◀

Proof of Theorem 3

By Claim 10, the running time of Algorithm 1 is $n^{O(p^4)}$. Now we will prove correctness. The algorithm always returns a feasible solution, because we replace S by T only if T is feasible, and thus S is always a feasible solution. Therefore, we only need to show that this algorithm returns a solution with cost at most the cost of the optimal solution.

Let the optimal solution be S^*. We assign P_i^* for all $i \in [p]$ as in Lemma 9. Recall that path P_i^* and P_j^* can share at most one (maximal) subpath for each $i, j \in [p]$ where $i \neq j$. Let Q^* be the endpoint set of the (maximal) subpaths which are shared by some P_i^* and P_j^*, and let $Q'^* = Q^* \cup \bigcup_{i=1}^{p} \{s_i, t_i\}$.

We can see that the optimal solution S^* can be partitioned to a collection of paths by Q^*. We use E'^* to represent whether two vertices in Q'^* are "adjacent" on some path P_i^*: for any $u, v \in Q'^*$ where $u \neq v$, the set E'^* contains $\{u, v\}$ if and only if there exists $i \in [p]$ such that $u, v \in P_i^*$, and there is no vertex $w \in Q'^* \setminus \{u, v\}$ which is in the subpath between u and v in P_i^*. For each $\{u, v\} \in E'^*$, let $P^*_{\{u,v\}}$ be the subpath between u and v on path P_i^*. This is well defined because by Lemma 9 the subpath is unique. We define $l'^*(\{u, v\})$ as the length of $P^*_{\{u,v\}}$ for each $\{u, v\} \in E'^*$

Note that for any $\{u, v\} \neq \{u', v'\} \in E'^*$, we also know that $P^*_{\{u,v\}}$ and $P^*_{\{u',v'\}}$ are edge-disjoint. To see this, assume that they do share an edge, and let u'' and v'' be the endpoints of the (maximal) shared subpath between $P^*_{\{u,v\}}$ and $P^*_{\{u',v'\}}$. Then u'' and v'' are both in Q'^*, and at least one of them is in $Q'^* \setminus \{u, v\}$ or in $Q'^* \setminus \{u', v'\}$, which contradicts our definition of E'^*.

Since the algorithm iterates over all possibilities for Q, E' and l', there is some iteration in which $Q = Q'^*$, $E' = E'^*$, and $l' \equiv l'^*$. We will show that the algorithm also must find an optimal feasible solution in this iteration.

For each $i \in [p]$, the path P_i^* is partitioned to edge-disjoint subpaths by Q'^*. Let q_i be the number of subpaths, and let the endpoints be $s_i = v_{i,0}, v_{i,1}, \ldots, v_{i,q_i-1}, v_{i,q_i} = t_i$. We further let these subpaths be $P^*_{\{s_i,v_{i,1}\}}, P^*_{\{v_{i,1},v_{i,2}\}}, \ldots, P^*_{\{v_{i,q_i-1},t_i\}}$. By the definition of l'^*, for each $j \in [q_i]$, there must be a path between $v_{i,j-1}$ and $v_{i,j}$ with length at most $l'^*(\{v_{i,j-1}, v_{i,j}\})$ in graph G. Thus after the algorithm visited $\{v_{i,j-1}, v_{i,j}\} \in E'^*$, the edge set T must contains a path between u and v with length at most $l'^*(\{v_{i,j-1}, v_{i,j}\})$. Therefore we know that the edge set T in this iteration contains a path between s_i and t_i with length $\sum_{j=1}^{q_i} l'^*(\{v_{i,j-1}, v_{i,j}\}) \leq L$, and thus it is a feasible solution.

Let $MinCost(u, v, d)$ be the lowest cost for a path between u and v with distance at most d in graph G, then the total cost of this solution is $\sum_{\{u,v\} \in E'^*} MinCost(u, v, l'^*(\{u, v\}))$. Moreover, for each $\{u, v\} \in E'^*$ and $\{u', v'\} \in E'^*$ with $\{u, v\} \neq \{u', v'\}$, the paths $P^*_{\{u,v\}}$ and $P^*_{\{u',v'\}}$ are edge-disjoint, and each $P^*_{\{u,v\}}$ has cost at least $MinCost(u, v, l'^*(\{u, v\}))$. Thus the cost of the optimal solution is at least $\sum_{\{u,v\} \in E'^*} MinCost(u, v, l'^*(\{u, v\}))$, and so the algorithm outputs an optimal solution and it runs in polynomial time. ◀

▶ **Corollary 11.** *The arbitrary-length unit-cost* SLSN *problem with parameter p is in* XP.

Proof. We can use the same technique, but instead of guessing the length l' we guess the cost c', and then find shortest path under cost bound c'. We can also use Bellman-Ford algorithm in this step. ◀

3.2 Star Demand Graphs (SLSN$_{\mathcal{C}^*}$)

We prove Theorem 4 by reducing SLSN$_{\mathcal{C}^*}$ to DST, which has a known FPT algorithm [10]. This reduction is essentially folklore, but is included in Section 3.2 of the full paper [2] for completeness. This reduction transforms a unit-length arbitrary-cost SLSN$_{\mathcal{C}^*}$ instance $(G, c, l \equiv 1, \{\{s_1, t_1\}, \ldots, \{s_p, t_p\}\}, L)$ into a DST instance by creating a layered graph G' with $L+1$ layers. Each layer includes $|V|$ vertices (one for each vertex in G). Letting $v^{(i)}$ represent vertex v in layer i, each vertex $v^{(i-1)}$ (for $i \in [1, L]$) is connected to vertex $v^{(i)}$ with a 0-cost edge $(v^{(i-1)}, v^{(i)})$. Each such $v^{(i-1)}$ is also connected to each vertex $u^{(i)}$ such that $(v, u) \in E(G)$ by an edge $(v^{(i-1)}, u^{(i)})$ with cost $c(u, v)$. For the demands of the DST instance, we require the demand-source $s = s_1, \ldots s_p$ of the SLSN$_{\mathcal{C}^*}$ instance in layer 0 (i.e., $s^{(0)}$) to be connected to layer-L endpoints $t_1^{(L)}, \ldots t_p^{(L)}$, giving us an instance $(G', c', s^{(0)}, t_1^{(L)}, \ldots, t_p^{(L)})$ of DST. We solve this DST instance using the algorithm of [10] and construct a solution to the SLSN$_{\mathcal{C}^*}$ by including each edge (v, u) such that edge $(v^{(i-1)}, u^{(i)})$ for some layer i appears in the DST solution.

4 W[1]-Hardness for Unit-Length Unit-Cost SLSN

In this section we prove our main hardness result, Theorem 5. We begin with some preliminaries, then give our reduction and proof.

4.1 Preliminaries

We prove Theorem 5 by constructing an FPT reduction from the Multi-Colored Clique (MCC) problem to the unit-length unit-cost SLSN$_{\mathcal{C}}$ problem for any $\mathcal{C} \not\subseteq \mathcal{C}_\lambda \cup \mathcal{C}^*$. We begin with the MCC problem.

▶ **Definition 12** (Multi-Colored Clique)**.** Given a graph $G = (V, E)$, a number $k \in \mathbb{N}$ and a coloring function $c : V \to [k]$. The objective of the MCC problem is to determine whether there is a clique $T \subseteq V$ in G with $|T| = k$ where $c(x) \neq c(y)$ for all $x, y \in T$.

For each $i \in [k]$, we define $C_i = \{v \in V : c(v) = i\}$ to be the vertices of color i. We can assume that the graph does not contain edges where both endpoints have the same color, since those edges do not affect the existence of a multi-colored clique. It has been proven that the MCC problem is W[1]-complete.

▶ **Theorem 13** ([12])**.** *The* MCC *problem is* W[1]*-complete with parameter* k*.*

We first define a few important classes of graphs. These are the major classes that fall outside of $\mathcal{C}^* \cup \mathcal{C}_\lambda$, so we will need to be able to reduce MCC to SLSN where the demand graphs are in these classes, and then this will allow us to prove the hardness for general $C \not\subseteq \mathcal{C}^* \cup \mathcal{C}_\lambda$. For every $k \in \mathbb{N}$, we define the following graph classes. Each of the first four classes is just one graph up to isomorphism, but classes 5 and 6 are sets of graphs, so we use the notation $\mathcal{H}$ instead of H for these classes. Note that each of the first three classes is just a star with an additional edge, so we use * to make this clear.

1. $H^*_{k,0}$: a star with $k(k-1)$ leaves and an edge with both endpoints *not* in the star.
2. $H^*_{k,1}$: a star with $(k(k-1)+1)$ leaves and an edge $\{u,v\}$ where u is a leaf of the star and v is not in the star.
3. $H^*_{k,2}$: a star with $(k(k-1)+2)$ leaves, and an edge $\{u,v\}$ where both u and v are leaves of the star.
4. $H_{k,k}$: $k(k-1)+1$ edges where all endpoints are different (i.e., a matching of size $k(k-1)+1$).
5. $\mathcal{H}_{2,k}$: the class of graphs that have exactly $k(k-1)+2$ vertices, and contain a 2 by $k(k-1)$ complete bipartite subgraph (not necessarily an induced subgraph).
6. $\mathcal{H}_k$: the class of graphs that contain at least one of the graphs in previous five classes as an induced subgraph.

We first prove the following lemma.

▶ **Lemma 14.** *For any $k \geq 2$, if a graph H is not a star and H has at least $8k^{10}$ edges, then $H \in \mathcal{H}_k$, and we can find an induced subgraph which is isomorphic to a graph in $\{H^*_{k,0}, H^*_{k,1}, H^*_{k,2}, H_{k,k}\} \cup \mathcal{H}_{2,k} \cup \mathcal{H}_k$ in poly$(|H|)$ time.*

Proof. We give a constructive proof. We first claim that either there is a vertex in H which has degree at least $2k^4$ or there is an induced matching in H of size k^2. Suppose that all vertices have degree less than $2k^4$. Then we can create an induced matching by adding an arbitrary edge $\{u,v\} \in H$ to a edge set M, removing all vertices that are adjacent to either u or v from H, and repeating until there are no more edges in H. In each iteration we reduce the total number of edges by at most $2 \cdot 2k^4 \cdot 2k^4$, thus $|M| \geq \frac{8k^{10}}{8k^8} = k^2$. Since when we add an edge $\{u,v\}$ we also remove all vertices adjacent to u or v, every future edge we add to M will have endpoints which are not adjacent to u or v, and thus M is an induced matching of H with size k^2.

If H has an induced matching of size k^2, then $H \in \mathcal{H}_k$ because it contains $H_{k,k}$ as an induced subgraph, and thus we are done.

Otherwise, H has a vertex s with degree at least $2k^4$. Let S be the neighbors of s. If there is any vertex other than s that is adjacent to at least $k(k-1)$ vertices in S, then H contains a 2 by $k(k-1)$ complete bipartite subgraph, so it contains an induced subgraph $H' \in \mathcal{H}_{2,k}$ and thus is in $\mathcal{H}_k$.

So suppose that there is no vertex other than s that is adjacent to at least $k(k-1)$ vertices in S. Consider the case that there is no edge between any pair of vertices in S; then, because H is not a star, there must be an edge $\{u,v\} \in H$ with at least one of u,v not in $S \cup \{s\}$. Since both u and v are adjacent to at most $k(k-1)$ vertices in S, there are at least $k^4 - 2 \cdot k(k-1) \geq k(k-1)$ vertices in S that are not adjacent to either u or v. Let the set of these vertices be T. Then the induced subgraph on vertex set $T \cup \{s,u,v\}$ is either $H^*_{k,0}$ or $H^*_{k,1}$, depending on whether $\{u,v\} \cap T$ is an empty set.

Now the only remaining case is that there is at least one edge in H with both endpoints in S. In this case, we can find $H^*_{k,2}$ as an induced subgraph as follows: We first let $S_0 = S$. Then, in each iteration t we let v_t be a vertex in S_{t-1} that is adjacent to the fewest number of other vertices in S_{t-1}. We add v_t to the vertex set T, and then delete v_t and all the vertices in S_{t-1} that are adjacent to v_t to get S_t. This process repeats until we have $|T| = k(k-1)$.

We can use induction to show that, after each iteration $t \leq k(k-1)$, there is always at least one edge in H where both endpoints are in S_t. The base case is $t = 0$, where such an edge clearly exists. Assume the claim holds for iteration $t-1$, consider the iteration $t \leq k(k-1)$. If v_t is not adjacent to any other vertex in S_{t-1}, then removing v_t does not affect the fact that there is at least one edge left, and thus the claim still holds. Otherwise,

v_t is adjacent to at least one vertex in S_{t-1}. Thus, each vertex in S_{t-1} must be adjacent to at least one vertex in S_{t-1}. Since there is no vertex other than s which is adjacent to at least $k(k-1)$ vertices in S, we know that at most k^2 vertices are deleted in each iteration, and thus there are still at least $2k^4 - k^2 \cdot k(k-1) \geq k^4$ vertices in S_{t-1}. Because removing v_t and its neighbors can only affect the degree of at most $k^2(k-1)^2$ vertices in S_{t-1}, there must still be an edge left between the vertices in S_t.

Let $\{u, v\}$ be one of the edges in H where both endpoints are in S_t, then the induced subgraph on vertex set $T \cup \{s, u, v\}$ is $H^*_{k,2}$. Thus $H \in \mathcal{H}_k$.

It is easy to see that all the previous steps directly find an induced subgraph which is isomorphic to a graph in $\{H^*_{k,0}, H^*_{k,1}, H^*_{k,2}, H_{k,k}\} \cup \mathcal{H}_{2,k} \cup \mathcal{H}_k$ and takes polynomial time, thus the lemma is proved. ◀

4.2 Reduction

In this subsection, we will prove the following reduction theorem.

▶ **Theorem 15.** *Let $(G = (V, E), c)$ be an* MCC *instance with parameter k, and let $H \in \mathcal{H}_k$ be a demand graph. Then a unit-length unit-cost* SLSN *instance (G', L) with demand graph H can be constructed in poly$(|V||H|)$ time, and there exists a function g (computable in time poly$(|H|)$) such that the* MCC *instance has a clique with size k if and only if the* SLSN *instance has a solution with cost $g(H)$.*

In order to prove this theorem, we first introduce a construction for any demand graph $H \in \{H^*_{k,0}, H^*_{k,1}, H^*_{k,2}, H_{k,k}\} \cup \mathcal{H}_{2,k}$, and then use the instances constructed in these cases to construct the instance for general $H \in \mathcal{H}_k$.

The construction for $H \in \mathcal{H}_{2,k}$ is similar to [11], which proves the W[1]-hardness of the DSN problem. We change all the directed edges in their construction to undirected, and add some edges and dummy vertices. This construction is presented in Appendix A.3. To handle $H^*_{k,0}$, $H^*_{k,1}$, $H^*_{k,2}$, and $H_{k,k}$, we need to change this basic construction due to the simplicity of the demand graphs. Because the constructions for these four graphs are quite similar, we first introduce the construction for $H^*_{k,0}$ in Section 4.2.1, and then show how to modify it for $H^*_{k,1}$, $H^*_{k,2}$, and $H_{k,k}$ in Appendix A.2.

4.2.1 Case 1: $H^*_{k,0}$

Given an MCC instance $(G = (V, E), c)$ with parameter k, we create a unit-length and unit-cost SLSN instance (G', L) with demand graph $H^*_{k,0}$ as follows.

We first create a graph G^*_k with integer edge lengths (we will later replace all non-unit length edges by paths). See Figure 1 for an overview of this graph. The vertex set V^*_k contains 6 layers of vertices and another group of vertices. The first layer V_1 is just a root r. The second layer V_2 contains a vertex $z_{\{i,j\}}$ for each $1 \leq i < j \leq k$, so there are $\binom{k}{2}$ vertices. The third layer V_3 contains a vertex z_e for each $e \in E$, so there are $|E|$ vertices. The fourth layer V_4 contains a vertex $x_{v,j}$ for each $v \in V$ and $j \in [k]$ with $j \neq c(v)$, so there are $|V| \cdot (k-1)$ vertices. The fifth layer V_5 again contains a vertex $x'_{v,j}$ for each $v \in V$ and $j \in [k]$ with $j \neq c(v)$. The sixth layer V_6 contains a vertex $l_{i,j}$ for each $i, j \in [k]$ where $i \neq j$, so there are $k(k-1)$ vertices. Finally, we have a vertex y_i for $i = 0, \ldots, k$, so there are $k+1$ vertices in the set V_y.

Let $f_i : \mathbb{N} \to \mathbb{N}$ be the function defined by $f_i(j) = j + 1$ if $j + 1 \neq i$ and $f_i(j) = j + 2$ if $j + 1 = i$. This function gives the next integer after j, but skips i. Let $f^t_i(j) = f_i(f_i(\ldots f_i(j)))$

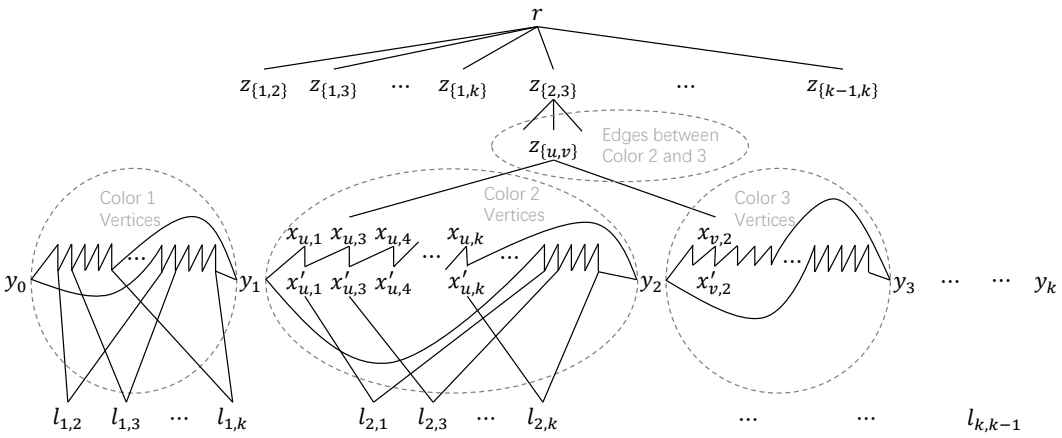


Figure 1 G^*_k.

denote this function repeated t times. Recall that $C_i = \{v \in V : c(v) = i\}$. The edge set E^*_k contains following edges, with lengths as indicated:

- $E_1 = \{\{r, z_{\{i,j\}}\} \mid 1 \le i < j \le k\}$, each edge in E_1 has length 2.
- $E_2 = \{\{z_{\{c(u),c(v)\}}, z_e\} \mid e = \{u, v\} \in E\}$, each edge in E_2 has length 1.
- $E_3 = \{\{z_e, x_{u,c(v)}\} \mid e = \{u, v\} \in E\}$, each edge in E_3 has length $2k^2 - 2$. Note that if $\{z_e, x_{u,c(v)}\} \in E_3$, then $\{z_e, x_{v,c(u)}\} \in E_3$
- $E_4 = \{\{x_{v,j}, x'_{v,j}\} \mid v \in V, j \ne c(v)\}$, each edge in E_4 has length 1.
- $E_5 = \{\{x'_{v,j}, l_{c(v),j}\} \mid v \in V, j \ne c(v)\}$, each edge in E_5 has length $2k^2 - 2$.
- $E_{yx} = \{\{y_{i-1}, x_{v,f_i(0)}\} \mid i \in [k], v \in C_i\}$, each edge in E_{yx} has length 4.
- $E_{xx} = \{\{x'_{v,j}, x_{v,f_{c(v)}(j)}\} \mid v \in V, j \in [k] \setminus \{c(v), f^{k-1}_{c(v)}(0)\}\}$, each edge in E_{xx} has length 3.
- $E_{xy} = \{\{x'_{v,f^{k-1}_i(0)}, y_i\} \mid i \in [k], v \in C_i\}$, each edge in E_{xy} has length 3.

Let G' be the graph obtained from G^*_k by replacing each edge $e \in E^*_k$ by a $length(e)$-hop path. We create an instance of SLSN on G' by setting the demands to be $\{r, l_{i,j}\}$ for all $i, j \in [k]$ where $i \ne j$, as well as $\{y_0, y_k\}$. Note that these demands form a star with $k(k-1)$ leaves and an edge with both endpoints not in the star, so it is isomorphic to $H^*_{k,0}$. We set the distance bound L to be $4k^2$.

This construction clearly takes $poly(|V||H^*_{k,0}|)$ time. Let $g(H^*_{k,0}) = 4k^4 - 4k^3 + \frac{3}{2}k^2 + \frac{5}{2}k$, which is clearly computable in $poly(H^*_{k,0})$ time. We will first prove the easy direction in the correctness of the construction.

▶ **Lemma 16.** *If there is a multi-colored clique of size k in G, then there is a solution S for the* SLSN *instance (G', L) with demand graph $H^*_{k,0}$, and the total cost of S is $g(H^*_{k,0})$.*

Proof. Let $v_1, \ldots, v_k$ be a multi-colored clique of size k in G, where $v_i \in C_i$ for all $i \in [k]$. We create a feasible solution S to our SLSN instance, which contains following paths in G' (i.e., edges in G^*_k):

- $\{r, z_{\{i,j\}}\}$ for each $1 \le i < j \le k$. The total cost of these edges is $2 \cdot \binom{k}{2} = k^2 - k$.
- $\{z_{\{i,j\}}, z_{\{v_i,v_j\}}\}$ for each $1 \le i < j \le k$. The total cost of these edges is $\binom{k}{2} = \frac{k^2-k}{2}$.
- $\{z_{\{v_i,v_j\}}, x_{v_i,j}\}$ and $\{z_{\{v_i,v_j\}}, x_{v_j,i}\}$ for each $1 \le i < j \le k$. The total cost of these edges is $2 \cdot (2k^2 - 2) \cdot \binom{k}{2} = 2k^4 - 2k^3 - 2k^2 + 2k$.
- $\{x_{v_i,j}, x'_{v_i,j}\}$ for each $i, j \in [k]$ where $i \ne j$. The total cost of these edges is $2 \cdot \binom{k}{2} = k^2 - k$.
- $\{x'_{v_i,j}, l_{i,j}\}$ for each $i, j \in [k]$ where $i \ne j$. The total cost of these edges is $2 \cdot (2k^2 - 2) \cdot \binom{k}{2} = 2k^4 - 2k^3 - 2k^2 + 2k$.

- $\{y_{i-1}, x_{v_i,f_i(0)}\}$ for each $i \in [k]$. The total cost of these edges is $4k$.
- $\{x'_{v_i,j}, x_{v_i,f_i(j)}\}$ for each $i \in [k]$ and $j \in [k] \setminus \{i, f_i^{k-1}(0)\}$. The total cost of these edges is $3 \cdot k(k-2) = 3k^2 - 6k$.
- $\{x'_{v_i,f_i^{k-1}(0)}, y_i\}$ for each $i \in [k]$. The total cost of these edges is $3k$.

Therefore, the total cost is $k^2 - k + \frac{k^2-k}{2} + 2k^4 - 2k^3 - 2k^2 + 2k + k^2 - k + 2k^4 - 2k^3 - 2k^2 + 2k + 4k + 3k^2 - 6k + 3k = 4k^4 - 4k^3 + \frac{3}{2}k^2 + \frac{5}{2}k = g(H^*_{k,0})$.

Now we show the feasibility of this solution. For each $i, j \in [k]$ where $i \neq j$, the path between r and $l_{i,j}$ is $r - z_{\{i,j\}} - z_{\{v_i,v_j\}} - x_{v_i,j} - x'_{v_i,j} - l_{i,j}$. The length of this path is $2 + 1 + 2k^2 - 2 + 1 + 2k^2 - 2 = 4k^2$, thus it is a feasible path.

The path between y_0 and y_k is $y_0 - x_{v_1,2} - x'_{v_1,2} - x_{v_1,3} - x'_{v_1,3} - \ldots - x_{v_1,k} - x'_{v_1,k} - y_1 - x_{v_2,1} - x'_{v_2,1} - x_{v_2,3} - x'_{v_2,3} - \ldots - y_2 - \ldots - y_k$. The length of this path is $(4 + 1 \cdot (k-1) + 3 \cdot (k-2) + 3) \cdot k = 4k^2$, thus it is a feasible path. ◀

For the other direction, we begin the proof with a few claims. We first show that the only feasible way to connect r and $l_{i,j}$ is to pick one edge between every two adjacent layers. We can also see in Figure 1 that for each $i \in [k]$, there are $|C_i|$ disjoint "zig-zag" paths between y_{i-1} and y_i, and each path corresponds to a vertex with color i. We will also show that the only feasible way to connect y_0 and y_k is to pick one zig-zag path between each y_{i-1} and y_i. The proof of these claims are in Appendix A.1. From these claims we can then prove that, if the cost of the optimal solution is at most $g(H^*_{k,0})$, then there is a multi-colored clique in G.

▶ **Claim 17.** *For all $i, j \in [k]$ where $i \neq j$, any path $P_{i,j}$ between r and $l_{i,j}$ with length at most $4k^2$ must be of the form $r - z_{\{i,j\}} - z_{\{u,v\}} - x_{u,j} - x'_{u,j} - l_{i,j}$, where $u \in C_i$, $v \in C_j$ and $\{u, v\} \in E$.*

▶ **Claim 18.** *Any path P_y between y_0 and y_k with length at most $4k^2$ can be divided to k subpaths as follows. For each $i \in [k]$, there is a subpath P_{v_i} between y_{i-1} and y_i with length $4k$, of the form $y_{i-1} - x_{v_i,f_i(0)} - x'_{v_i,f_i(0)} - x_{v_i,f_i^2(0)} - x'_{v_i,f_i^2(0)} - \cdots - x_{v_i,f_i^{k-1}(0)} - x'_{v_i,f_i^{k-1}(0)} - y_i$, where $v_i \in C_i$.*

Now, we can prove the other direction in the correctness of the construction.

▶ **Lemma 19.** *Let S be an optimal solution for the* SLSN *instance (G', L) with demand graph $H^*_{k,0}$. If S has cost at most $g(H^*_{k,0}) = 4k^4 - 4k^3 + \frac{3}{2}k^2 + \frac{5}{2}k$, then there is a multi-colored clique of size k in G.*

Proof. For each $i, j \in [k]$ with $i \neq j$, let $P_{i,j}$ be a (arbitrarily chosen) path in S which connects r and $l_{i,j}$ with length at most $L = 4k^2$. Let $\mathcal{P} = \{P_{i,j} \mid i, j \in [k], i \neq j\}$ be the set of all these paths. We also let P_y be a (arbitrary) path in S of length at most L which connects y_0 and y_k.

From Claim 18, P_y can be divided to k subpaths, each of which corresponds to a vertex v_i. We will show that $v_1, \ldots, v_k$ form a clique in G (i.e., for each $1 \leq i < j \leq k$, we have $\{v_i, v_j\} \in E$).

We first prove that these paths must share certain edges due to the cost bound of the optimal solution. From Claim 17, we know that each $P_{i,j}$ costs exactly $2 + 1 + 2k^2 - 2 + 1 + 2k^2 - 2 = 4k^2$. In addition, from the form of $P_{i,j}$ we can also see that these paths are almost disjoint, except that $P_{i,j}$ and $P_{j,i}$ may share a length 2 edge $\{r, z_{\{i,j\}}\} \in E_1$ and a length 1 edge $\{z_{\{i,j\}}, z_e\} \in E_2$. Therefore, in order to satisfy the demands between r and all of the $l_{i,j}$'s, the total cost of the edges in $S \cap (E_1 \cup E_2 \cup E_3 \cup E_4 \cup E_5)$ is at least

$4k^2 \cdot k(k-1) - \binom{k}{2} \cdot (2+1) = 4k^4 - 4k^3 - \frac{3}{2}k^2 + \frac{3}{2}k$, even if every $P_{i,j}$ and $P_{j,i}$ do share edge $\{r, z_{\{i,j\}}\}$ and edge $\{z_{\{i,j\}}, z_e\}$.

We now calculate the cost of the edges in $S \cap (E_{yx} \cup E_{xx} \cup E_{xy})$. From Claim 18, the total cost of edges in $P_y \cap (E_{yx} \cup E_{xx} \cup E_{xy})$ is at least $(4+3\cdot(k-1)+3)\cdot k = 3k^2+k$. Thus, the total cost is already at least $\left(4k^4 - 4k^3 - \frac{3}{2}k^2 + \frac{3}{2}k\right) + (3k^2+k) = 4k^4 - 4k^3 + \frac{3}{2}k^2 + \frac{5}{2}k = g(H^*_{k,0})$, so S cannot contain any edge which has not been counted yet.

Therefore, every edge in $P_y \cap E_4$ must appear in some path in $\mathcal{P}$. In fact, by the form of the paths in $\mathcal{P}$, we can see that for each $i, j \in [k]$ where $i \neq j$, the edge $\{x_{v_i,j}, x'_{v_i,j}\} \in P_y \cap E_4$ can only appear in path $P_{i,j}$, rather than any other $P_{i',j'}$. Thus $x_{v_i,j}$ is in path $P_{i,j}$, and similarly $x_{v_j,i}$ is in path $P_{j,i}$. Recall that $P_{i,j}$ and $P_{j,i}$ must share an edge $\{z_{\{i,j\}}, z_e\}$ for some $e \in E$ because of the cost bound, and $z_{\{v_i,v_j\}}$ is the only vertex which adjacent to both $x_{v_i,j}$ and $x_{v_j,i}$, we can see that e can only be $\{v_i, v_j\}$. Therefore $\{v_i, v_j\} \in E$, which proves the lemma. ◀

4.2.2 Cases 2: $H^*_{k,1}$, 3: $H^*_{k,2}$, 4: $H_{k,k}$, and 5: $\mathcal{H}_{2,k}$

For Cases 2, 3, and 4, we can use essentially the same reduction as in Case 1. For Case 2, we just need to add a new demand $\{r, y_0\}$, and do some extra analysis to show that adding this demand does not change anything. For Case 3, we similarly add another demand $\{r, y_k\}$. Case 4 requires only adding another layer of vertices and edges before the root r. The details are in Appendix A.2. Case 5 is a variant of the reduction in [11], and we prove this case in Appendix A.3.

4.2.3 Case 6: $\mathcal{H}_k$

We now want to construct an SLSN instance for a demand graph $H \in \mathcal{H}_k$ from an MCC instance $(G = (V, E), c)$ with parameter k; since all other cases have been handled, this will complete the proof of Theorem 15. By the definition of $\mathcal{H}_k$, for some $t \in [5]$ there is a graph $H^{(t)}$ of Case t that is an induced subgraph of H. We use Lemma 14 to find the graph $H^{(t)}$. Let $(G^{(t)}, L)$ be the SLSN instance obtained by applying our reduction for Case t to the MCC instance (G, c), and let the corresponding function be $g^{(t)}$.

We now want to transform the SLSN instance $(G^{(t)}, L)$ with demand graph $H^{(t)}$ into a new SLSN instance (G', L) with demand graph H, so that instance $(G^{(t)}, L, H^{(t)})$ has a solution with cost $g^{(t)}(H^{(t)})$ if and only if instance (G', L, H) has a solution with cost $g(H) = g^{(t)}(H^{(t)}) + L \cdot (|H| - |H^{(t)}|)$. If there is such a construction which runs in polynomial time, then there is a multi-colored clique of size k in G if and only if instance (G', L, H) has a solution with cost $g(H)$. This will then imply Theorem 15.

The graph G' is basically just $G^{(t)}$ with some additional vertices and edges from $H \setminus H^{(t)}$. For each vertex v in H but not in $H^{(t)}$, we add a new vertex v to G'. For each edge $\{u, v\} \in H \setminus H^{(t)}$, we add an L-hop path between u and v to G'.

The construction still takes $poly(|V||H|)$ time, because the construction for the previous cases takes $poly(|V||H^{(t)}|)$ time and the construction for Case 6 takes $poly(|G^{(t)}||H|)$ time. Here $|H^{(t)}| \leq |H|$, and we know that $|G^{(t)}|$ is polynomial in $|V|$ and $|H^{(t)}|$.

▶ **Lemma 20.** SLSN *instance* $(G^{(t)}, L, H^{(t)})$ *has a solution with cost* $g^{(t)}(H^{(t)})$ *if and only if instance* (G', L, H) *has a solution with cost* $g(H) = g^{(t)}(H^{(t)}) + L \cdot (|H| - |H^{(t)}|)$.

Proof. If instance $(G^{(t)}, L, H^{(t)})$ has a solution with cost $g^{(t)}(H^{(t)})$. Let the solution be $S^{(t)}$. For each $e = \{u, v\} \in H \setminus H^{(t)}$, let the new L-hop path between u and v in G' be P_e. Then $S^{(t)} \cup \bigcup_{e \in H \setminus H^{(t)}} P_e$ is a solution to G' with cost $g^{(t)}(H^{(t)}) + L \cdot (|H| - |H^{(t)}|)$.

If instance (G', L, H) has a solution with cost $g^{(t)}(H^{(t)}) + L \cdot (|H| - |H^{(t)}|)$, let the solution be S. Since for each $e = \{u, v\} \in H \setminus H^{(t)}$, the only path between u and v in G' within the length bound is the new L-hop path P_e, any valid solution must include all these P_e, which has total cost $L \cdot (|H| - |H^{(t)}|)$. In addition, for each demand $\{u, v\}$ which is also in H, any path between u and v in G' within the length bound will not include any new edge, because otherwise it will strictly contain an L-hop path, and have length more than L. Therefore, $S \setminus \bigcup_{e \in H \setminus H^{(t)}} P_e$ is a solution to $G^{(t)}$ with cost $g^{(t)}(H^{(t)})$. ◀

4.3 Proof of Theorem 5

If $\mathcal{C}$ is a recursively enumerable class, and $\mathcal{C} \nsubseteq \mathcal{C}_\lambda \cup \mathcal{C}^*$ for any constant λ, then for every $k \geq 2$, let H_k be the first graph in $\mathcal{C}$ where H_k is not a star and has at least $2k^{10}$ edges. The time for finding H_k is $f(k)$ for some function f. From Lemma 14 we know that $H_k \in \mathcal{H}_k$, so that we can use Theorem 15 to construct the SLSN$_\mathcal{C}$ instance with demand H_k.

The parameter $p = |H_k|$ of the instance is a function just of k, and the construction time is FPT from Theorem 15. Therefore this is a FPT reduction from the MCC problem to the unit-length unit-cost SLSN$_\mathcal{C}$ problem. Thus Theorem 13 implies that the unit-length unit-cost SLSN$_\mathcal{C}$ problem is W[1]-hard for parameter p. ◀

5 Overview of General Length and Cost Settings

As discussed in Section 2, we extended our results from the unit-length setting to the general length setting. We defer all detailed results to Section 5 and 6 of the full paper [2], and instead give only a brief overview of our results and techniques.

5.1 Upper bounds

Recall that we cannot have an exact FPT algorithm for SLSN$_{\mathcal{C}^*}$ and SLSN$_{\mathcal{C}_\lambda}$ since even if there is only a single demand the problem becomes the Restricted Shortest Path problem, which is known to be NP-hard [15]. But since Restricted Shortest Path admits an FPTAS [15, 17], it is natural to instead try to give a $(1+\varepsilon)$-approximation algorithm for both problems. We show that with some modifications of the algorithms in the unit-length case, we can give an FPTAS for arbitrary-length arbitrary-cost SLSN$_{\mathcal{C}_\lambda}$, and can give a $(1+\varepsilon)$-approximation algorithm in FPT time for arbitrary-length arbitrary-cost SLSN$_{\mathcal{C}^*}$.

For SLSN$_{\mathcal{C}_\lambda}$, Lemma 9 still holds, so we can still guess how the paths in the solution intersect with each other and what the endpoints of maximum shared subpaths are. However, we cannot guess the length of a subpath in this setting, since there are too many possibilities. We instead guess the cost of all the subpaths. Because we are aiming to find an approximation solution, we are allowed to have $(1+\varepsilon)$ error on the cost of each subpath, so this allows us to reduce the search space. However, this is still not enough: if the space of the possible values is too large, then $\log_{1+\varepsilon}$ of it is still too large. So we then use an additional procedure from [17] which gives valid upper bound U and lower bound L on the optimal solution such that $U/L \leq n^2$. This sufficiently decreases the space of possible guesses so that we get a $(1+\varepsilon)$-approximation in polynomial time. The full algorithm and analysis are in Section 5.2 of the full paper [2].

For the star demand graph, we cannot do the same reduction as in Section 3.2 because with arbitrary lengths the natural layered graph used in the reduction to DSN can have exponential layers. However, similar to Steiner Tree and DST, one can prove that the

optimal solution for $\text{SLSN}_{\mathcal{C}^*}$ is always a tree. Therefore we look at the original FPT algorithm for Steiner Tree and DST and attempt to modify it to work in our setting. Given a star demand graph where the center is s and the leaf set is T, both algorithms use dynamic programming to solve the subproblems $f(v, R)$, which are to find the minimum cost tree with root $v \in V$ that contains $R \subseteq T$, starting from $|R| = 1$ to $|R| = |T|$. The base case when $|R| = 1$ is essentially a shortest path algorithm. Then we can build up larger trees since a tree with more than two leaves can always be partitioned to two subtrees and a path from the root.

We use a similar approach, first discretizing the possible costs to be powers of $(1 + \varepsilon)$. We define the subproblem $d(v, j, R)$ to be the smallest height of a tree (with the given edge lengths) such that the root is v, the total cost is at most j, and it contains all vertices in R. Then, we find the smallest j for which $d(s, j, T)$ is at most the length bound L, and this j is actually a good approximation to the optimal solution. The full algorithm and analysis are in Section 5.3 of the full paper [2].

5.2 Lower bounds

For the lower bound on $\text{SLSN}_{\mathcal{C}}$ with $\mathcal{C} \not\subseteq (\mathcal{C}_\lambda \cup \mathcal{C}^*)$, the same reduction as in Section 4 already shows that it is W[1]-hard to obtain a $\left(1 + \frac{1}{O(p^2)}\right)$-approximation. However, we would like a stronger hardness of approximation, one which would rule out good approximations (like we gave for $\text{SLSN}_{\mathcal{C}^*}$ and $\text{SLSN}_{\mathcal{C}_\lambda}$) even for large p. With some modifications of the cost of some edges in the instance constructed in Section 4, and a stronger assumption of Gap-ETH, we can show that there is no $(\frac{5}{4} - \varepsilon)$-approximation for $\text{SLSN}_{\mathcal{C}}$ which runs in FPT time, even for the unit-length polynomial-cost setting.

Consider the reduction in Section 4. We showed that if there is a low-cost solution to the SLSN instance that we created, then the paths satisfying the demands must share some specific edges with each other, and the existence of these edges implies the existence of a clique in the given MCC instance. For the polynomial-cost setting, we reduce from a different problem known as the Multi-Colored Densest k-Subgraph, which is a gap version of the MCC instance. Under the assumption of Gap-ETH, a corollary of [7] shows that for any constant $0 < \varepsilon < 1$, no FPT algorithm can distinguish between the case that there is a multi-colored k-clique and the case that every subgraph with k vertices has at most $\varepsilon \cdot \binom{k}{2}$ edges. By modifying the cost of some edges and making a slightly delicate inclusion-exclusion argument, we can show that if the cost of the SLSN solution is not too large then many edges still need to be shared by different paths, which ensures that a subgraph with k vertices and $\varepsilon \cdot \binom{k}{2}$ edges must exist. The entire reduction and the correctness proof is in Section 6 of the full paper [2].

References

1 Amir Abboud and Greg Bodwin. Reachability Preservers: New Extremal Bounds and Approximation Algorithms. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 1865–1883, 2018.

2 Amy Babay, Michael Dinitz, and Zeyu Zhang. Characterizing Demand Graphs for (Fixed-Parameter) Shallow-Light Steiner Network. *arXiv preprint arXiv:1802.10566*, 2018.

3 Amy Babay, Emily Wagner, Michael Dinitz, and Yair Amir. Timely, Reliable, and Cost-Effective Internet Transport Service Using Dissemination Graphs. In *37th IEEE International Conference on Distributed Computing Systems, ICDCS 2017*, pages 1–12, 2017.

4 Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From gap-ETH to FPT-inapproximability: Clique, dominating set, and more. In *Foundations of Computer Science (FOCS), 2017 IEEE 58th Annual Symposium on*, pages 743–754. IEEE, 2017.

5 Moses Charikar, Chandra Chekuri, To-yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed Steiner problems. *Journal of Algorithms*, 33(1):73–91, 1999.

6 Chandra Chekuri, Guy Even, Anupam Gupta, and Danny Segev. Set connectivity problems in undirected graphs and the directed steiner network problem. *ACM Transactions on Algorithms (TALG)*, 7(2):18, 2011.

7 Rajesh Chitnis, Andreas Emil Feldmann, and Pasin Manurangsi. Parameterized Approximation Algorithms for Directed Steiner Network Problems. *arXiv preprint arXiv:1707.06499*, 2017.

8 Eden Chlamtác, Michael Dinitz, Guy Kortsarz, and Bundit Laekhanukit. Approximating Spanners and Directed Steiner Forest: Upper and Lower Bounds. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 534–553, 2017.

9 Stuart E Dreyfus and Robert A Wagner. The Steiner problem in graphs. *Networks*, 1(3):195–207, 1971.

10 Jon Feldman and Matthias Ruhl. The directed Steiner network problem is tractable for a constant number of terminals. *SIAM Journal on Computing*, 36(2):543–561, 2006.

11 Andreas Emil Feldmann and Dániel Marx. The Complexity Landscape of Fixed-Parameter Directed Steiner Network Problems. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*, volume 55 of *LIPIcs*, pages 27:1–27:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

12 Michael R Fellows, Danny Hermelin, Frances Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science*, 410(1):53–61, 2009.

13 Longkun Guo, Kewen Liao, and Hong Shen. On the shallow-light steiner tree problem. In *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2014 15th International Conference on*, pages 56–60. IEEE, 2014.

14 Mohammad Taghi Hajiaghayi, Guy Kortsarz, and Mohammad R Salavatipour. Approximating buy-at-bulk and shallow-light k-Steiner trees. *Algorithmica*, 53(1):89–103, 2009.

15 Refael Hassin. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations research*, 17(1):36–42, 1992.

16 Guy Kortsarz and David Peleg. Approximating shallow-light trees. Technical report, Association for Computing Machinery, New York, NY (United States), 1997.

17 Dean H Lorenz and Danny Raz. A simple efficient approximation scheme for the restricted shortest path problem. *Operations Research Letters*, 28(5):213–219, 2001.

18 Joseph Naor and Baruch Schieber. Improved approximations for shallow-light spanning trees. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 536–541. IEEE, 1997.

19 Leonid Zosin and Samir Khuller. On directed Steiner trees. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 59–63. Society for Industrial and Applied Mathematics, 2002.

A Proofs in Section 4

A.1 Proof of Claims in Case 1

A.1.1 Proof of Claim 17

We can see that G_k^* is a 6-layer graph with a few additional paths between the fourth layer and the fifth layer. Thus $P_{i,j}$ must contain at least one edge between each two adjacent layers. From the construction of G_k^*, all the edges between two adjacent layers have the same length. If we sum up the length from r to the fourth layer plus the length from the fifth layer to $l_{i,j}$, it is already $2+1+2k^2-2+2k^2-2=4k^2-1$. Thus, between the fourth layer and the fifth layer we can only choose one length 1 edge.

We know that the vertex in the fifth layer must adjacent to $l_{i,j}$, so it must be $x'_{u,j}$ for some $u \in C_i$. Thus, the edge between the fourth layer and the fifth layer must be $\{x_{u,j}, x'_{u,j}\}$, because this is the only length 1 edge adjacent to $x'_{u,j}$. In addition, the only way to go from r to $x_{u,j}$ with one edge per layer is to pass through vertex $z_{\{i,j\}}$ and $z_{\{u,v\}}$ for some $v \in C_j$ and $\{u,v\} \in E$. Therefore $P_{i,j}$ must correspond to an edge $\{u,v\} \in E$ where $u \in C_i$ and $v \in C_j$, and it has form $r - z_{\{i,j\}} - z_{\{u,v\}} - x_{u,j} - x'_{u,j} - l_{i,j}$. ◀

A.1.2 Proof of Claim 18

For the path connecting y_0 and y_k, we first prove another claim.

▶ **Claim 21.** *Any path P_y between y_0 and y_k with length at most $4k^2$ does not contain any edge in $E_1 \cup E_2 \cup E_3 \cup E_5$.*

Proof. We prove the claim by contradiction. If P_y contains an edge in $E_1 \cup E_2 \cup E_3 \cup E_5$, it must contain at least two edges with length $2k^2-2$ (one edge to go out of the fourth and the fifth layer, and another one to go back). Since any edge which has endpoint y_0 has length 4 and any edge which has endpoint y_k has length 3, the total length $2\cdot(2k^2-2)+4+3 = 4k^2+3$ already exceeds the length bound $4k^2$, giving a contradiction. ◀

Since we have Claim 21, it suffices to consider the edge set $E_4 \cup E_{yx} \cup E_{xx} \cup E_{xy}$. We can see that $E_4 \cup E_{yx} \cup E_{xx} \cup E_{xy}$ can be partitioned to $k|V|$ paths, where for each $i \in [k]$ and each $v \in C_i$, there is a path P_v which connects y_{i-1} and y_i with length $4k$. The path is $y_{i-1} - x_{v,f_i(0)} - x'_{v,f_i(0)} - x_{v,f_i^2(0)} - x'_{v,f_i^2(0)} - \ldots - x_{v,f_i^{k-1}(0)} - x'_{v,f_i^{k-1}(0)} - y_1$. We can see that these paths are vertex disjoint except for the endpoints $y_0, y_1, \ldots, y_k$.

Therefore, the only way to go from y_0 to y_k is by passing through $y_0, y_1, \ldots, y_k$ one-by-one. Thus, for each $i \in [k]$, P_y must contain a subpath P_{v_i} where $v_i \in C_i$. Because each of these subpaths has length $4k$, the total cost is already $4k \cdot k = 4k^2$, which is exactly the length bound. Therefore, P_y can not contain any other edge, which proves the lemma. ◀

A.2 Case 2, 3, and 4

Cases 2, 3, and 4 are basically the same as Case 1, so we discuss them in the same subsection.

Case 2: $H_{k,1}^*$

We use the same G_k^*, G', and L in the construction of the SLSN instance for demand graph $H_{k,0}^*$, and also set $g(H_{k,1}^*) = 4k^4 - 4k^3 + \frac{3}{2}k^2 + \frac{5}{2}k$. The only difference is the demand graph. Besides the demand of $\{r, l_{i,j}\}$ for all $i,j \in [k]$ where $i \neq j$, and $\{y_0, y_k\}$, there is a new

demand $\{r, y_0\}$. Clearly this new demand graph is a star with $(k(k-1)+1)$ leaves, and an edge in which exactly one of the endpoints is a leaf of the star, so it is isomorphic to $H^*_{k,1}$.

Assume there is a multi-colored clique of size k in G. The paths connecting previous demands in the solution of the SLSN instance are the same as Case 1. The path between r and y_0 is $r - z_{\{1,2\}} - z_{\{v_1,v_2\}} - x_{v_1,2} - y_0$. All the edges in this path are already in the previous paths, so the cost remains the same. The length of this path is $2+1+2k^2-2+4 = 2k^2+5 < 4k^2$, which satisfies the length bound.

Assume there is a solution for the SLSN instance $(G', L, H^*_{k,1})$ with cost $4k^4 - 4k^3 + \frac{3}{2}k^2 + \frac{5}{2}k$. The proof that there exists a multi-colored clique of size k in G is the same as Case 1.

Case 3: $H^*_{k,2}$

As in Case 2, only the demand graph changes. The new demand graph is the same as in Case 2 but again with a new demand $\{r, y_k\}$. Since $\{r, y_0\}$ was already a demand, our new demand graph is a star with $(k(k-1)+2)$ leaves (the $l_{i,j}$'s and y_0 and y_k), and an edge between two of its leaves (y_0 and y_k), which is isomorphic to $H^*_{k,2}$.

Assume there is a multi-colored clique of size k in G. The paths connecting previous demands in the solution of the SLSN instance are the same as Case 2. The path between r and y_k is $r - z_{\{k-1,k\}} - z_{\{v_{k-1},v_k\}} - x_{v_k,k-1} - y_k$. All the edges in this path are already in the previous paths, so the cost stays the same. The length of this path is $2 + 1 + 2k^2 - 2 + 4 = 2k^2 + 5 < 4k^2$, which satisfies the length bound.

Assume there is a solution for the SLSN instance $(G', L, H^*_{k,2})$ with cost $4k^4 - 4k^3 + \frac{3}{2}k^2 + \frac{5}{2}k$. The proof that there exists a multi-colored clique of size k in G is the same as Case 1.

Case 4: $H_{k,k}$

In order to get $H_{k,k}$ as our demand graph, we have to slightly change the construction from Case 1. We still first make a weighted graph $G_{k,k} = (V_{k,k}, E_{k,k})$ and then transform it to the unit-length unit-cost graph G'. For the vertex set $V_{k,k}$, we add another layer of vertices $V_0 = \{l'_{i,j} \mid i, j \in [k], i \neq j\}$ to V^*_k before the first layer V_1. For the edge set $E_{k,k}$, we include all the edges in E^*_k, but change the length of edges in E_1 to length 1. We also add another edge set $E_0 = \{\{l'_{i,j}, r\} \mid i, j \in [k], i \neq j\}$. Each edge in E_0 has length 1.

The demands are $\{l'_{i,j}, l_{i,j}\}$ for each $i, j \in [k]$ where $i \neq j$, as well as $\{y_0, y_k\}$. This is a matching of size $k(k-1)+1$, which is isomorphic to $H_{k,k}$. We still set the length bound to be $L = 4k^2$, and set $g(H_{k,k}) = 4k^4 - 4k^3 + 2k^2 + 2k$.

If there is a multi-colored clique of size k in G, the construction for the solution in G' is similar to Case 1. For each $i, j \in [k]$ where $i \neq j$, the path between $l'_{i,j}$ and $l_{i,j}$ becomes $l'_{i,j} - r - z_{\{i,j\}} - z_{\{v_i,v_j\}} - x_{v_i,j} - x'_{v_i,j} - l_{i,j}$ (i.e., one more layer before the root r). It is easy to see that the length bound and size bound are still satisfied.

Assume there is a solution for the SLSN instance $(G', L, H_{k,k})$ with cost $4k^4-4k^3+2k^2+2k$. The proof that there exists a multi-colored clique of size k in G is essentially the same as Case 1, except the path between $l'_{i,j}$ and $l_{i,j}$ has one more layer.

A.3 Case 5: $\mathcal{H}_{2,k}$

In this case, we slightly modify the reduction of [11]. We first change all the edges from directed to undirected. In addition, in [11] the demand graph is precisely a 2-by-$k(k-1)$ bipartite graph, but we also handle the generalization in which there may be more demands

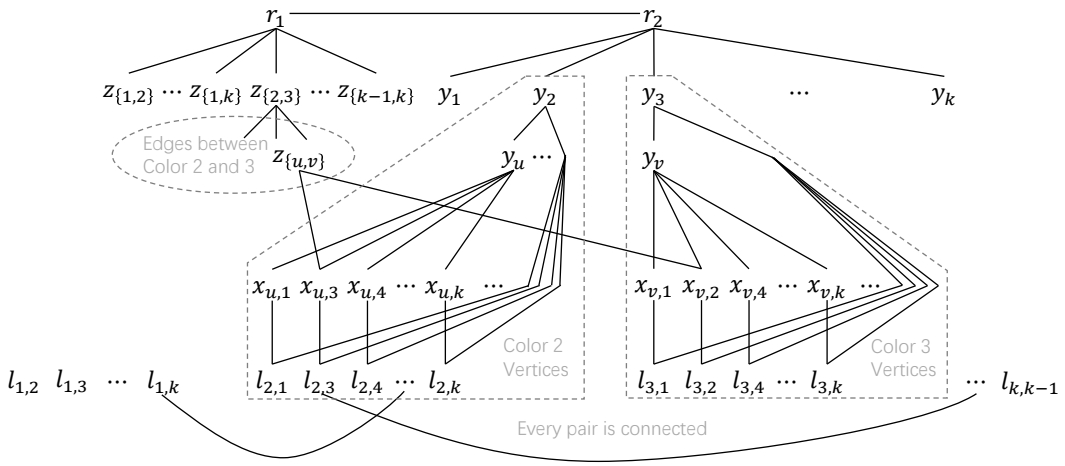


Figure 2 $G_{2,k}$.

between vertices on each sides (i.e., the 2-by-$k(k-1)$ bipartite graph is just a subgraph of our demands). In order to do this, we add some dummy vertices and some edges.

Given an MCC instance $(G = (V, E), c)$ with parameter k, and a demand graph $H \in \mathcal{H}_{2,k}$, we create a unit-length and unit-cost SLSN instance G' with demand isomorphic to H as follows.

We first create a weighted graph $G_{2,k} = (V_{2,k}, E_{2,k})$. The vertex set $V_{2,k}$ contains 5 layers of vertices. The first layer V_1 is just two roots r_1, r_2. The second layer V_2 contains a vertex $z_{\{i,j\}}$ for each $1 \le i < j \le k$, and a vertex y_i for each $i \in [k]$. The third layer V_3 contains a vertex z_e for each $e \in E$, and a vertex y_v for each $v \in V$. The fourth layer V_4 contains a vertex $x_{v,j}$ for each $v \in V$ and $j \ne c(v)$. The fifth layer V_5 contains a vertex $l_{i,j}$ for each $i, j \in [k]$ where $i \ne j$.

The edge set $E_{2,k}$ contains the following edges:

- $E_{11} = \{\{r_1, z_{\{i,j\}}\}, 1 \le i < j \le k\}$, each edge in E_{11} has length 1.
- $E_{12} = \{\{z_{\{c(u),c(v)\}}, z_e\} \mid e = \{u, v\} \in E\}$, each edge in E_{12} has length 1.
- $E_{13} = \{\{z_e, x_{u,c(v)}\} \mid e = \{u, v\} \in E\}$, each edge in E_{13} has length 1. Note that if $\{z_e, x_{u,c(v)}\} \in E_{13}$, then $\{z_e, x_{v,c(u)}\} \in E_{13}$
- $E_{21} = \{\{r_2, y_i\} \mid i \in [k]\}$, each edge in E_{21} has length 1.
- $E_{22} = \{\{y_{c(v)}, y_v\} \mid v \in V\}$, each edge in E_{22} has length 1.
- $E_{23} = \{\{y_v, x_{v,j}\} \mid v \in V, j \ne c(v)\}$, each edge in E_{23} has length 1.
- $E_{xl} = \{\{x_{v,j}, l_{c(v),j}\} \mid v \in V, j \ne c(v)\}$, each edge in E_{xl} has length 4.
- $E_{ll} = \{\{l_{i,j}, l_{i',j'}\} \mid i, j, i', j' \in [k], i \ne j, i' \ne j', (i, j) \ne (i', j')\}$, each edge in E_{ll} has length 7.

We get a unit-length graph G' from $G_{2,k}$ by replacing every edge $e \in E_{2,k}$ by a $length(e)$-hop path. Our SLSN instance consists of the graph G', length bound $L = 7$, and the following demands (which will be isomorphic to H). For each $r \in \{r_1, r_2\}$ and $i, j \in [k]$ with $i \ne j$, there is a demand between r and $l_{i,j}$ (note that these demands form a 2 by $k(k-1)$ complete bipartite graph. Let this complete bipartite subgraph be B. For the rest of the demands, we arbitrarily choose a mapping between $V_1 = \{r_1, r_2\}$ and the 2-side of the bipartite graph in H, as well as a mapping between $V_5 = \{l_{i,j} \mid i, j \in [k], i \ne j\}$ and the $k(k-1)$-side. There is a demand between two vertices $u, v \in V_1 \cup V_5$ if there is an edge between u, v in H.

This construction clearly takes $poly(|V||H|)$ time. Let $g(H) = 7|H| - 7k^2 + 9k - 7 \cdot \mathbb{1}_{\{r_1,r_2\}\in H}$, where $\mathbb{1}_{\{r_1,r_2\}\in H}$ is an indicator variable for $\{r_1, r_2\}$ being a demand in H. This function is also computable in time $poly(|H|)$. We first prove the easy direction in the correctness of the reduction.

▶ **Lemma 22.** *If there is a multi-colored clique of size k in G, then there is a solution S for the* SLSN *instance (G', L) with demand graph $H \in \mathcal{H}_{2,k}$, and the total cost of S is $7|H| - 7k^2 + 9k - 7 \cdot \mathbb{1}_{\{r_1,r_2\}\in H}$.*

Proof. Let $v_1, \ldots, v_k$ be a multi-colored clique of size k in G, where $v_i \in C_i$ for all $i \in [k]$. We create a feasible solution S to our SLSN instance, which contains following paths in G' (i.e., edges in $G_{2,k}$):

- $\{r_1, z_{\{i,j\}}\}$ for each $1 \leq i < j \leq k$. The total cost of these edges is $\binom{k}{2} = \frac{k^2-k}{2}$.
- $\{z_{\{i,j\}}, z_{\{v_i,v_j\}}\}$ for each $1 \leq i < j \leq k$. The total cost of these edges is $\binom{k}{2} = \frac{k^2-k}{2}$.
- $\{z_{\{v_i,v_j\}}, x_{v_i,j}\}$ and $\{z_{\{v_i,v_j\}}, x_{v_j,i}\}$ for each $1 \leq i < j \leq k$. The total cost of these edges is $2 \cdot \binom{k}{2} = k^2 - k$.
- $\{r_2, y_i\}$ for each $i \in [k]$. The total cost of these edges is k.
- $\{y_i, y_{v_i}\}$ for each $i \in [k]$. The total cost of these edges is k.
- $\{y_{v_i}, x_{v_i,j}\}$ for each $i, j \in [k]$ where $i \neq j$. The total cost of these edges is $2 \cdot \binom{k}{2} = k^2 - k$.
- $\{x_{v_i,j}, l_{i,j}\}$ for each $i, j \in [k]$ where $i \neq j$. The total cost of these edges is $4 \cdot 2 \cdot \binom{k}{2} = 4k^2 - 4k$.
- $\{u, v\}$ for each $\{u, v\} \in H \setminus (B \cup \{\{r_1, r_2\}\})$. The total cost of these edges is $7 \cdot (|H| - 2 \cdot k(k-1) - \mathbb{1}_{\{r_1,r_2\}\in H}) = 7|H| - 14k^2 + 14k - 7 \cdot \mathbb{1}_{\{r_1,r_2\}\in H}$.

Therefore, the total cost is $\frac{k^2-k}{2} + \frac{k^2-k}{2} + k^2 - k + k + k + k^2 - k + 4k^2 - 4k + 7|H| - 14k^2 + 14k - 7 \cdot \mathbb{1}_{\{r_1,r_2\}\in H} = 7|H| - 7k^2 + 9k - 7 \cdot \mathbb{1}_{\{r_1,r_2\}\in H}$.

Now we show the feasibility of this solution. For each $i, j \in [k]$ where $i \neq j$, the path between r_1 and $l_{i,j}$ is $r_1 - z_{\{i,j\}} - z_{\{v_i,v_j\}} - x_{v_i,j} - l_{i,j}$, and the path between r_2 and $l_{i,j}$ is $r_2 - y_i - y_{v_i} - x_{v_i,j} - l_{i,j}$. Both paths have length 7, which is within the length bound. For each $\{u, v\} \in H \setminus (B \cup \{\{r_1, r_2\}\})$, u and v have an edge with length 7, thus a path under the length bound exists. Finally, if there exists a demand between r_1 and r_2, we can follow the path $r_1 - z_{\{1,2\}} - z_{\{v_1,v_2\}} - x_{v_1,2} - y_{v_1} - y_1 - r_2$, which has length 6. ◀

Now we prove the other direction.

Let S be an optimal solution for the SLSN instance (G', L) with demand graph $H^*_{k,0}$. If S has cost at most $4k^4 - 4k^3 + \frac{3}{2}k^2 + \frac{5}{2}k$, then there is a multi-colored clique of size k in G.

▶ **Lemma 23.** *Let S be an optimal solution for the* SLSN *instance (G', L) with demand graph $H \in \mathcal{H}_{2,k}$. If S has cost at most $7|H| - 7k^2 + 9k - 7 \cdot \mathbb{1}_{\{r_1,r_2\}\in H}$, then there is a multi-colored clique of size k in G.*

Proof. For each $i, j \in [k]$ where $i \neq j$, let $P_{1,i,j} \subseteq S$ be a (arbitrarily chosen) path between r_1 and $l_{i,j}$ with length at most 7, and $P_{2,i,j} \subseteq S$ be a (arbitrarily chosen) path between r_2 and $l_{i,j}$ with length at most 7. Let $\mathcal{P}_1 = \{P_{1,i,j} \mid i, j \in [k], i \neq j\}$, and $\mathcal{P}_2 = \{P_{2,i,j} \mid i, j \in [k], i \neq j\}$. As in lemma 19, we first show that some edges must be shared by multiple paths by calculating the total cost.

In order to satisfy the demand for each $\{l_{i,j}, l_{i',j'}\} \in H \setminus (B \cup \{\{r_1, r_2\}\})$, the only way is to use the edge between $l_{i,j}$ and $l_{i',j'}$ in E_{ll}. Otherwise, suppose the path has more than one edge, since the only edges incident on any $l_{i,j}$ have length either 4 or 7, the cost of two of these edges already exceeds the length bound. Thus the total cost of the edges in $S \cap E_{ll}$ is at least $7|H| - 7|B| - 7 \cdot \mathbb{1}_{\{r_1,r_2\}\in H} = 7|H| - 14k^2 + 14k - 7 \cdot \mathbb{1}_{\{r_1,r_2\}\in H}$.

We can see that each of the paths in $\mathcal{P}_1 \cup \mathcal{P}_2$ must have exactly one edge between every two adjacent levels, and they cannot have any other edges because of the length bound. Thus, each path $P_{1,i,j} \in \mathcal{P}_1$ must have form $r_1 - z_{\{i,j\}} - z_{\{u,v\}} - x_{u,j} - l_{i,j}$ for some $\{u,v\} \in E$ with $u \in C_i$ and $v \in C_j$, and each path in $P_{2,i,j} \in \mathcal{P}_2$ must have form $r_2 - y_i - y_v - x_{v,j} - l_{i,j}$ for some $v \in C_i$.

By looking at the form of paths in $\mathcal{P}_1$, we can see that these paths are almost disjoint, except that $P_{1,i,j}$ and $P_{1,j,i}$ may share edge $\{r_1, z_{\{i,j\}}\} \in E_{11}$ and edge $\{z_{\{i,j\}}, z_e\} \in E_{12}$. Since paths in $\mathcal{P}_1$ only contain edges in $E_{11} \cup E_{12} \cup E_{13} \cup E_{xl}$, the cost of edges in $S \cap (E_{11} \cup E_{12} \cup E_{13} \cup E_{xl})$ must be at least $7 \cdot k(k-1) - \binom{k}{2} - \binom{k}{2} = 6k^2 - 6k$, even if every $P_{1,i,j}$ and $P_{1,j,i}$ do share edge $\{r_1, z_{\{i,j\}}\}$ and edge $\{z_{\{i,j\}}, z_e\}$.

We then look at the form of paths in $\mathcal{P}_2$. We can see that the first 3 hops of these paths only contain edges in $E_{21} \cup E_{22} \cup E_{23}$. In addition, these paths are all disjoint on edges in E_{23}. Moreover, in order to reach all $l_{i,j}$ from r_2 within length 7, these paths should contain all edges in E_{21} and at least k edges in E_{22}. Therefore, the total cost of edges in $S \cap (E_{21} \cup E_{22} \cup E_{23})$ should be at least $k(k-1) + k + k = k^2 + k$.

By summing up all these edges, the total cost of edges in S is already at least $7|H| - 14k^2 + 14k - 7 \cdot \mathbb{1}_{\{r_1,r_2\} \in H} + 6k^2 - 6k + k^2 + k = 7|H| - 7k^2 + 9k - 7 \cdot \mathbb{1}_{\{r_1,r_2\} \in H} = g(H)$, which means S cannot contain any edge that has not been counted before.

Therefore, S must contain exactly k edges in E_{22}, and each of these edges must have a different y_i as an endpoint. We let these edges be $\{y_1, y_{v_1}\}, \ldots, \{y_k, y_{v_k}\}$, where $v_i \in C_i$ for all $i \in [k]$. We claim that $v_1, \ldots, v_k$ forms a (multicolored) clique in G.

For each $1 \leq i < j \leq k$, by looking at the form of paths in $\mathcal{P}_2$, we know that the path $P_{2,i,j}$ must be $r_2 - y_i - y_{v_i} - x_{v_i,j} - l_{i,j}$. Because of the total cost limitation, the edge $\{x_{v_i,j}, l_{i,j}\} \in P_{2,i,j} \cap E_{xl}$ must also appear in some path in $\mathcal{P}_1$. By looking at the form of the paths in $\mathcal{P}_1$, the only possible path is $P_{1,i,j}$. Similarly, path $P_{2,j,i}$ must share edge $\{x_{v_j,i}, l_{j,i}\}$ with $P_{1,j,i}$. Again by looking at the form of the paths in $\mathcal{P}_1$, the edge in $\{z_{\{i,j\}}, z_e\} \in S \cap E_{12}$ which is shared by $P_{1,i,j}$ and $P_{1,j,i}$ must have $e = \{v_i, v_j\}$, which means $\{v_i, v_j\} \in E$.

Therefore, $v_1, \ldots, v_k$ forms a clique in G. ◀

On the Parameterized Complexity of $[1, j]$-Domination Problems

Mohsen Alambardar Meybodi
Department of Computer Science, Yazd University, Yazd, Iran
m.alambardar@stu.yazd.ac.ir

Fedor Fomin
Department of Informatics, University of Bergen, Norway
fomin@ii.uib.no

Amer E. Mouawad
Computer Science Department, American University of Beirut, Beirut, Lebanon
aa368@aub.edu.lb

Fahad Panolan
Department of Informatics, University of Bergen, Norway
fahad.panolan@ii.uib.no

Abstract

For a graph G, a set $D \subseteq V(G)$ is called a $[1, j]$-dominating set if every vertex in $V(G) \setminus D$ has at least one and at most j neighbors in D. A set $D \subseteq V(G)$ is called a $[1, j]$-total dominating set if every vertex in $V(G)$ has at least one and at most j neighbors in D. In the $[1, j]$-(Total) Dominating Set problem we are given a graph G and a positive integer k. The objective is to test whether there exists a $[1, j]$-(total) dominating set of size at most k. The $[1, j]$-Dominating Set problem is known to be NP-complete, even for restricted classes of graphs such as chordal and planar graphs, but polynomial-time solvable on split graphs. The $[1, 2]$-Total Dominating Set problem is known to be NP-complete, even for bipartite graphs. As both problems generalize the Dominating Set problem, both are W[1]-hard when parameterized by solution size. In this work, we study $[1, j]$-Dominating Set on sparse graph classes from the perspective of parameterized complexity and prove the following results when the problem is parameterized by solution size:

- $[1, j]$-Dominating Set is W[1]-hard on d-degenerate graphs for $d = j + 1$;
- $[1, j]$-Dominating Set is FPT on nowhere dense graphs.

We also prove that the known algorithm for $[1, j]$-Dominating Set on split graphs is optimal under the Strong Exponential Time Hypothesis (SETH). Finally, assuming SETH, we provide a lower bound for the running time of any algorithm solving the $[1, 2]$-Total Dominating Set problem parameterized by pathwidth.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms, Theory of computation → Design and analysis of algorithms

Keywords and phrases $[1, j]$-dominating set, parameterized complexity, sparse graphs

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.34

1 Introduction

A dominating set of a graph G is a subset D of vertices such that each vertex in $V(G) \setminus D$ is adjacent to at least one vertex in D. Various extensions of domination, such as independent, total, efficient, and perfect domination, have been introduced and widely studied both combinatorially and algorithmically. A discussion of these extensions can be found in [14]. A

38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018).
Editors: Sumit Ganguly and Paritosh Pandya; Article No. 34; pp. 34:1–34:14

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

$[1, j]$-dominating set, as first defined in [3], is a set that dominates the vertices of the graph but every vertex outside of the set must have at most j neighbors in it. In [3], it was shown that a minimum $[1, 2]$-dominating set and a minimum dominating set are of the same size in several classes of graphs such as claw-free graphs, P_4-free graphs, and caterpillars. It was also shown that the problem is NP-complete, even for bipartite graphs. In the $[1, j]$-Dominating Set problem the input is a graph G and a positive integer k. The objective is to test whether there is a $[1, j]$-dominating set of size at most k. The authors in [3] raised several open problems, including whether restricting to specific classes of graphs leads to strictly better upper bounds for the size of $[1, j]$-dominating sets and whether $[1, j]$-Dominating Set is efficiently solvable on trees. In [29] the first question was answered negatively for the classes of planar, bipartite, and triangle-free graphs in which the smallest $[1, 2]$-dominating set is the entire set of vertices. In [12] the second question was answered positively via a linear-time algorithm. In [2], the $[1, j]$-Dominating Set problem was shown to be NP-hard even for chordal and planar graphs. However, for a constant j, a polynomial-time algorithm running in time $O(n^j p(\lg n))$ where p is a polynomial function, was obtained for n-vertex split graphs [2]. This is in contrast to the classic Dominating Set problem which is NP-hard for this class of graphs.

The Dominating Set problem has been widely studied in the realm of parameterized complexity. In general, finding a dominating set of size k is a canonical W[2]-complete problem and therefore unlikely to admit FPT algorithms [7]. Moreover, the problem remains W[2]-complete for split and bipartite graphs [23]. Nevertheless, there are interesting classes of sparse graphs for which the Dominating Set problem admits FPT algorithms. For example, there is an $O^*(3^{\mathsf{tw}})$-time algorithm for graphs of treewidth at most tw [28, 15], and FPT algorithms for nowhere dense graphs [6] and d-degenarate graphs [1]. Also, an FPT algorithm was reported in [27] for t-biclique-free graphs, i.e., graphs that do not contain $K_{t,t}$ as a subgraph. To the best of our knowledge, this is the largest class of graphs for which the Dominating Set problem is known to be fixed-parameter tractable; d-degenerate and nowhere dense graphs are subclasses of t-biclique-free graphs.

Another variant of dominating sets and $[1, j]$-dominating sets is $[1, j]$-total dominating sets. For a graph G, a subset $D \subseteq V(G)$ is called a $[1, j]$-total dominating set if $1 \leq |N(v) \cap D| \leq j$ for all $v \in V(G)$, where $N(v)$ denotes the open neighbourhood of v in G. In the $[1, j]$-Total Dominating Set problem we are given a graph G and a positive integer k. The objective is to check whether G admits a $[1, j]$-total dominating set of size at most k. $[1, 2]$-Total Dominating Set is NP-complete even for bipartite graphs [24]. Sharp upper bounds on the $[1, 2]$-total domination number of a graph are investigated in [16, 29]. Using a result of Rooij et al. [28], we can show that $[1, j]$-Dominating Set and $[1, j]$-Total Dominating Set are solvable in time $O^*((j + 2)^{\mathsf{tw}})$ and $O^*((2j + 2)^{\mathsf{tw}})$, respectively, on graphs of treewidth at most tw.

Our Contribution. In this paper, we study $[1, j]$-Dominating Set and $[1, j]$-Total Dominating Set from the parameterized complexity perspective. We prove the following results.

1. For any $\epsilon > 0$, there is no algorithm with running time $O(n^{j-\epsilon})$ for $[1, j]$-Dominating Set on split graphs assuming the Strong Exponential Time Hypothesis (SETH).
2. For the case of d-degenerate graphs, we tighten the complexity results and show that the $[1, j]$-Dominating Set problem is W[1]-hard for $d = j + 1$.
3. $[1, j]$-Dominating Set is FPT on classes of nowhere dense graphs.
4. There is no algorithm for $[1, 2]$-Total Dominating Set running in time $O^*((4 - \epsilon)^{\mathsf{pw}})$ assuming SETH, where pw is the pathwidth of the input graph.

We begin by some basic terminology and notation in Section 2. Then the next sections each address one of the results mentioned above.

2 Preliminaries

Graphs. We assume G is a simple graph with vertex set $V(G)$ and edge set $E(G)$. For brevity, we often denote these sets by V and E. We let $n = |V(G)|$ denote the order of G. For a vertex $v \in V$, the open neighborhood of v, denoted by $N(v)$, is defined as $\{u : \{u, v\} \in E\}$ and the closed neighborhood $N[v]$ is defined as $N(v) \cup \{v\}$. For a set $S \subseteq V$, we use $N(S)$ and $N[S]$ to denote the open and closed neighborhood S, respectively. That is, $N[S] = \bigcup_{v \in S} N[v]$ and $N(S) = N[S] \setminus S$. For a set $U \subseteq V$, we use $G[U]$ to denote the subgraph of G induced on U. A tree decomposition of a graph G is a tree T in which each vertex $x \in T$ has an assigned set of vertices $B_x \subseteq V(G)$ (called a bag) which satisfies the following properties: (i) $\bigcup_{x \in T} B_x = V(G)$; (ii) For any $\{u, v\} \in E$, there exists $x \in V(T)$ such that $u, v \in B_x$; (iii) For any $v \in V(G)$, the subtree of T induced on $\{x \in V(T) \colon v \in B_x\}$ is connected. The *width* of a tree decomposition T is $\max_{x \in V(T)}(|B_x| - 1)$. The treewidth of G, denoted by $tw(G)$, is the minimum width over all tree decompositions of G. The pathwidth of G, denoted by $pw(G)$, is the minimum width over all tree decompositions T of G, where T is a path.

Parameterized complexity. We now review some necessary concepts from parameterized complexity. For more details we refer the reader to [4, 8]. Given a finite alphabet Σ, a parameterization of Σ^* is a function $p : \Sigma^* \to \mathbb{N}$. A parameterized language L is a subset of $\{(x, k) \mid x \in \Sigma^* \wedge k = p(x)\}$. Here k is called the parameter. A parameterized language $L \subseteq \Sigma^* \times \mathbb{N}$ is called fixed-parameter tractable (FPT) if there exist an algorithm A (called a *FPT algorithm*) and a computable function $f : \mathbb{N} \to \mathbb{N}$ such that given $(x, k) \in \Sigma^* \times \mathbb{N}$, the algorithm A correctly decides whether $(x, k) \in L$ in time $f(k) \cdot |(x, k)|^{O(1)}$. The class of all fixed-parameter tractable problems is denoted by FPT.

▶ **Definition 1.** Let L and L' be two parameterized languages with parameterization functions p and p'. An FPT reduction from L to L' is a mapping $\rho : \Sigma^* \to \Sigma^*$ such that the following holds.

- For all $x \in \Sigma^*$, $(x, p(x)) \in L$ if and only if $(\rho(x), p'(\rho(x))) \in L'$
- There exists a computable function $g : \mathbb{N} \to \mathbb{N}$ such that for all $x \in \Sigma^*$, $p'(\rho(x)) \leq g(p(x))$.
- ρ is computable in FPT time, i.e., there exists a computable function f such that $\rho(x)$ is computable in time $O(f(p(x)) \cdot |x|^{O(1)})$.

To classify problems that are not FPT, Downey and Fellows [8] introduced the W-hierarchy. The hierarchy consists of complexity class W[t] for every integer $t \in \mathbb{N}$ such that $W[t] \subseteq W[t + 1]$, for all t. More generally, FPT $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq \cdots \subseteq$ W[t]. For our purposes, it is sufficient to note that these classes are closed under FPT reduction.

ETH and SETH. For $q \geq 3$, let δ_q be the infimum of the set of constants c for which there exists an algorithm solving q-SAT with n variables and m clauses in time $2^{cn} \cdot m^{O(1)}$. The *Exponential-Time Hypothesis (ETH)* and *Strong Exponential-Time Hypothesis (SETH)* are then formally defined as follows. ETH conjectures that $\delta_3 > 0$ and SETH that $\lim_{q \to \infty} \delta_q = 1$. In other words, SETH conjectures that for all $0 < \epsilon < 1$, there exist a (large) $q = q(\epsilon)$ such that q-SAT cannot be solved in time $O(2^{(1-\epsilon)n})$, where n is the number of variables in the input formula.

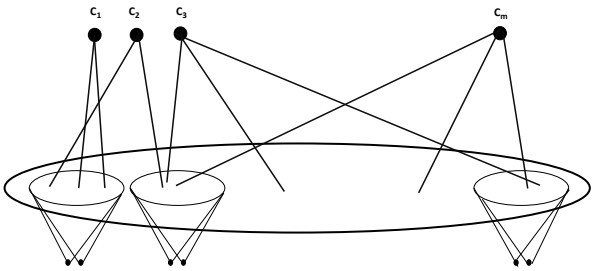


Figure 1 Constructed Split Graph.

3 Split Graphs

A graph G is called a split graph if its vertices can be partitioned into $V_1 \uplus V_2$ such that $G[V_1]$ is a complete graph and $G[V_2]$ is an empty graph (i.e., V_2 is an independent set in G). Several domination-like problems such as domination, total domination, and k-tuple domination are known to be NP-complete even on split graphs. On the other hand, there is a $n^j \cdot (\lg n)^{O(1)}$ time algorithm for $[1, j]$-Dominating Set on split graphs [2]. In this section, we prove that this is optimal, in the sense that one cannot obtain an $O(n^{j-\epsilon})$ algorithm for $[1, j]$-Dominating Set on split graphs unless SETH fails.

Reduction. Given an instance $I = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ of q-SAT over the set $X = \{x_1, \ldots, x_n\}$ of variables, we construct a graph G_I as follows.

- We partition the set X into j subsets $X_1, \ldots, X_j$ of size at most $\lceil n/j \rceil$.
- For each X_i we add a set S_i of $2^{|X_i|}$ vertices to the graph, each corresponding to one possible valuation of the variables in X_i. We also add two distinguished vertices u_i and v_i and connect them to all of S_i.
- We connect all the vertices in $\bigcup_{i=1}^{j} S_i$. That is $\bigcup_{i=1}^{j} S_i$ forms a clique in G_I.
- For every clause C_i, we add a vertex c_i and connect it to every vertex w of any S_j where the valuation corresponding to w satisfies C_i.

This completes the construction of G_I. See Figure 1 for an illustration. Our reduction algorithm will output an instance (G_I, j) of $[1, j]$-Dominating Set. We now proceed to proving the correctness of the reduction.

▶ **Lemma 2.** *If I is satisfiable then G_I has a $[1, j]$-dominating set of size j.*

Proof. Take one satisfying valuation of I and let $s_1 \in S_1, s_2 \in S_2, \ldots, s_j \in S_j$ be the vertices of G_I that correspond to this valuation. We claim that $S = \{s_1, s_2, \ldots, s_j\}$ is a $[1, j]$-dominating set. Every vertex in any of the S_i's is dominated by all of S, every $\{u_i, v_i\}$ pair is dominated only by the corresponding s_i and every c_i is dominated by a non-empty subset of S, i.e., the vertices whose corresponding valuation forces satisfaction of C_i. Given that $|S| = j$, there can be at most j such vertices. ◀

▶ **Lemma 3.** *If G_I has a $[1, j]$-dominating set of size j then I is satisfiable.*

Proof. First note that any dominating set of G_I of size j is also a $[1, j]$-dominating set. Let S be a dominating set of size at most j (and hence a $[1, j]$-dominating set) in G_I. Since $\{N[u_i] \colon i \in \{1, \ldots, j\}\}$ are pairwise vertex disjoint we have that $|S| = j$ and $|S \cap N[u_i]| = 1$

for all $i \in \{1, \ldots, j\}$. Moreover, since $N(u_i) = N(v_i)$, we conclude that $S \subseteq (\bigcup_{i=1}^{j} S_i)$. Consider the valuation of variables in X that corresponds to the vertices in S. This valuation satisfies every C_i, because S dominates every c_i. ◀

▶ **Theorem 4.** *For any $\varepsilon < 1$ and constant j, there is no $O(n^{j-\varepsilon})$ time algorithm for $[1, j]$-Dominating Set on split graphs unless SETH fails.*

Proof. For the sake of contradiction assume that there is an $O(n^{j-\varepsilon})$ time algorithm $\mathcal{A}$ for $[1, j]$-Dominating Set on split graphs. Then we claim that SETH is false. Let $\epsilon = (1 - \frac{\varepsilon}{j})$. Let $q = q(\epsilon)$ be the constant defined in the SETH conjecture. Given a q-SAT instance I we construct an instance (G_I, j) as mentioned in the reduction. By Lemmas 2 and 3 we know that I is satisfiable if and only if (G_I, j) is a yes-instance. Therefore, we use algorithm $\mathcal{A}$ to solve q-SAT. Now consider the running time for solving q-SAT using $\mathcal{A}$. The construction of (G_I, j) takes time $2^{\lceil n/j \rceil} \cdot n^{O(1)}$ and the number of vertices in G_I is at most $2j + m + j2^{\lceil n/j \rceil} = 2^{\lceil n/j \rceil} \cdot n^{O(1)}$. Thus the algorithm $\mathcal{A}$ on instance (G_I, j) takes time $2^{\lceil n/j \rceil (j-\varepsilon)} . n^{O(1)} = 2^{(n/j+(j-1))(j-\epsilon)} . n^{O(1)} = 2^{(n/j)(j-\varepsilon)} . n^{O(1)} = 2^{n(1-\epsilon/j)} . n^{O(1)} = 2^{\epsilon n} . n^{O(1)}$. This refutes SETH and the proof of the theorem is complete. ◀

4 Degenerate Graphs

A graph G is d-degenerate if every subgraph of G contains a vertex of degree at most d. Equivalently, a graph G is d-degenerate if and only if there exists an elimination ordering on its vertices such that every vertex has at most d neighbors appearing later in the ordering. In this section we prove the following result.

▶ **Theorem 5.** $[1, j]$-Dominating Set *parameterized by solution size is W[1]-hard on graphs of degeneracy $j + 1$.*

The following parameterized problem, proved to be W[1]-hard in [11], is used in our proof. In the Multicolored Independent Set problem, we are given a graph G and a proper vertex coloring of $V(G)$ with k colors. The parameter k is equal to the number of colors and the goal is to find a k-sized independent set in G containing exactly one vertex from each color class (such independents sets are called *k-colored independent sets*). We shall reduce the Multicolored Independent Set problem (with parameter k) to the problem of finding a $[1, j]$-dominating set of size at most $2k + j - 1$ in a graph of degeneracy $j + 1$.

The reduction. Let k be an integer and G be a proper k-vertex colored graph such that its vertices are partitioned into k groups $V_1, V_2, \ldots, V_k$, where each group corresponds to an independent set of the same color. Now we construct an instance $(G', 2k + j - 1)$ of $[1, j]$-Dominating Set as follows. For every edge $e = \{u, v\} \in E(G)$, we replace it by a path uv_ev, where v_e is a new vertex corresponding to the edge e. Let $S_E = \{v_e : e \in E(G)\}$. For each group V_i, $1 \leq i \leq k$, we build a $K_{1,2k+j}$ graph centered at a new vertex u_i. We also add new vertices x_1^i and x_2^i and connect the vertices u_i, x_1^i and x_2^i to the vertices of V_i. Moreover, we build $j - 1$ star graphs $K_{1,2k+j}$ centered at vertices $r_1, \ldots, r_{j-1}$ and make $r_1, \ldots, r_{j-1}$ adjacent to the vertices in S_E. This concludes the construction of G'. See Figure 2 for an illustration. Now we output $(G', 2k + j - 1)$ as an instance of $[1, j]$-Dominating Set. Clearly our reduction takes time polynomial in $|V(G)|$ and k.

▶ **Lemma 6.** *The constructed graph G' has degeneracy $j + 1$.*

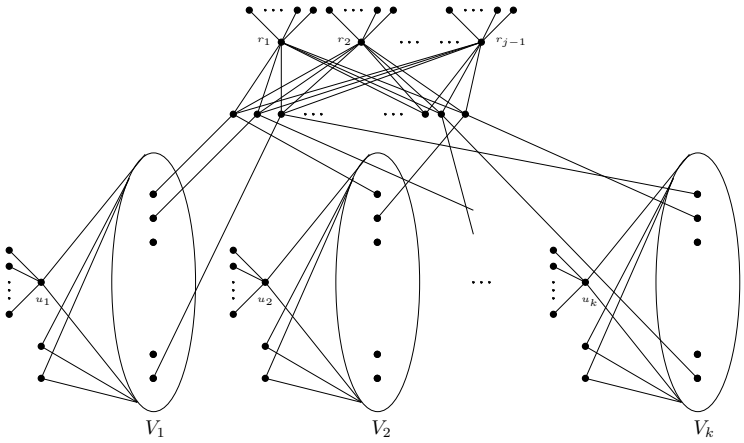


Figure 2 Reducing the Multicolored Independent Set problem to the $[1, j]$-Dominating Set problem.

Proof. The proof is by constructing a degeneracy ordering, which is an ordering on the vertices that we get from repeatedly removing a vertex of minimum degree in the remaining subgraph. First, we put all of the degree-one vertices in the degeneracy ordering and delete them. In the remaining subgraph, we select all the vertices in S_E and put them in ordering, because every vertex in S_E has degree $j+1$ in G'. After removing all vertices of S_E from the graph, each vertex in any block V_i, for $1 \leq i \leq k$, has degree three because after removing S_E such vertices are only connected to x_1^i, x_2^i, and u_i. So, next we can put the vertices $\bigcup_{i=1}^{k} V_i$ in the ordering. Finally we add all the remaining vertices. ◀

Lemmas 6, 7, and 8 below, imply Theorem 5.

▶ **Lemma 7.** *If there exists a k-colored independent set in G then there exists a $[1, j]$-dominating set of size $2k + j - 1$ in G'.*

Proof. Suppose that S is a k-colored independent set in G. We claim that $D = S \cup \{u_i : i \in \{1, \ldots, k\}\} \cup \{r_i : i \in \{1, \ldots, j-1\}\}$ is a $[1, j]$-dominating set of G'. Clearly, the size of D is $2k+j-1$. Each vertex $v_j \in V_i$ is dominated only by u_i in D. Moreover, each pair of x_1^i, x_2^i vertices is dominated by the single vertex in $V_i \cap S$. All the vertices in S_E are dominated by $\{r_i : i \in \{1, \ldots, j-1\}$ and by at most one vertex from S (since S is an independent set). Moreover all the degree one vertices in G' are dominated exactly once by D. ◀

▶ **Lemma 8.** *If there exists a $[1, j]$-dominating set of size $2k + j - 1$ in G' then there exists a k-colored independent set in G.*

Proof. Let D be a $[1, j]$-dominating set of size at most $2k + j - 1$. First, note that since $|D| \leq 2k + j - 1$, we have that $\{r_1, \ldots, r_{j-1}\} \cup \{u_i, \ldots, u_k\} \subseteq D$ (because each u_i and r_i is connected to $2k+j$ degree one vertices). We claim that $S = D \setminus (\{r_1, \ldots, r_{j-1}\} \cup \{u_i, \ldots, u_k\})$ is a k-colored independent set in G. Clearly $|S| \leq k$. Also, to dominate all the vertices x_1^i, x_i^2 for $1 \leq i \leq k$, we should have exactly one vertex from each V_i. Therefore $S \subseteq V(G)$ and $|S \cap V_i| = 1$ for all $1 \leq i \leq k$. Suppose $u, v \in S$ is adjacent in G. Then the vertex v_e, where $e = \{u, v\}$ is dominated $j+1$ times by D (because v_e is adjacent to $\{u, v\}$ and $\{r_1, \ldots, r_{j-1}\}$). Therefore since D is a $[1, j]$-dominating set in G', S is a k-colored independent set in G This completes the proof of the lemma. ◀

5 Nowhere Dense Graphs

The notion of nowhere denseness was introduced by Nešetřil and Ossona de Mendez [20, 21] as a general model of uniform sparseness of graphs. Many familiar classes of sparse graphs, like planar graphs, graphs of bounded tree-width, graphs of bounded degree, and all classes that exclude a fixed (topological) minor, are nowhere dense. An important and related concept is the notion of a graph class of bounded expansion, which was also introduced by Nešetřil and Ossona de Mendez [17, 18, 19].

▶ **Definition 9.** Let H be a graph and let $r \in \mathbb{N}$. An r-subdivision of H is obtained by replacing all edges of H by internally vertex disjoint paths of length at most r.

▶ **Definition 10.** A class $\mathcal{C}$ of graphs is nowhere dense if there exists a function $t\colon \mathbb{N} \to \mathbb{N}$ such that for all $q \in \mathbb{N}$ and for all $G \in \mathcal{C}$ we do not find an q-subdivision of the complete graph $K_{t(q)}$ as a subgraph of G. Otherwise, $\mathcal{C}$ is called somewhere dense.

▶ **Definition 11.** A class $\mathcal{C}$ of graphs has bounded expansion if there exists a function $d\colon \mathbb{N} \to \mathbb{N}$ such that for all $r \in \mathbb{N}$ and all graphs H, such that an r-subdivision of H is a subgraph of G for some $G \in \mathcal{C}$, satisfy $|E(H)|/|V(H)| \leq d(r)$.

Every class of bounded expansion is nowhere dense, which in turn excludes some biclique as a subgraph and hence is biclique-free. For extensive background on bounded expansion and nowhere dense graphs we refer to the textbook of Nešetřil and Ossona de Mendez [22].

Before we state our result, we quickly recall the necessary definitions from logic. For our purpose, it suffices to consider first-order logic over the vocabulary of graphs. We refer to the textbook [9] for extensive background on logic. A (relational) vocabulary is a finite set of relation symbols, each with a prescribed arity. We let σ be a vocabulary. A σ-structure A consist of a (not necessarily finite) set $V(A)$, called the universe or vertex set of A, and for each k-ary relation symbol $R \in \sigma$ a k-ary relation $R(A) \subseteq V(A)^k$. A structure A is finite if its universe is. For example, graphs may be viewed as $\{E\}$-structures, where E is a binary relation symbol. First-order formulas of vocabulary σ are formed from atomic formulas $x = y$ and $R(x_1, \ldots, x_k)$, where $R \in \sigma$ is a k-ary relation symbol and $x, y, x_1, \ldots, x_k$ are variables (we assume that we have an infinite supply of variables) by the usual Boolean connectives $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), and existential and universal quantifications $\exists x$ and $\forall x$, respectively. The set of all first-order formulas of vocabulary σ is denoted by FO$[\sigma]$, and the set of all first-order formulas by FO. The free variables of a formula are those not in the scope of a quantifier, and we write $\phi(x_1, \ldots, x_k)$ to indicate that the free variables of the formula ϕ are among $x_1, \ldots, x_k$. A sentence is a formula without free variables. To define the semantics, we inductively define a satisfaction relation $\models$, where for a σ-structure A, a formula $\phi(x_1, \ldots, x_k)$, and elements $a_1, \ldots, a_k \in V(A)$, $A \models \phi(a_1, \ldots, a_k)$ means that A satisfies ϕ if the free variables $x_1, \ldots, x_k$ are interpreted by $a_1, \ldots, a_k$, respectively. If $\phi(x_1, \ldots, x_k) = R(x_1, \ldots, x_k)$ is atomic, then $A \models \phi(a_1, \ldots, a_k)$ if $(a_1, \ldots, a_k) \in R(A)$. The meaning of the equality symbol, the Boolean connectives, and the quantifiers is the usual one. For example, consider the formula $\phi(x_1, x_2) = \forall y(x_1 = y \vee x_2 = y \vee E(x_1, y) \vee E(x_2, y))$ in the vocabulary $\{E\}$ of graphs. For every graph G and vertices $v_1, v_2 \in V(G)$ we have $G \models \phi(v_1, v_2)$ if any only if $\{v_1, v_2\}$ is a dominating set of G. Thus G satisfies the sentence $\exists x_1 \exists x_2 \phi(x_1, x_2)$ if and only if it has a (nonempty) dominating set of size at most 2.

▶ **Theorem 12.** *The* $[1, j]$-Dominating Set *problem parameterized by solution size* k *is fixed-parameter tractable on nowhere dense classes of graphs.*

Proof. Our proof is based on a result of Grohe, Kreutzer, and Siebertz [13], which states that for every first-order sentence ψ (or formula without free variables), every nowhere dense class $\mathcal{C}$ of graphs and every real $\epsilon > 0$, there exists a constant $f(|\psi|, \epsilon)$, such that given an n-vertex graph $G \in \mathcal{C}$, one can decide in time $f(|\psi|, \epsilon) \cdot n^{1+\epsilon}$ whether ψ holds in G.

It is easy to verify that the $[1, j]$-Dominating Set problem is expressible in FO. We let ψ be the following sentence.

$$\exists v_1, v_2, \ldots, v_k \forall u \Big((u = v_1 \vee u = v_2, \ldots, u = v_k) \vee ((\phi_1(u, v_1, v_2, \ldots, v_k) \vee \phi_2(u, v_1, v_2, \ldots, v_k)$$

$$\cdots \vee \phi_j(u, v_1, v_2, \ldots, v_k)) \Big),$$

where the function $\phi_i(u, v_1, v_2, \ldots, v_k)$ is true where the vertex u is adjacent to exactly i vertices of $v_1, v_2, \ldots, v_k$, which can be represented using a formula of length bounded by a function of i and k. Note that the length of ψ is bounded by a function depending only on k (and on j, though only as a fixed constant). Then, by fixing any $\epsilon > 0$ and using the result of [13], we conclude that $[1, j]$-Dominating Set is fixed-parameter tractable parameterized by solution size k on every nowhere dense class $\mathcal{C}$. ◀

6 Bounded Treewidth Graphs

It is well-known [4] that the Dominating Set problem can be solved in $O^*(3^t)$ time on graphs of treewidth at most t. Lokshtanov et al. [15] showed that this is essentialy optimal, i.e., they showed that the problem cannot be solved in $O^*((3 - \epsilon)^t)$ time unless SETH fails. The situation is almost identical for the Connected Dominating Set problem. The problem can be solved in $O^*(4^t)$ time, but cannot be solved in time $O^*(4 - \epsilon)^t$ unless SETH fails [5]. The $[1, j]$-Dominating Set problem is a special case of the (σ, ρ)-domination problem. The concept of (σ, ρ)-domination was introduced by Telle [26]. Let σ, ρ be a pair of non-empty sets of non-negative integers. A set S of vertices of a graph G is called (σ, ρ)-dominating if for every vertex $v \in S, |S \cap N(v)| \in \sigma$, and for every $v \notin S, |S \cap N(v)| \in \rho$. $[1, j]$-Dominating Set and $[1, 2]$-Total Dominating Set are special cases of the (σ, ρ)-Dominating Set problem. For $[1, j]$-Dominating Set, we set $\sigma = \{0, 1, \ldots\}$ and $\rho = \{1, \ldots, j\}$. On the other hand for $[1, 2]$-Total Dominating Set we set $\sigma = \rho = \{1, \ldots, j\}$. The following result is due to Rooij et al. [28].

▶ **Proposition 13.** *Let $\sigma, \rho \subseteq \mathcal{N}$ be finite or cofinite. There exists an algorithm with running time $O^*(s^t)$ for finding a (σ, ρ)-dominating set in graphs of treewidth at most t where s is the number of states per vertex used in the representations of the solution.*

▶ **Corollary 14.** $[1, j]$-Dominating Set *and* $[1, j]$-Total Dominating Set *are solvable in time $O^*((j + 2)^{\mathsf{tw}})$ and $O^*((2j + 2)^{\mathsf{tw}})$, respectively, on graphs of treewidth at most* tw.

Proof. As mentioned before, $[1, j]$-Dominating Set is a special case of (σ, ρ)-Dominating Set, where $\sigma = \{0, 1, \ldots\}$ and $\rho = \{1, \ldots, j\}$. The states needed in the representation of the solution in this special case are $\{\rho_0, \rho_1, \ldots, \rho_j\}$ and $*$, where ρ_i, $0 \leq i \leq j$ means the vertex in the bag is not in the dominating set and dominated i times and $*$ means that the vertex is in the dominating set. Then by Proposition 13, $[1, j]$-Dominating Set is solvable in time $O^*((j + 2)^{\mathsf{tw}})$.

$[1, j]$-Total Dominating Set is the special case of (σ, ρ)-Dominating Set, with $\sigma = \rho = \{1, \ldots, j\}$. In this case the states needed in the representation of the solution are $\{\sigma_0, \rho_0, \sigma_1, \rho_1, \ldots, \sigma_j, \rho_j\}$, where ρ_i, $0 \leq i \leq j$ means the vertex in the bag is not in

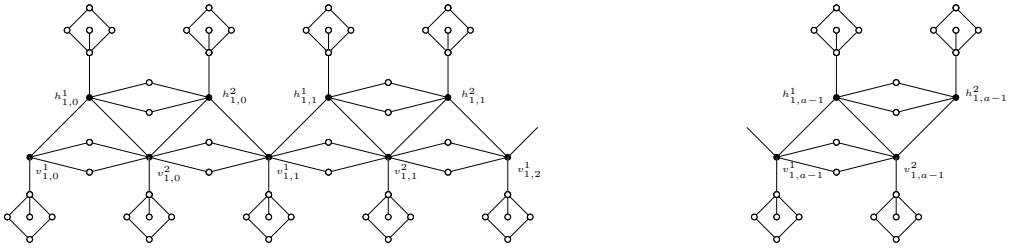


Figure 3 Parts of the construction.

the dominating set and dominated i times and σ_i, $0 \leq i \leq j$ means the vertex in the bag is in the dominating set and i of its neighbours are also in the dominating set. Then by Proposition 13, $[1, j]$-Total Dominating Set is solvable in time $O^*((2j+2)^{\mathsf{tw}})$. ◀

In what follows, we prove that the $[1, 2]$-Total Dominating Set problem cannot be solved in time $O^*(4-\epsilon)^{\mathsf{pw}}$ (unless SETH fails) and which is also a lower bound in terms of the treewidth of the input graph. It remains open whether a similar result holds for $[1, 2]$-Dominating Set. Our proof closely follow the work of Cygan et al. [5] and we use the notions of path decompositions, pathwidth, tree decompositions, treewidth, and mixed search games. We give a reduction from CNF-SAT to the $[1, 2]$-Total Dominating Set problem and prove that the reduced graph has pathwidth at most $\frac{n}{2} + O(1)$, where n is the number of variables in the input CNF-SAT formula.

The reduction. Given $\epsilon > 0$ and an instance Φ of CNF-SAT with n variable and m clauses, we construct a graph G as follows. We assume that the number of variables n is even, otherwise we add a single dummy variable. We partition the variables of Φ into groups $F_1, F_2, \ldots, F_{n'}$, each of size two, where $n' = n/2$. We let $a = m(n+1)$.

For each $1 \leq t \leq n'$ we create a path P_t of length $4a = 4m(n+1)$ consisting of vertices $v^\alpha_{t,q}$ and $h^\alpha_{t,q}$, where $1 \leq \alpha \leq 2$ and $0 \leq q < a$. The vertices are arranged on the path in the following order: $v^1_{t,0}, h^1_{t,0}, v^2_{t,0}, h^2_{t,0}, v^1_{t,1}, h^1_{t,1}, v^2_{t,1}, h^2_{t,1}, \ldots, v^1_{t,a-1}, h^1_{t,a-1}, v^2_{t,a-1}, h^2_{t,a-1}$. We let $\mathcal{V}$ and $\mathcal{H}$ denote the sets of all $v^\alpha_{t,q}$ vertices and $h^\alpha_{t,q}$ vertices, respectively.

For each vertex $v^\alpha_{t,q}$, we add a forcing gadget consisting of a 4-cycle with one additional pendant vertex connected to a vertex which we denote as the root vertex $r^\alpha_{t,q}$. We add an edge between $v^\alpha_{t,q}$ and the root of the cycle $r^\alpha_{t,q}$. Similarly, for each vertex $h^\alpha_{t,q}$, we add a forcing gadget consisting of a 4-cycle rooted at vertex $s^\alpha_{t,q}$ (i.e. the pendant vertex is connected to $s^\alpha_{t,q}$). We add an edge between $h^\alpha_{t,q}$ and $s^\alpha_{t,q}$ (see Figure 3). Note that any dominating set in the graph will contain at least two vertex from a forcing gadget and if a $[1, 2]$-total dominating set contains only two vertex from a forcing gadget, then it include the root vertex and one of its neighbours on the 4-cycle (by the definition of $[1, 2]$-total dominating set).

Next, we add three pairs of guard vertices $p^1_{t,q}$, $p^2_{t,q}$, and $p^3_{t,q}$, for each $1 \leq t \leq n'$ and $0 \leq q < a$. Each of the vertices in these pairs are of degree two and are connected to other vertices as follows: (i) vertices in $p^1_{t,q}$ are adjacent to $v^1_{t,q}$ and $v^2_{t,q}$; (ii) vertices in $p^2_{t,q}$ are adjacent to $v^2_{t,q}$ and $v^2_{t,q+1}$; and (iii) vertices in $p^3_{t,q}$ are adjacent to $h^1_{t,q}$ and $h^2_{t,q}$. This structure forces a dominating set to either contain the vertices from the guard sets or one of their two neighbors. For instance, to dominate the pair $p^1_{t,q}$, either both vertices in $p^1_{t,q}$ must be in the dominating set or one of $v^1_{t,q}$ or $v^2_{t,q}$. We use $\mathcal{G}$ to denote the set of all guard vertices.

The intuition of the construction made so far is as follows. For each two-variable block F_t we encode any assignment of the variables in F_t as a choice of whether to take $v^1_{t,q}$ or $v^2_{t,q}$ and $h^1_{t,q}$ or $h^2_{t,q}$ into the dominating set. This concludes the construction of the "variable gadgets" which are required for encoding an assignment. The forcing gadgets attached to vertices in $\mathcal{V}$ and $\mathcal{H}$ will guarantee that those vertices are all dominated at least once.

We now add "clause gadgets" required for checking the satisfiability of Φ. For each clause C_i, we build $(n+1)$ vertices $c_{i,j}$, one for each $0 \leq j \leq n$. Consider a clause C_i and a group of variables $F_t = \{x^1_t, x^2_t\}$. If x^1_t occurs positively as the ℓ^{th} literal in C_i, then connect $v^1_{t,mj+i}$ to $c_{i,j}$ via a path of length five by adding four vertices $w^\ell_{i,j}$, $x^\ell_{i,j}$, $y^\ell_{i,j}$ and $z^\ell_{i,j}$ and edges $\{c_{i,j}, w^\ell_{i,j}\}$, $\{w^\ell_{i,j}, x^\ell_{i,j}\}$, $\{x^\ell_{i,j}, y^\ell_{i,j}\}$, $\{y^\ell_{i,j}, z^\ell_{i,j}\}$, and $\{z^\ell_{i,j}, v^1_{t,mj+i}\}$. If x^1_t occurs negatively as the ℓ^{th} literal in C_i, we connect $v^2_{t,mj+i}$ to $c_{i,j}$ again via a path of length five by adding four vertices $w^\ell_{i,j}$, $x^\ell_{i,j}$, $y^\ell_{i,j}$ and $z^\ell_{i,j}$ and edges $\{c_{i,j}, w^\ell_{i,j}\}$, $\{w^\ell_{i,j}, x^\ell_{i,j}\}$, $\{x^\ell_{i,j}, y^\ell_{i,j}\}$, $\{y^\ell_{i,j}, z^\ell_{i,j}\}$, and $\{z^\ell_{i,j}, v^2_{t,mj+i}\}$. Similarly, if x^2_t occurs positively as the ℓ^{th} literal in C_i, we connect $h^1_{t,mj+i}$ to $c_{i,j}$ via a path of length five and if it occurs negatively, we connect $h^2_{t,mj+i}$ to $c_{i,j}$ via a path of length five. We let $\mathcal{W}$, $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$ denote the sets of all w, x, y, and z vertices added between clause vertices and vertices in $\mathcal{V} \cup \mathcal{H}$, respectively. This concludes the construction of the graph G.

▶ **Lemma 15.** *If Φ is satisfiable then G has a $[1,2]$-total dominating set D of size $k = 10an' + 2(n+1)\Sigma_{i=1}^{m}|C_i|$.*

Proof. Consider a satisfying assignment ϕ for Φ. We construct a $[1,2]$-total dominating set D of size k in G as follows. We first add the root vertex and one of its neighbors on the 4-cycle of each forcing gadget to D. Then, for each block $F_t = \{x^1_t, x^2_t\}$ and each $0 \leq q < a$, we add the vertex $v^1_{t,q}$ to D if the value of x^1_t is true, otherwise we add $v^2_{t,q}$ to D. Similarly, if the value of x^2_t is true, we add $h^1_{t,q}$ to D, otherwise we add $h^2_{t,q}$. Since the clause C_i is satisfied, at least one of the vertices $v^\alpha_{t,q}$ or $h^\alpha_{t,q}$ (for some t, α) will be in the dominating set D. Hence, the corresponding $z^\ell_{i,j}$ vertex will be dominated, for some $1 \leq \ell \leq |C_i|$. We can therefore add vertices $x^\ell_{i,j}$ and $w^\ell_{i,j}$ to D as to dominate $c_{i,j}$ and $y^\ell_{i,j}$ (and maintain the $[1,2]$-total dominating set property). To dominate the remaining vertices in the clause gadget for $c_{i,j}$, we add all $y^{\ell'}_{i,j}$ and $x^{\ell'}_{i,j}$ vertices, where $1 \leq \ell' \leq |C_i|$ and $\ell' \neq \ell$.

It is clear that for each clause C_i, we add $2(n+1)|C_i|$ vertices to D, for a total of $2(n+1)\Sigma_{i=1}^{m}|C_i|$ vertices. Moreover, for each path P_t, we add $10a$ vertices to D. Therefore, accounting for the fact that we have n' paths in total, the total number of vertices in D is $10an' + 2(n+1)\Sigma_{i=1}^{m}|C_i| = k$. It remains to show that D is in fact a $[1,2]$-total dominating set. Every vertex in $\mathcal{V} \cup \mathcal{H}$ is dominated at most twice; once by a vertex in its corresponding forcing gadget and possibly once by a neighbor in $\mathcal{V} \cup \mathcal{H}$. Each vertex in the guard sets is dominated exactly once and each vertex in $\mathcal{W}$, $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$ is dominated at most twice (as they have degree two). Finally, each vertex $c_{i,j}$ is dominated exactly once (by a vertex in $\mathcal{W}$), as needed. This concludes the proof of the lemma. ◀

▶ **Lemma 16.** *If G has a $[1,2]$-total dominating set of size at most $k = 10an' + 2(n+1)\Sigma_{i=1}^{m}|C_i|$ then Φ is satisfiable.*

Proof. Consider a $[1,2]$-total dominating set D of size at most k. The set D must contain at least 2 vertex from each forcing gadget. This constitute at least $8an'$ vertices from forcing gadgets. Since $2(n+1)|C_i|$ vertices are required to total dominate the vertices in the clause gadget for C_i, we know that $|D \cap (\mathcal{W} \cup \mathcal{X} \cup \mathcal{Y} \cup \mathcal{Z})| \geq 2(n+1)\Sigma_{i=1}^{m}|C_i|$ and, therefore, $|D \cap (\mathcal{V} \cup \mathcal{H} \cup \mathcal{G})| \leq 2an'$. Moreover, G contains n' paths each of length $4a$ and the guard vertices in $\mathcal{G}$ ensure that at least $2a$ vertices are required to dominate the vertices of each path.

Finally, since we have $2an'$ pairs of guards with disjoint neighbors, exactly one vertex from $\{v^1_{t,q}, v^2_{t,q}\}$ and one vertex from $\{h^1_{t,q}, h^2_{t,q}\}$ must be in D, for each $1 \leq t \leq n'$ and $0 \leq q < a$. This implies that each forcing gadget will contribute two vertices to D– the root vertex and one of its neighbour in the 4-cycle. Moreover $|D \cap (\mathcal{W} \cup \mathcal{X} \cup \mathcal{Y} \cup \mathcal{Z})| = 2(n+1)\Sigma_{i=1}^{m}|C_i|$. This implies that exactly two vertex from a path from $c_{i,j}$ to a vertex in $\mathcal{V} \cup \mathcal{H}$ is included in D. Moreover, this in turn implies that $c_{i,j} \notin D$ for any i, j.

Now, for each $0 \leq q < a$, we construct an assignment ϕ_q as follows. For each block $F_t = \{x^1_t, x^2_t\}$, we define $\phi_q(x^1_t)$ as true if $v^1_{t,q} \in D$ and we define $\phi_q(x^1_t)$ to be false if $v^2_{t,q} \in D$. Similarly, for x^2_t, we define $\phi_q(x^2_t)$ to be true if $h^1_{t,q} \in D$ and we define $\phi_q(x^2_t)$ to be false if $h^2_{t,q} \in D$. Note that for each block $F_t = \{x^1_t, x^2_t\}$ and each $0 \leq q < a$, we have:

- If $\phi_q(x^1_t)$ is true then $\phi_{q+1}(x^1_t)$ is true. Otherwise, both $v^2_{t,q}$ and $v^1_{t,q+1}$ are not in the dominating set and vertices in $p^2_{t,q}$ is not dominated by D.
- If $\phi_q(x^2_t)$ is false then $\phi_{q+1}(x^2_t)$ is false. Otherwise, both $h^2_{t,q}$ and $h^1_{t,q+1}$ are in the dominating set and vertex $v^1_{t,q+1}$ is dominated three times in D; by $h^2_{t,q}$, $h^1_{t,q+1}$, and the root of the forcing gadget corresponding to $v^1_{t,q+1}$.

For each variable x, we define a sequence $\hat{\phi}_x = \phi_0(x)\phi_1(x)\ldots\phi_{a-1}(x)$. By the discussion above we infer that for each variable x, the sequence $\hat{\phi}_x$ can change its value at most once. Hence, as $a = m(n+1)$, we conclude that there exists $0 \leq j < n+1$ such that for all $0 \leq i < m$, the assignment $\phi_{mj+i}(x)$ are equal. We claim that the assignment $\phi = \phi_{mj}(x)$ satisfies ϕ. Consider a clause C_i and focus on the vertex $c_{i,j}$. We know that $c_{i,j} \notin D$. Thus one of its neighbors from $\mathcal{W}$ (say $w^{\ell'}_{i,j}$) is contained in D. Therefore, by the definition of a $[1,2]$-total dominating set, $x^{\ell'}_{i,j}$ is in D. We have already argued that any path from $c_{i,j}$ to any vertex in $\mathcal{V} \cup \mathcal{H}$ can contain at most 2 vertices in D. This implies that $z^{\ell'}_{i,j}$ is dominated by a vertex in $\mathcal{V} \cup \mathcal{H}$ and the literal corresponding to it will be set to true by ϕ. This completes the proof of the lemma. ◀

▶ **Lemma 17.** *$pw(G) \leq n' + O(1)$.*

Proof. Let us first recall the definition of a mixed search game. In a mixed search game, the graph G represents a "system of tunnels". Initially, all edges are contaminated by a gas. An edge is cleared by placing searchers at both its endpoints simultaneously or by sliding a searcher along the edge. A cleared edge is re-contaminated if there is a path from an uncleared edge to the cleared edge without any searchers on its vertices or edges. A search is a sequence of operations that can be of the following types: (i) placement of a new searcher on a vertex; (ii) removal of a searcher from a vertex; (iii) sliding a searcher on a vertex along an incident edge and placing the searcher on the other end. A search strategy is winning if after its termination all edges are cleared. The mixed search number of a graph G, denoted by $ms(G)$, is the minimum number of searchers required for a winning strategy of mixed searching on G. Takahashi et al. [25] obtained the following relationship between $pw(G)$ and $ms(G)$: $pw(G) \leq ms(G) \leq pw(G) + 1$.

We give a mixed strategy to clean the graph with $n' + O(1)$ searchers. The cleaning process will be done in a rounds. At each round $0 \leq q < a$, we put a searcher on clause variable $c_{i,j}$ and keep this searcher there until the cleaning round is completed. It is clear we could clean each forcing gadget by four searcher and keep searcher on the root vertex after cleaning the edges in the force gadget. For each $1 \leq t \leq n'$ and $0 \leq q < a$, after cleaning four connected forcing gadgets we slide the searchers from the root of forcing gadgets to the vertices $v^1_{t,q}$, $h^1_{t,q}$, $v^2_{t,q}$, $h^2_{t,q}$. Then we put searchers on the the (at most two) z vertices connected to them and the guard vertices in sets $p^1_{t,q}$, $p^2_{t,q}$ and $p^3_{t,q}$. As we have a searcher

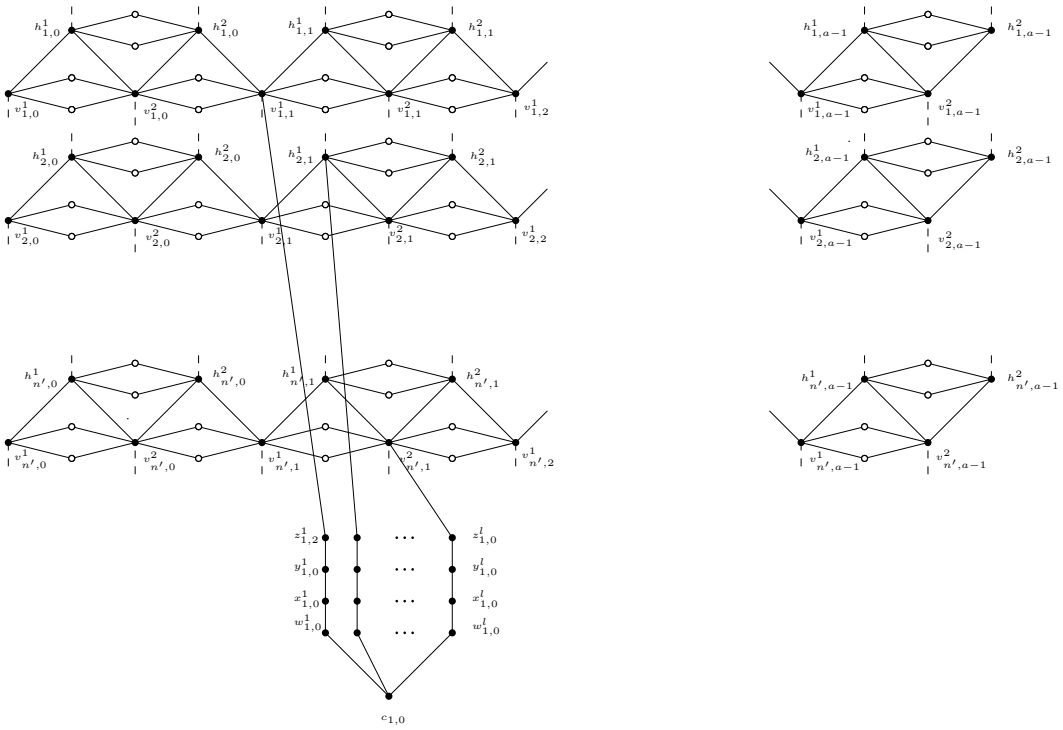


Figure 4 Parts of the construction. Dashed edges are connecting vertices with a forcing gadget.

on $c_{i,j}$ unil the end of this round we could clean the paths between vertex $c_{i,j}$ and those z vertices using one more searcher. The last step of the round is removing the searchers from vertices $v^1_{t,q}, h^1_{t,q}, v^2_{t,q}, h^2_{t,q}$ and searchers on the gaurd sets $p^1_{t,q}, p^2_{t,q}$ and $p^3_{t,q}$. We just keep the searcher on $v^1_{t,q+1}$. Also, we remove searchers in path between $c_{i,j}$ and $\mathcal{V} \cup \mathcal{H}$ except for the one standing on the $c_{i,j}$. To commence the next round, the searcher in $c_{i,j}$ is deleted and a new searcher is put on $c_{i,j+1}$. After the last round the whole graph G is cleaned. Since at any point in time we need at most 14 searchers (they are on $c_{i,j}$, two z vertices, $v^1_{t,q}$, $h^1_{t,q}$, $v^2_{t,q}$, $h^2_{t,q}$, guard vertices in sets $p^1_{t,q}$, $p^2_{t,q}$ and $p^3_{t,q}$, one searcher to clean the paths between two z vertices and $c_{i,j}$) and we reuse 14 searchers in the cleaning process, $n' + 14$ searchers suffice to clean the graph. ◀

▶ **Theorem 18.** *Assuming SETH, for any $\epsilon > 0$ there is no algorithm running in time $(4-\epsilon)^{\mathsf{pw}}|V(G)|^{O(1)}$ for* $[1,2]$-Total Dominating Set *on a graph G with pathwidth* pw.

Proof. Suppose that $[1,2]$-Total Dominating Set can be solved in time $(4-\epsilon)^{\mathsf{pw}}|V(G)|^{O(1)}$. Given an instance of CNF-SAT, we can construct an instance of $[1,2]$-Total Dominating Set using the above construction and solve it with a $(4-\epsilon)^{\mathsf{pw}}|V(G)|^{O(1)}$ time algorithm. The correctness of the algorithm follows from Lemmata 15 and 16. Lemma 17 implies that the running time of the algorithm is $(4-\epsilon)^{\frac{n}{2}}|V(G)|^{O(1)}$. However, we have $(4-\epsilon)^{\frac{n}{2}} = (\sqrt{4-\epsilon})^n$ and $\sqrt{4-\epsilon} < 2$ which refutes SETH. This concludes the proof. ◀

7 Conclusion

We have shown that the $[1, j]$-Dominating Set problem parameterized by solution size is W[1]-hard on graphs of degeneracy $(j + 1)$. It is thus natural to ask whether the problem becomes fixed-parameter tractable when the degeneracy of the graph is smaller or equal to j. In particular, is the $[1, j]$-Dominating Set problem fixed-parameter tractable on j-degenerate graphs? There is a very rich literature [6, 1, 10] on kernelization for the Dominating Set problem on sparse graphs and it would be interesteing to see where (if anywhere) the known techniques for the Dominating Set problem become applicable to the $[1, j]$-Dominating Set problem. Finally, we have shown a lower bound of $O^*((4 - \epsilon)^{\mathsf{tw}})$ for $[1, 2]$-Total Dominating Set assuming SETH, while the known upper bound is $O^*(6^{\mathsf{tw}})$. Closing this gap is an interesting open problem. On the other hand $[1, 2]$-Dominating Set can be solved in time $O^*(4^{\mathsf{tw}})$ and getting a matching lower bound is another open problem. Moreover, can we get lower bounds for more general problems such as $[1, j]$-Dominating Set and $[1, j]$-Total Dominating Set?

References

1 Noga Alon and Shai Gutner. Linear time algorithms for finding a dominating set of fixed size in degenerated graphs. *Algorithmica*, 54(4):544, 2009.

2 Arijit Bishnu, Kunal Dutta, Arijit Ghosh, and Subhabrata Paul. (1, j)-set problem in graphs. *Discrete Mathematics*, 339(10):2515–2525, 2016.

3 Mustapha Chellali, Teresa W Haynes, Stephen T Hedetniemi, and Alice McRae. [1, 2]-sets in graphs. *Discrete Applied Mathematics*, 161(18):2885–2893, 2013.

4 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 3. Springer, 2015.

5 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Joham MM van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 150–159. IEEE, 2011.

6 Anuj Dawar and Stephan Kreutzer. Domination Problems in Nowhere-Dense Classes. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, volume 4 of *LIPIcs*, pages 157–168. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2009. doi:10.4230/LIPIcs.FSTTCS.2009.2315.

7 Rod G Downey and Michael R Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.

8 Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.

9 Heinz-Dieter Ebbinghaus, Jörg Flum, and Wolfgang Thomas. *Mathematical logic (2. ed.)*. Undergraduate texts in mathematics. Springer, 1994.

10 Eduard Eiben, Mithilesh Kumar, Amer E. Mouawad, Fahad Panolan, and Sebastian Siebertz. Lossy Kernels for Connected Dominating Set on Sparse Graphs. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPIcs*, pages 29:1–29:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.29.

11 Michael R Fellows, Danny Hermelin, Frances Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science*, 410(1):53–61, 2009.

12 Amir Kafshdar Goharshady, Mohammad Reza Hooshmandasl, and M Alambardar Meybodi. [1, 2]-sets and [1, 2]-total sets in trees with algorithms. *Discrete Applied Mathematics*, 198:136–146, 2016.

13 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. In *STOC 2014*, pages 89–98. ACM, 2014.

14 Teresa W Haynes, Stephen Hedetniemi, and Peter Slater. *Fundamentals of domination in graphs*. CRC Press, 1998.

15 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known Algorithms on Graphs of Bounded Treewidth Are Probably Optimal. In *Proceedings of the Twenty-second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, pages 777–789, Philadelphia, PA, USA, 2011. Society for Industrial and Applied Mathematics. URL: `http://dl.acm.org/citation.cfm?id=2133036.2133097`.

16 Xuezheng Lv and Baoyindureng Wu. Total [1, 2]-domination in graphs. *arXiv preprint arXiv:1503.04939*, 2015.

17 Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion I. Decompositions. *European Journal of Combinatorics*, 29(3):760–776, 2008.

18 Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion II. Algorithmic aspects. *European Journal of Combinatorics*, 29(3):777–791, 2008.

19 Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion III. Restricted graph homomorphism dualities. *European Journal of Combinatorics*, 29(4):1012–1024, 2008.

20 Jaroslav Nešetřil and Patrice Ossona de Mendez. First order properties on nowhere dense structures. *The Journal of Symbolic Logic*, 75(03):868–887, 2010.

21 Jaroslav Nešetřil and Patrice Ossona de Mendez. On nowhere dense graphs. *European Journal of Combinatorics*, 32(4):600–617, 2011.

22 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity — Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012.

23 Venkatesh Raman and Saket Saurabh. Short cycles make W-hard problems hard: FPT algorithms for W-hard problems in graphs with no short cycles. *Algorithmica*, 52(2):203–225, 2008.

24 P Sharifani and MR Hooshmandasl. Some Results on [1, k]-sets of Lexicographic Products of Graphs. *arXiv preprint arXiv:1708.00219*, 2017.

25 Atsushi Takahashi, Shuichi Ueno, and Yoji Kajitani. Mixed searching and proper-pathwidth. *Theoretical Computer Science*, 137(2):253–268, 1995.

26 Jan Arne Telle. Complexity of domination-type problems in graphs. *Nord. J. Comput.*, 1(1):157–171, 1994.

27 Jan Arne Telle and Yngve Villanger. FPT algorithms for domination in biclique-free graphs. In *ESA*, pages 802–812. Springer, 2012.

28 Johan MM Van Rooij, Hans L Bodlaender, and Peter Rossmanith. Dynamic Programming on Tree Decompositions Using Generalised Fast Subset Convolution. In *ESA*, volume 9, pages 566–577. Springer, 2009.

29 Xiaojing Yang and Baoyindureng Wu. [1, 2]-domination in graphs. *Discrete Applied Mathematics*, 175:79–86, 2014.

Sub-Exponential Time Parameterized Algorithms for Graph Layout Problems on Digraphs with Bounded Independence Number

Pranabendu Misra
University of Bergen, Norway
pranabendu.misra@ii.uib.no

Saket Saurabh
Institute of Mathematical Sciences, HBNI, India
saket@imsc.res.in

Roohani Sharma
Institute of Mathematical Sciences, HBNI, India
roohani@imsc.res.in

Meirav Zehavi
Ben-Gurion University, Beersheba, Israel
meiravze@bgu.ac.il

Abstract

Fradkin and Seymour [Journal of Combinatorial Graph Theory, Series B, 2015] defined the class of digraphs of bounded independence number as a generalization of the class of tournaments. They argued that the class of digraphs of bounded independence number is structured enough to be exploited algorithmically. In this paper, we further strengthen this belief by showing that several cut problems that admit sub-exponential time parameterized algorithms (a trait uncommon to parameterized algorithms) on tournaments, including Directed Feedback Arc Set, Directed Cutwidth and Optimal Linear Arrangement, also admit such algorithms on digraphs of bounded independence number. Towards this, we rely on the generic approach of Fomin and Pilipczuk [ESA, 2013], where to get the desired algorithms, it is enough to bound the number of k-cuts in digraphs of bounded independence number by a sub-exponential FPT function (Fomin and Pilipczuk bounded the number of k-cuts in transitive tournaments). Specifically, our main technical contribution is that the yes-instances of the problems above have a sub-exponential number of k-cuts. We prove this bound by using a combination of chromatic coding, an inductive argument and structural properties of the digraphs.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability, Mathematics of computing → Combinatorial algorithms

Keywords and phrases sub-exponential fixed-parameter tractable algorithms, directed feedback arc set, directed cutwidth, optimal linear arrangement, bounded independence number digraph

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.35

1 Introduction

Tournaments form one of the most well studied families of digraphs, both algorithmically and structurally. In particular, whenever we try to generalize results that hold for undirected graphs to digraphs, arguably, one of the first families to consider is that of tournaments.

38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018).
Editors: Sumit Ganguly and Paritosh Pandya; Article No. 35; pp. 35:1–35:19

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

Indeed, this has been the case when designing parameterized algorithms or approximation algorithms. Two problems that have been extensively studied on tournaments are Directed Feedback Vertex Set (DFVS) and Directed Feedback Arc Set (DFAS).

In the realm of approximation, we know that DFVS admits a 7/2-approximation algorithm on tournaments [20], and DFAS admits a PTAS on tournaments [18]. Here, it is worth to point out that whether or not DFAS is NP-complete on tournaments was a well known open problem in the area [4]. First, Ailon et al. [1] proved that unless NP⊆BPP, DFAS admits no polynomial-time algorithm. Shortly afterwards, the proof that DFAS is NP-complete was attained simultaneously and independently by Alon [2] and Charibt et al. [6].

For DFVS on tournaments, the best known parameterized algorithm runs in time $1.618^k \cdot n^{\mathcal{O}(1)}$ [19]. Prior to this the fastest known parameterized algorithm for DFVS ran in time $2^k \cdot n^{\mathcal{O}(1)}$ [10], based on iterative compression. As in the case of approximation, from the viewpoint of Parameterized Complexity, DFAS on tournaments is "easier" than DFVS on tournaments. Here, we mean that for DFAS on tournaments, sub-exponential time parameterized algorithms are known. The quest for sub-exponential time parameterized algorithms for DFAS has a rich history. For a long time (even after the $2^k \cdot n^{\mathcal{O}(1)}$-time algorithm for DFVS was discovered), the question of the existence of an algorithm for DFAS that runs in time $2^k \cdot n^{\mathcal{O}(1)}$ was still being posed as an open problem.

Based on a generic method called chromatic coding (also used in our paper), Alon et al. [3] gave the first sub-exponential time parameterized algorithm for DFAS, which runs in time $2^{\mathcal{O}(\sqrt{k}\log^2 k)} \cdot n^{\mathcal{O}(1)}$. This was the the first problem not confined to planar graphs (or generalizations such as apex-minor-free graphs) that was shown to admit a sub-exponential time parameterized algorithm. Later, simultaneously and independently, Feige [11] and, Karpinski and Schudy [17] gave faster algorithms that run in time $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$. Fomin and Pilipczuk [13] presented a general approach, based on a bound on the number of k-cuts (defined below) in transitive tournaments, that achieved the same running time for DFAS. Using this framework they also designed the first sub-exponential time algorithms for Directed Cutwidth and Optimal Linear Arrangement (OLA) (defined later) on tournaments. Barbero et al. [5] studied Directed Cutwidth and OLA on semi-complete digraphs (that is, digraphs where for any two vertices u and v, at least one of the arcs (u, v) and (v, u) is present) and showed that these problems are NP-complete on semi-complete digraphs. Furthermore, they showed that Directed Cutwidth does not a admit polynomial kernel on semi-complete digraphs but admits a polynomial Turing kernel. Finally, they obtained a linear vertex kernel for OLA on semi-complete digraphs.

The measure of directed cutwidth plays a key role in the work of Chudnovsky and Seymour [8] where it is shown that tournaments are well-quasi-ordered under immersion. This measure was considered by Chudnovsky et al. [7] also in their algorithmic study of Immersion on tournaments. Later, Fradkin and Seymour [14] showed that the Directed Pathwidth and Topological Containment problems on tournaments are fixed parameter tractable (FPT). Fomin and Pilipczuk [12, 13], and Pilipczuk [21] revisited these problems and gave the best known algorithms for them on tournaments. Fradkin and Seymour [15], to generalize their results from tournaments to broader families of graphs, introduced the idea of digraphs that have bounded independence number. In particular, tournaments have independence number 1. They showed that Edge disjoint Paths admits an XP algorithm (that is, an algorithm with running time of the form of $n^{f(k)}$, where n is the number of vertices in the input graph and k is the number of pairs between which one is asked to find edge-disjoint paths) on this family of graphs.

In this paper, we study well-known cut problems (DFAS, Directed Cutwidth and OLA) on digraphs of bounded independence number. Our main contribution is a bound on the number of k-cuts (defined below), which shows that the sub-exponential behavior of these problems on tournaments generalizes to digraphs of bounded independence number.

1.1 Problem statements and our algorithms

For a simple digraph D (every pair of vertices has at most one arc), denote $n = |V(D)|$ and $m = |E(D)|$. Let us formally define the class of digraphs relevant to our work. Given a digraph D, a vertex subset $I \subseteq V(D)$ is called an *independent set* if there are no arcs between any pair of vertices in I. For any positive integer α, let

$$\mathcal{D}_\alpha = \{D \mid \text{ maximum independent set in } D \text{ has size at most } \alpha\}.$$

Observe that for $\alpha = 1$, $\mathcal{D}_\alpha$ is a family of tournaments. *For simplicity, we assume to work with simple digraphs. However, all our results hold also when the digraph is not simple. That is, for any pair of vertices u, v, both the arcs (u, v) and (v, u) can be present in the digraph.* A digraph is a *DAG* (Directed Acyclic Graph) if it has no directed cycles.

We first study the following problem.

Directed Feedback Arc Set (DFAS)	**Parameter:** k
Input: A digraph D and an integer k.	
Question: Does there exist $S \subseteq E(D)$ of size at most k such that $D - S$ is a DAG?	

Our first theorem gives a sub-exponential time algorithm for DFAS on $\mathcal{D}_\alpha$.

▶ **Theorem 1.** DFAS *on $\mathcal{D}_\alpha$ is solvable in time $2^{\mathcal{O}(\alpha^2\sqrt{k}\log(\alpha k))} \cdot n^{\mathcal{O}(\alpha)}$.*

Towards the definition of the second problem, let D be a digraph. For $X, Y \subseteq V(D)$, let $E(X, Y) = \{(u, v) \in E(D) \mid u \in X, v \in Y\}$ denote the set of arcs from X to Y. For an integer q, denote $[q] = \{1, \ldots, q\}$. The *width* of an ordering $(v_1, \ldots, v_n)$ of $V(D)$ is $\max_{i \in [n-1]} |E(\{v_{i+1}, \ldots, v_n\}, \{v_1, \ldots, v_i\})|$. The *cutwidth* of D, denoted by $\mathbf{ctw}(D)$, is the smallest possible width of an ordering of $V(D)$. Now, the second problem is defined as follows.

Directed Cutwidth	**Parameter:** k
Input: A digraph D and an integer k.	
Question: Is $\mathbf{ctw}(D) \leq k$?	

We present a sub-exponential time algorithm for Directed Cutwidth on $\mathcal{D}_\alpha$.

▶ **Theorem 2.** Directed Cutwidth *on $\mathcal{D}_\alpha$ is solvable in time $2^{\mathcal{O}(\alpha^2\sqrt{k}\log(\alpha k))} \cdot n^{\mathcal{O}(\alpha)}$.*

Towards the definition of the third problem, let D be a digraph. For two integers i, j, let $[i > j]$ evaluate to 1 if $i > j$, and to 0 otherwise. The *cost* of an ordering $\sigma = (v_1, \ldots, v_n)$ of $V(D)$ is $\sum_{(v_i, v_j) \in E(D)} (i - j) \cdot [i > j]$. In other words, every arc (v_i, v_j) directed backward in σ costs a value equal to its length, where the length of (v_i, v_j) is the distance between v_i and v_j in σ. Our last problem seeks an ordering of cost at most k.

Optimal Linear Arrangement (OLA)	**Parameter:** k
Input: A digraph D and an integer k.	
Question: Is there an ordering of $V(D)$ of cost at most k?	

Our third theorem gives a sub-exponential time algorithm for OLA on $\mathcal{D}_\alpha$.

▶ **Theorem 3.** OLA *on* $\mathcal{D}_\alpha$ *is solvable in time* $2^{\mathcal{O}(\alpha^2 k^{\frac{1}{3}}\log(\alpha k))} \cdot n^{\mathcal{O}(\alpha)}$.

1.2 Main contribution and methods

Our algorithms are based on the general framework of Fomin and Pilipczuk [13] to design parameterized sub-exponential time algorithms. The main ingredient to prove in order to employ this framework is a combinatorial upper bound on the number of "k-cuts" in graphs that are YES-instances of the problem at hand. The proof for the combinatorial bound in our case is completely different from the proof given by Fomin and Pilipczuk [13] for transitive tournaments. The bound of Fomin and Pilipczuk [13] is achieved by mapping the set of k-cuts in a transitive tournament to the set of partitions of the integer k. Then, an asymptotic bound on the partition number of an integer yields a bound on the number of k-cuts in a transitive tournament. In the case of digraphs with bounded independence number, we do not know how to attain the desired bound by utilizing such partitions of integers.

Before we go further, we define the notion of k-cuts.

▶ **Definition 4** (k-cut)**.** A *k-cut* in a digraph D is a partition of $V(D)$ into two parts L and R (that is, $V(D) = L \uplus R$) such that $|E(R, L)| \leq k$. The k-cut is denoted by the ordered pair (L, R). The set L is called the *left* part of the cut, and the set R is called the *right* part of the cut. The arcs in $E(R, L)$ are the *cut-arcs* of (L, R).

Our first technical contribution is an upper bound on the number of k-cuts in $\mathcal{D}_\alpha$.

▶ **Lemma 5.** *If $D \in \mathcal{D}_\alpha$, then for any positive integer k, the number of k-cuts in D is at most $2^{c\sqrt{k}\log k} \cdot (n+1)^{2\alpha\lceil\sqrt{k}\rceil} \cdot \log n$, where c is a fixed absolute constant.*

The upper bound in Lemma 5 is of the form $n^{\mathcal{O}(f(\alpha)\sqrt{k})}$. That is, it shows that the number of k-cuts in digraphs in $\mathcal{D}_\alpha$ is upper bounded by a sub-exponential function in n. Clearly, such a bound is not sufficient to design sub-exponential time parameterized algorithms. If any of the problems DFAS, DIRECTED CUTWIDTH or OLA on $\mathcal{D}_\alpha$ admits a polynomial kernel, then Lemma 5 can readily yield a sub-exponential time parameterized algorithm for the corresponding problem. However, we do not know whether these problems admit such kernels, and the resolution of these questions remains an interesting open problem.

Our second main technical contribution is an upper bound on the number of k-cuts in a *subfamily* of $\mathcal{D}_\alpha$. This bound suffices to prove Theorems 1, 2 and 3 by embedding it in the framework of Fomin and Pilipczuk [13]. Let us first define this subfamily. Given a vertex $v \in V(D)$, denote the set of out-neighbors of v in D by $N^+_D(v) = \{u \in V(D) \mid (v, u) \in E(D)\}$.

▶ **Definition 6** (d-out-degenerate digraph)**.** For any positive integer d, a digraph D is *d-out-degenerate* if for every subgraph H of D, there exists a vertex $v \in V(H)$ such that $d^+_H(v) \leq d$. An ordering $(v_1, \ldots, v_n)$ of the vertex set of D is a *d-out-degeneracy sequence* of D if for any $i \in \{2, \ldots, n\}$, $|N(v_i) \cap \{v_j \mid j < i\}| \leq d$.

Observe that a digraph is d-out degenerate if and only if it has a d-out-degeneracy sequence, that is there is an ordering of the vertex set of the digraph such that each vertex has at most d edges to the vertices before it. Also observe that DAGs are 0-out-degenerate. Next, we define a class of digraphs having small independence number and bounded out-degeneracy. Formally, $\mathcal{D}_{\alpha,d} = \{D \mid D \in \mathcal{D}_\alpha$ and D is d-out-degenerate$\}$. Note that if $D \in \mathcal{D}_{\alpha,d}$, then every induced subgraph D' of D belongs to $\mathcal{D}_{\alpha,d}$. Our second main technical contribution is formally stated as follows.

▶ **Lemma 7.** *If $D \in \mathcal{D}_{\alpha,d}$, then for any positive integer k, the number of k-cuts in D is at most $2^{c(\alpha+1)\sqrt{k}\log k} \cdot (d+1+\alpha(2k+1))^{2\alpha(\alpha+1)\lceil\sqrt{k}\rceil} \cdot \log(d+\alpha(2k+1)) \cdot n^{\alpha+1}$, where c is a fixed absolute constant.*

One can easily see that if (D,k) is a YES-instance of DFAS, DIRECTED CUTWIDTH or OLA, then D is k-out-degenerate. Thus, Lemma 7 implies a sub-exponential (in k) upper bound on the number of k-cuts for YES-instances of these problems. In fact, for OLA one can show that D is $2k^{2/3}$-out-degenerate, and thus obtain an improved upper bound on the number of k-cuts for YES-instances. Since the k-cuts of any digraph can be enumerated with polynomial delay [13], hence the upper bounds in Lemmas 5 and 7 are constructive.

In what follows, we present our proof strategies for the results stated above.

Proof Strategy of Lemma 5. We first make a very simple observation, which serves as the starting point of our proof. Let $V(D) = V_1 \uplus \cdots \uplus V_\ell$ be some partition of $V(D)$. Then, the number of k-cuts in D is upper bounded by the product of the number of k-cuts in the digraph induced by each V_i. Thus, we aim to partition $V(D)$ into parts that induce "sufficiently structured" subdigraphs – we want the number of k-cuts in $D[V_i]$, for any $i \in [\ell]$, to be "easier" to upper bound than the number of k-cuts in D directly. Moreover, since our aim is to achieve a bound of $n^{o(k)}$ for the total number of k-cuts in D, we want a partition $V(D) = V_1 \uplus \cdots \uplus V_\ell$ where $\ell = o(k)$. To this end, we utilize Gallai-Milgram's Theorem (explained next) under the canvas of chromatic coding.

On the one hand, Gallai-Milgram's Theorem states that if the size of a maximum independent set in a digraph is α, then its vertex set can be partitioned into at most α parts such that the digraph induced by each of these parts has a directed Hamiltonian path. On the other hand, chromatic coding (in its derandomized form) provides a family $\mathcal{F}$ of partitions of $V(D)$ such that *(i)* $|\mathcal{F}| = 2^{o(k)}\log n$, *(ii)* for each k-cut (L,R) in D, there exists a partition $\mathcal{P} \in \mathcal{F}$ such that all the cut arcs of (L,R) go across the parts of $\mathcal{P}$, and *(iii)* the number of parts of each partition in $\mathcal{F}$ is upper bounded by $\mathcal{O}(\sqrt{k})$. If the cut-arcs of (L,R) go across the parts of a partition $\mathcal{P}$, we say that (L,R) *respects* $\mathcal{P}$. To see how to combine these two tools, let $\mathcal{F}$ be a family provided by chromatic coding. Since the number of partitions in $\mathcal{F}$ is $2^{o(k)}\log n$, and for each k-cut (L,R) there exists a partition in $\mathcal{F}$ that it respects, it suffices to bound the number of k-cuts that respect a particular (arbitrary) partition in $\mathcal{F}$. Then, the total number of k-cuts in the digraph will be the product of the number of k-cuts that respect a partition in $\mathcal{F}$, over all partitions in $\mathcal{F}$.

Consider an arbitrary partition $\mathcal{P} \in \mathcal{F}$ (of $V(D)$). Let $\mathcal{P} = P_1 \uplus \cdots \uplus P_\ell$. Recall that $\ell = \mathcal{O}(\sqrt{k})$, and the number of k-cuts in D is at most the product of the number of k-cuts in $D[P_i]$, over all $i \in [\ell]$. Here, a crucial insight is that the number of k-cuts in D that respect $\mathcal{P}$ is at most the product of the number of 0-cuts in $D[P_i]$, over all $i \in [\ell]$. Thus, we have reduced our problem to upper bounding the number of 0-cuts in a digraph. Now, to upper bound the number of 0-cuts in $D[P_i]$ by $n^{o(k)}$, we utilize Gallai-Milgram's Theorem. Since $D[P_i] \in \mathcal{D}_\alpha$, Gallai-Milgram's Theorem implies that P_i can be partitioned into at most α parts, say $P_i = P_{i1} \uplus \ldots \uplus P_{iq}$, $q \leq \alpha$, such that for each $j \in [q]$, $D[P_{ij}]$ has a directed Hamiltonian path. Thus, we have finally reduced our problem to finding 0-cuts in digraphs that have a directed Hamiltonian path. As we will see later, the number of 0-cuts in such digraphs is linear in its number of vertices. Combining everything together, we are able to bound the number of k-cuts in D by $n^{\mathcal{O}(\alpha\sqrt{k})}$.

Proof Strategy of Lemma 7. Each vertex in a digraph D has two choices of how to participate in a cut – it can belong either to its left side or to its right side. Thus, if $|V(D)| = n$, a trivial upper bound on the total number of k-cuts in D is 2^n. Suppose that

we have (somehow) reached a "situation" where most of the vertices *must* belong to only one of the sides of a k-cut. Then, the arguments to attain the 2^n bound imply that the number of k-cuts is at most 2^q, where q is the number of vertices which possibly have both choices. By the bound in Lemma 5, we can further conclude that the number of k-cuts is, in fact, at most $q^{\mathcal{O}(\alpha\sqrt{k})}$. Thus, if $q = k^{\mathcal{O}(1)}$ (that is, only $k^{\mathcal{O}(1)}$ vertices can choose a side), we get a bound of $2^{o(k)}$.

On a different note, suppose that we can identify a set of vertices in D, say V_1, such that $D[V_1]$ has at most $2^{o(k)}$ k-cuts. If V_1 is large enough, say $|V_1|$ is such that $|V(D) \setminus V_1| = k^{\mathcal{O}(1)}$, then we can bound the number of k-cuts in $D[V(D) \setminus V_1]$ by $2^{o(k)}$ (by Lemma 5). Since the number of k-cuts in D is bounded by the product of the number of k-cuts in $D[V_1]$ and the number of k-cuts in $D[V(D) \setminus V_1]$, we attain the bound of $2^{o(k)}$ on the number of k-cuts in D.

Our algorithm combines the two ideas above to obtain the desired bound. For any vertex $v \in V(D)$, we aim to bound the number of k-cuts in D where v is "forced" to belong to the left part. We exploit the position of v in a fixed d-out-degeneracy sequence of D to conclude that a large number of vertices are forced to belong to one side of these cuts. Then, building on the second idea, we inductively find a set of vertices such that the digraph induced on it has independence number strictly smaller than the independence number of D. For such a set of vertices, we can inductively assume that the number of k-cuts in the digraph induced by them is $2^{o(k)}$. Having this bound at hand, we are able to conclude the proof.

Proof Strategy of Theorems 1, 2 and 3. To obtain sub-exponential FPT algorithms for DFAS, Directed Cutwidth and OLA on $\mathcal{D}_\alpha$, we first use Lemma 7 to bound the number of k-cuts in the digraphs of the Yes-instances of these problems by $2^{o(k)}$. Here, we rely on the observation that these digraphs must be k-out-degenerate. Though we do not explicitly state this, the procedures to bound the number of k-cuts in both Lemmas 5 and 7 are constructive. However, constructiveness is not necessary since a standard branching procedure can also enumerate all k-cuts in a digraph with polynomial delay [13]. To actually solve any of the three problem, we design a dynamic programming procedure over the k-cuts.

The last two steps of this proof (namely, the enumeration and the dynamic programming procedures) are quite standard, based on the work by Fomin and Pilipczuk [13] to obtain subexponential FPT algorithms for DFAS, Directed Cutwidth and OLA on tournaments. Since the proofs of Theorems 1, 2 and 3 are standard and resemble [13], we defer them to the appendix.

1.3 Preliminaries

For any $i, j \in \mathbb{Z}^+$, denote $[i] = \{1, \ldots, i\}$, $[i]_0 = \{0, 1, \ldots, i\}$ and $[i, j] = \{i, i+1, \ldots, j-1, j\}$. For a partition $\mathcal{P} = P_1 \uplus \cdots \uplus P_\ell$, each P_i is referred to as a *part* of $\mathcal{P}$. For a digraph D, $V(D)$ denotes its vertex set and $E(D)$ its arc set. We write $(u, v) \in E(D)$ if there is an arc in D with u as its tail and v as its head. Given a vertex $v \in V(D)$, the set of in-neighbors of v in D, denoted by $N_D^-(v)$, is the set of all vertices $u \in V(D)$ such that $(u, v) \in E(D)$. The set of out-neighbors of v in D, denoted by $N_D^+(v)$, is the set of all vertices $u \in V(D)$ such that $(v, u) \in E(D)$. The set of neighbors of v, denoted by $N_D(v)$, is the union of $N_D^-(v)$ and $N_D^+(v)$. For a set $X \subseteq V(D)$, we let $N_X^-(v)$ denote the set of in-neighbors of v in X, that is, $N_X^-(v) = N_D^-(v) \cap X$ (respectively, $N_X^+(v) = N_D^+(v) \cap X$, $N_X(v) = N_D(v) \cap X$). Similarly, $N_X^+(v) = N_D^+(v) \cap X$ and $N_X(v) = N_D(v) \cap X$. Whenever the digraph is clear from the context, we drop the subscript D. For $X, Y \subseteq V(D)$, $E(X, Y) = \{(u, v) \mid (u, v) \in E(D), u \in X \text{ and } v \in Y\}$ denotes the set of arcs from X to Y. By $D[X]$, we denote the directed subgraph induced by the vertices of X. A set $X \subseteq V(D)$ is called an *independent set* of D if

for any $u, v \in X$, $(u, v) \notin E(D)$ and $(v, u) \notin E(D)$. In other words, X is an independent set in the underlying undirected graph of D. The independence number of a digraph is equal to the size of the maximum independent set it contains. A *directed Hamiltonian path* in D is a directed simple path on all vertices in D. For a set of vertices $\{v_1, \ldots, v_n\}$, let $(v_1, \ldots, v_n)$ denote the ordering where for any $i \in [n]$, v_i is the ith vertex of the ordering.

2 Bounding the number of k-cuts for digraphs in $\mathcal{D}_\alpha$

In this section, we prove that the number of k-cuts in any digraph on n vertices with bounded independence number is $n^{o(k)}$. In particular we prove Lemma 5. Let us recall that a k-cut in a directed graph D is a partition of the vertex set of D into two parts, $V(D) = L \uplus R$, such that $|E(R, L)| \leq k$. Let us note that a 0-cut in a digraph D is a partition (L, R) of the vertex set $V(D)$ such that there are no arcs from R to L in D.

At the heart of the proof of Lemma 5 is a simple observation that helps us focus on parts of the digraph for which bounding the number of k-cuts is easier. This simple observation is then exploited to its fullest using two main tools - (1) the Gallai-Milgram's Theorem and (2) chromatic coding. Let us state them formally. We begin by stating this key observation, followed by formally defining both these ideas.

▶ **Lemma 8.** *Let D be a digraph and $k \in \mathbb{Z}^+$. Let $V(D) = V_1 \uplus \ldots \uplus V_q$ be some partition of $V(D)$. For any $i \in [q]$, let N_i be the number of k-cuts in $D[V_i]$, then the number of k-cuts in D is at most $\prod_{i \in [q]} N_i$.*

Proof. To prove the lemma, observe that, it is enough to prove that for any k-cut (L, R) in D, there exists k-cuts (L_i, R_i), for each $i \in [q]$, in $D[V_i]$, such that $L = \cup_{i \in [q]} L_i$ and $R = \cup_{i \in [q]} R_i$. To see this, for any $i \in [q]$, let $L_i = L \cap V_i$ and $R_i = R \cap V_i$. Observe that, each (L_i, R_i) is a k-cut in $D[V_i]$, otherwise (L, R) is not a k-cut in D. ◀

Thus, if we can partition the vertex set of D into $o(k)$ parts such that it is "easier" to bound the number of k-cuts in each of these parts, then we are done. At a high level, we will first partition the vertex set of D using chromatic coding, and then further partition each part of this partition using Gallai-Milgram's Theorem. We will then conclude by proving that the number of k-cuts in each of the sub-parts is linear in the number of vertices. We now state the Gallai-Milgram's Theorem formally.

▶ **Proposition 9** ([16], Gallai-Milgram Theorem, 1960)**.** *For any $\alpha \in \mathbb{Z}^+$ and $D \in \mathcal{D}_\alpha$, there exists a partition of $V(D) = V_1 \uplus \ldots \uplus V_q$, such that $q \leq \alpha$ and for each $i \in [q]$, $D[V_i]$ has a directed Hamiltonian path.*

Next, we state the technique of chromatic coding in its derandomized form. To this end, we first define *universal (n, k, r)-coloring family* and then state the known results about the existence of such a families of bounded size. This result is called the chromatic coding lemma. For any graph G, a *proper vertex coloring* of G is a function $f : V(G) \to \mathbb{Z}^+$, such that for any $(u, v) \in E(G)$, $f(u) \neq f(v)$.

▶ **Definition 10** ([3], Universal (n, k, r)-Coloring Family)**.** For integers n, k and r, a family $\mathcal{H}$ of functions from $[n]$ to $[r]$ is called a universal (n, k, r)-coloring family, if for any graph G on the vertex set $[n]$ with at most k edges, there exists an $h \in \mathcal{H}$ which is a proper vertex coloring of G.

Observe that the above mentioned definition holds for digraphs too, where the notion of proper vertex coloring is defined on its underlying undirected graph.

▶ **Proposition 11** ([3], Chromatic Coding Lemma)**.** *For any $n, k \geq 1$, there exists a universal $(n, k, 2\lceil\sqrt{k}\rceil)$-coloring family of size at most $2^{\mathcal{O}(\sqrt{k}\log k)} \cdot \log n$.*

A formulation of the Chromatic Coding lemma, in the way that is useful to us, can be seen in the following corollary.

▶ **Corollary 12.** *For any digraph D on n vertices, and an integer k, there exists a family $\mathcal{F}$ of partitions of $V(D)$ into at most $2\sqrt{k}$ parts, such that,*

1. *for any k-cut (L, R) in D, there exists a partition $\mathcal{P} = \{P_1, \ldots, P_q\}$ in the family $\mathcal{F}$, such that for any cut-arc (u, v) of (L, R), there exists $i, j \in [q]$, $i \neq j$, such that $u \in P_i$ and $v \in P_j$, and*
2. *$|\mathcal{F}| = 2^{\mathcal{O}(\sqrt{k}\log k)} \cdot \log n$.*

Proof. Let $\mathcal{H}$ be a $(n, k, 2\lceil\sqrt{k}\rceil)$-universal coloring family from Proposition 11, of size at most $2^{\mathcal{O}(\sqrt{k}\log k)} \cdot \log n$. We construct a family $\mathcal{F}$ of partitions of $V(D)$ from the family $\mathcal{H}$ as follows. For each $h \in \mathcal{H}$, there is a partition $\mathcal{P}_h = P_1 \uplus \cdots \uplus P_{\lceil 2\sqrt{k}\rceil}$ in $\mathcal{F}$, where for any $i \in [2\lceil\sqrt{k}\rceil]$, $P_i = h^{-1}(i)$. Here, if for a certain i, $P_i = \emptyset$, then we discard this part from the partition $\mathcal{P}_h$.

We will now show that $\mathcal{F}$ is indeed the family with the required properties. Since $|\mathcal{H}| = 2^{\mathcal{O}(\sqrt{k}\log k)} \cdot \log n$, clearly $|\mathcal{F}| = 2^{\mathcal{O}(\sqrt{k}\log k)} \cdot \log n$. Let (L, R) be some k-cut in D. Consider the digraph, say $D_{(L,R)}$, on the vertex set of D with only the cut-arcs of (L, R). Note that $|E(D_{(L,R)})| \leq k$. Thus, from the definition of $(n, k, 2\lceil\sqrt{k}\rceil)$-universal coloring family, there exists a function $h : V(D_{(L,R)}) \to [2\lceil 2\sqrt{k}\rceil]$ in $\mathcal{H}$, such that h is a proper vertex coloring of $D_{(L,R)}$. Consider the partition $\mathcal{P}_h \in \mathcal{F}$. Let $\mathcal{P}_h = P_1 \uplus \cdots \uplus P_{2\lceil\sqrt{k}\rceil}$. Since h is a proper coloring of $D_{(L,R)}$ and all the cut-arcs of (L, R) are in $D_{(L,R)}$, for any cut-arc (u, v) of (L, R), $h(u) \neq h(v)$. Thus, if $h(u) = i$ and $h(v) = j$, $i \neq j$, then $u \in P_i$ and $v \in P_j$. ◀

For the rest of this section, let $\mathcal{F}$ denote the family described in Corollary 12 for the digraph D and integer k. For any arc (u, v) of a digraph and a partition $\mathcal{P} = P_1 \uplus \cdots \uplus P_q$ of the vertex set of the digraph, we say that the arc (u, v) *goes across the parts of this partition* $\mathcal{P}$, if $u \in P_i$, $v \in P_j$ and $i \neq j$. For any partition $\mathcal{P}$ of the vertex set of the digraph D, we say that a k-cut (L, R) in D *respects* $\mathcal{P}$ if all the cut-arcs of (L, R) go across the parts of $\mathcal{P}$. The next lemma states that, the number of k-cuts in D is at most the sum of the number of k-cuts that respect a partition $\mathcal{P}$, over all partitions $\mathcal{P} \in \mathcal{F}$. Since $|\mathcal{F}| = 2^{o(k)}$, it is enough to bound the number of k-cuts that respect an arbitrary partition in $\mathcal{F}$ by $n^{o(k)}$. For the digraph D, an integer k and $\mathcal{P} \in \mathcal{F}$, let $N_{\mathcal{P}}$ be the number of k-cuts in D that respect $\mathcal{P}$.

▶ **Lemma 13.** *The total number of k-cuts in D is at most $\sum_{\mathcal{P}\in\mathcal{F}} N_{\mathcal{P}}$.*

Proof. To prove the lemma, we need to prove that for any k-cut (L, R) in D, there exists $\mathcal{P} \in \mathcal{F}$ such that (L, R) respects $\mathcal{P}$. This follows from Corollary 12. ◀

Henceforth, let us fix $\mathcal{P} = P_1 \uplus \cdots \uplus P_q$, $q \leq 2\lceil\sqrt{k}\rceil$, where $\mathcal{P}$ is an arbitrary partition in $\mathcal{F}$. We are now only interested in bounding the number of k-cuts in D that respect $\mathcal{P}$. It follows from Lemma 8, that to bound the total number of k-cuts in D, it is sufficient to bound the number of k-cuts in $D[P_i]$, for each $i \in [q]$. We now have the following lemma, that says something much stronger. To bound the number of k-cuts in D that respect $\mathcal{P}$, it is sufficient to bound the number of just the 0-cuts in $D[P_i]$, for all $i \in [q]$.

▶ **Lemma 14.** *For any digraph D, let $\mathcal{P} = P_1 \uplus \ldots \uplus P_q$ be some partition of the vertex set of D. For any $i \in [q]$, let N_i be the number of 0-cuts in $D[P_i]$. Then the number of k-cuts in D that respect $\mathcal{P}$ is at most $\prod_{i\in[q]} N_i$.*

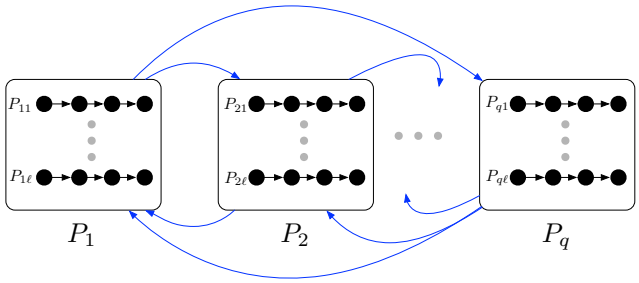


Figure 1 The Vertex Partition for the Sub-exponential XP bound. $\mathcal{P} = \{P_1 \uplus \cdots \uplus P_q\}$ is the vertex partition obtained using chromatic coding and $P_i = P_{i1} \uplus \cdots \uplus P_{i\ell}$ is the partition obtained using Gallai-Milgram's Theorem. Each P_{ij} contains a Directed Hamiltonian Path. The cut arcs of all the cuts that respect $\mathcal{P}$ are marked in blue.

Proof. Observe that to prove the lemma it is enough to prove that for any k-cut (L, R) of D that respects $\mathcal{P}$, there exists 0-cuts (L_i, R_i) in $D[P_i]$, for each $i \in [q]$ such that $L = \cup_{i \in q} L_i$ and $R = \cup_{i \in [q]} R_i$. Let (L, R) be some k-cut in D that respects $\mathcal{P}$. For each $i \in [q]$, let $L_i = L \cap P_i$ and $R_i = R \cap P_i$. Observe that, for each $i \in [q]$, (L_i, R_i) is a 0-cut in $D[P_i]$. Suppose not. Then there exists a cut-arc of (L_i, R_i), say (u, v), such that $u, v \in P_i$ and $u \in R_i$, $v \in L_i$. Since $L = \cup_{i \in [q]} L_i$ and $R = \cup_{i \in [q]} R_i$, $u \in R$ and $v \in L$. This contradicts that (L, R) respects $\mathcal{P}$. ◀

Thus, we have further narrowed down the class of k-cuts that we want to bound. More precisely, we are now interested in bounding the number of 0-cuts in $D[P_i]$, for any part P_i of $\mathcal{P}$. Since $D \in \mathcal{D}_\alpha$, for any $P_i \in \mathcal{P}$, $D[P_i] \in \mathcal{D}_\alpha$. Thus, from Gallai-Milgram Theorem, the vertex set of P_i can be partitioned into at most α parts, say $P_i = P_{i1} \uplus \cdots \uplus P_{i\ell}$, $\ell \leq \alpha$, such that for each $j \in [\ell]$, $D[P_{ij}]$ has a directed Hamiltonian path. We will now prove that for any digraph that has a directed Hamiltonian path, the number of 0-cuts in it are linear in the number of its vertices.

▶ **Lemma 15.** *Let D be a digraph on n vertices that has a directed Hamiltonian path. Then the number of 0-cuts in D is $n + 1$.*

Proof. Since D has a directed Hamiltonian path, let $\{v_1, \ldots, v_n\}$ be the vertex set of D such that for each $i \in [n-1]$, $(v_i, v_{i+1}) \in E(D)$. Consider any 0-cut (L, R) in D. Let i be the smallest integer such that $v_i \in R$. By the choice of i, for all $j < i$, $v_j \in L$. We now claim that, for all $j > i$, $v_j \in R$. Suppose not. Then there exist a $j > i$, such that $v_j \in L$. Since $j > i$, and v_i appears before v_j in the Hamiltonian path ordering. Thus, there is a directed path from v_i to v_j in D. Since $v_i \in R$ and $v_j \in L$, an arc of this directed path is a cut-arc for (L, R), which contradicts that (L, R) is a 0-cut.

Thus, for any $i \in [n]$, the number of 0-cuts in D where v_i is the first vertex in the ordering $(v_1, \ldots, v_n)$ that belongs to the right part of these cuts is exactly 1. Since any cut in D, either does not contain any vertex in its right part (there is only one such cut) or contains some vertex, the total number of 0-cuts in D is $n + 1$. ◀

We are now ready to prove Lemma 5. An illustration depicting the partitioning used in the proof of Lemma 5 is given in Figure 1.

Proof of Lemma 5. Let N be the total number of k-cuts in D. Consider the family $\mathcal{F}$ of Corollary 12 for the digraph D and integer k. From Corollary 12, $|\mathcal{F}| \leq 2^{\mathcal{O}(\sqrt{k} \log k)} \cdot \log n$. For each partition $\mathcal{P} \in \mathcal{F}$, let $N_{\mathcal{P}}$ be the number of k-cuts in D that respect $\mathcal{P}$. From

Lemma 13, $N \leq \sum_{\mathcal{P} \in \mathcal{F}} N_{\mathcal{P}}$. Consider any arbitrary partition $\mathcal{P} \in \mathcal{F}$. Let $\mathcal{P} = P_1 \uplus \ldots \uplus P_q$, and from Corollary 12 we have $q \leq 2\lceil\sqrt{k}\rceil$. For any $i \in [q]$, let N_{P_i} be the number of 0-cuts in $D[P_i]$. From Lemma 14, $N_{\mathcal{P}} \leq \prod_{i \in [q]} N_{P_i}$. Since $D \in \mathcal{D}_\alpha$, for any P_i, $D[P_i] \in \mathcal{D}_\alpha$. Thus, from Gallai-Milgram Theorem, the vertex set of P_i can be partitioned into at most α parts, say $P_i = P_{i1} \uplus \ldots \uplus P_{i\ell}$, $\ell \leq \alpha$, such that such that for each $j \in [\ell]$, $D[P_{ij}]$ has a directed Hamiltonian path. From Lemma 15, the number of 0-cuts in $D[P_{ij}]$ is $n+1$. From Lemma 8, $N_{P_i} \leq \prod_{j \in [\ell]}(n+1) \leq (n+1)^\ell \leq (n+1)^\alpha$. Combining everything stated above, we get that, $N \leq |\mathcal{F}| \cdot N_{\mathcal{P}} \leq |\mathcal{F}| \cdot (N_{P_i})^{2\lceil\sqrt{k}\rceil} \leq |\mathcal{F}| \cdot (n+1)^{2\alpha\lceil\sqrt{k}\rceil} \leq 2^{\mathcal{O}(\sqrt{k}\log k)} \cdot (n+1)^{2\alpha\lceil\sqrt{k}\rceil} \cdot \log n)$. ◀

3 Improved bounds for digraphs in $\mathcal{D}_\alpha$ with bounded out-degeneracy

In this section we give the proof of Lemma 7. Recall from the introduction that a digraph D is said to be d-out-degenerate, if for every subgraph H of D, there exists a vertex $v \in V(H)$, such that $d^+_H(v) \leq d$. Furthermore, a digraph D d-out degenerate if and only if it has a d-out-degeneracy sequence.

Throughout this section, D is a digraph on n vertices and $D \in \mathcal{D}_{\alpha,d}$. Let $(v_1, \ldots, v_n)$ be a d-out-degeneracy sequence of D. For any $i \in [n]$, we say that a k-cut (L, R) in D is of *type-i*, if $v_i \in L$ and for all $j > i$, $v_j \in R$. We say that a k-cut (L, R) in D is of *type*-0 if $L = \emptyset$. Note that the collection of the sets of type-i cuts for all $i \in [n]_0$, forms a partition of the set of all the k-cuts. Observe that there is exactly 1 type-0 cut in any digraph.

▶ **Observation 16.** *For any $i \in [n]_0$, let N_i be the number of k-cuts in D of type-i. Then the number of k-cuts in D is at most $\sum_{i \in [n]_0} N_i$.*

Henceforth, our goal is to bound the number of k-cuts in D of type-i, for an arbitrary $i \in [n]$. Recall from Lemma 8 that if $V(D) = V_1 \uplus \cdots \uplus V_c$ is a partition of the vertex set of D, then to bound the number of k-cuts in D, it is enough to bound the number of k-cuts in each $D[V_j]$, $j \in [c]$. This remains our underlying strategy. However, this time we use a different partition of the vertex set of D, where the number of parts of this partition is 4, compared to $o(k)$ in Lemma 5. This partition of the vertex set, is presented in Lemma 17.

▶ **Lemma 17.** *For a digraph $D \in \mathcal{D}_{\alpha,d}$ and any positive integer k, for any fixed $i \in [n]$, there exists a partition $V(D) = V_{\text{induct}} \uplus V_{\text{forceL}} \uplus V_{\text{forceR}} \uplus V_{\text{small}}$ such that:*

1. *If $\alpha = 1$, then $V_{\text{induct}} = \emptyset$, otherwise $D[V_{\text{induct}}] \in \mathcal{D}_{\alpha',d}$, where $\alpha' < \alpha$.*
2. *For any k-cut (L, R) in D of type-i, $V_{\text{forceL}} \subseteq L$.*
3. *For any k-cut (L, R) in D of type-i, $V_{\text{forceR}} \subseteq R$.*
4. *$|V_{\text{small}}| \leq d + \alpha(2k+1)$.*

Lemma 17 states that the vertex set of D can be partitioned into 4 parts with the following properties. The digraph induced on the *first* part is either empty or belongs to $\mathcal{D}_{\alpha',d}$, for $\alpha' < \alpha$. To bound the number of k-cuts in such a digraph we will use an induction on α. For the *second* part of this partition, we prove that for any k-cut (L, R) of type-i, all the vertices of this part belong to L. Similarly, for the *third* part of this partition, we prove that for any k-cut (L, R) of type-i, all the vertices of this part belong to R. Therefore, there is a unique k-cut of type-i in the digraph induced by the second and third part. The *last* part of the partition has the property that the number of vertices in this part is "small". For the digraph induced by this part, we will get the desired bound by using Lemma 5 on this digraph.

The proof of Lemma 17 is deferred for later. We will now proceed towards the proof of Lemma 7 using Lemma 17 and induction on α. At any inductive step we use the partition of Lemma 17 and bound the number of k-cuts of type-i in the digraph induced on each part of the partition, thereby bounding the number of k-cuts in D because of Observation 16.

Proof of Lemma 7. We prove the lemma using induction on α. For any positive integer α, let us denote the bound of Lemma 5 on the number of k-cuts in $D \in \mathcal{D}_\alpha$, on at most $d+\alpha(2k+1)$ vertices, by $\eta(\alpha, d, k)$. That is, $\eta(\alpha, d, k) = 2^{c\sqrt{k}\log k} \cdot (d+1+\alpha(2k+1))^{2\alpha\lceil\sqrt{k}\rceil} \cdot \log(d+\alpha(2k+1))$, where c is the absolute constant hidden in the $\mathcal{O}$ notation of the expression in Proposition 11. Let $\mathcal{N}_k(n, \alpha, d)$ denote the maximum number of k-cuts in D for any digraph $D \in \mathcal{D}_{\alpha,d}$ on n vertices. We claim that for any positive integers n, d and $\alpha > 1$, $\mathcal{N}_k(n, 1, d) \leq 1 + n \cdot \eta(1, d, k)$ and $\mathcal{N}_k(n, \alpha, d) \leq 1 + \mathcal{N}_k(n, \alpha-1, d) \cdot \eta(\alpha, d, k) \cdot n$. Solving the recurrence, we will get the desired bound on the number of k-cuts in D.

Let us first prove that for any positive integers n and d, $\mathcal{N}_k(n, 1, d) \leq 1 + n \cdot \eta(1, d, k)$. If the independence number of the digraph D is 1, then from Lemma 17, there exists a partition $V(D) = V_{\text{forceL}} \uplus V_{\text{forceR}} \uplus V_{\text{small}}$ of D such that for any k-cut (L, R) in D of type-i, $V_{\text{forceL}} \subseteq L$ and $V_{\text{forceR}} \subseteq R$. Thus, from Lemma 8, we conclude that the number of k-cuts of type-i in D is at most the number of k-cuts in $D[V_{\text{small}}]$. Since $D[V_{\text{small}}]$ is an induced subgraph of D, the independence number of $D[V_{\text{small}}]$ is at most α and $D[V_{\text{small}}]$ is a digraph on $d + 2k + 1$ vertices. Thus, we conclude that the number of k-cuts in D of type-i are at most $\eta(1, d, k)$. From Observation 16, we conclude that the number of k-cuts in D are at most $1 + \eta(1, d, k) \cdot n$.

By induction hypothesis, let us assume that for any positive integers n, d and for all $\alpha' < \alpha$, the number of k-cuts in any digraph $D' \in \mathcal{D}_{\alpha',d}$ on at most n vertices is $\mathcal{N}_k(n, \alpha', d)$. We will now prove that the number of k-cuts in the digraph $D \in \mathcal{D}_{\alpha,d}$ is $\mathcal{N}_k(n, \alpha, d) \leq 1 + \mathcal{N}_k(n, \alpha-1, d) \cdot \eta(\alpha, d, k)$. From Lemma 17, there exists a partition $V(D) = V_{\text{induct}} \uplus V_{\text{forceL}} \uplus V_{\text{forceR}} \uplus V_{\text{small}}$, such that for any k-cut (L, R) in D of type-i, $V_{\text{forceL}} \subseteq L$ and $V_{\text{forceR}} \subseteq R$. Thus, from Lemma 8, the number of k-cuts of type-i in D is at most the product of the number of k-cuts in $D[V_{\text{induct}}]$ and the number of k-cuts in $D[V_{\text{small}}]$. Since $D[V_{\text{induct}}] \in \mathcal{D}_{\alpha',d}$, where $\alpha' < \alpha$, from inductive hypothesis we get that the number of k-cuts in $D[V_{\text{induct}}]$ is at most $\mathcal{N}_k(n, \alpha', d) \leq \mathcal{N}_k(n, \alpha-1, d)$. Since $|V_{\text{small}}| \leq d + \alpha(2k+1)$, from Lemma 5, the number of k-cuts in $D[V_{\text{small}}]$ is at most $\eta(\alpha, d, k)$. Thus, the number of k-cuts of type-i in D is at most $\mathcal{N}_k(n, \alpha-1, d) \cdot \eta(\alpha, d, k)$. From Observation 16, we conclude that the number of k-cuts in D is at most $1 + \mathcal{N}_k(n, \alpha-1, d) \cdot \eta(\alpha, k, d) \cdot n$. ◀

Proof of Partitioning Lemma. We start by a lemma that gives an upper bound on the size of a digraph in $\mathcal{D}_\alpha$ when every vertex has small out-degree.

▶ **Lemma 18.** *For any digraph $D \in \mathcal{D}_\alpha$ and a positive integer k, if for all $v \in V(D)$, $d^+(v) \leq k$, then $|V(D)| \leq \alpha(2k+1)$.*

Proof. Let $|V(D)| = n$. We will first prove that if $D \in \mathcal{D}_\alpha$, then there exists $v \in V(D)$ such that $d^+(v) \geq \frac{(n-\alpha)}{2\alpha}$. Since $d^+(v) \leq k$, for all $v \in V(D)$, this implies that $\frac{(n-\alpha)}{2\alpha} \leq k$, thereby implying that $n \leq \alpha(2k+1)$.

To prove the above-mentioned claim, we invoke Turan's Theorem ([9]), which states that for any graph G and integer r, if G does not contain a clique of size $r+1$, then $|E(G)| \leq (1 - \frac{1}{r}) \cdot \frac{|V(G)|^2}{2}$. Let G be the underlying undirected graph of D. Let $\bar{G}$ be the complement graph of G. Since $D \in \mathcal{D}_\alpha$, $\bar{G}$ does not contain a clique of size $\alpha+1$. Thus, by Turan's Theorem, $|E(\bar{G})| \leq (1 - \frac{1}{\alpha}) \cdot \frac{n^2}{2}$. Since $\bar{G}$ is the complement graph of G, $|E(G)| \geq \frac{n(n-1)}{2} - (1 - \frac{1}{\alpha}) \cdot \frac{n^2}{2} \geq \frac{(n^2-n\alpha)}{2\alpha}$. Since G is the underlying undirected graph of D, $|E(D)| \geq \frac{(n^2-n\alpha)}{2\alpha}$. Since $|E(D)| = \sum_{v \in V(D)} d^+(v) \geq \frac{(n^2-n\alpha)}{2\alpha}$, there exists $v \in V(D)$, such that $d^+(v) \geq \frac{(n-\alpha)}{2\alpha}$. ◀

Intuitive Ideas for the proof Lemma 17. Let us begin by recalling that $(v_1, \ldots, v_n)$ is a d-out-degeneracy sequence of D. Also recall that, the aim of proving Lemma 17 is to be able to use it to bound the number of k-cuts in D of type-i. Consider any k-cut in D of type-i. By definition, $v_i \in L$ and for all $j > i$, $v_j \in R$. Thus, $v_i \in V_{\text{forceL}}$ and $\{v_j \mid j > i\} \subseteq V_{\text{forceR}}$. Thus, to prove Lemma 17, we essentially need to partition the vertices that appear before v_i in $(v_1, \ldots, v_n)$. Consider the non-neighbors of v_i. They induce a digraph whose independence number is strictly less than the independence number of D. Thus, they go to V_{induct}. Thus, we are now left with the goal of partitioning the set of neighbors of v_i that appear before v_i in $(v_1, \ldots, v_n)$. Since $(v_1, \ldots, v_n)$ is a d-out-degeneracy sequence of D, the number of out-neighbors of v_i that appear before v_i in $(v_1, \ldots, v_n)$ is at most d. This set of neighbors goes to the set V_{small}. Finally, we are left with the set, say X, of vertices that appear before v_i in $(v_1, \ldots, v_n)$ and are in-neighbors of v_i. Here, we observe that, if any vertex $v \in X$ has out-degree at least $k+1$ in $D[X]$, then there are at least $k+1$ arc-disjoint paths from v to v_i in $D[X \cup \{v_i\}]$, and hence in D. Thus, such a vertex v should always belong to same part as v_i in any k-cut. Thus, such vertices goes to V_{forceL}. Finally, the remaining vertex set, say X', has the property that each vertex in X has out-degree at most k. By Lemma 18, in such a case the size of X' is at most $\alpha(2k+1)$, and hence X' goes to V_{small}. We are now ready to prove Lemma 17 formally.

Proof of Lemma 17. Let $(v_1, \ldots, v_n)$ be a d-out-degeneracy sequence of D. Consider the partition of $V(D)$ into three parts: $\{v_i\}$, the predecessors of v_i in this ordering, V_P and the successors of v_i in this ordering V_S. Formally, consider $V(D) = \{v_i\} \uplus V_P \uplus V_S$, where $V_P = \{v_j : j < i\}$ and $V_S = \{v_j : j > i\}$. Further consider the partition of V_P into the set of vertices of V_P that are neighbors of v_i, say V_P^N, and the set of vertices of V_P that are non-neighbors of v_i, say V_P^{NN}. That is, $V(P) = V_P^N \uplus V_P^{NN}$. Next consider the partition of V_P^N into two parts: V_P^{ON} and V_P^{IN} such that V_P^{ON} is the set of vertices in V_P^N that are out-neighbors of v_i and V_P^{IN} is the set of vertices in V_P^N that are in-neighbors of v_i. Finally, consider the digraph induced on V_P^{IN}. We partition the set V_P^{IN} based on the out-degree of the vertices in $D' = D[V_P^{IN} \cup \{v_i\}]$. We partition the set V_P^{IN} into two parts: $V_{P,L}^{IN}$ and $V_{P,S}^{IN}$, in the following way. If $d^+_{D'}(v) \geq k+1$, $v \in V_{P,L}^{IN}$, otherwise $v \in V_{P,S}^{IN}$. Observe that, for each $v \in V_{P,S}^{IN}$, $d^+_{D''(v)} \leq k$, where $D'' = D[V_{P,S}^{IN} \cup \{v_i\}]$. We have the following from the above discussion.

$$\begin{aligned} V(D) &= \{v_i\} \uplus V_P \uplus V_S = \{v_i\} \uplus V_P^N \uplus V_P^{NN} \uplus V_S = \{v_i\} \uplus V_P^{ON} \uplus V_P^{IN} \uplus V_P^{NN} \uplus V_S \\ &= \{v_i\} \uplus V_P^{ON} \uplus V_{P,L}^{IN} \uplus V_{P,S}^{IN} \uplus V_P^{NN} \uplus V_S. \end{aligned}$$

We now claim that the desired partition $V(D) = V_{\text{induct}} \uplus V_{\text{forceL}} \uplus V_{\text{forceR}} \uplus V_{\text{small}}$ is such that, (1) $V_{\text{induct}} = V_P^{NN}$, (2) $V_{\text{forceL}} = \{v_i\} \cup V_{P,L}^{IN}$, (3) $V_{\text{forceR}} = V_S$, and (4) $V_{\text{small}} = V_P^{ON} \cup V_{P,S}^{IN}$. An illustration depicting this partitioning can be found in Figure 2. Let us now prove that the sets $V_{\text{induct}}, V_{\text{forceL}}, V_{\text{forceR}}$ and V_{small} satisfy the desired properties.

1. $\boldsymbol{V_{\text{induct}}}$: Observe that when $\alpha = 1$, that is, when D is a tournament, $V_P^{NN} = \emptyset$. Therefore, in this case, $V_{\text{induct}} = \emptyset$. Otherwise, since $D[V_{\text{induct}}]$ is a subgraph of D and $D \in \mathcal{D}_{\alpha,d}$, $D[V_{\text{induct}}] \in \mathcal{D}_{\alpha,d}$. Since V_{induct} only contains vertices that are non-neighbors of v_i, if $D[V_{\text{induct}}]$ has an independent set, say X, of size α then $X \cup \{v_i\}$ is an independent set in D of size $\alpha+1$, which contradicts the fact that the size of any independent set in D is bounded by α. Thus, $D[V_{\text{induct}}] \in \mathcal{D}_{\alpha',d}$, where $\alpha' < \alpha$.
2. $\boldsymbol{V_{\text{forceL}}}$: By the definition of a type-i cut, for any k-cut (L, R) of type-i in D, $v_i \in L$. We will now show that for any $v_j \in V_{P,L}^{IN}$, there exists $k+1$ arc-disjoint paths from v_j to v_i. Thus, if (L, R) is a k-cut in D and $v_i \in L$, then for all $v_j \in V_{P,L}^{IN}$, $v_j \in L$. Consider any $v_j \in V_{P,L}^{IN}$. Recall that $d^+_{D'}(v_j) \geq k+1$ where $D' = D[V_P^{IN} \cup \{v_i\}]$ and V_P^{IN} is the

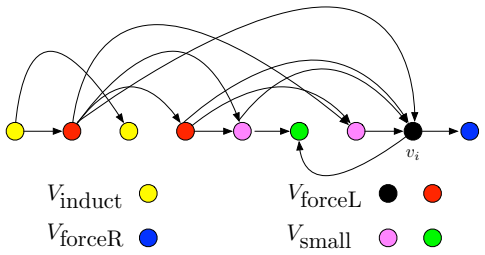


Figure 2 The vertex partition for the Subexponential FPT bound. Here the vertices are arranged in the linear order respecting the d-out-degeneracy sequence of D. Here $k = 2$ and the partition of the vertices into the respective sets is demonstrated using appropriate colors.

set of in-neighbors of v_i in V_P. Consider the set of out-neighbours of v_j in D'. Since the number of such out-neighbors is at least $k+1$ and each of these out-neighbors is an in-neighbor of v_i, we conclude that there are at least $k+1$ arc-disjoint paths from v_j to v_i.

3. $\boldsymbol{V_{\textbf{forceR}}}$: By the definition of type-i cut, $V_S \subseteq R$, for any type-i cut (L, R).
4. $\boldsymbol{V_{\textbf{small}}}$: Since $(v_1, \ldots, v_n)$ is a d-out-degeneracy sequence of D, $|V_P^{NO}| \leq d$. We need to show that $|V_{P,S}^{IN}| \leq \alpha(2k+1)$. Recall that, as observed before, for each $v \in V_{P,S}^{IN}$, $d^+_{D''}(v) \leq k$, where $D'' = D[V_{P,S}^{IN} \cup \{v_i\}]$. Since D'' is an induced subgraph of D, $D'' \in \mathcal{D}_{\alpha,d}$. Also for each $v \in V(D'')$, $d^+_{D''}(v) \leq k$. Thus, from Lemma 18, $|V(D'')| \leq \alpha(2k+1)$. This proves that $|V_{P,S}^{IN}| \leq \alpha(2k+1)$.

This concludes the proof. ◀

4 Conclusion

In this paper, we designed sub-exponential time parameterized algorithms for DFAS, Directed Cutwidth and OLA on digraphs of bounded independence number. We thus significantly generalized known results for the restricted case of input digraphs that are tournaments. Towards this, we obtained an upper bound on the number of k-cuts in digraphs in $\mathcal{D}_\alpha$. This bound is our main contribution, which we believe to find further implications in the future, and to be of independent interest. We conclude with an open problem: Do DFAS, Directed Cutwidth and OLA admit polynomial kernels on $\mathcal{D}_\alpha$?

References

1 Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *J. ACM*, 55(5):23:1–23:27, 2008.
2 Noga Alon. Ranking Tournaments. *SIAM J. Discrete Math.*, 20(1):137–142, 2006.
3 Noga Alon, Daniel Lokshtanov, and Saket Saurabh. Fast FAST. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, pages 49–58, 2009.
4 Jørgen Bang-Jensen and Carsten Thomassen. A Polynomial Algorithm for the 2-Path Problem for Semicomplete Digraphs. *SIAM J. Discrete Math.*, 5(3):366–376, 1992.

5 Florian Barbero, Christophe Paul, and Michal Pilipczuk. Exploring the Complexity of Layout Parameters in Tournaments and Semi-Complete Digraphs. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 70:1–70:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

6 Pierre Charbit, Stéphan Thomassé, and Anders Yeo. The Minimum Feedback Arc Set Problem is NP-Hard for Tournaments. *Combinatorics, Probability & Computing*, 16(1):1–4, 2007.

7 Maria Chudnovsky, Alexandra Ovetsky Fradkin, and Paul D. Seymour. Tournament immersion and cutwidth. *J. Comb. Theory, Ser. B*, 102(1):93–101, 2012.

8 Maria Chudnovsky and Paul D. Seymour. A well-quasi-order for tournaments. *J. Comb. Theory, Ser. B*, 101(1):47–53, 2011.

9 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

10 Michael Dom, Jiong Guo, Falk Hüffner, Rolf Niedermeier, and Anke Truß. Fixed-parameter tractability results for feedback set problems in tournaments. *J. Discrete Algorithms*, 8(1):76–86, 2010.

11 Uriel Feige. Faster FAST(Feedback Arc Set in Tournaments). *CoRR*, abs/0911.5094, 2009. URL: `http://arxiv.org/abs/0911.5094`.

12 Fedor V. Fomin and Michal Pilipczuk. Jungles, bundles, and fixed parameter tractability. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 396–413, 2013.

13 Fedor V. Fomin and Michal Pilipczuk. Subexponential Parameterized Algorithm for Computing the Cutwidth of a Semi-complete Digraph. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, pages 505–516, 2013.

14 Alexandra Ovetsky Fradkin and Paul D. Seymour. Tournament pathwidth and topological containment. *J. Comb. Theory, Ser. B*, 103(3):374–384, 2013.

15 Alexandra Ovetsky Fradkin and Paul D. Seymour. Edge-disjoint paths in digraphs with bounded independence number. *J. Comb. Theory, Ser. B*, 110:19–46, 2015.

16 T Gallai and AN Milgram. Verallgemeinerung eines graphentheoretischen Satzes von Rédei. *Acta Sc. Math*, 21:181–186, 1960.

17 Marek Karpinski and Warren Schudy. Faster Algorithms for Feedback Arc Set Tournament, Kemeny Rank Aggregation and Betweenness Tournament. In *Algorithms and Computation - 21st International Symposium, ISAAC 2010, Jeju Island, Korea, December 15-17, 2010, Proceedings, Part I*, pages 3–14, 2010.

18 Claire Kenyon-Mathieu and Warren Schudy. How to rank with few errors. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 95–103, 2007.

19 Mithilesh Kumar and Daniel Lokshtanov. Faster Exact and Parameterized Algorithm for Feedback Vertex Set in Tournaments. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPIcs*, pages 49:1–49:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

20 Matthias Mnich, Virginia Vassilevska Williams, and László A. Végh. A 7/3-Approximation for Feedback Vertex Sets in Tournaments. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 67:1–67:14, 2016.

21 Michal Pilipczuk. Computing cutwidth and pathwidth of semi-complete digraphs via degree orderings. In *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 - March 2, 2013, Kiel, Germany*, volume 20 of *LIPIcs*, pages 197–208. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

A Appendix: Sub-exponential FPT algorithms for DFAS, Directed Cutwidth and OLA for digraphs in $\mathcal{D}_\alpha$

In this section, we will give sub-exponential FPT algorithms for DFAS, Directed Cutwidth and Optimal Linear Arrangement when the input graph belongs to $\mathcal{D}_\alpha$, for some positive integer α. All these algorithms are based on a three step procedure. The *first* is observing that the digraphs that are Yes-instances of these problems have *sub-exponential FPT* many k-cuts. The proofs for DFAS and Directed Cutwidth are based on showing that the digraph in the Yes-instances of the problems are k-out-degenerate, and hence, the bounds follow from Lemma 7. For OLA, we show that if there is an ordering of the vertex set of a digraph of cost at most k then the cutwidth of this digraph is $\mathcal{O}(k^{2/3})$. Hence, from the results for Directed Cutwidth, the number of k-cuts in the Yes-instances of OLA is also bounded. The *second* step is a procedure to enumerate all k-cuts of the input digraph. And the *third* is to do some dynamic programming procedure over these enumerated cuts to solve the respective problems. The last part of the algorithm (doing dynamic programming over k-cuts) is standard and is identical to the algorithm given by Fomin and Pilipczuk [13]. Proofs are given for completeness.

Before proceeding further, we make a small remark that the proofs of Lemma 5 and 7 can be made constructive by using the constructive versions of the Gallai-Milgram's Theorem, Chromatic Coding lemma and a polynomial time procedure to output a d-out-degeneracy sequence of a digraph. Thus, one can actually enumerate all the k-cuts in the input digraphs of these Lemmas using our algorithm. However, for the sake of completeness, we state in Lemma 19, a different procedure that using a standard branching, enumerates all the k-cuts in any digraph with polynomial delay.

▶ **Lemma 19** (Lemma 7, [13]). *k-cuts of a digraph D can be enumerated with polynomial-time delay.*

A.1 Sub-exponential algorithm for Directed Feedback Arc Set

We begin by recalling the problem definition.

Directed Feedback Arc Set (DFAS)	**Parameter:** k
Input: A digraph D and an integer k.	
Question: Does there exist $S \subseteq E(D)$ such that $D - S$ is a DAG?	

Such a set $S \subseteq E(D)$ is called a dfas of D. Observe that, a digraph D has a dfas of size at most k if and only if there exists an ordering, say $(v_1, \ldots, v_n)$, of $V(D)$ such that $|\sum_{i\in[n]} N^+(v_i) \cap \{v_j \mid j < i\}| \leq k$, that is, the number of backward arcs in this ordering is at most k. Next we bound the number of k-cuts in the Yes-instances of DFAS.

▶ **Lemma 20.** *If (D, k) is a* Yes*-instance of* DFAS *and $D \in \mathcal{D}_\alpha$, then the number of k-cuts in D is at most $2^{c(\alpha+1)\sqrt{k}\log k} \cdot 2^{2\alpha(\alpha+1)\lceil\sqrt{k}\rceil \log((k(2\alpha+1)+\alpha+1))} \cdot \log(k+\alpha(2k+1)) \cdot n^{\alpha+1}$, where c is a fixed absolute constant.*

Proof. Since (D, k) is a Yes-instance of DFAS, there exists an ordering, say $(v_1, \ldots, v_n)$, of $V(D)$, such that $|\sum_{i\in[n]} N^+(v_i) \cap \{v_j \mid j < i\}| \leq k$. In particular, for any $i \in [n]$, $|N^+(v_i) \cap \{v_j \mid j < i\}| \leq k$. Thus, $(v_1, \ldots, v_n)$ is a k-out-degeneracy sequence of $V(D)$. Therefore, the bound follows from Lemma 7. ◀

Now we give the proof of Theorem 1.

Proof of Theorem 1. Using the algorithm of Lemma 19, we enumerate all k-cuts in D. If during the enumeration we exceed the bound given in Lemma 20, then we correctly conclude that (D, k) is a No-instance of DFAS. Otherwise, from Lemma 19, in time $2^{\mathcal{O}(\alpha^2\sqrt{k}\log(\alpha k))} \cdot n^{\mathcal{O}(\alpha)}$, we would have enumerated the set of all k-cuts in D. Let us denote this set by $\mathcal{C}$. We will solve the DFAS problem by doing a dynamic programming over the set $\mathcal{C}$ of k-cuts. Let T be the dynamic programming table indexed by a k-cut $(L, R) \in \mathcal{C}$ and an integer $i \in [k]$. For any $(L, R) \in \mathcal{C}$ and $i \in [k]$, we want $T((L, R), i)$ to store the following information.

$$T((L,R),i) = \begin{cases} 1 & \text{if there exists an ordering } (v_1, \ldots, v_\ell) \text{ of } L \\ & \text{witnessing that } D[L] \text{ has a dfas of size } i, \text{ and} \\ & (L \setminus \{v_\ell\}, R \cup \{v_\ell\}) \in \mathcal{C} \\ 0 & \text{otherwise} \end{cases}$$

Note that $T((V(D), \emptyset), k) = 1$ if and only if D has a dfas of size at most k. We now describe how we compute $T((L, R), i)$, for any $(L, R) \in \mathcal{C}$ and $i \in [k]$. For all $i \in [k]$, $T((\emptyset, V(D)), i) = 1$. For any $(L, R) \in \mathcal{C}$, such that $L \neq \emptyset$, and any $i \in [k]$, $T((L, R), i) = 1$ if and only if there exists $v \in L$ such that $(L \setminus \{v\}, R \cup \{v\}) \in \mathcal{C}$ and, if $|N^+_L(v)| = j$, then $T((L \setminus \{v\}, R \cup \{v\}), i - j) = 1$.

We now prove that for any $(L, R) \in \mathcal{C}$ and $i \in [k]$, $T((L, R), i) = 1$ if and only if there exists an ordering $(v_1, \ldots, v_\ell)$ of L witnessing that $D[L]$ has a dfas of size i, and $(L \setminus \{v_\ell\}, R \cup \{v_\ell\}) \in \mathcal{C}$. We prove this by induction on $|L|$. When $|L| = 0$, this is true because of the base case. By inductive hypothesis, assume that it holds for any $(L', R') \in \mathcal{C}$ such that $|L'| = \ell - 1$, and for any $i \in [k]$. We will first prove that if $T((L, R), i) = 1$, then there exists an ordering $(v_1, \ldots, v_\ell)$ of L witnessing that $D[L]$ has a dfas of size i, and $(L \setminus \{v_\ell\}, R \cup \{v_\ell\}) \in \mathcal{C}$.

Since $T((L, R), i) = 1$, there exists a vertex, say $v_\ell \in L$, such that $(L \setminus \{v_\ell\}, R \cup \{v_\ell\}) \in \mathcal{C}$ and if $|N^+_L(v_\ell)| = j$ then $T((L \setminus \{v_\ell\}, R \cup \{v_\ell\}), i - j) = 1$. Since $T((L \setminus \{v_\ell\}, R \cup \{v_\ell\}), i - j) = 1$, from induction hypothesis, $D[L \setminus \{v_\ell\}]$ has a dfas of size at most $i - j$. Let $(v_1, \ldots, v_{\ell-1})$ be the ordering of $L \setminus \{v_\ell\}$ witnessing this, that is, $\sum_{p \in [\ell-1]} |N^+(v_p) \cap \{v_q \mid q < p\}| \leq i - j$. Since $|N^+_L(v_\ell)| = j$, $\sum_{p \in [\ell]} |N^+(v_p) \cap \{v_q \mid q < p\}| \leq i$. Thus, the ordering $(v_1, \ldots, v_{\ell-1}, v_\ell)$ is a witness to the fact that $D[L]$ has a dfas of size at most i.

We will now prove that if $D[L]$ has a dfas of size at most i and $(v_1, \ldots, v_\ell)$ is an ordering witnessing this such that $(L \setminus \{v_\ell\}, R \cup \{v_\ell\}) \in \mathcal{C}$, then $T((L, R), i) = 1$. Clearly, if $|N^+(v_\ell)| = j$, then the ordering $(v_1, \ldots, v_{\ell-1})$ witnesses that $D[L \setminus \{v_\ell\}]$ has a dfas of size at most $i - j$. Thus, $T((L \setminus \{v_\ell\}, R \cup \{v_\ell\}), i - j) = 1$. ◀

A.2 Sub-exponential algorithm for Directed Cutwidth

Let D be a digraph. For an ordering $(v_1, \ldots, v_n)$ of $V(D)$, the *width* of this ordering is $\max_{i \in [n-1]} |E(\{v_{i+1}, \ldots, v_n\}, \{v_1, \ldots, v_i\})|$. The *cutwidth* of D, denoted by $\mathbf{ctw}(D)$, is the smallest possible width of an ordering of $V(D)$.

Directed Cutwidth	**Parameter:** k
Input: A digraph D and an integer k.	
Question: Is $\mathbf{ctw}(D) \leq k$?	

Next we bound the number of k-cuts in the Yes-instances of DFAS.

▶ **Lemma 21.** *If (D,k) is a* YES*-instance of* DIRECTED CUTWIDTH *and $D \in \mathcal{D}_\alpha$, then the number of k-cuts in D is at most $2^{c(\alpha+1)\sqrt{k}\log k} \cdot 2^{2\alpha(\alpha+1)\lceil\sqrt{k}\rceil\log((k(2\alpha+1)+\alpha+1))} \cdot \log(k+\alpha(2k+1)) \cdot n^{\alpha+1}$, where c is a fixed absolute constant.*

Proof. If (D,k) is a YES-instance of DFAS, then there is an ordering, say $(v_1,\ldots,v_n)$, of $V(D)$ of width at most k. Recall that, the width of an ordering $(v_1,\ldots,v_n)$ is $\max_{i\in[n-1]} |E(\{v_{i+1},\ldots,v_n\},\{v_1,\ldots,v_i\})|$. Observe that if $\max_{i\in[n-1]} |E(\{v_1,\ldots,v_i\},\{v_{i+1},\ldots,v_n\})| \leq k$, then for each $i \in [n]$, $|N^+(v_i) \cap \{v_j : j < i\}| \leq k$. Thus, D is k-out-degenerate. Thus, the bound follows from Lemma 7. ◀

Now we give the proof of Theorem 2.

Proof of Theorem 2. Using the algorithm of Lemma 19, we enumerate all k-cuts in D. If during the enumeration we exceed the bound given in Lemma 21, then we correctly conclude that (D,k) is a NO-instance of DIRECTED CUTWIDTH. Otherwise, from Lemma 19, in time $2^{\mathcal{O}(\alpha^2\sqrt{k}\log(\alpha k))} \cdot n^{\mathcal{O}(\alpha)}$, we would have enumerated the set of all k-cuts in D. Let us denote this set by $\mathcal{C}$. We will solve the DIRECTED CUTWIDTH problem by doing a dynamic programming over the set $\mathcal{C}$ of k-cuts. Let T be the dynamic programming table indexed by a k-cut $(L,R) \in \mathcal{C}$. For any $(L,R) \in \mathcal{C}$, we want $T((L,R))$ to store the following information.

$$T((L,R)) = \begin{cases} 1 & \text{if there exists an ordering of } L, \text{ say } (v_1,\ldots,v_\ell), \\ & \text{such that for all } j \in [\ell-1], |E(\{v_{j+1},\ldots,v_n\},\{v_1,\ldots,v_j\})| \leq k \\ 0 & \text{otherwise} \end{cases}$$

Note that $T((V(D),\emptyset)) = 1$ if and only if $\mathbf{ctw}(D) \leq k$. We now describe how we compute $T((L,R))$ for any $(L,R) \in \mathcal{C}$. Set $T((\emptyset,V(D))) = 1$. For any $(L,R) \in \mathcal{C}$ such that $L \neq \emptyset$, $T((L,R)) = 1$ if and only if there exists $v \in L$ such that $(L\setminus\{v\}, R\cup\{v\}) \in \mathcal{C}$ and $T((L\setminus\{v\}, R\cup\{v\})) = 1$.

We now prove that for any $(L,R) \in \mathcal{C}$, $T((L,R)) = 1$ if and only if there exists an ordering of L, say $(v_1,\ldots,v_\ell)$, such that for all $j \in [\ell-1]$, $|E(\{v_{j+1},\ldots,v_n\},\{v_1,\ldots,v_j\})| \leq k$. We prove this by induction on $|L|$. When $|L| = 0$, this is true because of the base case. By inductive hypothesis, assume that for any $(L',R') \in \mathcal{C}$, such that $|L'| = \ell-1$, $T((L',R')) = 1$ if and only if there exists an ordering of L', say $(v_1,\ldots,v_{\ell-1})$, such that for all $j \in [\ell-2]$, $|E(\{v_{j+1},\ldots,v_n\},\{v_1,\ldots,v_j\})| \leq k$. Let $(L,R) \in \mathcal{C}$ be such that $|L| = \ell$. We will first prove that if $T((L,R)) = 1$, then there exists an ordering of L, say $(v_1,\ldots,v_\ell)$, such that for all $j \in [\ell-1]$, $|E(\{v_{j+1},\ldots,v_n\},\{v_1,\ldots,v_j\})| \leq k$. Since $T((L,R)) = 1$, there exists a vertex in L, say v_ℓ, such that $(L\setminus\{v_\ell\}, R\cup\{v_\ell\})$ and $T((L\setminus\{v_\ell\}, R\cup\{v_\ell\})) = 1$. Since $T((L\setminus\{v_\ell\}, R\cup\{v_\ell\})) = 1$, from inductive hypothesis, there exists an ordering of $L\setminus\{v_\ell\}$, say $(v_1,\ldots,v_{\ell-1})$, such that for all $j \in [\ell-2]$, $|E(\{v_{j+1},\ldots,v_n\},\{v_1,\ldots,v_j\})| \leq k$. Also, since $(L\setminus\{v_\ell\}, R\cup\{v_\ell\}) \in \mathcal{C}$, $|E(\{v_\ell,\ldots,v_n\},\{v_1,\ldots,v_{\ell-1}\})| \leq k$. Thus, for the ordering $(v_1,\ldots,v_\ell)$ of L, for all $j \in [\ell-1]$, $|E(\{v_{j+1},\ldots,v_n\},\{v_1,\ldots,v_j\})| \leq k$.

We will now prove that if there exists an ordering of L, say $(v_1,\ldots,v_\ell)$, such that for all $j \in [\ell-1]$, $|E(\{v_{j+1},\ldots,v_n\},\{v_1,\ldots,v_j\})| \leq k$, then $T((L,R)) = 1$. Since $|E(\{v_\ell,\ldots,v_n\}, \{v_1,\ldots,v_{\ell-1}\})| \leq k$, $(L\setminus\{v_\ell\}, R\cup\{v_\ell\}) \in \mathcal{C}$. Also, since for all $j \in [\ell-2]$, $|E(\{v_{j+1},\ldots,v_n\}, \{v_1,\ldots,v_j\})| \leq k$, therefore, $T((L\setminus\{v_\ell\}, R\cup\{v_\ell\})) = 1$. Thus, $T((L,R)) = 1$. This concludes the proof. ◀

A.3 Sub-exponential algorithm for Optimal Linear Arrangement

Let D be a digraph. For an ordering $\sigma = (v_1, \ldots, v_n)$ of $V(D)$, the *cost* of σ is

$$\sum_{(v_i,v_j)\in E(D)} (i-j)\cdot[i>j],$$

that is, every arc directed backward in the ordering contributes a cost that is equal to the length of this arc, which is the distance between the end-points of this arc in the ordering. Recall that $[i > j]$, evaluates to 1 if $i > j$, to 0 otherwise.

Optimal Linear Arrangement (OLA)	**Parameter:** k
Input: A digraph D and an integer k.	
Question: Is there an ordering of $V(D)$ of cost at most k?	

The following proposition gives an alternate definition of the cost of an ordering.

▶ **Proposition 22** ([13]). *For a digraph D and an ordering $(v_1, \ldots, v_n)$ of $V(D)$, the cost of this ordering is equal to $\sum_{i\in[n-1]} |E(\{v_{i+1}, \ldots, v_n\}, \{v_1, \ldots, v_i\})|$.*

Lemma 23 shows a relation between the cost of an ordering and its width. Note that this lemma was already proved in [13], but the authors state the result for the case when the input digraph is a semi-complete digraph. We observe that the same proof works for any digraph. For the sake of completeness, we give the same proof here.

▶ **Lemma 23.** *For any digraph D, if there is an ordering say $(v_1, \ldots, v_n)$ of $V(D)$, of cost at most k, then $\mathbf{ctw}(D) \leq (2k)^{\frac{2}{3}}$.*

Proof. Since $(v_1, \ldots, v_n)$ is an ordering of cost at most k, from Proposition 22, $\sum_{i\in[n-1]} |E(\{v_{i+1}, \ldots, v_n\}, \{v_1, \ldots, v_i\})| \leq k$. Fix an arbitrary $i \in [n-1]$. We will show that $|E(\{v_{i+1}, \ldots, v_n\}, \{v_1, \ldots, v_i\})| \leq (2k)^{\frac{2}{3}}$. Let $|E(\{v_{i+1}, \ldots, v_n\}, \{v_1, \ldots, v_i\})| = \ell$. For any arc $(v_p, v_q) \in E(D)$, such that $p < q$, the length of the arc (v_p, v_q) is equal to $q - p$. Observe that, for any r, the number of arcs of length exactly r with tail in $\{v_{i+1}, \ldots, v_n\}$ and head in $\{v_1, \ldots, v_i\}$ is at most r. Thus, for any r, the total number of arcs of length at most r, with tail in $\{v_{i+1}, \ldots, v_n\}$ and head in $\{v_1, \ldots, v_i\}$, is at most $\frac{r(r+1)}{2}$. In particular, the number of arcs of length at most $\sqrt{\ell} - 1$, with tail in $\{v_{i+1}, \ldots, v_n\}$ and head in $\{v_1, \ldots, v_i\}$ is at most $\frac{\sqrt{\ell}(\sqrt{\ell}-1)}{2} \leq \frac{\ell}{2}$. Since $|E(\{v_{i+1}, \ldots, v_n\}, \{v_1, \ldots, v_i\})| = \ell$, the number of arcs of length at least $\sqrt{\ell}$ with tail in $\{v_{i+1}, \ldots, v_n\}$ and head in $\{v_1, \ldots, v_i\}$ is at least $\frac{\ell}{2}$. Since $\sum_{i\in[n-1]} |E(\{v_{i+1}, \ldots, v_n\}, \{v_1, \ldots, v_i\})| \leq k$, we have that $k \geq \sqrt{\ell} \cdot \frac{\ell}{2}$. Thus, $\ell \leq (2k)^{\frac{2}{3}}$. ◀

Next we bound the number of k-cuts in the Yes-instances of OLA.

▶ **Lemma 24.** *If (D, k) is a* Yes*-instance of* OLA *and $D \in \mathcal{D}_\alpha$, then the number of k-cuts in D is at most $2^{c(\alpha+1)k^{\frac{1}{3}}\log k} \cdot 2^{2\alpha(\alpha+1)\lceil k^{\frac{1}{3}}\rceil \log((k(2\alpha+1)+\alpha+1))} \cdot \log(k+\alpha(2k+1)) \cdot n^{\alpha+1}$, where c is a fixed absolute constant.*

Proof. Since D is a Yes-instance of OLA, from Lemma 23, $\mathbf{ctw}(D) \leq (2k)^{\frac{2}{3}}$. Thus, $(D, (2k)^{\frac{2}{3}})$ is a Yes-instance of Directed Cutwidth. Hence, from Lemma 21, the number of k-cuts in D are bounded by the desired function. ◀

Proof of Theorem 3. Using the algorithm of Lemma 19, we enumerate all k-cuts in D. If during the enumeration we exceed the bound given in Lemma 24, then we correctly conclude that (D, k) is a No-instance of OLA. Otherwise, from Lemma 19, in time $2^{\mathcal{O}(\alpha^2 k^{\frac{1}{3}}\log(\alpha k))}$.

$n^{\mathcal{O}(\alpha)}$, we would have enumerated the set of all k-cuts in D. Let us denote this set by $\mathcal{C}$. We will solve OLA by doing a dynamic programming over the set $\mathcal{C}$ of k-cuts. Let T be the dynamic programming table indexed by a k-cut $(L,R) \in \mathcal{C}$ and an integer $i \in [k]$. For any $(L,R) \in \mathcal{C}$ and $i \in [k]$, we want $T((L,R),i)$ to store the following information.

$$T((L,R),i) = \begin{cases} 1 & \text{if there exists an ordering of } L, \text{ say } (v_1,\ldots,v_\ell), \\ & \text{such that } \sum_{j\in[\ell]} |E(\{v_{j+1},\ldots,v_n\},\{v_1,\ldots,v_j\})| \leq i \\ 0 & \text{otherwise} \end{cases}$$

Note that $T((V(D),\emptyset),k) = 1$ if and only if D has an ordering of cost at most k. We now describe how we compute $T((L,R),i)$ for any $(L,R) \in \mathcal{C}$ and $i \in [k]$. For all $i \in [k]$, $T((\emptyset,V(D)),i) = 1$. For any $(L,R) \in \mathcal{C}$ such that $L \neq \emptyset$, and any $i \in [k]$, $T((L,R)) = 1$ if and only if there exists $v \in L$ such that $(L\setminus\{v_\ell\}, R\cup\{v_\ell\})$ and $T((L\setminus\{v\}, R\cup\{v\}), i-j) = 1$, where $j = |E(R,L)|$.

We now prove that for any $(L,R) \in \mathcal{C}$ and integer $i \in [k]$, $T((L,R),i) = 1$ if and only if there exists an ordering of L, say $(v_1,\ldots,v_\ell)$, such that $\sum_{j\in[\ell]} |E(\{v_{j+1},\ldots,v_n\}, \{v_1,\ldots,v_j\})| \leq i$. We prove this by induction on $|L|$. When $|L| = 0$, this is true because of the base case. By inductive hypothesis, assume that for any $(L',R') \in \mathcal{C}$ such that $|L'| = \ell-1$, and for any $p \in [k]$, $T((L',R'),p) = 1$ if and only if there exists an ordering of L, say $(v_1,\ldots,v_\ell)$, such that $\sum_{j\in[\ell]} |E(\{v_{j+1},\ldots,v_n\},\{v_1,\ldots,v_j\})| \leq i$. Let $(L,R) \in \mathcal{C}$ be such that $|L| = \ell$ and $i \in [k]$. We will first prove that if $T((L,R),i) = 1$, then there exists an ordering of L, say $(v_1,\ldots,v_\ell)$, such that $\sum_{j\in[\ell]} |E(\{v_{j+1},\ldots,v_n\},\{v_1,\ldots,v_j\})| \leq i$. Let $j = |E(R,L)|$. Since $T((L,R),i) = 1$, there exists a vertex in L, say v_ℓ, such that $(L\setminus\{v_\ell\}, R\cup\{v_\ell\}) \in \mathcal{C}$ and $T((L\setminus\{v_\ell\}, R\cup\{v_\ell\}), i-j) = 1$. From inductive hypothesis, there exists an ordering of $L\setminus\{v_\ell\}$, say $(v_1,\ldots,v_{\ell-1})$, such that $\sum_{p\in[\ell-1]} |E(\{v_{p+1},\ldots,v_n\},\{v_1,\ldots,v_p\})| \leq i-j$. Since $j = |E(R,L)|$, for the ordering $(v_1,\ldots,v_\ell)$ of L, $\sum_{p\in[\ell]} |E(\{v_{p+1},\ldots,v_n\},\{v_1,\ldots,v_p\})| \leq i$.

We will now prove that if there exists an ordering of L, say $(v_1,\ldots,v_\ell)$, such that $\sum_{j\in[\ell]} |E(\{v_{j+1},\ldots,v_n\},\{v_1,\ldots,v_j\})| \leq i$, then $T((L,R),i) = 1$. Observe from the definition of this ordering $(v_1,\ldots,v_\ell)$ that $(L\setminus\{v_\ell\}, R\cup\{v_\ell\})$ is an i-cut in D. Since $i \leq k$, $(L\setminus\{v_\ell\}, R\cup\{v_\ell\}) \in \mathcal{C}$. Clearly, if $|E(R,L)| = j$, then $\sum_{p\in[\ell-1]} |E(\{v_{p+1},\ldots,v_n\},\{v_1,\ldots,v_p\})| \leq i-j$. Thus, $T((L\setminus\{v_\ell\}, R\cup\{v_\ell\}), i-j) = 1$ implying that $T((L,R),i) = 1$. This concludes the proof. ◀

Safe and Optimal Scheduling for Hard and Soft Tasks

Gilles Geeraerts
Université libre de Bruxelles, Brussels, Belgium
gigeerae@ulb.ac.be

Shibashis Guha
Université libre de Bruxelles, Brussels, Belgium
shibashis.guha@ulb.ac.be

Jean-François Raskin
Université libre de Bruxelles, Brussels, Belgium
jraskin@ulb.ac.be

Abstract

We consider a stochastic scheduling problem with both hard and soft tasks on a single machine. Each task is described by a discrete probability distribution over possible execution times, and possible inter-arrival times of the job, and a fixed deadline. Soft tasks also carry a penalty cost to be paid when they miss a deadline. We ask to compute an *online* and *non-clairvoyant* scheduler (i.e. one that must take decisions without knowing the future evolution of the system) that is *safe* and *efficient*. Safety imposes that deadline of hard tasks are never violated while efficient means that we want to minimise the mean cost of missing deadlines by soft tasks.

First, we show that the dynamics of such a system can be modelled as a finite Markov Decision Process (MDP). Second, we show that our scheduling problem is PP-hard and in EXPTime. Third, we report on a prototype tool that solves our scheduling problem by relying on the Storm tool to analyse the corresponding MDP. We show how antichain techniques can be used as a potential heuristic.

2012 ACM Subject Classification Theory of computation → Probabilistic computation, Computer systems organization → Real-time system specification, Computer systems organization → Embedded systems

Keywords and phrases Non-clairvoyant scheduling, hard and soft tasks, automatic synthesis, Markov decision processes

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.36

Acknowledgements This work was supported by the ARC project "Non-Zero Sum Game Graphs: Applications to Reactive Synthesis and Beyond" (Fédération Wallonie-Bruxelles).

1 Introduction

In modern real-time systems, we usually need to distinguish between two types of tasks: *hard tasks* that ought to be scheduled so that they meet their deadline with *absolute certainty* and *soft tasks* for which missing a deadline can be tolerated. Typically, hard tasks are vital for the correct execution of the system and missing a deadline for such tasks may have catastrophic consequences while missing a deadline of a soft task only degrades the overall performances of the system (as in a video decoding system, for example, where missing a deadline means skipping some video frames). An example of a system with both hard and soft tasks may consist of the computer system of a commercial aircraft, that must, at the same time, run

the avionics control, and the on-board entertainment system. Clearly, avionics controls are vital and most of these tasks will be *hard*, while missing a few frames in the video stream of the entertainment system only degrades the quality of the video; hence those tasks are *soft*. It is also usual to distinguish between tasks for which the inter-arrival time is fixed (such tasks are often called *periodic* tasks); and tasks for which the inter-arrival time is subject to uncertainty and specified by an interval constraint (such task are often called *sporadic* tasks). Most real systems naturally contain both periodic and sporadic tasks.

In this paper, we consider a rich formal model of infinite duration scheduling on a *single processor* that is applicable to systems with both periodic/aperiodic and hard/soft tasks. The tasks can be *preempted*. Additionally, we assume our schedulers to be *non-clairvoyant* in the sense that the execution and inter-arrival times of tasks are not known in advance and subject to uncertainty modelled by stochastic distributions. More precisely, each hard and soft task is characterised by a fixed deadline, and two discrete finite support distributions specifying its possible inter-arrival times and durations respectively. When a job associated to a task (i.e. a new instance of the task) arrives in the system, its execution time and the arrival time of the next job are not known but only the probability distribution over the possible execution times and arrival times are known. In addition, each soft task comes with a cost that is incurred each time a job of this task misses a deadline. The objective of a scheduler in this model is two-fold:

(i) the deadline of all jobs corresponding to hard tasks must be met with certainty; and

(ii) the expected mean cost of missing deadlines of jobs associated to soft tasks must be minimised.

Contributions. We define formally our scheduling problem as a non-standard optimisation problem on an MDP. That is, we consider MDPs with two *simultaneous* objectives: a *safety* objective asking that the deadline of each job associated to a hard task is met; and an *optimisation* objective asking to minimise the expected mean-cost of missing job deadlines associated to soft tasks. Second, we provide a worst-case exponential time algorithm (see Theorem 4) that decides the existence of a safe and optimal schedule, and provide a PP-hard lower bound[1], which is an improvement on the NP-Complete and coNP-Complete lower bounds that can be deduced from the literature (see related works below). Third, we propose a heuristic based on antichain techniques [16]: we identify a naturally occurring ordering on the states of the MDP, that can be used to prune the state space while computing the set of *safe* states. Thanks to the antichain technique, this set of safe states can also be described compactly. Finally, we have implemented a prototype tool for computing safe schedules on top of the probabilistic model-checker Storm [13], which we use to solve the *optimisation* part of our problem. We rely on the classical attractor algorithm to compute the set of states of the MDP from which the safety objective can always be satisfied. We also have implemented the antichain-based heuristic, and our experiments are encouraging: using the compact description of the safe states, we manage to produce a much smaller input file for Storm. Our algorithm works well for a small number of tasks and each task can have infinitely many jobs. Further an optimal schedule can be implemented simply as a *table lookup* and during runtime it requires minimal computation.

[1] Recall that PP is the class of problems that can be solved by a *probabilistic* Turing machine that operates in polynomial time [21].

To the best of our knowledge, there does not exist any scheduling algorithm in the literature that considers a cost model as ours as well as stochastic behaviour. In the appendix, we show that adapting the classical EDF scheduling policy to our setting yields scheduler that can be arbitrarily worse when compared to our optimal algorithm in terms of the expected mean-costs.

Related work. The schedulability of (hard) periodic tasks is a classical problem that has been studied in details in the literature, see e.g. [29, 26, 27, 14, 8]; and which has been shown to be coNP-complete in [26, 8] (see also [34, 25, 24, 7] where the tasks are not strictly periodic). It can be seen as a special case of our problem, where there are only hard tasks.

The *clairvoyant* scheduling of soft tasks (only) is also a classical problem that has attracted ample attention in the scheduling literature, see e.g. [41, 28, 6]. In [30], the authors consider a setting in which all the tasks have a mandatory (hard) part and an optional (soft) part that incurs a penalty when not executed; and show that cost minimisation is NP-complete when the optional tasks have arbitrary processing times. Again, this setting is a particular case of ours.

Finally, there are works in the literature that consider scheduling problems with both hard and soft tasks, see e.g. [10, 40, 11, 1]. A prominent line of works among them is based on the notion of *servers* [10, 40] to handle soft tasks. Algorithms for preemptive uniprocessor scheduling following this approach include Priority Exchange [40, 25], Sporadic Server [40, 38], Total Bandwidth Server [40], Earliest Deadline Late (EDL) Server [11, 39, 40], Constant Bandwidth Server [1], etc. However, those algorithms do not take into account a stochastic model of the tasks as in our problem nor a notion of deadline and cost for the soft tasks. The algorithm EDL is known to be optimal for dynamic priority assignment [11]. In Appendix B, we show that a version of EDL adapted to our setting can be arbitrarily worse in terms of the expected mean-cost when compared to our optimal algorithm.

The non-standard optimisation problem that we consider on MDP and which simultaneously asks for satisfying a safety and an expected mean-cost constraint is related to a recent line of works that mixes two-player zero sum games and MDPs, see e.g. [9, 3, 12].

2 Preliminaries

We denote by $\mathbb{N}$ the set of non-negative integer numbers, and by $\mathbb{Q}$ the set of rational numbers. For $n \in \mathbb{N}$, we use $[n]$ to denote $\{1, \ldots, n\}$ and $[n]_0$ to denote $\{0, 1, \ldots, n\}$. Given a finite set A, a (rational) *probability distribution* over A is a function $p\colon A \to [0,1] \cap \mathbb{Q}$ such that $\sum_{a \in A} p(a) = 1$. We denote the set of probability distributions on A by $\mathcal{D}(A)$. The *support* of the probability distribution p on A is $\mathsf{Supp}(p) = \{a \in A \mid p(a) > 0\}$. A distribution is called *Dirac* if $|\mathsf{Supp}(p)| = 1$.

Job Scheduling for both soft and hard tasks. We consider a system of n preemptive tasks $\{\tau_1, \ldots, \tau_n\}$ to be scheduled on a *single processor*. We identify all tasks τ_i with their unique respective index i. We assume that the time is discrete and measured in CPU ticks. Each task τ_i generates an infinite number of instances $\tau_{i,j}$, that we call *jobs*, with $j = 1, 2, \ldots$. We assume that all tasks are either *hard* or *soft* and denote by F and H the set of indexes of soft and hard tasks respectively (i.e. $i \in F$ iff τ_i is a soft task). Jobs generated by both hard and soft tasks are equipped with deadlines, which are relative to the respective arrival times of the jobs in the system. Jobs generated by hard tasks must complete before their

respective deadlines, but this is not mandatory for jobs generated by soft tasks. We also assume that tasks are independent, i.e. the scheduling a job of one task does not depend on another job belonging to some other task.

In order to make our model more realistic, we rely on a probabilistic model for the computation times of the jobs and on the time between the arrival of two successive jobs of the same task. Formally, for all $i \in [n]$, task τ_i is defined as a tuple $\langle I_i, \mathcal{C}_i, D_i, \mathcal{A}_i \rangle$, where:

- **(i)** $I_i \in \mathbb{N}$ is the arrival time of the first job of τ_i;
- **(ii)** $\mathcal{C}_i$ is a discrete probability distribution on the (finitely many) possible computation times of the jobs generated by τ_i;
- **(iii)** $D_i \in \mathbb{N}$ is the deadline of all jobs generated by τ_i which is relative to the arrival time of the jobs; and
- **(iv)** $\mathcal{A}_i$ is a discrete probability distribution on the (finitely many) possible inter-arrival times of the jobs generated by τ_i.

We note that $\max(\mathsf{Supp}(\mathcal{C}_i)) \leq D_i$, and throughout the paper, we assume that $D_i \leq \min(\mathsf{Supp}(\mathcal{A}_i))$ for all $i \in [n]$. In addition, we model the potential degradation in the quality when a soft task misses its deadline by a cost function $cost : F \to \mathbb{Q}_{\geq 0}$, that associates, to each soft task τ_j, a cost $c(j)$ which is incurred every time a job of τ_j misses its deadline.

One of the main contributions of this paper is to provide a formal model for such a system (see Section 3). We provide an intuitive explanation for now: each task τ_i releases a first job $\tau_{i,1}$ at time I_i. This job, like all other jobs of τ_i will request a CPU time which is chosen randomly according to $\mathcal{C}_i$. The deadline of $\tau_{i,1}$ is at time $I_i + D_i$. The next job $\tau_{i,2}$ will be released by τ_i at a time $I_i + \delta_2$, where δ_2 is chosen randomly according to $\mathcal{A}_i$, and so forth.

▶ **Example 1.** Consider a system with one hard task $\tau_h = \langle 0, \mathcal{C}_h, 2, \mathcal{A}_h \rangle$ s.t. $\mathcal{C}_h(1) = 1$ and $\mathcal{A}_h(3) = 1$; one soft task $\tau_s = \langle 0, \mathcal{C}_s, 2, \mathcal{A}_s \rangle$ s.t. $\mathcal{C}_s(1) = 0.4$, $\mathcal{C}_s(2) = 0.6$, and $\mathcal{A}_s(3) = 1$; and the cost function c s.t. $c(\tau_s) = 10$. This means that both tasks will submit their first job at time 0, both with deadlines at time $0 + 2 = 2$. Then, $\tau_{h,1}$ will have a computation time of 1, while $\tau_{s,1}$ will have a computation time which is either 1 (with probability 0.4) or 2 (with probability 0.6). Both tasks will submit new jobs $\tau_{h,2}$ and $\tau_{s,2}$ at time $0 + 3 = 3$. Each time a job of τ_s misses its deadline, a cost of 10 will be incurred.

Our goal is to find a *scheduler*, i.e. a function that, given the current state of the system, returns the identifier of the task that needs to be granted CPU access and ensuring that:

- **(i)** no job of the hard tasks misses its respective deadline; and
- **(ii)** the expected mean-cost incurred by the soft tasks missing their deadlines is minimised.

In Section 3, we model the problem as a game between two players: the *Scheduler* whose objectives are sketched above, and *TaskGen*, the task generator that generates jobs according to the semantics of the tasks, and whose goal is *antagonistic* to the scheduler's. Then, computing a scheduler will amount to computing a winning strategy of the *Scheduler* player. We now introduce the necessary notions to model this game with stochastic features.

Labelled Directed Graphs. A *labelled directed graph* (or graph for short) is a tuple $\mathcal{G} = \langle V, E, L \rangle$ where:

- **(i)** V is the finite set of vertices;
- **(ii)** $E \subseteq V \times V$ is the set of directed edges (sometimes called *transitions*); and
- **(iii)** $L : E \to A$ is the function labelling the edges by elements from some set A.

For a transition $e = (v, v')$, v is its *source*, denoted $\mathsf{src}(e)$, and v' its *destination* denoted $\mathsf{trg}(e)$.

Given $v \in V$, let $\mathsf{Succ}(v) = \{v' \in V \mid \exists (v, v') \in E\}$ be its set of successors, and $E(v) = \{e \mid \mathsf{src}(e) = v\}$ be its set of outgoing edges. We assume that for all $v \in V$: $\mathsf{Succ}(v) \neq \emptyset$, i.e. there is no deadlock. A *play* in a graph $\mathcal{G}$ from an initial vertex $v_{\mathsf{init}} \in V$ is an infinite sequence of transitions $\pi = e_0 e_1 e_2 \dots$ such that $\mathsf{src}(e_0) = v_{\mathsf{init}}$ and $\mathsf{trg}(e_i) = \mathsf{src}(e_{i+1})$ for all $i \geq 0$. The *prefix* up to the n-th vertex of π is the finite sequence $\pi(n) = e_0 e_1 \dots e_n$. We denote its last vertex by $\mathsf{Last}(\pi(n)) = \mathsf{trg}(e_n)$. The set of plays of $\mathcal{G}$ is denoted by $\mathsf{Plays}(\mathcal{G})$ and the corresponding set of prefixes is denoted by $\mathsf{Prefs}(\mathcal{G})$.

Weighted Markov Chains. A finite weighted *Markov chain* (MC, for short) is a tuple $M = \langle \mathcal{G}, Prob \rangle$, where $\mathcal{G} = \langle V, E, L \rangle$ is a graph with $L : E \mapsto \mathbb{Q}$ (i.e. edges are labelled by rational numbers that we call the *costs* of the edges), and $Prob : V \to \mathcal{D}(E)$ is a function that assigns a probability distribution on the set $E(v)$ of outgoing edges to all vertices $v \in V$. Given an initial vertex $v_{\mathsf{init}} \in V$, we define the set of possible *outcomes* in M as $\mathsf{Outs}_M(v_{\mathsf{init}}) = \{\pi = e_0 e_1 e_2 \dots \in \mathsf{Plays}(\mathcal{G}) \mid \mathsf{src}(e_0) = v_{\mathsf{init}} \wedge (\forall\, n \in \mathbb{N},\, e_{n+1} \in \mathsf{Supp}(\mathsf{trg}(e_n)))\}$. Let $V_{\mathsf{Outs}_M(v_{\mathsf{init}})} \subseteq V$ denote the set of vertices visited in the set of possible outcomes $\mathsf{Outs}_M(v_{\mathsf{init}})$. Finally, let us assume some measurable function $f : \mathsf{Plays}(\mathcal{G}) \to \mathbb{R}_{\geq 0}$ associating a rational value to each play of the MC. Since the set of plays of M forms a probability space, f is a random variable, and we denote by $\mathbb{E}^M_{v_{\mathsf{init}}}(f)$ the *expected value* of f over the set of plays starting from v_{init}.

Markov decision processes. A finite *Markov decision process* (MDP, for short) is a tuple $\Gamma = \langle V, E, L, (V_\Box, V_\bigcirc), A, Prob \rangle$, where:

- **(i)** A is a finite set of actions;
- **(ii)** $\langle V, E, L \rangle$ is a graph;
- **(iii)** the set of vertices V is partitioned into $V_\Box$ and $V_\bigcirc$;
- **(iv)** the graph is bipartite i.e. $E \subseteq (V_\Box \times V_\bigcirc) \cup (V_\bigcirc \times V_\Box)$, and the labeling function is s.t. $L(v, v') \in A$ if $v \in V_\Box$, and $L(v, v') \in \mathbb{Q}$ if $v \in V_\bigcirc$; and
- **(v)** $Prob$ assigns to each vertex $v \in V_\bigcirc$ a rational probability distribution on $E(v)$.

For all edges e, we let $\mathsf{cost}(e) = L(e)$ if $L(e) \in \mathbb{Q}$, and $\mathsf{cost}(e) = 0$ otherwise. We further assume that, for all $v \in V_\Box$, for all e, e' in $E(v)$: $L(e) = L(e')$ implies $e = e'$, i.e. an action identifies uniquely and outgoing edge.

An MDP can be interpreted as a game between two players: $\Box$ and $\bigcirc$ (Scheduler and TaskGen respectively), who own the vertices in $V_\Box$ and $V_\bigcirc$ respectively. A play in an MDP is a play in its underlying graph $\langle V, E, A \cup \mathbb{Q} \rangle$. We say that a prefix $\pi(n)$ of a play π belongs to player $i \in \{\Box, \bigcirc\}$, iff $\mathsf{Last}(\pi(n)) \in V_i$. The set of prefixes that belong to player i is denoted by $\mathsf{Prefs}_i(G)$. A play in the MDP is then obtained by the interaction of the two players as follows: if the current play prefix $\pi(n)$ belongs to $\Box$, she plays by picking an edge $e \in E(\mathsf{Last}(\pi(n)))$ (or, equivalently, an action that labels a necessarily unique edge from $\mathsf{Last}(\pi(n))$). Otherwise, when $\pi(n)$ belongs to $\bigcirc$, the next edge $e \in E(\mathsf{Last}(\pi(n)))$ is chosen randomly according to $Prob(\mathsf{Last}(\pi(n)))$. In both cases, the plays prefix is extended by e and the game goes *ad infinitum*.

A *strategy* of $\Box$ is a function $\sigma_\Box : \mathsf{Prefs}_\Box(G) \to E$, such that $\sigma_\Box(\rho) \in E(\mathsf{Last}(\rho))$ for all prefixes. A strategy $\sigma_\Box$ is *memoryless* if for all finite prefixes ρ_1 and $\rho_2 \in \mathsf{Prefs}(\mathcal{G})$: $\mathsf{Last}(\rho_1) = \mathsf{Last}(\rho_2)$ implies $\sigma_\Box(\rho_1) = \sigma_\Box(\rho_2)$. From now on, we will consider mainly memoryless strategies. Let $\Gamma = \langle V, E, L, (V_\Box, V_\bigcirc), A, Prob \rangle$ be an MDP and let $\sigma_\Box$ be a *memoryless* strategy. Then, assuming that $\Box$ plays according to $\sigma_\Box$, we can express the behaviour of Γ as an MC $\Gamma[\sigma_\Box]$, where the probability distributions reflect the stochastic choices of $\bigcirc$. Formally, $\Gamma[\sigma_\Box] = \langle V_\bigcirc, E', L', Prob' \rangle$, where $(v, v') \in E'$ iff there is $\hat{v}$ s.t.:

(i) $(v, \hat{v}) \in E$;
(ii) $\sigma_\Box(\hat{v}) = v'$; and
(iii) $Prob(\hat{v}, v') = Prob'(v, v')$.
Further, for all $e \in E'$, we have $L'(e) = L(e)$.

Safety synthesis. Given an MDP $\Gamma = \langle V, E, L, (V_\Box, V_\bigcirc), A, Prob\rangle$, an initial vertex $v_{\mathsf{init}} \in V$, and a set $V_{\mathsf{safe}} \subseteq V$ of so-called *safe vertices*, the *safety synthesis problem* is to decide whether $\Box$ has a strategy $\sigma_\Box$ such that $V_{\mathsf{Outs}_{\Gamma[\sigma_\Box]}(v_{\mathsf{init}})} \subseteq V_{\mathsf{safe}}$, that is, all the plays obtained when $\Box$ plays according to $\sigma_\Box$ visit only the safe vertices. The safety synthesis problem is decidable in polynomial time for MDPs. Indeed, since probabilities do not matter for this problem, the MDP can be regarded as a plain two-player game played on graphs (like in [42]), and the classical *attractor* algorithm can be used (see Appendix A).

Expected mean cost threshold synthesis. Let us first associate a value, called the *mean cost* $\mathsf{MC}(\pi)$ to all plays π in an MDP $\Gamma = \langle V, E, L, (V_\Box, V_\bigcirc), A, Prob\rangle$. First, for a prefix $\rho = e_0 e_1 \dots e_{n-1}$, we define $\mathsf{MC}(\rho) = \frac{1}{n}\sum_{i=0}^{i=n-1} \mathsf{cost}(e_i)$ (recall that $\mathsf{cost}(e) = 0$ when $L(e)$ is an action). Then, for a play $\pi = e_0 e_1 \dots$, we have $\mathsf{MC}(\pi) = \limsup_{n\to\infty} \mathsf{MC}(\pi(n))$. Observe that MC is a measurable function. Then, the *expected mean-payoff threshold synthesis* problem is to decide whether $\Box$ has a strategy $\sigma_\Box$ such that $\mathbb{E}^{\Gamma[\sigma_\Box]}_{v_{\mathsf{init}}}(\mathsf{MC}) \leq \lambda$ for some initial vertex $v_{\mathsf{init}} \in V$ and threshold $\lambda \in \mathbb{Q}$. Such strategies are called *optimal*, and it is well-known that, if such an optimal strategy exists, then, there is one which is *memoryless*. Moreover, this problem can be solved in polynomial time through linear programming [17] or in practice using value iteration (as implemented, for example, in the tool Storm [13]).

3 Modelling the system as an MDP

Let us fix a system of tasks $\tau = \{\tau_1, \tau_2, \dots, \tau_n\}$ and a cost function *cost* and let us model our scheduling problem by means of an MDP Γ_τ. This will provide us with a precise and formal definition of the problem, and will allow us to rely on automatic tools (such as Storm [13], see Section 6) to solve it. In order to define Γ_τ, it is easier to first build an *infinite* MDP $\Gamma = \langle V, E, L, (V_\Box, V_\bigcirc), \ell, Prob\rangle$. Then, Γ_τ will be the (finite) portion of Γ that is reachable from some designated initial state v_{init}. In our model, $\Box$ models the *Scheduler* and $\bigcirc$ models the task generator (abbreviated *TaskGen*).

Modelling the system states. Since $D_i \leq \min(\mathsf{Supp}(\mathcal{A}_i))$ for all tasks τ_i, there can be at most one job of each task at a time t that can be scheduled at t or later. Thus when the system executes, we keep information related to only one job per task. For each task τ_i, at every time, in the vertices of the MDP we maintain the following information about the current job in the system at that time:
(i) a *distribution* c_i over the job's possible remaining computation times (rct);
(ii) the time d_i up to its deadline; and
(iii) a distribution a_i over the possible times up to the next arrival of a new job of τ_i.
We also have a special vertex $\bot$ that will be reached when a hard task misses a deadline. Formally: $V_\Box = \big(\mathcal{D}([C_{\max}]_0) \times [D_{\max}]_0 \times \mathcal{D}([A_{\max}]_0)\big)^n \times \{\Box\} \cup \{\bot\}$ and $V_\bigcirc = \big(\mathcal{D}([C_{\max}]_0) \times [D_{\max}]_0 \times \mathcal{D}([A_{\max}]_0)\big)^n \times \{\bigcirc\}$; where $C_{\max} = \max_i(\max(\mathsf{Supp}(\mathcal{C}_i)))$, $D_{\max} = \max_i(\{D_i\})$ and $A_{\max} = \max_i(\max(\mathsf{Supp}(\mathcal{A}_i)))$. For a vertex $v = \big((c_1, d_1, a_1), \dots (c_n, d_n, a_n)\big)$, we let $\mathsf{active}(v) = \{i \mid c_i(0) \neq 1 \text{ and } d_i > 0\}$ and $\mathsf{dlmiss}(v) = \{i \mid c_i(0) = 0 \text{ and } d_i = 0\}$. Intuitively, $\mathsf{active}(v)$ is the set of tasks that have an active job in v, that is one which has not finished

and whose deadline has not passed yet; and $\mathsf{dlmiss}(v)$ is the set of tasks that have missed a deadline *for sure* in v (observe that for $c_i(0) > 0$, the task *could* complete now and does not miss a deadline for sure). In v, for every task $i \in [n]$, the tuple (c_i, d_i, a_i) is called its *configuration*.

Distribution updates. Let us now introduce the dec and norm functions that will be useful when we will need to update the knowledge of the Scheduler. For example, consider a state where $c_i(1) = 0.5$, $c_i(4) = 0.1$ and $c_i(5) = 0.4$ for some i, and where τ_i is granted one CPU time unit. Then, all elements in the support of c_i should be decremented, yielding c'_i with $c'_i(0) = 0.5$, $c'_i(3) = 0.1$ and $c'_i(4) = 0.4$. Since $0 \in \mathsf{Supp}(c'_i)$, the current job of τ_i *could* now terminate with probability $c'_i(0) = 0.5$, or continue running, which will be observed by the Scheduler player. In the case where the job does not terminate, the probability mass must be redistributed to update Scheduler's knowledge, yielding the distribution c''_i with $c''_i(3) = 0.2$ and $c''_i(4) = 0.8$. Formally, let p and p' be probability distributions on $\mathbb{N}$ s.t. $0 \notin \mathsf{Supp}(p)$. Then, we let $\mathsf{dec}(p)$ and $\mathsf{norm}(p')$ be probability distributions on $\{x - 1 \mid x \in \mathsf{Supp}(p)\}$ and $\mathsf{Supp}(p') \setminus \{0\}$ respectively s.t.:

$$\text{for all } x \in \mathsf{Supp}(p) : \mathsf{dec}(p)(x-1) = p(x)$$
$$\text{for all } x \in \mathsf{Supp}(p') \setminus \{0\} : \mathsf{norm}(p')(x) = \frac{p'(x)}{\sum_{x \geq 1} p'(x)}.$$

Observe that, when $0 \notin \mathsf{Supp}(p')$, then $\mathsf{norm}(p') = p'$.

Possible moves of the Scheduler. The possible actions of Scheduler are to schedule an active task or to idle the CPU. We model this by having, from all states $v \in V_\Box$ one transition labelled by some element from $\mathsf{active}(v)$, or by ε (no job gets scheduled). The effect of the transition models the elapsing of one clock tick.

Formally, fix $v = \big((c_1, d_1, a_1), \ldots, (c_n, d_n, a_n), \Box\big) \in V_\Box$ s.t. $0 \notin \mathsf{Supp}(c_i)$ and $0 \notin \mathsf{Supp}(a_i)$ for all $i \in [n]$. Then, there is $e = (v, v') \in E$ with $L(e) \in [n] \cup \{\varepsilon\}$ and $v' = \big((c'_1, d'_1, a'_1), \ldots, (c'_n, d'_n, a'_n), \bigcirc\big)$ iff the following four conditions hold:

- **(i)** $L(e) = i \in [n]$ implies that $i \in \mathsf{active}(v)$ and $c'_i = \mathsf{dec}(c_i)$, i.e. if a task is scheduled, it must be active, and its rct is decremented;
- **(ii)** for all $j \in [n] \setminus \{L(e)\}$: $c'_j = c_j$, i.e. the rct of all the other tasks does not change;
- **(iii)** for all $j \in [n]$: $d'_j = \max(d_j - 1, 0)$, i.e. the deadline is one time unit closer, if not reached yet;
- **(iv)** for all $j \in [n]$: $a'_j = \mathsf{dec}(a_j)$, i.e. we decrement the time to next arrival of all tasks.

Observe that when a *soft* task misses a deadline, we maintain the positive rct of the job in the next state: this will be used as a marker to ensure that the associated cost is paid when a new job of the same task will arrive. For example, consider a state (with one soft task) $\big((c, 0, a), \Box\big)$ with $c(2) = 1$ and $a(1) = 1$ i.e. the task has reached its deadline but still need 2 time units to complete and the next job arrives in 1 time unit. Then, the successor state will be $\big((c, 0, a'), \bigcirc\big)$ with $a'(0) = 1$, from which a new job will be submitted, which will incur a cost (see action $killANDsub$ in the next paragraph).

Possible moves of the Task Generator. The moves of TaskGen (modelled by $\bigcirc$) consist in selecting, for each task one possible *action* out of four: either nothing (ε); or

- **(i)** to finish the current job without submitting a new one (fin); or
- **(ii)** to submit a new job while the previous one is already finished (sub); or
- **(iii)** to submit a new job and kill the previous one, in the case of a soft task ($killANDsub$), which will incur a cost.

Formally, let $Actions = \{fin, sub, killANDsub, \varepsilon\}$. To define Γ_τ, we introduce a function $\mathcal{L} : (V_\bigcirc \times V_\Box) \mapsto Actions^n$, i.e. $\mathcal{L}(e,i)$ is the action of $\bigcirc$ corresponding to τ_i on edge e. Fix a state $v = \big((c_1,d_1,a_1),\ldots,(c_n,d_n,a_n),\bigcirc\big) \in V_\bigcirc$. Let $\hat{v} = \big((\hat{c_1},\hat{d_1},\hat{a_1}),\ldots,(\hat{c_n},\hat{d_n},\hat{a_n}),\Box\big) \in V_\Box$ be such that that $\hat{v} \xrightarrow{i} v$. Note that there is a unique such $\hat{v}$ from which action i can be done to reach v. We consider two cases. Either $\mathsf{dlmiss}(v) \cap H \neq 0$, i.e., a hard task has just missed a deadline. In this case, the only transition from v is $e = (v, \bot)$ with $\mathcal{L}(e,i) = \varepsilon$ for all $i \in [n]$. Otherwise, there is an edge $e = (v,v')$ with $v' = \big((c'_1,d'_1,a'_1),\ldots,(c'_n,d'_n,a'_n),\Box\big)$ iff for all $i \in [n]$, one of the following holds:

1. $\mathcal{L}(e,i) = fin$, $\min(\mathsf{Supp}(\hat{c}_i = 1))$, $a_i(0) \neq 1$, $c'_i(0) = 1$, $d'_i = d_i$, and $a'_i = \mathsf{norm}(a_i)$. The current job of τ_i finishes now ($c_i(0) > 0$) and the next arrival will occur in the future ($a_i(0) \neq 1$), according to the probability distribution $\mathsf{norm}(a_i)$.
2. $\mathcal{L}(e,i) = sub$, $c_i(0) > 0$, $a_i(0) > 0$, $c'_i = \mathcal{C}_i$, $d'_i = D_i$, and $a'_i = \mathcal{A}_i$. In this case, we assume that the previous job of τ_i has completed ($c_i(0) > 0$) and we let τ_i submit a new job (see the new values c'_i, d'_i and a'_i); or
3. $\mathcal{L}(e,i) = killANDsub$, $c_i(0) \neq 1$, $a_i(0) > 0$, $c'_i = \mathcal{C}_i$, $d'_i = D_i$, and $a'_i = \mathcal{A}_i$. In this case, τ_i (necessarily a soft task) submits a new job, and kills the previous one (there is possibly some remaining rct as $c_i(0) \neq 1$); or
4. $\mathcal{L}(e,i) = \varepsilon$, $a_i(0) \neq 1$, either $c'_i = \mathsf{norm}(c_i)$ or $c'_i(0) = c_i(0) = \hat{c}_i(0) = 1$, $d'_i = d_i$ and $a'_i = \mathsf{norm}(a_i)$. No action is performed on τ_i which must not submit a new job now ($a_i(0) \neq 1$) and does not finish now. For $c'_i(0) = c_i(0) = \hat{c}_i(0) = 1$, it denotes that the job already finished during a previous clock tick. The knowledge of the scheduler (c' and a') is updated accordingly.

The cost of and edge e is: $L(e) = c = \sum_{i:L(e,i)=killANDsub} cost(i)$. As said earlier, the cost is incurred when the $killANDsub$ action is performed by some task τ_i, although the deadline miss might have occurred earlier. Finally, the probability of an edge e is $Prob(e) = \prod_{i\in[n]} p_i$, where, for all $i \in [n]$:

$$p_i = \begin{cases} c_i(0)\cdot(1-a_i(0)) & \text{if } \mathcal{L}(e,i) = fin \\ c_i(0)\cdot a_i(0) & \text{if } \mathcal{L}(e,i) = sub \\ (1-c_i(0))\cdot a_i(0) & \text{if } \mathcal{L}(e,i) = killANDsub \\ (1-c_i(0))\cdot(1-a_i(0)) & \text{if } \mathcal{L}(e,i) = \varepsilon \text{ and } c_i(0) \neq 1 \\ 1-a_i(0) & \text{if } \mathcal{L}(e,i) = \varepsilon \text{ and } c_i(0) = 1. \end{cases}$$

Finally, the initial vertex is $v_{\mathsf{init}} = \big((c_0,d_0,a_0),\ldots,(c_n,d_n,a_n),\Box\big) \in V_\Box$ s.t. for all $i \in [n]$: $(c_i,d_i,a_i) = (\mathcal{C}_i, D_i, \mathcal{A}_i)$ if $I_i = 0$; and $(c_i,d_i,a_i) = (c,0,a)$ with $c(0) = 1$ and $a(I_i) = 1$ otherwise. The finishes the definition of Γ. As explained above, the MDP Γ_τ modelling our problem is the portion of Γ that is reachable from v_{init}. One can check, by a careful inspection of the definitions, that Γ_τ is indeed finite. Let us now illustrate these definitions.

▶ **Example 2.** Figure 1 presents an excerpt of the MDP Γ_τ built from the set of tasks $\tau = \{\tau_h, \tau_s\}$ of Example 1. We denote a distribution p with support $\{x_1, x_2, \ldots, x_n\}$ by $[x_1, p(x_1); x_2, p(x_2); \ldots; x_n p(x_n)]$. When p is s.t. $p(x) = 1$ for some x, we simply denote p by x. States from $V_\Box$ and $V_\bigcirc$ are depicted by rectangles and rounded rectangles respectively. Each state is labelled by (c_h, d_h, a_h) on the top and (c_s, d_s, a_s) below.

A strategy to avoid missing a deadline of τ_h consists in first scheduling τ_s, then τ_h. One then reaches the left-hand part of the graph from which $\Box$ can avoid $\bot$ irrespective of whatever $\bigcirc$ does. Note that other safe strategies are possible: the first step of our algorithm

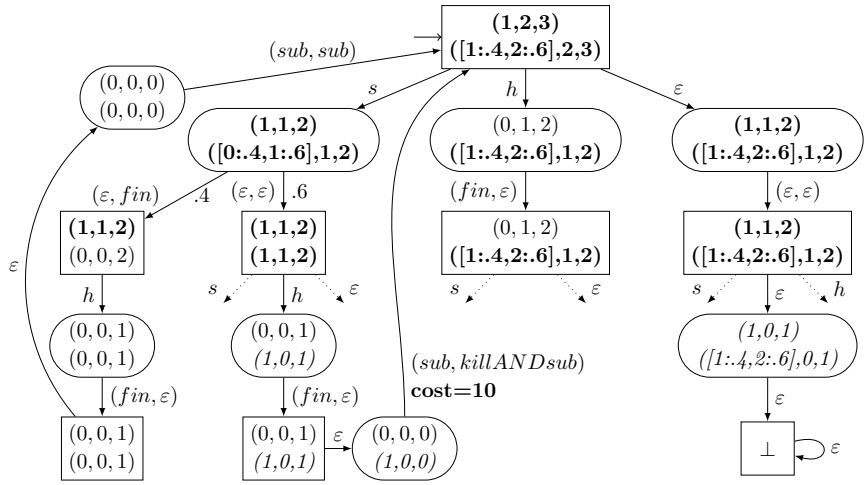


Figure 1 An excerpt of the MDP for Example 1. Tasks in **bold** are active, tasks in *italics* have missed a deadline.

is actually to compute all the *safe* nodes (i.e. those from which $\Box$ can ensure to avoid $\bot$), and then to look for an optimal one (wrt to the cost of missing deadlines for soft tasks) among those.

From state $\langle (1,1,2), ([0{:}.4,1{:}.6],1,2)\rangle$, $\bigcirc$ chooses whether τ_s finishes or not with respective probabilities 0.4 and 0.6. In the latter case, τ_s will miss its deadline, which incurs a cost on the edge where the $killANDsub$ action occurs.

Equipped with these definitions, we can define the problem that we want to solve. The **Safe and optimal scheduler synthesis** problem is stated as, given a set of real-time tasks τ partitioned into hard and soft tasks, and a rational threshold λ whether there exists, in the MDP Γ_τ a strategy $\sigma_\Box$ of $\Box$ s.t.:

- **(i)** $\bot \notin V_{\mathsf{Outs}_{\Gamma_\tau[\sigma]}(v^\tau_{\mathsf{init}})}$, i.e. no hard task misses its deadline. Strategies that enforce this objective are called *safe* strategies; and
- **(ii)** $\mathbb{E}^{\Gamma_\tau[\sigma]}_{v^\tau_{\mathsf{init}}}(\mathsf{MC}) \leq \lambda$, i.e. the expectation of the mean-cost (due to the deadline misses by the soft tasks) is at most λ.

This strategy $\sigma_\Box$ constitutes our scheduler. We will see in the next section that when such a strategy exists, then there exists one which is also *memoryless*.

▶ **Example 3.** Let us continue with Example 2. There are two optimal memoryless strategies, one in which the Scheduler first chooses to execute τ_h, then τ_s; and another where τ_s is scheduled for 1 time unit, and then preempted to let τ_h execute. Since the period of τ_s is 3 and the cost of missing a deadline is 10, for both of these optimal strategies, the soft task's deadline is missed with probability 0.6 during each period and hence the mean-cost is 2. Observe that there is another safe schedule that is not optimal is one in which only τ_h is granted CPU access, and τ_s is never scheduled thus giving a mean-cost of $\frac{10}{3}$.

4 Algorithm and Complexity

In this section, we show that the safe and optimal scheduler synthesis problem is PP-hard and in EXPTIME. We start with the upper bound first.

▶ **Theorem 4.** *The safe and optimal scheduler synthesis problem can be solved in* EXPTIME.

Proof. We sketch an exponential time algorithm that solves our problem. First, we build the MDP $\Gamma_\tau = \langle V, E, A, Prob\rangle$ according to the above definitions. Note that the supports of the distributions and the deadline given in binary, the size of this MDP is exponential in the size of the task set, hence Γ_τ can be built in exponential time. Then, we run the attractor algorithm on Γ_τ (see Appendix A), using $\{\bot\}$ as the set of *unsafe* vertices. This takes polynomial time in the size of Γ_τ, hence exponential time in the size of τ. We obtain a set $V_{unsafe} \subseteq V$ of *losing vertices*. That is, V_{unsafe} contains all the vertices from which $\Box$ cannot guarantee that all the hard tasks will never miss deadlines. We then prune Γ_τ by removing all the vertices of V_{unsafe}, and obtain $\Gamma' = \langle V \setminus V_{unsafe}, E', A, Prob\rangle$. Observe that, by definition of the attractor, whenever a vertex $v \in V_\bigcirc$ has a successor in V_{unsafe}, then $v \in V_{unsafe}$ too. So, the pruning operation either keeps a vertex $v \in V_\bigcirc$ with all its successors, or remove it. The corresponding edges are also removed. Hence, the $Prob$ function is still a probability distribution and Γ' is still an MDP. By property of the attractor, the possible strategies for $\Box$ in Γ' are exactly all the *safe* strategies in Γ_τ. Hence, we can now solve our problem by applying some classical polynomial time algorithm [36, 35] to solve the mean-cost threshold synthesis problem in Γ', and we have the guarantee that for all strategies $\sigma_\Box$: $\mathbb{E}^{\Gamma'[\sigma_\Box]}_{v^\tau_{\mathsf{init}}}(\mathsf{MC}) \leq \lambda$ iff $\bot \notin V_{\mathsf{Outs}_{\Gamma_\tau[\sigma_\Box]}(v^\tau_{\mathsf{init}})}$ and $\mathbb{E}^{\Gamma_\tau[\sigma_\Box]}_{v^\tau_{\mathsf{init}}}(\mathsf{MC}) \leq \lambda$.

Observe that those algorithms compute *memoryless* optimal strategies that map all vertices in $V_\Box$ to an action in $[n] \cup \{\varepsilon\}$, representing which task (if any) should be granted CPU access. This strategy is thus the actual scheduler, and we are done. ◀

Let us now turn our attention to a lower bound on the complexity. As explained in the *related works* section, our problem subsumes classical scheduling problems that are known to be coNP-complete, like the periodic (hard) task scheduling problem [26, 8], and NP-complete, like the clairvoyant scheduling problem for soft tasks [30]. The proof of task scheduling problem for sporadic hard tasks in [7] also applies to our case giving coNP-completeness in a system with only hard tasks that are neither periodic nor sporadic[2]. We now provide a stronger lower bound by establishing PP-hardnes. Recall that PP is the class of languages $L \subseteq \Sigma^*$ recognised by a probabilistic polynomial-time Turing machine M with access to a fair coin such that for all $w \in \Sigma^*$, we have $w \in L$ if and only if M accepts w with probability at least $\frac{1}{2}$. The class PP contains NP, is closed under complement [37] and hence also contains the class coNP. Further, the class PP is contained in PSPACE.

▶ **Theorem 5.** *The safe and optimal schedule synthesis problem is* PP*-hard.*

Proof. We show a reduction from k-th largest subset which has recently been shown to be PP-complete [22]. The k-th largest subset problem is stated as given a finite set A, a size function $h : A \to \mathbb{N}$ assigning strictly positive integer values to elements of A, and two naturals $K, L \in \mathbb{N}$, decide if there exist K or more distinct subsets $S_j \subseteq A$, where $1 \leq j \leq K$, such that $\sum_{o \in S_j} h(o) \leq L$ for all these K or more subsets.

Let $|A| = n$, and $B = \sum_{o \in A} h(o)$. Given an instance of k-th largest subset, we construct a system of n hard tasks and one soft task. Let H denote the set of hard tasks. The first instance of each hard task arrives at time 0. The computation time is $e_i = h(o_i)$ with probability $\frac{1}{2}$, and $e_i = 0$ with probability $\frac{1}{2}$, the deadline $d = B$, and the inter-arrival time $p_i = B$ for all $1 \leq i \leq n$.

The arrival time of the first instance of the soft task is L, its computation time is $B - L$, (relative) deadline is $B - L$ and inter-arrival time is B.

[2] Note that in sporadic tasks, only the minimum inter-arrival time for a task is specified.

Suppose there is a solution to the instance of k-th largest subset problem, i.e. there are at least $r \geq K$ subsets of A such that the sum of the elements in each S_j, where $j \in [r]$ is less than L. Hence in the system constructed above, there are $r \geq K$ subsets of the set H such that corresponding to each subset S_j, where $i \in [r]$, for each $i \in [n]$, hard task i executes with time $h(o_i)$ if $o_i \in S_j$, else i executes with time 0.

Note that since each hard task has two possible computation times that are 0 and $h(o_j)$, there are 2^n combinations of computation times for all the hard tasks, each such combination having a probability of $\frac{1}{2^n}$, and r of them finish before L and for each of these r combinations, the soft task executes to completion. Thus the cost is 0 for r of these combinations, that is, with probability $\frac{r}{2^n}$, while the cost is 1 with probability $1 - \frac{r}{2^n}$. Hence the expected cost is $1 - \frac{r}{2^n}$ over a time period B. The expected mean-cost is $\frac{1}{B} \cdot (1 - \frac{r}{2^n})$. So the expected mean-cost is indeed less than or equal to $\frac{1}{B} \cdot (1 - \frac{K}{2^n})$ iff there are at least K subsets of A each of whose elements sum up to L or less. ◀

As a consequence, our EXPTIME upper bound cannot be improved substantially unless P=PP. We note that our proof implies that we cannot have a short certificate indicating the absence of a safe schedule that meets some minimal performance for the soft tasks unless PP = coNP = NP.

Finally, we note that in our reduction, we associate a system with only one soft task to each instance of the K-th largest subset sum problem. But our reduction can be adapted by turning all *hard tasks* into *soft tasks* with costs > 1. In place of each hard task, we have a soft task, each parameter of the soft task remains the same as the hard task, and the cost of missing the deadline for the soft task is strictly greater than 1 while we have only one soft task as in the previous case whose cost is 1. Arguing as above, we see that minimum expected mean-cost is less than or equal to $\frac{1}{B} \cdot (1 - \frac{K}{2^n})$ iff there are at least K subsets of A each of whose elements sum up to L or less. Hence, we obtain the same lower bound in the case where we have only soft tasks:

▶ **Corollary 6.** *The safe and optimal scheduler synthesis problem is* PP*-hard even in a system with only soft tasks.*

This shows that the non-clairvoyant scheduling of soft tasks where the tasks are described using probability distributions is computationally more difficult than the clairvoyant version (unless NP = PP).

5 A symbolic data-structure for the safety game

Our last theoretical contribution is to propose an antichain-based [16] heuristic to mitigate the high complexity of the problem, and solve the *safety* part of the game in an efficient way (in practice). The core of this approach consists in identifying an ordering $\succeq \subseteq V_\Box \times V_\Box$ on the vertices of $\Box$ and that can be interpreted intuitively as follows: $v_1 \succeq v_2$ means that v_1 is *as difficult* as v_2 (from the point of view of $\Box$, i.e. the Scheduler player). Thus, if $\Box$ has a safe strategy from v_1, then she also has a safe strategy from v_2 (Theorem 9). This implies that the set of safe $\Box$ vertices, (that our algorithm computes in the first place) is *downward-closed* for $\succeq$, a special structure that we will exploit in our implementation (see Section 6).

Let $v^1 = \big((s_1^1, s_2^1, \ldots, s_n^1), \Box\big)$, and $v^2 = \big((s_1^2, s_2^2, \ldots, s_n^2), \Box\big)$ be two vertices of $\Box$ where $s_i^j = (c_i^j, d_i^j, a_i^j)$ for all $j \in \{1, 2\}$ and $i \in [n]$. Intuitively, in order to make sure that, for all tasks $i \in [n]$ its configuration s_i^1 is *at least as difficult as* s_i^2, we could request that:

(i) for all $e_i^2 \in \mathsf{Supp}(c_i^2)$, there is $e_i^1 \in \mathsf{Supp}(c_i^1)$ s.t.: $d_i^2 - e_i^2 \geq d_i^1 - e_i^1$; and
(ii) $d_i^1 \leq d_i^2$; and
(iii) $\mathsf{Supp}(a_i^1) \supseteq \mathsf{Supp}(a_i^2)$.

Indeed, condition (i) means that the amount of work needed for τ_i to complete in v^1 is at least the amount of work needed in v^2, whatever the choices of the task generator (observe that we ignore the probabilities here since we are only interested in safety). Condition (ii) says that the deadline is closer in v^1 than in s^2. Condition (iii) ensures that all next arrivals that can occur in v^2 can also occur in v^1, hence, all the actions that $\bigcirc$ will be able to play from the successors of v^2 should also be present from the successors of v^1.

Let us now argue that we can actually simplify these conditions. Assume the three conditions hold on v^1 and v^2, and consider (iii) i.e. $\mathsf{Supp}(a_i^1) \supseteq \mathsf{Supp}(a_i^2)$. Since the a_i parameters of all tasks are initialised to the same value $\mathcal{A}_i$, this implies that the job of τ_i in v^2 has been submitted earlier than the corresponding job in v^1, which implies that $d_i^1 \geq d_i^2$. This last inequality together with (ii) implies that $d_i^1 = d_i^2$, which means that the two jobs of τ_i in both states have actually been submitted at the same time, hence we must also have $a_i^1 = a_i^2$. This motivates our definition of the $\succeq$ order:

▶ **Definition 7.** Let $v^1 = \big((s_1^1, s_2^1, \ldots, s_n^1), \square\big)$, and $v^2 = \big((s_1^2, s_2^2, \ldots, s_n^2), \square\big)$ be two states of $\square$ where $s_i^j = (c_i^j, d_i^j, a_i^j)$ for all $j \in \{1, 2\}$ and $i \in [n]$. Then, $v^1 \succeq v^2$ iff: for all $i \in [n]$, there exists an injective function $f : \mathsf{Supp}(c_i^2) \to \mathsf{Supp}(c_i^1)$ s.t. for all $e_2 \in \mathsf{Supp}(c_i^2)$: $f(e_2) \geq e_2$, $d_i^1 = d_i^2$ and $a_i^1 = a_i^2$.

Observe that this ordering is defined on the *structure* of the states of the MDP, so it is easy to test simply by inspecting v^1 and v^2. In order to show that $\succeq$ has the right properties, we rely on a variation of the notion of alternating simulation [4], which we adapt to our setting. Fix an MDP $\gamma_\tau = \langle V, E, L, (V_\square, V_\bigcirc), \ell, Prob\rangle$. Then, a relation $\mathcal{R} \subseteq V_\square \times V_\square$ is an alternating simulation relation iff for all $(v_1, v_2) \in \mathcal{R}$, the following holds. For all $v_1' \in \mathsf{Succ}(v_1)$, there is $v_2' \in \mathsf{Succ}(v_2)$ s.t.:

(i) $L(v_2, v_2') \in \{L(v_1, v_1'), \varepsilon\}$; and
(ii) for all $v_2'' \in \mathsf{Succ}(v_2')$ there is $v_1'' \in \mathsf{Succ}(v_1')$ s.t. $\mathcal{L}(v_1', v_1'') = \mathcal{L}(v_2', v_2'')$ and $(v_1'', v_2'') \in \mathcal{R}$.

Then, we can show that:

▶ **Lemma 8.** *$\succeq$ is an alternating simulation relation.*

Proof. Let $(v_1, v_2) \in \succeq$ with $v_1 = (s_{1,1}, s_{2,1}, \ldots s_{n,1}, \square)$ and $v_1 = (s_{1,2}, s_{2,2}, \ldots s_{n,2}, \square)$ Recall that $v_1, v_2 \in V_\square$. Let $i \in [n] \cup \{\varepsilon\}$ be the action of $\square$ from v_1, let $v_1 \xrightarrow{i} v_1'$, and let $s_{i,1} = \langle c_1, d, a\rangle$ and $s_{i,2} = \langle c_2, d, a\rangle$. Since $v_1 \succeq v_2$, by definition of $\succeq$, there exists an injective function $f : \mathsf{Supp}(c_2) \to \mathsf{Supp}(c_1)$ as defined above. Let $e = \min(\mathsf{Supp}(c_2))$. Consider the action from v_2 as $v_2 \xrightarrow{\varepsilon} v_2'$ if $f(e) > e$, else $v_2 \xrightarrow{i} v_2'$.

Now since a_i is the same in both v_1' and v_2', we see that for every action b of $\bigcirc$ such that $v_2' \xrightarrow{b} v_2''$, we have a v_1'' such that $v_1' \xrightarrow{b} v_1''$, and $v_1'', v_2'' \in V_\square$. From the transitions of the MDP as defined in Section 3, it is not difficult to see that $(v_1'', v_2'') \in \succeq$, and we are done. ◀

From this Lemma, and from the fact that the objective of the *safety* part of the game is to avoid reaching $\bot$, we can now deduce that $\succeq$ has the desired property. More precisely, we show that, if $v_1 \succeq v_2$ and Scheduler can schedule all hard tasks from v_1, then we can find a so-called $\succeq$-strategy σ' that is also winning *and* that takes the same choices (when possible, that is, when the same tasks are active) from all pairs of states which are comparable (in particular, from s_2). Formally, we call a strategy $\sigma : V_\square \to [n] \cup \{\varepsilon\}$ *$\succeq$-compatible* iff for all v_1, v_2 with $v_1 \succeq v_2$: either $\sigma(v_1) = \sigma(v_2)$ or $\sigma(v_2) = \varepsilon$. That is, either σ schedules the same task from both states, or it idles the CPU in the "easier" state v_2.

▶ **Theorem 9.** *For all $\Box$ vertices v_1 and v_2 with $v_1 \succeq v_2$, if $\Box$ (Scheduler player) has a safe strategy from v_1, then she has a $\succeq$-compatible safe strategy σ from v_2 (also safe from v_1).*

Proof. Let $v_1 = \big((c_1^1, d_1^1, a_1^1), \ldots, (c_n^1, d_n^1, a_n^1), \Box\big)$ and $v_2 = \big((c_1^2, d_1^2, a_1^2), \ldots, (c_n^2, d_n^2, a_n^2), \Box\big)$ respectively. Given a winning strategy σ' from v_1, we construct an $\succeq$-compatible winning strategy σ from v_1 as follows. Consider $V_{\mathsf{Outs}_{\Gamma(\sigma')}(v_1)}$, the set of all vertices visited from v_1 when σ' is played from v_1. We define σ such that for all $\tilde{v}_1 \in V_{\mathsf{Outs}_{\Gamma(\sigma')}(v_1)}$, we have $\sigma(\tilde{v}_1) = \sigma'(\tilde{v}_1)$. For all vertices $\hat{v}_1 = \big((\hat{c}_1, \hat{d}_1, \hat{a}_1), (\hat{c}_2, \hat{d}_2, \hat{a}_2), \ldots, (\hat{c}_n, \hat{d}_n, \hat{a}_n), \Box\big) \in V_\Box$ s.t. there exists a $\tilde{v}_1 \in V_{\mathsf{Outs}_{\Gamma(\sigma')}(v_1)}$ with $\tilde{v}_1 \succeq \hat{v}_1$, we have $\sigma(\hat{v}_1) = \sigma(\tilde{v}_1)$, when $\sigma(\tilde{v}_1) = i$ for some $i \in [n]$ and $f(\min(\mathsf{Supp}(\hat{c}_{\sigma(\tilde{v}_1)}))) = \min(\mathsf{Supp}(\hat{c}_{\sigma(\tilde{v}_1)}))$ for the injective function f defined above and $\sigma(\hat{v}_1) = \varepsilon$ otherwise. For all the remaining vertices v, we have $\sigma(v) = \sigma'(v)$. Clearly, σ is $\succeq$-compatible. From Lemma 8, since $\succeq$ is an alternating simulation relation and since an ε action can be scheduled from every $\Box$ vertex, it is always possible to construct such a strategy σ.

Now we show that σ is safe from v_2. We show by induction on the length of the play that playing σ from v_2 does not visit $\bot$. Let $\sigma(v_1) = a$, and $\sigma(v_2) = \alpha$, where $a, \alpha \in [n] \cup \{\varepsilon\}$. Let $v_1 \xrightarrow{a} v_{1,\bigcirc}$ and $v_2 \xrightarrow{\alpha} v_{2,\bigcirc}$. Recall that for all $i \in [n]$, we have $d_i^1 = d_i^2$ and $a_i^1 = a_i^2$. Further from the definition of σ, we have that $\bigcirc$ player has the same set of actions from both $v_{2,\bigcirc}$ and $v_{1,\bigcirc}$. Let $v_{2,\bigcirc} \xrightarrow{b} v_2'$ and $v_{1,\bigcirc} \xrightarrow{b} v_1'$, where b is a $\bigcirc$ action and $v_1', v_2' \in V_\Box$. Clearly, $v_1' \succeq v_2'$, and since $v_1' \neq \bot$ then $v_2' \neq \bot$ too (because $\bot$ is not comparable to any other vertex). By induction hypothesis, σ is a $\succeq$-compatible winning strategy from v_2', and we are done. ◀

A consequence of this theorem is that the set $V_{safe}^\Box = V_{safe} \cap V_\Box$ of *safe* $\Box$ *vertices* has a special structure wrt $\succeq$. Formally, $V_{safe}^\Box$ is *downward-closed*, i.e. if $v_1 \in V_{safe}^\Box$, then for all $v_2 \in V_\Box$: $v_1 \succeq v_2$ implies that $v_2 \in V_{safe}^\Box$ as well. We can thus represent $V_{safe}^\Box$ in a compact way by keeping its *maximal elements only*. Formally, for a set S of vertices, we let $\lceil S \rceil = \{v \mid \nexists v' \in S : v' \neq v \text{ and } v' \succeq v\}$ be its *maximal antichain*. When the set S is downward-closed, $\lceil S \rceil$ can be regarded as a symbolic (compact) representation of S. In the next section, we will rely on $A_{safe}^\Box = \lceil V_{safe}^\Box \rceil$.

6 Implementation and Experiments

In this section, we discuss a prototype tool implementing the techniques described so far. The tool uses the networkx library [23] to build the pruned MDP (see the steps outlined in the proof of Theorem 4 in Section 4) containing only the transitions that allow all possible schedules in which no hard task misses a deadline. This is given as an input to the Storm model-checker which analyses the MDP and finds an optimal schedule among the set of safe schedules.

We have run our prototype on a small benchmark with different numbers of hard and soft tasks. We measure the size of the system as the number of vertices in the MDP which is analysed by Storm. We ran our experiments in a MacBook Air with 1.7 GHz Intel Core i5 processor with 2 cores and having 4GB memory. We ran three different experiments:

1. We compute the time taken by our tool to remove the unsafe schedules for increasing number of hard tasks and increasing number of vertices. The results are shown in Table 1. We have only hard tasks in these experiments and the values of all the parameters is less than or equal to 12. We perform the following optimisation when constructing the set of safe vertices. We call a vertex $s = \langle s_1, \ldots, s_n, X\rangle$ with $X \in \{\Box, \bigcirc\}$ to be *immediately unsafe* if there is a task $j \in H$ s.t. $s_j = (c_j, d_j, a_j)$ with $\max(\mathsf{Supp}(c_j)) > d_j$, i.e. in

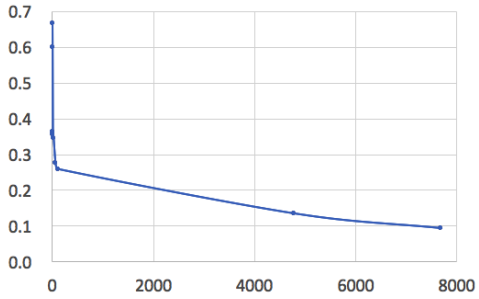


Figure 2 Fraction of vertices in the antichain (Y-axis) with increasing number of safe vertices (X-axis).

Table 1 Removing the unsafe schedules for a set of $|H|$ hard tasks. $|V|$ and $|V_{safe}|$ are the number of initial and safe vertices in the MDP respectively, T_{MDP} is the time to compute the entire MDP and T_{safe} is the time required to remove the unsafe vertices (all times in second)

	$\|H\|$	$\|V\|$	$\|V_{safe}\|$	T_{MDP}	T_{safe}
1	2	32	0	0.02	0.02
2	3	1,155	0	5.49	0.4
3	3	6,550	6,015	231.37	5.29
4	5	4,397	0	122.41	1.58
5	6	2,875	0	53.05	1.21
6	6	7,685	0	339.52	3.88

Table 2 Experiments where the number $|S|$ of soft tasks varies. $|V_{safe}|$ is the number of safe vertices in the MDP that is analysed by STORM and T is the running time of STORM to compute the mean-cost C.

	$\|S\|$	$\|V_{safe}\|$	T	C
1	1	230	0.03	0
2	2	5,369	0.39	0.07
3	3	150,895	73.09	0.28

Table 3 Experiments where the number $|H|$ of hard tasks varies. $|V_{safe}|$ is the number of safe vertices in the MDP that is analysed by STORM and T is the running time of STORM to compute the mean-cost C.

	$\|H\|$	$\|V_{safe}\|$	T	C
1	1	560	0.05	0
2	2	8,040	2.35	0
3	3	9,626	6.08	0

the rct distribution, the maximum remaining computation time exceeds the remaining time before the deadline. While computing the set of reachable vertices, once we detect that a vertex is immediately unsafe, we stop exploring from that node further. We note that both T_{MDP} and T_{safe} are proportional to the number $|V|$ of vertices rather than the number of hard tasks in the system and the number of vertices may not be directly related to the number of tasks in the system, but also depends on the parameters of the tasks. Most of these experiments were designed so that a safe schedule does not exist for the set of hard tasks considered, that is, all the vertices in V were found to be unsafe. We however have one experiment (number 3 in which most of the vertices are safe and we note that T_{safe} is not negligible in this case.

2. We run our complete algorithm on examples where all the task parameters are less than or equal to 10, we have only one hard task and the number of *soft* tasks varies, see Table 2.
3. Symmetrically, we run our complete algorithm on examples where all the task parameters are less than or equal to 9, we have only one soft task and the number of *hard* tasks varies, see Table 3.

Symbolic vertices and transitions with antichain. Finally, let us explain how we have exploited the relation defined in Section 5 in our prototype. As explained earlier, the

set $V^{\Box}_{safe}$ of *safe* $\Box$ vertices of Γ_τ is *downward-closed* for the relation $\succeq$, and can be thus represented by its maximal antichain $A^{\Box}_{safe} = \lceil V^{\Box}_{safe} \rceil$. Our experiments show that $A^{\Box}_{safe}$ is, in practice, much smaller that $V^{\Box}_{safe}$, as can be seen in Figure 2, where we plot the ratio $\frac{|A^{\Box}_{safe}|}{|V^{\Box}_{safe}|}$ against $|V^{\Box}_{safe}|$ on a set of randomly generated systems where some have only hard tasks while some are with both hard and soft tasks. The number of different tasks varies from 1 to 6. We notice that the fraction reduces with increasing number of vertices and it reduces to less than 10% when the number of safe vertices is a few thousands.

We then exploit $A^{\Box}_{safe}$ to obtain a more succinct and more structured input file to the Storm model checker. The input syntax of Storm uses a guarded command language, where the states of the system are described by means of variables that are tested and updated by a transition. So, a possible transition could be of the form:

```
rct1_1=1 & d1=3 & p1_1=4 & ... ->  (rct1_1'=1) & (d1'=2) & (p1_1'=3) & ...;
```

where `rct1_1`, `d1`, etc are the variables that encode the system state, and their primed versions characterise the successors from $V_\Box$, as the vertices from $V_\bigcirc$ are not explicitly encoded in the Storm models. In the experiments we have described above, this is how we have encoded the Storm models: all transitions from all safe states are encoded explicitly by means of conjunctions of equalities on the systems states.

Now that we have a compact representation $A^{\Box}_{safe} = \{v_1, v_2, \ldots, v_n\}$ of the safe states, we further improve the encoding of the Storm model, by testing, in the guard of the transition, whether the potential successor state v' is s.t. $v_i \succeq v'$ for some i. We then use inequalities in the guards of the transition and describe several possible transitions at once, for example:

```
rct1_1>=0 & d1=3 & ... &
((rct1_1<=1 & rct2_1-1<=2 & rct2_2-1<=4) | (rct1_1<=2 & ...) | ...)
-> (rct1_1'=rct1_1) & ...;
```

Observe that now we test that variable `rct_1` is ≥ 0 and constrain that it is unmodified (`rct1_1'=rct1_1`), so the guard is satisfied by potentially several vertices at once. The part of the guard that appears on the second line tests whether for a vertex v that is reached after the transition, there exists a $v' \in A^{\Box}_{safe}$ such that $v' \succeq v$. See Appendix C for a more detailed discussion of this new encoding. We however noted that though this approach produces a succinct input file for Storm, the latter constructs an MDP with all the vertices and transitions among them appearing explicitly in the MDP.

Another possible optimisation that our preorder $\succeq$ allows would be to compute $A^{\Box}_{safe}$ *directly* by maintaining, during the fix point computation of the attractor, sets that are antichains only, in the spirit of [20]. This has the potential to speed up the computation of the safe states. We leave this implementation for future works.

7 Discussion and Future Work

In this paper, given a set of hard and soft tasks, we described an algorithm to compute a strategy that is safe and has the minimal expected mean-cost. We formalise the construction of an MDP and show that a safe and optimal schedule can be implemented as a simple table lookup unlike many of the existing scheduling algorithm that requires more computations during scheduling the tasks. We also implemented a prototype of a tool and use Storm model-checker to show that our approach can indeed be used in practice.

Using reinforcement learning. Our algorithm relies on a stochastic model for the arrival of the tasks and the duration of the tasks. If this stochastic model is not available, techniques like reinforcement learning can be used for the online construction of a stochastic model during interaction with the tasks. The challenge here is to combine reinforcement learning with techniques to maintain the safety for the deadline of the hard tasks. Algorithms to combine learning with hard guarantees are currently explored [44].

References

1 L. Abeni and G. Buttazzo. Integrating Multimedia Applications in Hard Real-Time Systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, RTSS '98, pages 4–13, Washington, DC, USA, 1998. IEEE Computer Society.

2 L. Abeni and G. C. Buttazzo. Stochastic Analysis of a Reseveration Based System. In *Proceedings of the 15th International Parallel & Distributed Processing Symposium (IPDPS-01), San Francisco, CA, April 23-27, 2001*, pages 92–98, 2001.

3 S. Almagor, O. Kupferman, and Y. Velner. Minimizing Expected Cost Under Hard Boolean Constraints, with Applications to Quantitative Synthesis. In *Proc. 27th Int. Conf. on Concurrency Theory*, volume 59 of *LIPIcs*, pages 9:1–9:15, 2016.

4 R. Alur, T.A. Henzinger, O. Kupferman, and M.Y. Vardi. Alternating refinement relations. In *Proc. 9th Int. Conf. on Concurrency Theory*, volume 1466 of *Lecture Notes in Computer Science*, pages 163–178. Springer, 1998.

5 A. Atlas and A. Bestavros. Statistical Rate Monotonic Scheduling. In *Proceedings of the 19th IEEE Real-Time Systems Symposium, Madrid, Spain, December 2-4, 1998*, pages 123–132, 1998.

6 H. Aydin, R. G. Melhem, D. Mossé, and P. Mejía-Alvarez. Optimal Reward-Based Scheduling of Periodic Real-Time Tasks. In *Proceedings of the 20th IEEE Real-Time Systems Symposium, Phoenix, AZ, USA, December 1-3, 1999*, pages 79–89, 1999.

7 S. K. Baruah, A. K. Mok, and L. E. Rosier. Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor. In *Proceedings of the Real-Time Systems Symposium - 1990, Lake Buena Vista, Florida, USA, December 1990*, pages 182–190, 1990.

8 S. K. Baruah, L. E. Rosier, and R. R. Howell. Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor. *Real-Time Systems*, 2(4):301–324, 1990.

9 V. Bruyère, E. Filiot, M. Randour, and J-F. Raskin. Meet Your Expectations With Guarantees: Beyond Worst-Case Synthesis in Quantitative Games. In *Proc. 31th Symp. on Theoretical Aspects of Computer Science*, volume 25 of *LIPIcs*, pages 199–213, 2014.

10 G. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer Publishing Company, Incorporated, 3rd edition, 2011.

11 H. Chetto and M. Chetto. Some Results of the Earliest Deadline Scheduling Algorithm. *IEEE Trans. Software Eng.*, 15(10):1261–1269, 1989.

12 L. Clemente and J-F. Raskin. Multidimensional beyond Worst-Case and Almost-Sure Problems for Mean-Payoff Objectives. In *Proc. 30th IEEE Symp. on Logic in Computer Science*, pages 257–268, 2015.

13 C. Dehnert, S. Junges, J-P. Katoen, and M. Volk. A Storm is Coming: A Modern Probabilistic Model Checker. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, Proceedings, Part II*, pages 592–600, 2017.

14 S. K. Dhall and C. L. Liu. On a Real-Time Scheduling Problem. *Oper. Res.*, 26(1):127–140, February 1978.

15 J. L. Díaz, D. F. García, K. Kim, C-G. Lee, L. L. Bello, J. M. López, S. L. Min, and O. Mirabella. Stochastic Analysis of Periodic Real-Time Systems. In *Proceedings of the*

23rd IEEE Real-Time Systems Symposium, RTSS '02, pages 289–300, Washington, DC, USA, 2002. IEEE Computer Society.

16 L. Doyen and J.-F. Raskin. Antichains for the Automata-Based Approach to Model-Checking. *Logical Methods in Computer Science*, 5(1), 2009.

17 J. Filar and K. Vrieze. *Competitive Markov decision processes*. Springer, 1997.

18 M. K. Gardner. *Probabilistic Analysis and Scheduling of Critical Soft Real-Time Systems*. PhD thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 1999.

19 M. K. Gardner and J. W.-S. Liu. Analyzing Stochastic Fixed-Priority Real-Time Systems. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, TACAS '99, pages 44–58, London, UK, UK, 1999. Springer-Verlag.

20 G. Geeraerts, J. Goossens, T-V-A Nguyen, and A. Stainer. Synthesising succinct strategies in safety games with an application to real-time scheduling. *Theoretical Computer Science*, 735:24–49, 2018.

21 John Gill. Computational Complexity of Probabilistic Turing Machines. *SIAM Journal on Computing*, 6(4):675–695, 1977.

22 C. Haase and S. Kiefer. The complexity of the Kth largest subset problem and related problems. *Information Processing Letters*, 116(2):111–115, 2016.

23 A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring Network Structure, Dynamics, and Function Using NetworkX. In *Proceedings of the 7th Python in Science Conference*, pages 11–15, 2008.

24 K. S. Hong and J. Y.-T. Leung. On-Line Scheduling of Real-Time Tasks. In *Proceedings of the 9th IEEE Real-Time Systems Symposium (RTSS '88), December 6-8, 1988, Huntsville, Alabama, USA*, pages 244–250, 1988.

25 J. P. Lehoczky, L. Sha, and I. Strosnider. Enhanced aperiodic responsiveness in a hard real-time environment. In *Real-Time Systems Symposium*, pages 261–270, 1987.

26 J. Y.-T. Leung and M. L. Merrill. A Note on Preemptive Scheduling of Periodic, Real-Time Tasks. *Information Processing Letters*, 11(3):115–118, 1980.

27 J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Perform. Eval.*, 2(4):237–250, 1982.

28 K-J Lin, S. Natarajan, and J. W.-S. Liu. Imprecise Results: Utilizing Partial Comptuations in Real-Time Systems. In *Proceedings of the 8th IEEE Real-Time Systems Symposium (RTSS '87), December 1-3, 1987, San Jose, Califorinia, USA*, pages 210–217, 1987.

29 C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, January 1973.

30 J. W. S. Liu, K. J. Lin, W. K. Shih, A. C. Yu, J. Y. Chung, and W. Zhao. *Algorithms for Scheduling Imprecise Computations*, pages 203–249. Springer US, Boston, MA, 1991.

31 Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean. A Statistical Response-Time Analysis of Real-Time Embedded Systems. In *Proceedings of the 33rd IEEE Real-Time Systems Symposium, RTSS 2012, San Juan, PR, USA, December 4-7, 2012*, pages 351–362, 2012.

32 Y. Lu, T. Nolte, J. Kraft, and C. Norström. A Statistical Approach to Response-Time Analysis of Complex Embedded Real-Time Systems. In *16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2010, Macau, SAR, China, 23-25 August 2010*, pages 153–160, 2010.

33 D. Maxim, R. I. Davis, L. Cucu-Grosjean, and A. Easwaran. Probabilistic analysis for mixed criticality systems using fixed priority preemptive scheduling. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems, RTNS 2017, Grenoble, France, October 04 - 06, 2017*, pages 237–246, 2017.

34 A. K. Mok. *Fundamental design problems of distributed systems for the hard-real-time environment*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1983.

35 C. H. Papadimitriou and J. N. Tsitsiklis. The Complexity of Markov Decision Processes. *Math. Oper. Res.*, 12(3):441–450, 1987.

36 M.L. Puterman. *Markov Decision Processes*. Wiley, 1994.

37 J. Simon. *On Some Central Problems in Computational Complexity*. PhD thesis, Cornell University, Ithaca, NY, USA, 1975.

38 B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic task scheduling for Hard-Real-Time systems. *Real-Time Systems*, 1(1):27–60, June 1989.

39 M. Spuri and G. Buttazzo. Efficient Aperiodic Service Under Earliest Deadline Scheduling. In *Proceedings of the 15th IEEE Real-Time Systems Symposium (RTSS '94), San Juan, Puerto Rico, December 7-9, 1994*, pages 2–11, 1994.

40 M. Spuri and G. Buttazzo. Scheduling aperiodic tasks in dynamic priority systems. *Real-Time Systems*, 10(2):179–210, 1996.

41 A. Srinivasan and J. H. Anderson. Efficient Scheduling of Soft Real-time Applications on Multiprocessors. *J. Embedded Comput.*, 1(2):285–302, April 2005.

42 Wolfgang Thomas. On the Synthesis of Strategies in Infinite Games. In *STACS*, pages 1–13, 1995. doi:10.1007/3-540-59042-0_57.

43 T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J. W.-S. Liu. Probabilistic Performance Guarantee for Real-time Tasks with Varying Computation Times. In *Proceedings of the Real-Time Technology and Applications Symposium*, RTAS '95, pages 164–173, Washington, DC, USA, 1995. IEEE Computer Society.

44 M. Wen and U. Topcu. Probably Approximately Correct Learning in Stochastic Games with Temporal Logic Specifications. In *Proc. 25th Int. Joint Conf. on Artificial Intelligence*, pages 3630–3636. IJCAI/AAAI Press, 2016.

A Attractor algorithm for Safety Synthesis

The algorithm consists of computing all the vertices from which $\Box$ *cannot avoid* reaching the *unsafe vertices*. To this end, the algorithm computes a sequence of sets of vertices $(A_i)_{i\geq 0}$ defined as follows:

(i) $A_0 = V \setminus V_{\mathsf{safe}}$; and

(ii) for all $i \geq 0$: $A_{i+1} = A_i \cup \{v \in V_\Box \mid \mathsf{Succ}(v) \subseteq A_i\} \cup \{v \in V_\bigcirc \mid \mathsf{Succ}(v) \cap A_i \neq \emptyset\}$.

That is, the sequence $(A_i)_{i\geq 0}$ is initialised to the set of *unsafe* vertices. Then, the algorithm grows this set of vertices by adding:

(i) vertices belonging to $\Box$ whose set of successors has been entirely identified as unsafe in a previous step; and

(ii) vertices belonging to $\bigcirc$ having at least one unsafe successor.

It is easy to check that this sequence converges after at most $|V|$ steps (the graph of the MDP being finite) and returns the set of vertices $\mathsf{Attr}(V \setminus V_{\mathsf{safe}})$ from which $\Box$ has no strategy to stay within V_{safe}. Hence, $\Box$ has a strategy $\sigma_\Box$ to stay within V_{safe} from all vertices in which is s.t. $\sigma_\Box(v) \notin \mathsf{Attr}(V \setminus V_{\mathsf{safe}})$ (any successor of v satisfying this criterion yields a safe strategy).

B Comparison with Relevant Scheduling Algorithms

Several works in the literature consider real-time scheduling of systems with both soft and hard tasks. A prominent line of works among them is based on the notion of *servers* [10, 40] to handle soft tasks. Algorithms for preemptive uniprocessor scheduling following this approach

include Priority Exchange [40, 25], Sporadic Server [40, 38], Total Bandwidth Server [40], Earliest Deadline Late (EDL) Server [11, 39, 40], Constant Bandwidth Server [1], etc. Their performance are measured according to the *responsiveness* to each of the soft task requests without compromising the schedulability of the hard tasks. The responsiveness is defined as the difference between the time of completion of a request and the time of its arrival in the system. However, those algorithms do not take into account a stochastic model of the tasks as in our problem nor a notion of deadline and cost for the soft tasks. Hence they do not provide solutions to the problem that we consider in this paper.

The algorithm EDL is known to be optimal for dynamic priority assignment [11]. An EDL server algorithm is a dynamic slack-stealing algorithm in which the active periodic tasks are processed as late as possible. The basic idea behind the EDL server is to use the idle times of an EDL schedule to execute aperiodic requests (soft tasks) as soon as possible. When there are no aperiodic activities in the system, periodic tasks are scheduled according to the earliest deadline first (EDF) algorithm. An important property of EDL is that it guarantees the maximum available idle time that is used for an optimal server mechanism for soft aperiodic activities.

In order to evaluate the potential of those solutions for our problem, we consider the following modified version of EDL:

- The hard tasks are scheduled as late as possible following an EDF among the hard tasks, without compromising their schedulability.
- At every time, when a hard task is not scheduled, the active soft tasks are also scheduled according to EDF.

The algorithm is preemptive as in the original setting.

The example below shows that in our setting, the ratio of the expected costs obtained with the modified EDL and the the expected cost of the optimal strategy can be arbitrarily large.

▶ **Example 10.** Consider a system with three tasks: one hard task h and two soft tasks s_1 and s_2. The hard task h has execution time, deadline and a period of 2, 3 and 3 respectively. The execution time, deadline and period of s_1 are 1, 3 and 3 respectively. For s_2, the execution time and the deadline are 1 and 2 respectively while the inter-arrival time is 3 with probability 0.1 and 6 with probability 0.9. Let the cost of missing the deadline for an instance of s_1 and s_2 be c_1 and c_2 respectively and let $c_2 > c_1$. The first instance of each of h and s_1 arrives at time 0 while the first instance of s_2 arrives at time 1. We divide the entire timeline into *blocks* of 3 time units with consecutive odd and even blocks and the first block is an odd block. In every even block, an instance of task s_2 appears with probability 0.1 while in every odd block excluding the first block, it appears with probability 0.9. Thus in the optimal strategy, in every even block, s_1 misses its deadline with probability 0.1 while in every odd block it misses its deadline with probability 0.9. The optimal schedule prioritises scheduling s_2 over scheduling s_1 since $c_2 > c_1$. A strategy that produces the minimum expected mean cost thus has an associated cost proportional to $0.1 \cdot c_1 + 0.9 \cdot c_1 = c_1$.

The modified version of the EDL, on the other hand, in both odd and even blocks, schedules the soft task s_1. Thus with probability 0.1, an instance of the soft task s_2 misses its deadline in every even interval and with probability 0.9, it misses its deadline in every odd interval and hence the minimum expected mean cost is proportional to c_2.

As we increase c_2, we see that the ratio of the costs obtained from using the modified EDL and the optimal strategy cannot be bounded above.

Note that the above example could be simplified by considering a Dirac distribution on the inter-arrival time of task s_2 that can be set to 3. However, the example above illustrates the robustness of our approach in the sense that it can find an optimal schedule even when we consider arbitrary probability distributions. The optimal schedule generated by our approach indeed changes with the change in the distribution. We thus have the following proposition.

▶ **Proposition 11.** *There exists a family of systems S in which the ratio of the expected mean costs of missing the deadlines of the soft tasks by using the modified EDL and the optimal strategy (as obtained by our algorithm) can be arbitrarily large.*

Further, EDL suffers from the following problems. Although optimal with respect to the finishing time of the soft tasks, it involves a heavy overload for the computation of the idle times (slacks) that makes it less practical [10]. As seen in Proposition 11, since EDL does not consider any cost associated to missing deadlines of the soft tasks, it is not optimal for our setting. Furthermore, it does not take advantage of the information that is given by the stochastic model of the tasks.

We also consider a simple adaptation of the EDF algorithm to schedule hard and soft tasks to our setting that we call the two- stage EDF.

Two-stage EDF. The two-stage EDF is described as follows. Among the set of safe strategies, first the hard tasks are scheduled by EDF strategy. Next the soft tasks are scheduled by EDF strategy only when there does not exist an active hard task in the system. We show that in the worst case, the two-stage EDF algorithm can be arbitrarily bad in terms of the mean cost.

▶ **Proposition 12.** *There exists a family of systems S in which the ratio of the expected mean costs of missing the deadlines of the soft tasks by using the two-stage EDF algorithm and the optimal strategy (as obtained by our algorithm) can be arbitrarily large.*

Proof. Consider a deterministic system with one hard and one soft task. Let the execution time, deadline and period of the hard task be $1, 2$ and 2 respectively. The parameters for the soft task are $1, 1$ and 2 respectively. The first instance of both the hard and the soft task arrives the system at time 0 and let the cost of missing the deadline for each job of the soft task be 20. The two-phase EDF first schedules the hard task and thus for every instance of the soft task, it misses the deadline. Since the period is 2, the expected mean cost for missing the deadline for the soft task instances is 10.

On the other hand, the optimal schedule generated by our approach always schedules the soft task before the hard task and hence the soft task always finishes execution before the deadline and hence the mean cost is 0 and we are done. ◀

We also have an implementation of the two-stage-EDF and compare the time taken to compute the mean-cost by our optimal algorithm and the two-stage EDF. Given a specific schedule (two-stage EDF in our experiments) it corresponds to a fixed strategy of Scheduler, and hence the state space of the MDP that is analysed by Storm to compute the expected mean-cost for this particular strategy is usually only a small part entire state space of the MDP that is given as an input to Storm. Computing the cost corresponding to an optimal schedule giving the minimum expected mean-cost, on the other hand, requires Storm to find the corresponding optimal strategy of Scheduler which involves exploring the entire state space of this MDP.

In Table 4, $|Soft|$ denotes the number of soft tasks, S_{EDF} denotes the number of vertices in the system that is analysed by Storm when we consider the two-stage EDF schedule and T_{EDF} is the time required to analyse the system for two-stage EDF and C_{EDF} denotes the

Table 4 Comparison between two-stage EDF and our optimal algorithm with different number of soft tasks.

	$\|Soft\|$	S_{EDF}	S_{OPT}	T_{EDF}	T_{OPT}	C_{EDF}	C_{OPT}
1	1	71	230	0.03	0.03	0.18	0
2	2	876	5369	0.08	0.39	0.22	0.07
3	3	18273	150895	4.93	73.09	0.68	0.28

Table 5 Comparison between two-stage EDF and our optimal algorithm with different number of hard tasks.

	$\|Hard\|$	S_{EDF}	S_{OPT}	T_{EDF}	T_{OPT}	C_{EDF}	C_{OPT}
1	1	109	560	0.04	0.05	0.07	0
2	2	810	8040	0.38	2.35	0.26	0
3	3	808	9626	0.93	6.08	0.24	0

mean-cost that we obtain for two-stage EDF. Similarly, we have S_{OPT}, T_{OPT} and C_{OPT} for our optimal algorithm. The columns S_{OPT}, T_{OPT} and C_{OPT} are the same as S_{safe}, T and C in Table 2 and Table 3 respectively. In these experiments, all the parameters, that is, the supports in the execution time distribution, the deadline and the supports in the inter-arrival time distribution for every task have values less than or equal to 10.

In Table 5, the first column $|Hard|$ denotes the number of hard tasks in the system. We use one soft task. The trends of the results are similar to those in Table 4. The values of all the parameters used in these experiments is less than or equal to 9.

Further, there have been several studies to analyse quality of service driven applications. These studies use stochastic tools to analyse the execution times of different tasks and their effects on the quality of service [15, 43, 18, 19, 5, 2, 32, 31, 33]. Those methods do not propose synthesis techniques and consider that the scheduler preexists.

C Optimising the Storm input file with antichains

An example of a transition using concrete vertices is like the following. Here every state is represented concretely. `[hard2]` is the task that is executed, that is, it corresponds to the second hard task as specified in the input file.

```
[hard2] rct1_1=1 & d1=3 & p1_1=4 & p1_2=5 &
rct2_1=3 & rct2_2=5 & d2=5 & p2_1=6 & p2_2=7 ->
(rct1_1'=1) & (d1'=2) & (p1_1'=3) & (p1_2'=4) &
(rct2_1'=2) & (rct2_2'=4) & (d2'=4) & (p2_1'=5) & (p2_2'=6);
```

The variable `[rct1_1]` denotes the first element in the support of the rct distribution of task τ_1 when the elements of the support are arranged in an increasing order, while `[rct2_1]` denotes the first element of the support in the rct distribution of task τ_2 and so on. `d1` is the remaining deadline of task τ_1, and `d2` denotes the remaining deadline for task τ_2. `p1_1` and `p1_2` respectively denote the first and the second elements in an increasing order in the support of the remaining time before the arrival of the next job for task τ_1. The part of the transition on the left side of `->` is the guard while the part on its right is the new state reached. Note that in this particular example, we have only state that is reached following the transition.

A transition using symbolic states is the following.

```
[hard2] rct1_1>=0 & d1=3 & p1_1=4 & p1_2=5 & rct2_1>=2 &
rct2_2>=4 & d2=5 & p2_1=6 & p2_2=7 &
((rct1_1<=1 & rct2_1-1<=2 & rct2_2-1<=4) |
(rct1_1<=2 & rct2_1-1<=1 & rct2_2-1<=3)) ->
(rct1_1'=rct1_1) & (d1'=2) & (p1_1'=3) & (p1_2'=4) &
(rct2_1'=rct2_1-1) & (rct2_2'=rct2_2-1) & (d2'=4) & (p2_1'=5) & (p2_2'=6);
```

Note that in the rct distributions, we have inequalities instead of equalities and hence transitions corresponding to several concrete states are represented by this. The part `((rct1_1<=1 & rct2_1-1<=2 & rct2_2-1<=4) | (rct1_1<=2 & rct2_1-1<=1 & rct2_2-1<=3))` denotes that the state reachable following the transition should be less difficult than at least one of the two elements of the antichain. Each disjunct corresponds to a comparison with an element of the antichain.

The Δ-Framework

Furio Honsell
Dept. of Mathematics, Computer Science and Physics, University of Udine, Via delle Scienze, 206, 33100 Udine, Italy
furio.honsell@uniud.it

Luigi Liquori
Université Côte d'Azur, INRIA Sophia Antipolis – Méditerranée 2004 Route des Lucioles – BP 93 FR-06902 Sophia Antipolis, France
Luigi.Liquori@inria.fr

Claude Stolze
Université Côte d'Azur, INRIA Sophia Antipolis – Méditerranée 2004 Route des Lucioles – BP 93 FR-06902 Sophia Antipolis, France
Claude.Stolze@inria.fr

Ivan Scagnetto
Dept. of Mathematics, Computer Science and Physics, University of Udine, Via delle Scienze, 206, 33100 Udine, Italy
ivan.scagnetto@uniud.it

Abstract

We introduce the Δ-framework, LF_Δ, a dependent type theory based on the Edinburgh Logical Framework LF, extended with the *strong proof-functional connectives*, *i.e.* strong intersection, minimal relevant implication and strong union. Strong proof-functional connectives take into account the shape of logical proofs, thus reflecting polymorphic features of proofs in formulæ. This is in contrast to classical or intuitionistic connectives where the meaning of a compound formula depends only on the truth value or the provability of its subformulæ. Our framework encompasses a wide range of type disciplines. Moreover, since relevant implication permits to express subtyping, LF_Δ subsumes also Pfenning's refinement types. We discuss the design decisions which have led us to the formulation of LF_Δ, study its metatheory, and provide various examples of applications. Our strong proof-functional type theory can be plugged in existing common proof assistants.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Logic of programs, type theory, λ-calculus

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.37

1 Introduction

This paper provides a unifying framework for two hitherto unreconciled understandings of types: *i.e.* types-as-predicates *à la* Curry and types-as-propositions (sets) *à la* Church. The key to our unification consists in introducing *strong proof-functional connectives* [40, 3, 4] in a dependent type theory such as the Edinburgh Logical Framework (LF) [22]. Both Logical Frameworks and Proof-Functional Logics consider proofs as first class citizens, albeit differently. Strong proof-functional connectives take seriously into account the shape of logical proofs, thus allowing for polymorphic features of proofs to be made explicit in formulæ. Hence they provide a finer semantics than classical/intuitionistic connectives, where the meaning of a compound formula depends only on the *truth value* or the *provability* of its

subformulæ. However, existing approaches to strong proof-functional connectives are all quite idiosyncratic in mentioning proofs. Existing Logical Frameworks, on the other hand, provide a uniform approach to proof terms in object logics, but they do not fully capitalize on subtyping.

This situation calls for a natural combination of the two understandings of types, which should benefit both worlds. On the side of Logical Frameworks, the expressive power of the metalanguage would be enhanced thus allowing for shallower encodings of logics, a more principled use of subtypes [37], and new possibilities for formal reasoning in existing interactive theorem provers. On the side of type disciplines for programming languages, a principled framework for proofs would be provided, thus supporting a uniform approach to "proof reuse" practices based on type theory [38, 12, 20, 9, 6].

Therefore, in this paper, we extend LF with the connectives of *strong intersection*, *strong union*, and *minimal relevant implication* of Proof-Functional Logics [40, 3, 4]. We call this extension the Δ-framework (LF_Δ), since it builds on the Δ-calculus [31]. Moreover, we illustrate by way of examples, that LF_Δ subsumes many expressive type disciplines in the literature [37, 3, 4, 38, 12].

It is not immediate to extend the judgments-as-type, Curry-Howard paradigm to logics supporting strong proof-functional connectives, since these connectives need to compare the shapes of derivations and do not just take into account the provability of propositions, *i.e.* the inhabitation of the corresponding type. In order to capture successfully strong logical connectives such as ∩ or ∪, we need to be able to express the rules:

$$\frac{\mathcal{D}_1 : A \quad \mathcal{D}_2 : B \quad \mathcal{D}_1 \equiv \mathcal{D}_2}{A \cap B}\ (\cap I) \qquad \frac{\mathcal{D}_1 : A \supset C \quad \mathcal{D}_2 : B \supset C \quad A \cup B \quad \mathcal{D}_1 \equiv \mathcal{D}_2}{C}\ (\cup E)$$

where ≡ is a suitable equivalence between logical proofs. Notice that the above rules suggest immediately intriguing applications in polymorphic constructions, *i.e.* the same evidence can be used as a proof for different statements. Pottinger [40] was the first to study the strong connective ∩. He contrasted it to the intuitionistic connective ∧ as follows: *"The intuitive meaning of ∩ can be explained by saying that to assert $A \cap B$ is to assert that one has a reason for asserting A which is also a reason for asserting B ... (while) ... to assert $A \wedge B$ is to assert that one has a pair of reasons, the first of which is a reason for asserting A and the second of which is a reason for asserting B".* A logical theorem involving intuitionistic conjunction which does not hold for strong conjunction is $(A \supset A) \wedge (A \supset B \supset A)$, otherwise there should exist a closed λ-term having simultaneously both one and two abstractions. Lopez-Escobar [32] and Mints [35] investigated extensively logics featuring both strong and intuitionistic connectives especially in the context of *realizability* interpretations.

Dually, it is in the ∪-elimination rule that proof equality needs to be checked. Following Pottinger, we could say that *asserting $(A \cup B) \supset C$ is to assert that one has a reason for $(A \cup B) \supset C$, which is also a reason to assert $A \supset C$ and $B \supset C$.* The two connectives differ since the intuitionistic theorem $((A \supset B) \vee B) \supset A \supset B$ is not derivable for ∪, otherwise there would exist a term which behaves both as **I** and as **K**.

Following Barbanera and Martini [4], *Minimal Relevant Implication*, $\supset_r$, can be viewed as a special case of implication whose related function space is the simplest possible one, namely the one containing only the identity function. The operators ⊃ and $\supset_r$ differ, since $A \supset_r B \supset_r A$ is not derivable. Relevant implication allows for a natural introduction of subtyping, in that $A \supset_r B$ morally means $A \leqslant B$. Relevant implication amounts to a notion of "proof-reuse". Combining the remarks in [4, 3], minimal relevant implication, strong

$$\frac{B \vdash M : \sigma \quad B \vdash M : \tau}{B \vdash M : \sigma \cap \tau}\ (\cap I) \qquad \frac{B \vdash M : \sigma \cap \tau}{B \vdash M : \sigma}\ (\cap E_i) \qquad \frac{B \vdash M : \sigma \cap \tau}{B \vdash M : \tau}\ (\cap E_r)$$

$$\frac{B \vdash M : \sigma}{B \vdash M : \sigma \cup \tau}\ (\cup I_l) \qquad \frac{B \vdash M : \tau}{B \vdash M : \sigma \cup \tau}\ (\cup I_r)$$

$$\frac{B, x{:}\sigma \vdash M : \rho \quad B, x{:}\tau \vdash M : \rho \quad B \vdash N : \sigma \cup \tau}{B \vdash M[N/x] : \rho}\ (\cup E) \qquad \frac{B \vdash M : \sigma \quad \sigma \leq \tau}{B \vdash M : \tau}\ (Sub)$$

$$\frac{x{:}\sigma \in B}{B \vdash x : \sigma}\ (Var) \qquad \frac{B \vdash M : \sigma \to \tau \quad B \vdash N : \sigma}{B \vdash M\,N : \tau}\ (App) \qquad \frac{B, x{:}\sigma \vdash M : \tau}{B \vdash \lambda x.M : \sigma \to \tau}\ (Abs)$$

(1)	$\sigma \leqslant \sigma \cap \sigma$	(8)	$\sigma_1 \leqslant \sigma_2, \tau_1 \leqslant \tau_2 \Rightarrow \sigma_1 \cup \tau_1 \leqslant \sigma_2 \cup \tau_2$
(2)	$\sigma \cup \sigma \leqslant \sigma$	(9)	$\sigma \leqslant \tau, \tau \leqslant \rho \Rightarrow \sigma \leqslant \rho$
(3)	$\sigma \cap \tau \leqslant \sigma, \sigma \cap \tau \leqslant \tau$	(10)	$\sigma \cap (\tau \cup \rho) \leqslant (\sigma \cap \tau) \cup (\sigma \cap \rho)$
(4)	$\sigma \leqslant \sigma \cup \tau, \tau \leqslant \sigma \cup \tau$	(11)	$(\sigma \to \tau) \cap (\sigma \to \rho) \leqslant \sigma \to (\tau \cap \rho)$
(5)	$\sigma \leqslant \omega$	(12)	$(\sigma \to \rho) \cap (\tau \to \rho) \leqslant (\sigma \cup \tau) \to \rho$
(6)	$\sigma \leqslant \sigma$	(13)	$\omega \leqslant \omega \to \omega$
(7)	$\sigma_1 \leqslant \sigma_2, \tau_1 \leqslant \tau_2 \Rightarrow \sigma_1 \cap \tau_1 \leqslant \sigma_2 \cap \tau_2$	(14)	$\sigma_2 \leqslant \sigma_1, \tau_1 \leqslant \tau_2 \Rightarrow \sigma_1 \to \tau_1 \leqslant \sigma_2 \to \tau_2$

Figure 1 The type assignment system $\mathcal{B}$ of [3] and the subtype theory Ξ.

intersection and strong union correspond respectively to the implication, conjunction and disjunction operators of Meyer and Routley's Minimal Relevant Logic B^+ [34][1].

Strong connectives arise naturally in investigating the propositions-as-types analogy for intersection and union type assignment systems. Intersection types were introduced by Coppo, Dezani *et al.* in the late 70's [13, 15, 16, 5] to support a form of *ad hoc* polymorphism, for untyped λ-calculi, *à la* Curry. Intersection types were used originally as an (undecidable) type assignment system for pure λ-calculi, *i.e.* for finitary descriptions of denotational semantics [14]. This line of research was later explored by Abramsky [1] in a full-fledged Stone duality. Union types were introduced semantically, by MacQueen, Plotkin, and Sethi [33, 3]. In [3] strong intersection, union and subtyping were thoroughly studied in the context of type-assignment systems, see Figure 1. A classical example of the expressiveness of union types is due to Pierce [38]: without union types, the best information we can get for (Is_0 Test) is a boolean type.

$$\begin{array}{rcl} \text{Test} & \stackrel{def}{=} & \text{if } b \text{ then } 1 \text{ else } -1 : Pos \cup Neg \\ \text{Is\_0} & : & (Neg \to F) \cap (Zero \to T) \cap (Pos \to F) \\ (\text{Is\_0 Test}) & : & F \end{array}$$

[1] A terminological comment is in order. We refer to $(\supset_r)$ as "relevant implication" in order to be faithful to the original logical literature, since this constructor satisfies the logical properties of implication in the minimal relevant logical system introduced in [34]. And precisely in this sense it was used later in [4]. This use of the word "relevant" is therefore considerably *stronger* than, but not totally unrelated to, the one arising in the context of λI$-calculus$ and linear logic, where it expresses the requirement that the variable "is used at least once" in the function, in contrast to affine "at most one use" and linear "exactly one use".

$$\dfrac{\dfrac{\dfrac{}{x{:}\sigma \vdash x{:}\sigma}\,(Var)}{\vdash \lambda x{:}\sigma.x{:}\sigma \to \sigma}\,(\to I) \qquad \dfrac{\dfrac{}{x{:}\tau \vdash x{:}\tau}\,(Var)}{\vdash \lambda x{:}\tau.x{:}\tau \to \tau}\,(\to I)}{\vdash \lambda x{:}???.x{:}(\sigma \to \sigma) \cap (\tau \to \tau)}\,(\cap I)$$

Figure 2 Polymorphic identity.

Designing a λ-calculus *à la* Church with intersection and union types is problematic. The usual approach of simply adding types to binders does not work, as shown in Figure 2. Same difficulties can be found with union types. Intersection and union type disciplines started to be investigated in a explicitly typed programming language settings *à la* Church, much later by Reynolds and Pierce [41, 38], Wells *et al.* [48, 49], Liquori *et al.* [29, 18], Frisch *et al.* [21] and Dunfield [19]. From a logical point of view, there are many proposals to find a suitable logics to fit intersection: among them we cite [35, 37, 47, 42, 36, 11, 10, 39].

The LF_Δ, introduced in this paper extends [31] with union types, dependent types and minimal relevant implication. The novelty of LF_Δ in the context of Logical Frameworks, lies in the full-fledged use of strong proof-functional connectives, which to our knowledge has never been explored before. Clearly, all Δ-terms have a computational counterpart.

Pfenning's work on Refinement Types [37] pioneered an extension of the Edinburgh Logical Framework with subtyping and intersection types. His approach capitalises on a tame and essentially *ad hoc* notion of subtyping, but the logical strength of that system does not go beyond the LF (*i.e.* simple types). The logical power of LF_Δ allows to type all strongly normalizing terms. Furthermore, subtyping in LF_Δ arises naturally as a derived notion from the more fundamental concept of minimal relevant implication, as illustrated in Section 2.

Miquel [36] discusses an extension of the Calculus of Constructions with implicit typing, which subsumes a kind of proof-functional intersection. His approach has opposite motivations to ours. While LF_Δ provides a Church-style version of Curry-style type assignment systems, Miquel's Implicit Calculus of Constructions encompasses some features of Curry-style systems in an otherwise Church-style Calculus of Constructions. In LF_Δ we can discuss also *ad hoc* polymorphism, while in the Implicit Calculus only structural polymorphism is encoded. Indeed, he cannot assign the type $((\sigma \cap \tau) \to \sigma) \cap (\rho \to \rho))$ to the identity $\lambda x.x$ [28]. Kopylov [27] adds a dependent intersection type constructor $x{:}A \cap B[x]$ to NuPRL, allowing the resulting system to support dependent records (which are a very useful data structure to encode mathematics). The implicit product-type of Miquel, together with the dependent intersection type of Kopylov, and a suitable equality-type is used by Stump [46] to enrich the impredicative second-order system $\lambda P2$, in order to derive induction.

In order to achieve our goals, we could have carried out simply the encoding of LF_Δ in LF. But, due to the side-conditions characterizing proof-functional connectives, this would have be achieved only through a deep encoding. As an example of this, in Figure 8, we give an encoding of a subsystem of [3], where subtyping has been simulated using relevant arrows. This encoding illustrates the expressive power of LF in treating proofs as first-class citizens, and it was also a source of inspiration for LF_Δ.

All the examples discussed in this paper have been checked by an experimental proof development environment for LF_Δ [45] (see Bull and Bull-Subtyping in [44]).

Synopsis. In Section 2, we introduce LF_Δ and outline its metatheory, together with a discussion of the main design decisions. In Section 3, we provide the motivating examples. In Section 4, we outline the details of the implementation and future work.

Kinds

K	::=	$\mathsf{Type} \mid \Pi x{:}\sigma.K$	as in LF

Families

σ, τ	::=	$a \mid \Pi x{:}\sigma.\tau \mid \sigma\,\Delta \mid$	as in LF
		$\sigma \to^{\mathsf{r}} \tau \mid$	relevant family
		$\sigma \cap \tau \mid$	intersection family
		$\sigma \cup \tau$	union family

Objects

Δ	::=	$c \mid x \mid \lambda x{:}\sigma.\Delta \mid \Delta\,\Delta \mid$	as in LF
		$\lambda^{\mathsf{r}} x{:}\sigma.\Delta \mid$	relevant abstraction
		$\Delta\,{}^{\mathsf{r}}\,\Delta \mid$	relevant application
		$\langle \Delta\ ,\ \Delta \rangle \mid$	intersection objects
		$[\Delta\ ,\ \Delta] \mid$	union objects
		$pr_l\,\Delta \mid pr_r\,\Delta \mid$	projections objects
		$in_l^{\sigma}\,\Delta \mid in_r^{\sigma}\,\Delta$	injections objects

Figure 3 The syntax of the Δ-framework.

$$
\begin{array}{lll}
\wr\langle \Delta_1\ ,\ \Delta_2\rangle\wr \stackrel{def}{=} \wr\Delta_1\wr & \wr[\Delta_1\ ,\ \Delta_2]\wr \stackrel{def}{=} \wr\Delta_1\wr & \wr pr_i\,\Delta\wr \stackrel{def}{=} \wr\Delta\wr \\
\wr\lambda^{\mathsf{r}} x{:}\sigma.\Delta\wr \stackrel{def}{=} \lambda x.\wr\Delta\wr & \wr\Delta_1\,\Delta_2\wr \stackrel{def}{=} \wr\Delta_1\wr\wr\Delta_2\wr & \wr in_i\,\Delta\wr \stackrel{def}{=} \wr\Delta\wr \\
\wr\lambda x{:}\sigma.\Delta\wr \stackrel{def}{=} \lambda x.\wr\Delta\wr & \wr\Delta_1\,{}^{\mathsf{r}}\,\Delta_2\wr \stackrel{def}{=} \wr\Delta_2\wr & \wr c\wr \stackrel{def}{=} c \\
 & & \wr x\wr \stackrel{def}{=} x
\end{array}
$$

Figure 4 The essence function.

2 The Δ-framework: LF with proof-functional operators

The syntax of LF_Δ pseudo-terms is given in Figure 3. For the sake of simplicity, we suppose that α-convertible terms are equal. Signatures and contexts are defined as finite sequence of declarations, like in LF. Observe that we could formulate LF_Δ in the style of [23], using only canonical forms and without reductions, but we prefer to use the standard LF format to support better intuition. There are three proof-functional objects, namely strong conjunction (typed with $\sigma \cap \tau$) with two corresponding projections, strong disjunction (typed with $\sigma \cup \tau$) with two corresponding injections, and strong (or relevant) λ-abstraction (typed with $\to^{\mathsf{r}}$). Indeed, a relevant implication is not a dependent one because the essence of the inhabitants of type $\sigma \to^{\mathsf{r}} \tau$ is essentially the identity function as enforced in the typing rules. Note that injections in_i need to be decorated with the injected type σ in order to ensure the unicity of typing.

We need to generalize the notion of *essence*, introduced in [17, 30] to syntactically connect pure λ-terms (denoted by M) and type annotated LF_Δ terms (denoted by Δ). The essence function compositionally erases all type annotations, see Figure 4.

One could argue that the choice of Δ_1 in the definition of strong pairs/co-pairs is arbitrary and could have been replaced with Δ_2: however, the typing rules will ensure that, if $\langle \Delta_1\ ,\ \Delta_2\rangle$ (resp. $[\Delta_1\ ,\ \Delta_2]$) is typable, then we have that $\wr\Delta_1\wr =_\eta \wr\Delta_2\wr$. Thus, strong pairs/co-pairs are constrained. The rule for the essence of a relevant application is justified by the fact that the operator amounts to just a type decoration.

The six basic reductions for LF_Δ objects appear on the left in Figure 5. Congruence rules are as usual, except for the two cases dealing with pairs and co-pairs which appear on the right of Figure 5. Here redexes need to be reduced "in parallel" in order to preserve identity of essences in the components. We denote by $=_\Delta$ the symmetric, reflexive, and transitive closure of $\to_\Delta$, *i.e.* the compatible closure of the reduction induced by the first six

$$
\begin{array}{rcl}
(\lambda x{:}\sigma.\Delta_1)\,\Delta_2 & \longrightarrow_{\beta} & \Delta_1[\Delta_2/x] \\
pr_l\,\langle \Delta_1\ ,\ \Delta_2\rangle & \longrightarrow_{pr_l} & \Delta_1 \\
pr_r\,\langle \Delta_1\ ,\ \Delta_2\rangle & \longrightarrow_{pr_r} & \Delta_2 \\
[\Delta_1\ ,\ \Delta_2]\,in_l^{\sigma}\,\Delta_3 & \longrightarrow_{in_l} & \Delta_1\,\Delta_3 \\
[\Delta_1\ ,\ \Delta_2]\,in_r^{\sigma}\,\Delta_3 & \longrightarrow_{in_r} & \Delta_2\,\Delta_3 \\
(\lambda\!\!\!^{\mathsf{r}}\, x{:}\sigma.\Delta_1)\,{}^{\mathsf{r}}\Delta_2 & \longrightarrow_{\beta r} & \Delta_1[\Delta_2/x]
\end{array}
$$

$$
\frac{\Delta_1 \to_{\Delta} \Delta_1' \quad \Delta_2 \to_{\Delta} \Delta_2' \quad \wr\,\Delta_1'\,\wr \equiv \wr\,\Delta_2'\,\wr}{\langle \Delta_1\ ,\ \Delta_2\rangle \to_{\Delta} \langle \Delta_1'\ ,\ \Delta_2'\rangle}\ (Congr_{\cap})
$$

$$
\frac{\Delta_1 \to_{\Delta} \Delta_1' \quad \Delta_2 \to_{\Delta} \Delta_2' \quad \wr\,\Delta_1'\,\wr \equiv \wr\,\Delta_2'\,\wr}{[\Delta_1\ ,\ \Delta_2] \to_{\Delta} [\Delta_1'\ ,\ \Delta_2']}\ (Congr_{\cup})
$$

Figure 5 The reduction semantics.

rules on the left in Figure 5, with the addition of the last two congruence rules in the same figure. In order to make this definition truly functional as well as to be able to prove a simple subject reduction result, we need to constrain pairs and co-pairs, *i.e.* objects of the form $\langle \Delta_i\ ,\ \Delta_j\rangle$ and $[\Delta_i\ ,\ \Delta_j]$ to have congruent components up-to erasure of type annotations. This is achieved by imposing $\wr\,\Delta_i\,\wr \equiv \wr\,\Delta_j\,\wr$ in both constructs. We will therefore assume that such pairs and co-pairs are simply not well defined terms, if the components have a different "infrastructure". The effects of this choice are reflected in the congruence rules in the reduction relation, in order to ensure that reductions can only be carried out in parallel along the two components.

The restriction on reductions in pairs/co-pairs and the new constructs do not cause any problems in showing that $\to_{\Delta}$ is locally confluent:

▶ **Theorem 1** (Local confluence)**.**
The reduction relation on well-formed LF_{Δ}*-terms is locally confluent.*

The extended type theory LF_{Δ} is a formal system for deriving judgements of the forms:

	$\vdash$	Σ	Σ is a valid signature	Γ	$\vdash_{\Sigma}$	$\sigma : K$	σ has kind K in Γ and Σ
	$\vdash_{\Sigma}$	Γ	Γ is a valid context in Σ	Γ	$\vdash_{\Sigma}$	$\Delta : \sigma$	Δ has type σ in Γ and Σ
Γ	$\vdash_{\Sigma}$	K	K is a kind in Γ and Σ				

The set of rules for object formation is defined in Figure 6, while the sets of rules for signatures, contexts, kinds and families are defined as usual in the Appendix: all typing rules are syntax-directed. Note that proof-functionality is enforced by the essence side-conditions in rules $(\to^{\mathsf{r}} I)$, $(\cap I)$, and $(\cup E)$. In the rule $(Conv)$ we rely on the external notion of equality $=_{\Delta}$. An option could have be to add an internal notion of equality directly in the type system ($\Gamma \vdash_{\Sigma} \sigma =_{\Delta} \tau$), and prove that the external and the internal definitions of equality are equivalent, as was proved for semi-full Pure Type Systems [43]. Yet another possibility could be to compare type essences $\wr\,\sigma\,\wr =_{\Delta} \wr\,\tau\,\wr$, for a suitable extension of essence to types and kinds. Unfortunately, this would lead to undecidability of type checking, in connection with relevant implication, as the following example shows. Consider two constants c_1 of type $\sigma \to^{\mathsf{r}} (\Pi y{:}\sigma.\sigma)$ and c_2 of type $(\Pi y{:}\sigma.\sigma) \to^{\mathsf{r}} \sigma$: the following Δ-term is typable with σ and its essence is Ω.

$$
\Delta_{\Omega} \stackrel{def}{=} (\lambda x{:}\sigma.c_1\,{}^{\mathsf{r}}\,x\,x)\,(c_2\,{}^{\mathsf{r}}\,(\lambda x{:}\sigma.c_1\,{}^{\mathsf{r}}\,x\,x)) \qquad\qquad \wr\,\Delta_{\Omega}\,\wr = \Omega
$$

Since the intended meaning of relevant implication is "essentially" the identity, introducing variables or constants whose type is a relevant implication, amounts to assuming axioms

Valid Objects

$$\frac{\vdash_\Sigma \Gamma \quad c{:}\sigma \in \Sigma}{\Gamma \vdash_\Sigma c : \sigma}\ (Const) \qquad \frac{\vdash_\Sigma \Gamma \quad x{:}\sigma \in \Gamma}{\Gamma \vdash_\Sigma x : \sigma}\ (Var)$$

$$\frac{\Gamma, x{:}\sigma \vdash_\Sigma \Delta : \tau}{\Gamma \vdash_\Sigma \lambda x{:}\sigma.\Delta : \Pi x{:}\sigma.\tau}\ (\Pi I) \qquad \frac{\Gamma \vdash_\Sigma \Delta_1 : \Pi x{:}\sigma.\tau \quad \Gamma \vdash_\Sigma \Delta_2 : \sigma}{\Gamma \vdash_\Sigma \Delta_1\, \Delta_2 : \tau[\Delta_2/x]}\ (\Pi E)$$

$$\frac{\Gamma, x{:}\sigma \vdash_\Sigma \Delta : \tau \quad \wr \Delta \wr =_\eta x}{\Gamma \vdash_\Sigma \lambda^{\mathsf{r}} x{:}\sigma.\Delta : \sigma \to^{\mathsf{r}} \tau}\ (\to^{\mathsf{r}} I) \qquad \frac{\Gamma \vdash_\Sigma \Delta_1 : \sigma \to^{\mathsf{r}} \tau \quad \Gamma \vdash_\Sigma \Delta_2 : \sigma}{\Gamma \vdash_\Sigma \Delta_1\, {}^{\mathsf{r}}\Delta_2 : \tau}\ (\Pi^{\mathsf{r}} E)$$

$$\frac{\Gamma \vdash_\Sigma \Delta_1 : \sigma \quad \Gamma \vdash_\Sigma \Delta_2 : \tau \quad \wr \Delta_1 \wr =_\eta \wr \Delta_2 \wr}{\Gamma \vdash_\Sigma \langle \Delta_1\ ,\ \Delta_2 \rangle : \sigma \cap \tau}\ (\cap I) \qquad \frac{\Gamma \vdash_\Sigma \Delta : \sigma \cap \tau}{\Gamma \vdash_\Sigma pr_l\, \Delta : \sigma}\ (\cap E_l) \qquad \frac{\Gamma \vdash_\Sigma \Delta : \sigma \cap \tau}{\Gamma \vdash_\Sigma pr_r\, \Delta : \tau}\ (\cap E_r)$$

$$\frac{\Gamma \vdash_\Sigma \Delta : \sigma \quad \Gamma \vdash_\Sigma \sigma \cup \tau : \mathsf{Type}}{\Gamma \vdash_\Sigma in_l^\tau\, \Delta : \sigma \cup \tau}\ (\cup I_l) \qquad \frac{\Gamma \vdash_\Sigma \Delta : \tau \quad \Gamma \vdash_\Sigma \sigma \cup \tau : \mathsf{Type}}{\Gamma \vdash_\Sigma in_r^\sigma\, \Delta : \sigma \cup \tau}\ (\cup I_r)$$

$$\frac{\begin{array}{ll}\Gamma \vdash_\Sigma \Delta_1 : \Pi y{:}\sigma.\rho[in_l^\tau\ y/x] & \wr \Delta_1 \wr =_\eta \wr \Delta_2 \wr \\ \Gamma \vdash_\Sigma \Delta_2 : \Pi y{:}\tau.\rho[in_r^\sigma\ y/x] & \Gamma, x{:}\sigma \cup \tau \vdash_\Sigma \rho : \mathsf{Type}\end{array}}{\Gamma \vdash_\Sigma [\Delta_1\ ,\ \Delta_2] : \Pi x{:}\sigma \cup \tau.\rho}\ (\cup E) \qquad \frac{\begin{array}{l}\Gamma \vdash_\Sigma \Delta : \sigma \\ \Gamma \vdash_\Sigma \tau : \mathsf{Type} \quad \sigma =_\Delta \tau\end{array}}{\Gamma \vdash_\Sigma \Delta : \tau}\ (Conv)$$

Figure 6 The type rules for valid objects.

corresponding to type inclusions such as those that equate σ and $\sigma \to \sigma$. As a consequence, β-equality of essences becomes undecidable. Thus, we rule out such options in relating relevant implications in LF_Δ to subtypes in the type assignment system $\mathcal{B}$ of [3].

2.1 Relating LF_Δ to $\mathcal{B}$

We compare and contrast certain design decisions of LF_Δ to the type assignment system $\mathcal{B}$ of [3]. The proof of strong normalization for LF_Δ will rely, in fact, on a forgetful mapping from LF_Δ to $\mathcal{B}$. As pointed out in [3], the elimination rule for union types in $\mathcal{B}$ breaks subject reduction for one-step β-reduction, but this can be recovered using a suitable parallel β-reduction. The well-known counter-example for one-step reduction, due to Pierce is

$$x\,((\mathsf{I}\,y)\,z)\,((\mathsf{I}\,y)\,z) \longrightarrow_\beta \begin{array}{l} \nearrow^\beta x\,(y\,z)\,((\mathsf{I}\,y)\,z)\,\searrow_\beta \\ \searrow_\beta x\,((\mathsf{I}\,y)\,z)\,(y\,z)\,\nearrow^\beta \end{array} x\,(y\,z)\,(y\,z),$$

where I is the identity. In the typing context $B \stackrel{def}{=} x{:}(\sigma_1 \to \sigma_1 \to \tau) \cap (\sigma_2 \to \sigma_2 \to \tau), y{:}\rho \to (\sigma_1 \cup \sigma_2), z{:}\rho$, the first and the last terms can be typed with τ, while the terms in the fork cannot. The reason is that the subject in the conclusion of the $(\cup E)$ rule uses a context which can have more than one hole, as in the present case[2]. In LF_Δ, the formulation of the $(\cup E)$ rule takes a different route which does not trigger the counterexample. Indeed, we have introduction and elimination constructs $in_l\,, in_r$ and $[\ ,\ ]$ which allow to reduce the term

[2] The problem would not arise if $(\cup E)$ is replaced by the rule schema

$$\frac{B, x_1{:}\sigma, \ldots, x_n{:}\sigma \vdash M : \rho \quad B, x_1{:}\tau, \ldots, x_n{:}\tau \vdash M : \rho \quad B \vdash N_i : \sigma \cup \tau \quad N_i =_\beta N_j \quad i,j = 1 \ldots n}{B \vdash M[N_1/x_1 \ldots N_n/x_n] : \rho}\ (\cup E')$$

Removing the non-static clause on the N_i's would yield a more permissive type system than $\mathcal{B}$.

only if we know that the argument, stripped of the introduction construct, has one of the types of the disjunction. Pierce's critical term can be expressed and typed in LF_Δ with the following judgment (the full derivation is in the Appendix):

$$\Gamma \vdash_\Sigma [\underbrace{(\lambda x_1{:}\sigma_1.(pr_l\, x)\, x_1\, x_1)}_{\Delta_1}\, ,\, \underbrace{(\lambda x_2{:}\sigma_2.(pr_r\, x)\, x_2\, x_2)}_{\Delta_2}]\, (\underbrace{(\lambda x_3{:}\rho \to \sigma_1 \cup \sigma_2.x_3)}_{\Delta_3}\, y\, z) : \tau$$

where $\Gamma \stackrel{def}{=} x{:}(\Pi x_1{:}\sigma_1.\Pi x_2{:}\sigma_1.\tau) \cap (\Pi x_1{:}\sigma_2.\Pi x_2{:}\sigma_2.\tau)$, $y{:}\rho \to \sigma_1 \cup \sigma_2, z{:}\rho$, and $\Sigma \stackrel{def}{=} \tau{:}\mathsf{Type}$. Notice that there is only one redex, namely $\Delta_3\, y$, and the reduction of this redex leads to $[\Delta_1, \Delta_2]\, (y\, z)$, and no other intermediate (untypable) Δ-terms are possible.

The following result will be useful in the following section.

▶ **Theorem 2.** *The system $\mathcal{B}$ without ω gives types only to strongly normalizing terms.*

A proof is embedded in Theorem 4.8 of [3]. It can also be obtained using the general computability method presented in [25] Section 4, by interpreting intersection and union types precisely as intersections and unions in the lattice of computability sets.

2.2 LF_Δ metatheory

LF_Δ can play the role of a Logical Framework only if decidable. Due to the lack of space, we list here only the main results: the complete list appears in the Appendix. The first important step states that if a Δ-term is typable, then its type is unique up to $=_\Delta$.

▶ **Theorem 3** (Unicity of types and kinds)**.**
1. *If $\Gamma \vdash_\Sigma \Delta : \sigma$ and $\Gamma \vdash_\Sigma \Delta : \tau$, then $\sigma =_\Delta \tau$.*
2. *If $\Gamma \vdash_\Sigma \sigma : K$ and $\Gamma \vdash_\Sigma \sigma : K'$, then $K =_\Delta K'$.*

Strong normalization is proved as in LF. First we encode LF_Δ-terms into terms of the type assignment system $\mathcal{B}$ such that redexes in the source language correspond to redexes in the target language and we use Theorem 2. Then, we introduce two forgetful mappings, namely $\|\cdot\|$ and $|\cdot|$, defined in Figure 11 of the Appendix, to erase dependencies in types and to drop proof-functional constructors in Δ-terms and we conclude. Special care is needed in dealing with redexes occurring in type-dependencies, because these need to be flattened at the level of terms.

▶ **Theorem 4** (Strong normalization)**.**
1. LF_Δ *is strongly normalizing, i.e.,*
   a. *If $\Gamma \vdash_\Sigma K$, then K is strongly normalizing.*
   b. *If $\Gamma \vdash_\Sigma \sigma : K$, then σ is strongly normalizing.*
   c. *If $\Gamma \vdash_\Sigma \Delta : \sigma$, then Δ is strongly normalizing.*
2. *Every strongly normalizing pure λ-term can be annotated so as to be the essence of a Δ-term.*

Local confluence and strong normalization entail confluence, so we have

▶ **Theorem 5** (Confluence)**.** LF_Δ *is confluent, i.e.:*
1. *If $K_1 \longrightarrow^*_\Delta K_2$ and $K_1 \longrightarrow^*_\Delta K_3$, then $\exists K_4$ such that $K_2 \longrightarrow^*_\Delta K_4$ and $K_3 \longrightarrow^*_\Delta K_4$.*
2. *If $\sigma_1 \longrightarrow^*_\Delta \sigma_2$ and $\sigma_1 \longrightarrow^*_\Delta \sigma_3$, then $\exists \sigma_4$ such that $\sigma_2 \longrightarrow^*_\Delta \sigma_4$ and $\sigma_3 \longrightarrow^*_\Delta \sigma_4$.*
3. *If $\Delta_1 \longrightarrow^*_\Delta \Delta_2$ and $\Delta_1 \longrightarrow^*_\Delta \Delta_3$, then $\exists \Delta_4$ such that $\Delta_2 \longrightarrow^*_\Delta \Delta_4$ and $\Delta_3 \longrightarrow^*_\Delta \Delta_4$.*

Then, we have subject reduction, whose proof relies on technical lemmas about inversion and subderivation properties (see Appendix).

▶ **Theorem 6** (Subject reduction of LF_Δ).
1. *If* $\Gamma \vdash_\Sigma K$ *and* $K \to_\Delta K'$, *then* $\Gamma \vdash_\Sigma K'$.
2. *If* $\Gamma \vdash_\Sigma \sigma : K$ *and* $\sigma \to_\Delta \sigma'$, *then* $\Gamma \vdash_\Sigma \sigma' : K$.
3. *If* $\Gamma \vdash_\Sigma \Delta : \sigma$ *and* $\Delta \to_\Delta \Delta'$, *then* $\Gamma \vdash_\Sigma \Delta' : \sigma$.

Finally, we define a possible algorithm for checking judgements in LF_Δ by computing a type or a kind for a term, and then testing for definitional equality, *i.e.* $=_\Delta$, against the given type or kind. This is achieved by reducing both to their unique normal forms and checking that they are identical up to α-conversion. Therefore we finally have:

▶ **Theorem 7** (Decidability). *All the type judgments of* LF_Δ *are recursively decidable.*

Minimal Relevant Implications and Type Inclusion. Type inclusion and the rules of subtyping are related to the notion of minimal relevant implication, see [4, 17]. The insight is quite subtle, but ultimately very simple. This is what makes it appealing. The apparently intricate rules of subtyping and type inclusion, which occur in many systems, and might even appear *ad hoc* at times, can all be explained away in our principled approach, by proving that the relevant implication type is inhabited by a term whose essence is essentially a variable.

In the following theorem we show how relevant implication subsumes the type-inclusion rules of the theory Ξ of [3], without rules (5) and (13) (dealing with ω) and rule (10) (distributing $\cap$ over $\cup$) in Figure 1: we call Ξ' such restricted subtype theory. Note that the reason to drop subtype rule (10) is due to the fact that we cannot inhabit the type $\sigma \cap (\tau \cup \rho) \to^{\mathsf{r}} (\sigma \cap \tau) \cup (\sigma \cap \rho)$[3].

▶ **Theorem 8** (Type Inclusion). *The judgement* $\langle\rangle \vdash_\Sigma \Delta : \sigma \to^{\mathsf{r}} \tau$ *(where both* σ *and* τ *do not contain dependencies or relevant families) holds iff* $\sigma \leq \tau$ *holds in the subtype theory* Ξ' *of* $\mathcal{B}$ *enriched with new axioms of the form* $\sigma_1 \leq \sigma_2$ *for each constant* $c : \sigma_1 \to^{\mathsf{r}} \sigma_2 \in \Sigma$.

As far as the $\lambda^{\Pi\&}$ system of Refinement Types introduced by Pfenning in [37], we have the following theorem:

▶ **Corollary 9** (Pfenning's Refinement Types). *The judgment* $\vdash_\Sigma \sigma \leq \tau$ *in* $\lambda^{\Pi\&}$ *can be encoded in* LF_Δ *by adding a constant of type* $\sigma \to^{\mathsf{r}} \tau$ *to* Σ', *where the latter is the signature obtained from* Σ *by replacing each clause of the form* $a_1 :: a_2$ *or* $a_1 \leq a_2$ *in* Σ *by a constant of type* $a_1 \to^{\mathsf{r}} a_2$.

Moreover, while Pfenning needs to add explicitly the rules of subtyping (*i.e.* the theory of $\leq$) in $\lambda^{\Pi\&}$, we inherit them naturally in LF_Δ from the rules for minimal relevant implication.

3 Examples

As we have argued in the previous sections, the point of this paper is a uniform and principled approach to the encoding of a plethora of type disciplines and systems which ultimately stem or can capitalize from strong proof-functional connectives and subtyping.

[3] To encompass also the subtype rule (10) of the type theory Ξ, besides adding a special constant, we can strengthen the form of the $(\cup E)$ type rule as follows:

$$\frac{\begin{array}{ll}\Gamma \vdash_\Sigma \Delta_1 : \Pi y{:}\chi \cap \sigma.\rho\,\langle pr_l\ y\ ,\ in_l^\tau\ pr_r\ y\rangle & \wr\,\Delta_1\,\wr =_\eta \wr\,\Delta_2\,\wr \\ \Gamma \vdash_\Sigma \Delta_2 : \Pi y{:}\chi \cap \tau.\rho\,\langle pr_l\ y\ ,\ in_r^\sigma\ pr_r\ y\rangle & \Gamma \vdash_\Sigma \rho : \Pi y{:}\chi \cap (\sigma \cup \tau).\mathsf{Type}\end{array}}{\Gamma \vdash_\Sigma [\Delta_1\ ,\ \Delta_2] : \Pi x{:}\chi \cap (\sigma \cup \tau).\rho\,x}\ (\cup E)$$

Similarly we can treat the remaining rules of the type theory Π in [3].

Atomic propositions, non-atomic goals and non-atomic programs: $\alpha, \gamma_0, \pi_0 : \mathsf{Type}$
Goals and programs: $\gamma = \alpha \cup \gamma_0 \qquad \pi = \alpha \cup \pi_0$
Constructors (implication, conjunction, disjunction).

$$\begin{array}{rcl}
\mathsf{impl} & : & (\pi \to \gamma \to \gamma_0) \cap (\gamma \to \pi \to \pi_0) \\
\mathsf{impl}_1 & = & \lambda x{:}\pi.\lambda y{:}\gamma.in_r^{\alpha}\ (pr_l\ \mathsf{impl}\ x\ y) \qquad \mathsf{impl}_2 = \lambda x{:}\gamma.\lambda y{:}\pi.in_r^{\alpha}\ (pr_r\ \mathsf{impl}\ x\ y) \\
\mathsf{and} & : & (\gamma \to \gamma \to \gamma_0) \cap (\pi \to \pi \to \pi_0) \\
\mathsf{and}_1 & = & \lambda x{:}\gamma.\lambda y{:}\gamma.in_r^{\alpha}\ (pr_l\ \mathsf{and}\ x\ y) \qquad \mathsf{and}_2 = \lambda x{:}\pi.\lambda y{:}\pi.in_r^{\alpha}\ (pr_r\ \mathsf{and}\ x\ y) \\
\mathsf{or} & : & (\gamma \to \gamma \to \gamma_0) \qquad \mathsf{or}_1 = \lambda x{:}\gamma.\lambda y{:}\gamma.in_r^{\alpha}\ (\mathsf{or}\ x\ y)
\end{array}$$

$\mathsf{solve}\ p\ g$ indicates that the judgment $p \vdash g$ is valid.
$\mathsf{bchain}\ p\ a\ g$ indicates that, if $p \vdash g$ is valid, then $p \vdash a$ is valid.

$$\mathsf{solve} : \pi \to \gamma \to \mathsf{Type} \qquad \mathsf{bchain} : \pi \to \alpha \to \gamma \to \mathsf{Type}$$

Rules for solve:

$$\begin{array}{rcl}
- & : & \Pi_{(p:\pi)(g_1,g_2:\gamma)}\mathsf{solve}\ p\ g_1 \to \mathsf{solve}\ p\ g_2 \to \mathsf{solve}\ p\ (\mathsf{and}_1\ g_1\ g_2) \\
- & : & \Pi_{(p:\pi)(g_1,g_2:\gamma)}\mathsf{solve}\ p\ g_1 \to \mathsf{solve}\ p\ (\mathsf{or}_1\ g_1\ g_2) \\
- & : & \Pi_{(p:\pi)(g_1,g_2:\gamma)}\mathsf{solve}\ p\ g_2 \to \mathsf{solve}\ p\ (\mathsf{or}_1\ g_1\ g_2) \\
- & : & \Pi_{(p_1,p_2:\pi)(g:\gamma)}\mathsf{solve}\ (\mathsf{and}_2\ p_1\ p_2)\ g \to \mathsf{solve}\ p_1\ (\mathsf{impl}_1\ p_2\ g) \\
- & : & \Pi_{(p:\pi)(a:\alpha)(g:\gamma)}\mathsf{bchain}\ p\ a\ g \to \mathsf{solve}\ p\ g \to \mathsf{solve}\ p\ (in_l^{\gamma_0}\ a)
\end{array}$$

Rules for bchain:

$$\begin{array}{rcl}
- & : & \Pi_{(a:\alpha)(g:\gamma)}\mathsf{bchain}\ (\mathsf{impl}_2\ g\ (in_l\ \pi_0 a))\ a\ g \\
- & : & \Pi_{(p_1,p_2:\pi)(a:\alpha)(g:\gamma)}\mathsf{bchain}\ p_1\ a\ g \to \mathsf{bchain}\ (\mathsf{and}_2\ p_1\ p_2)\ a\ g \\
- & : & \Pi_{(p_1,p_2:\pi)(a:\alpha)(g:\gamma)}\mathsf{bchain}\ p_2\ a\ g \to \mathsf{bchain}\ (\mathsf{and}_2\ p_1\ p_2)\ a\ g \\
- & : & \Pi_{(p:\pi)(a:\alpha)(g,g_1,g_2:\gamma)}\mathsf{bchain}\ (\mathsf{impl}_2\ (\mathsf{and}_1\ g_1\ g_2)\ p)\ a\ g \to \mathsf{bchain}\ (\mathsf{impl}_2\ g_1\ (\mathsf{impl}_2\ g_2\ p))\ a\ g \\
- & : & \Pi_{(p_1,p_2:\pi)(a:\alpha)(g,g_1:\gamma)}\mathsf{bchain}\ (\mathsf{impl}_2\ g_1\ p_1)\ a\ g \to \mathsf{bchain}\ (\mathsf{impl}_2\ g_1\ (\mathsf{and}_2\ p_1\ p_2))\ a\ g \\
- & : & \Pi_{(p_1,p_2:\pi)(a:\alpha)(g,g_1:\gamma)}, \mathsf{bchain}\ (\mathsf{impl}_2\ g_1\ p_2)\ a\ g \to \mathsf{bchain}\ (\mathsf{impl}_2\ g_1\ (\mathsf{and}_2\ p_1\ p_2))\ a\ g
\end{array}$$

Figure 7 The LF_Δ encoding of Hereditary Harrop Formulæ.

The framework LF_Δ, presented in this paper, is the first to accommodate all the examples and counterexamples that have appeared in the literature. The complete developments of both the implementation of the Δ-framework and example encodings can be found in [44].

We start the section showing the expressive power of LF_Δ in encoding classical features of typing disciplines with strong intersection and union.

Auto application. The judgement $\vdash_{\mathcal{B}} \lambda x.x\,x : \sigma \cap (\sigma \to \tau) \to \tau$ in $\mathcal{B}$, is rendered in LF_Δ by the LF_Δ-judgement $\vdash_\Sigma \lambda x{:}\sigma \cap (\sigma \to \tau).(pr_r\ x)\ (pr_l\ x) : \sigma \cap (\sigma \to \tau) \to \tau$.

Polymorphic identity. The judgement $\vdash_{\mathcal{B}} \lambda x.x : (\sigma \to \sigma) \cap (\tau \to \tau)$ in $\mathcal{B}$, is rendered in LF_Δ by the judgement $\vdash_{\langle\rangle} \langle \lambda x{:}\sigma.x\ ,\ \lambda x{:}\tau.x\rangle : (\sigma \to \sigma) \cap (\tau \to \tau)$.

Commutativity of union. The judgement $\lambda x.x : (\sigma \cup \tau) \to (\tau \cup \sigma)$ in $\mathcal{B}$ is rendered in LF_Δ by the judgement $\lambda x{:}\sigma \cup \tau.[\lambda y{:}\sigma.in_r^{\tau}\ y\ ,\ \lambda y{:}\tau.in_l^{\sigma}\ y]\,x : (\sigma \cup \tau) \to (\tau \cup \sigma)$.

Pierce's expression of page 2. The expressive power of union types highlighted by Pierce is rendered in LF_Δ by

$$\begin{array}{rcl}
Neg : \mathsf{Type} \quad Zero : \mathsf{Type} \quad Pos : \mathsf{Type} \quad T : \mathsf{Type} \quad F : \mathsf{Type} \quad \mathrm{Test} : Pos \cup Neg \\
\mathrm{Is_0} & : & (Neg \to F) \cap (Zero \to T) \cap (Pos \to F) \\
\mathrm{Is_0_Test} & \stackrel{def}{=} & [\lambda x{:}Neg.(pr_l\ pr_l\ \mathrm{Is_0})\,x\ ,\ \lambda x{:}Pos.(pr_r\ \mathrm{Is_0})\,x]\ \mathrm{Test}
\end{array}$$

The above example illustrates the advantages of taking LF_Δ as a framework. In LF we would render it only encoding $\mathcal{B}$ deeply, ending up with the verbose code in pierce_program.v [44].

Hereditary Harrop Formulæ. The encoding of Hereditary Harrop's Formulæ is one of the motivating examples given by Pfenning for introducing refinement types in [37]. In LF_Δ it can be expressed as in Figure 7 and type checked in the environment [45] using our concrete syntax (file pfenning_harrop.bull [44]), without any reference to intersection types, by a subtle use of union types. We add also rules for solving and backchaining. Hereditary Harrop formulæ can be recursively defined using two mutually recursive syntactical objects called programs (π) and goals (γ):

$$\gamma \quad := \quad \alpha \mid \gamma \wedge \gamma \mid \pi \Rightarrow \gamma \mid \gamma \vee \gamma \qquad\qquad \pi \quad := \quad \alpha \mid \pi \wedge \pi \mid \gamma \Rightarrow \pi$$

Using Corollary 9, we can provide an alternative encoding of atoms, goals and programs which is more faithful to the one by Pfenning. Namely, we can introduce in the signature the constants $c_1 : \alpha \to^{\mathsf{r}} \gamma$ and $c_2 : \alpha \to^{\mathsf{r}} \pi$ in order to represent the axioms $atom \leq goal$ and $atom \leq prog$ in Pfenning's encoding. Our approach based on union types, while retaining the same expressivity permits to shortcut certain inclusions and to rule out also certain exotic goals and exotic programs. Indeed, for the purpose of establishing the adequacy of the encoding, it is sufficient to avoid variables involving union types in the derivation contexts.

Natural Deductions in Normal Form. The second motivating example for intersection types given in [37] is *natural deductions in normal form*. We recall that a natural deduction is in normal form if there are no applications of elimination rules of a logical connective immediately following their corresponding introduction, in the main branch of a subderivation.

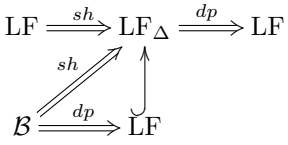
The encoding we give in LF_Δ is a slightly improved version of the one in [37]: as Pfenning, we restrict to the purely implicational fragment.

$$\begin{array}{rcl} o & : & Type \quad \supset : o \to o \to o \quad Elim, \mathrm{Nf}^0 : o \to \mathsf{Type} \\ \mathrm{Nf} & \equiv & \Pi A{:}o.\mathrm{Nf}^0(A) \cup Elim(A) \\ \supset_I & : & \Pi A, B{:}o.(Elim(A) \to \mathrm{Nf}(B)) \to \mathrm{Nf}^0(A \supset B) \\ \supset_E & : & \Pi A, B{:}o.Elim(A \supset B) \to \mathrm{Nf}^0(A) \to Elim(B). \end{array}$$

As in the previous example, we use union types to define normal forms ($\mathrm{Nf}(A)$) either as pure elimination-deductions from hypotheses ($Elim(A)$) or normal form-deductions ($\mathrm{Nf}^0(A)$). As above we could have used also intersection types. This example is interesting in itself, being the prototype of the encoding of type systems using canonical and atomic syntactic categories [23] and also of Fitch Set Theory [26].

Adequacy, Canonical Forms, Exotic terms. In the presence of union types, we have to pay special attention to the exact formulation of Adequacy Theorems, as in the Harrop's formulæ example above. Otherwise exotic terms arise, such as $[\lambda x{:}\sigma.C(x)\ ,\ \lambda x{:}\tau.D(x)]\, y$, where $C(\cdot)$ and $D(\cdot)$ are distinct contexts (*i.e.* terms with holes), which cannot be naturally simplified even if $\wr C \wr \equiv \wr D \wr$. More work needs to be done to streamline how to exclude, or even capitalize on exotic terms.

Metacircular Encodings. The following diagram summarizes the network of adequate encodings/inclusions between LF_Δ, LF, and $\mathcal{B}$ that can be defined.



```
(* Define our types *)
Axiom o : Set.
(* Axiom omegatype : o. *)
Axioms (arrow inter union : o → o → o).

(* Transform our types into LF types *)
Axiom OK : o → Set.

(* Define the essence equality as an equivalence relation *)
Axiom Eq : forall (s t : o), OK s → OK t → Prop.
Axiom Eqrefl : forall (s : o) (M : OK s), Eq s s M M.
Axiom Eqsymm : forall (s t : o) (M : OK s) (N : OK t), Eq s t M N → Eq t s N M.
Axiom Eqtrans : forall (s t u : o) (M : OK s) (N : OK t) (O : OK u), Eq s t M N → Eq t u N O → Eq s u M O.

(* constructors for arrow (→ I and → E) *)
Axiom Abst : forall (s t : o), ((OK s) → (OK t)) → OK (arrow s t).
Axiom App : forall (s t : o), OK (arrow s t) → OK s → OK t.

(* constructors for intersection *)
Axiom Proj_l : forall (s t : o), OK (inter s t) → OK s.
Axiom Proj_r : forall (s t : o), OK (inter s t) → OK t.
Axiom Pair : forall (s t : o) (M : OK s) (N : OK t), Eq s t M N → OK (inter s t).

(* constructors for union *)
Axiom Inj_l : forall (s t : o), OK s → OK (union s t).
Axiom Inj_r : forall (s t : o), OK t → OK (union s t).
Axiom Copair : forall (s t u : o) (X : OK (arrow s u)) (Y : OK (arrow t u)), OK (union s t) →
 Eq (arrow s u) (arrow t u) X Y → OK u.

(* define equality wrt arrow constructors *)
Axiom Eqabst : forall (s t s' t' : o) (M : OK s → OK t) (N : OK s' → OK t'),
 (forall (x : OK s) (y : OK s'), Eq s s' x y → Eq t t' (M x) (N y)) →
 Eq (arrow s t) (arrow s' t') (Abst s t M) (Abst s' t' N).
Axiom Eqapp : forall (s t s' t' : o) (M : OK (arrow s t)) (N : OK s) (M' : OK (arrow s' t')) (N' : OK s'),
 Eq (arrow s t) (arrow s' t') M M' → Eq s s' N N' → Eq t t' (App s t M N) (App s' t' M' N').

(* define equality wrt intersection constructors *)
Axiom Eqpair : forall (s t : o) (M : OK s) (N : OK t) (pf : Eq s t M N), Eq (inter s t) s (Pair s t M N pf) M.
Axiom Eqproj_l : forall (s t : o) (M : OK (inter s t)), Eq (inter s t) s M (Proj_l s t M).
Axiom Eqproj_r : forall (s t : o) (M : OK (inter s t)), Eq (inter s t) t M (Proj_r s t M).

(* define equality wrt union *)
Axiom Eqinj_l : forall (s t : o) (M : OK s), Eq (union s t) s (Inj_l s t M) M.
Axiom Eqinj_r : forall (s t : o) (M : OK t), Eq (union s t) t (Inj_r s t M) M.
Axiom Eqcopair : forall (s t u : o) (M : OK (arrow s u)) (N : OK (arrow t u)) (O : OK (union s t))
 (pf: Eq (arrow s u) (arrow t u) M N) (x : OK s),
 Eq s (union s t) x O → Eq u u (App s u M x) (Copair s t u M N O pf).
```

Figure 8 The LF encoding of $\mathcal{B}$ (Coq syntax).

We denote by $\mathcal{S}_1 \Longrightarrow \mathcal{S}_2$ the encoding of system $\mathcal{S}_1$ in system $\mathcal{S}_2$, where the label *sh* (resp. *dp*), denotes a shallow (resp. deep) embedding. The notation $\mathcal{S}_1 \hookrightarrow \mathcal{S}_2$ denotes that $\mathcal{S}_2$ is an extension of $\mathcal{S}_1$. Due to lack of space, but with the intention of providing a better formal understanding of the semantics of strong intersection and union types in a logical framework, we provide in Figure 8 a deep LF encoding of a presentation of $\mathcal{B}$ *à la* Church [17]. A shallow encoding of $\mathcal{B}$ in LF_Δ (file intersection_union.bull [44]) can be mechanically type checked in the environment [45]. A shallow encoding of LF in LF_Δ (file lf.bull) making essential use of intersection types can be also type checked.

LF encoding of $\mathcal{B}$. Figure 8 presents a pure LF encoding of a presentation of $\mathcal{B}$ *à la* Church in Coq syntax using HOAS. We use HOAS in order to take advantage of the higher-order features of the frameworks: other abstract syntax representation techniques would not be much different, but more verbose. The `Eq` predicate plays the same role of the essence function in LF_Δ, namely, it encodes the judgement that two proofs (*i.e.* two terms of type `(OK _)`) have the same structure. This is crucial in the `Pair` axiom (*i.e.* the introduction

rule of the intersection type constructor) where we can inhabit the type `(inter s t)` only when the proofs of its component types `s` and `t` share the same structure (*i.e.* we have a witness of type `(Eq s t M N)`, where `M` has type `(OK s)` and `N` has type `(OK t)`). A similar role is played by the `Eq` premise in the `Copair` axiom (*i.e.* the elimination rule of the union type constructor). We have an `Eq` axiom for each proof rule. Examples of this encoding can be found in intersection_union.v [44].

4 Implementation and Future Work

In a previous paper [45], we have implemented in OCaml suitable algorithms for type reconstruction, as well as type checking. In [30] we have implemented the subtyping algorithm which extends the well-known Hindley algorithm for intersection types [24] with union types. The subtyping algorithm has been mechanically proved correct in Coq, extending the Bessai's mechanized proof of a subtyping algorithm for intersection types [8].

A Read-Eval-Print-Loop allows to define axioms and definitions, and performs some basic terminal-style features like error pretty-printing, subexpressions highlighting, and file loading. Moreover, it can type-check a proof or normalize it, using a strong reduction evaluator. We use the syntax of Pure Type Systems [7] to improve the compactness and the modularity of the kernel. Binders are implemented using de Brujin indexes. We implemented the conversion rule in the simplest way possible: when we need to compare types, we syntactically compare their normal form. Abstract and concrete syntax are mostly aligned: the concrete syntax is similar to the concrete syntax of Coq (see Bull and Bull-Subtyping [44]).

We are currently designing a higher-order unification algorithm for Δ-terms and a bidirectional refinement algorithm, similar to the one found in [2]. The refinement can be split into two parts: the essence refinement and the typing refinement. In the same way, there will be a unification algorithm for the essence terms, and a unification algorithm for Δ-terms. The bidirectional refinement algorithm aims to have partial type inference, and to give as much information as possible to a hypothetical solver, or the unifier. For instance, if we want to find a $?y$ such that $\vdash_{\Sigma} \langle \lambda x{:}\sigma.x \ , \ \lambda x{:}\tau.?y\rangle : (\sigma \to \sigma) \cap (\tau \to \tau)$, we can infer that $x{:}\tau \vdash ?y : \tau$ and that $\wr\, ?y \,\wr = x$.

$\mathbf{LF_\Delta}$ in Canonical Form. We presented LF_Δ in the standard LF format in order to support intuition. It would be worthwhile however, to attempt to formulate LF_Δ in the style of [23], using only canonical forms without reductions, especially in view of Adequacy Theorems. The term constructs peculiar to LF_Δ would then introduce new clauses in the definition of canonical and atomic terms. The principle to follow in this task is that atomic terms synthesize their type, while canonical terms are checked against their type. We are currently exploring with the following extension:

$$\begin{aligned} M &::= \ldots \mid \lambda^{r} x.M \mid \langle M \ , \ M\rangle \mid [M \ , \ M] \mid in_l \, M \mid in_r \, M \\ R &::= \ldots \mid pr_l \, R \mid pr_r \, R \mid R^{r} M \end{aligned}$$

Notice the somewhat surprising treatment of the [,] constructor, which is not really an elimination construct but rather behaves as another form of abstraction. Accordingly hereditary substitution needs to be extended.

An intriguing issue raised by one of the referees is to explore the connections between strong implication and the *singleton type* of the identity function. This could lead also to an internalization of the essence function.

References

1 Samson Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51(1):1–77, 1991.

2 Andrea Asperti, Wilmer Ricciotti, Claudio Sacerdoti Coen, and Enrico Tassi. A Bi-Directional Refinement Algorithm for the Calculus of (Co)Inductive Constructions. *Logical Methods in Computer Science*, 8(1), 2012.

3 Franco Barbanera, Mariangiola Dezani-Ciancaglini, and Ugo de'Liguoro. Intersection and union types: syntax and semantics. *Inf. Comput.*, 119(2):202–230, 1995.

4 Franco Barbanera and Simone Martini. Proof-functional connectives and realizability. *Archive for Mathematical Logic*, 33:189–211, 1994.

5 Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A Filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic*, 48(4):931–940, 1983.

6 Gilles Barthe and Olivier Pons. Type Isomorphisms and Proof Reuse in Dependent Type Theory. In *Foundations of Software Science and Computation Structures, 4th International Conference, FOSSACS 2001*, pages 57–71, 2001.

7 Stefano Berardi. *Towards a mathematical analysis of the Coquand–Huet calculus of constructions and the other systems in Barendregt's cube.* PhD thesis, Dipartimento Matematica, Universita di Torino, 1988.

8 Jan Bessai. Extracting a formally verified Subtyping Algorithm for Intersection Types from Ideals and Filters. Talk at COST Types, 2016.

9 Olivier Boite. Proof Reuse with Extended Inductive Types. In *Theorem Proving in Higher Order Logics, 17th International Conference, TPHOLs 2004*, pages 50–65, 2004.

10 Viviana Bono, Betti Venneri, and Lorenzo Bettini. A typed lambda calculus with intersection types. *Theor. Comput. Sci.*, 398(1-3):95–113, 2008.

11 Beatrice Capitani, Michele Loreti, and Betti Venneri. Hyperformulae, Parallel Deductions and Intersection Types. *BOTH, Electr. Notes Theor. Comput. Sci.*, 50(2):180–198, 2001.

12 Joshua E. Caplan and Mehdi T. Harandi. A Logical Framework for Software Proof Reuse. In *SSR*, pages 106–113, 1995.

13 Mario Coppo and Mariangiola Dezani-Ciancaglini. A new type assignment for λ-terms. *Arch. Math. Log.*, 19(1):139–156, 1978.

14 Mario Coppo, Mariangiola Dezani-Ciancaglini, Honsell Furio, and Longo Giuseppe. Extended Type Structures and Filter Lambda Models. In *Logic Colloquium*, pages 241–262, 1983.

15 Mario Coppo, Mariangiola Dezani-Ciancaglini, and Patrick Sallé. Functional characterization of some semantic equalities inside λ-calculus. In *International Colloquium on Automata, Languages, and Programming*, pages 133–146. Springer-Verlag, 1979.

16 Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. Functional characters of solvable terms. *Zietschrift für Mathematische Logik und Grundlagen der Mathematik*, 27(2-6):45–58, 1981.

17 Daniel J. Dougherty, Ugo de'Liguoro, Luigi Liquori, and Claude Stolze. A Realizability Interpretation for Intersection and Union Types. In *APLAS*, volume 10017 of *Lecture Notes in Computer Science*, pages 187–205. Springer-Verlag, 2016.

18 Daniel J. Dougherty and Luigi Liquori. Logic and Computation in a Lambda Calculus with Intersection and Union Types. In *LPAR*, volume 6355 of *Lecture Notes in Computer Science*, pages 173–191. Springer-Verlag, 2010.

19 Joshua Dunfield. Elaborating intersection and union types. *J. Funct. Program.*, 24(2-3):133–165, 2014.

20 Amy P. Felty and Douglas J. Howe. Generalization and Reuse of Tactic Proofs. In *Proc. of Logic Programming and Automated Reasoning, 5th International Conference, LPAR*, pages 1–15, 1994.

21 Alain Frisch, Giuseppe Castagna, and Véronique Benzaken. Semantic subtyping: Dealing set-theoretically with function, union, intersection, and negation types. *Journal of the ACM (JACM)*, 55(4):19, 2008.

22 Robert Harper, Furio Honsell, and Gordon Plotkin. A Framework for Defining Logics. *J. ACM*, 40(1):143–184, 1993.

23 Robert Harper and Daniel R. Licata. Mechanizing metatheory in a logical framework. *J. Funct. Program.*, 17(4-5):613–673, 2007.

24 J. Roger Hindley. The simple semantics for Coppo-Dezani-Sallé types. In *International Symposium on Programming*, pages 212–226, 1982.

25 Furio Honsell and Marina Lenisa. Semantical Analysis of Perpetual Strategies in lambda-Calculus. *Theor. Comput. Sci.*, 212(1-2):183–209, 1999.

26 Furio Honsell, Marina Lenisa, Luigi Liquori, and Ivan Scagnetto. Implementing Cantor's Paradise. In *Proc. of Programming Languages and Systems - 14th Asian Symposium, APLAS*, pages 229–250, 2016.

27 Alexei Kopylov. Dependent intersection: a new way of defining records in type theory. In *Proc. of 18th Annual IEEE Symposium of Logic in Computer Science, LICS*, pages 86–95, 2003.

28 Luigi Liquori, Andreas Nuyts, and Claude Stolze. Privates communications, 2017.

29 Luigi Liquori and Simona Ronchi Della Rocca. Intersection Typed System *à la* Church. *Information and Computation*, 9(205):1371–1386, 2007.

30 Luigi Liquori and Claude Stolze. A Decidable Subtyping Logic for Intersection and Union Types. In *Proc of TTCS*, volume 10608 of *Lecture Notes in Computer Science*, pages 74–90. Springer-Verlag, 2017.

31 Luigi Liquori and Claude Stolze. The Delta-calculus: syntax and types. Research report, Inria, July 2018. URL: `https://arxiv.org/abs/1803.09660`.

32 Edgar G. K. Lopez-Escobar. Proof functional connectives. In *Methods in Mathematical Logic*, volume 1130 of *Lecture Notes in Mathematics*, pages 208–221. Springer-Verlag, 1985.

33 David B. MacQueen, Gordon D. Plotkin, and Ravi Sethi. An Ideal Model for Recursive Polymorphic Types. *Information and Control*, 71(1/2):95–130, 1986.

34 Robert K Meyer and Richard Routley. Algebraic analysis of entailment I. *Logique et Analyse*, 15:407–428, 1972.

35 Grigori Mints. The Completeness of Provable Realizability. *Notre Dame Journal of Formal Logic*, 30(3):420–441, 1989.

36 Alexandre Miquel. The Implicit Calculus of Constructions. In *TLCA*, pages 344–359, 2001.

37 Frank Pfenning. Refinement Types for Logical Frameworks. In *TYPES*, pages 285–299, 1993.

38 Benjamin C. Pierce. *Programming with intersection types, union types, and bounded polymorphism.* PhD thesis, Technical Report CMU-CS-91-205. Carnegie Mellon University, 1991.

39 Elaine Pimentel, Simona Ronchi Della Rocca, and Luca Roversi. Intersection Types from a Proof-theoretic Perspective. *Fundam. Inform.*, 121(1-4):253–274, 2012.

40 Garrel Pottinger. A Type Assignment for the Strongly Normalizable λ-terms. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 561–577. Academic Press, 1980.

41 John C. Reynolds. Preliminary Design of the Programming Language Forsythe. Report CMU–CS–88–159, Carnegie Mellon University, 1988.

42 Simona Ronchi Della Rocca and Luca Roversi. Intersection logic. In *CSL*, volume 2142 of *Lecture Notes in Computer Science*, pages 421–428. Springer-Verlag, 2001.

43 Vincent Siles and Hugo Herbelin. Equality Is Typable in Semi-full Pure Type Systems. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS*, pages 21–30, 2010.

44 Claude Stolze. Δ-framework implementation. Bull and Bull-Subtyping, 2017.

45 Claude Stolze, Luigi Liquori, Furio Honsell, and Ivan Scagnetto. Towards a Logical Framework with Intersection and Union Types. In *11th International Workshop on Logical Frameworks and Meta-languages, LFMTP*, pages 1–9, 2017.

46 Aaron Stump. From realizability to induction via dependent intersection. *Annals of Pure and Applied Logic*, 169(7):637–655, 2018.

47 Betti Venneri. Intersection Types as Logical Formulae. *J. Log. Comput.*, 4(2):109–124, 1994.

48 Joe B. Wells, Allyn Dimock, Robert Muller, and Franklyn Turbak. A calculus with polymorphic and polyvariant flow types. *J. Funct. Program.*, 12(3):183–227, 2002.

49 Joe B. Wells and Christian Haack. Branching Types. In *ESOP*, volume 2305 of *Lecture Notes in Computer Science*, pages 115–132. Springer-Verlag, 2002.

A Appendix

Let Figure 9 denote Valid Signatures and Contexts and Figure 10 denote Valid Kinds and Families.

LF_Δ can play the role of a logical framework only if decidable. The road map which we follow to establish decidability is the standard one, see *e.g.* [22]. In particular, we prove in order: uniqueness of types and kinds, structural properties, normalization for raw well-formed terms, and hence confluence. Then we prove the inversion property, the subderivation property, subject reduction, and finally decidability.

▶ **Lemma 10.** *Let α be either $\sigma : K$ or $\Delta : \sigma$. Then:*

1. *Weakening: If $\Gamma \vdash_\Sigma \alpha$ and $\vdash_\Sigma \Gamma, \Gamma'$, then $\Gamma, \Gamma' \vdash_\Sigma \alpha$.*
2. *Strengthening: If $\Gamma, x{:}\sigma, \Gamma' \vdash_\Sigma \alpha$, then $\Gamma, \Gamma' \vdash_\Sigma \alpha$, provided that $x \notin FV(\Gamma') \cup FV(\alpha)$.*
3. *Transitivity: If $\Gamma \vdash_\Sigma \Delta : \sigma$ and $\Gamma, x{:}\sigma, \Gamma' \vdash_\Sigma \alpha$, then $\Gamma, \Gamma'[\Delta/x] \vdash_\Sigma \alpha[\Delta/x]$.*
4. *Permutation: If $\Gamma, x_1{:}\sigma, \Gamma', x_2{:}\tau, \Gamma'' \vdash_\Sigma \alpha$, then $\Gamma, x_2{:}\tau, \Gamma', x_1{:}\sigma, \Gamma''$ $\vdash_\Sigma \alpha$, provided that x_1 does not occur free in Γ' or in τ, and that τ is valid in Γ.*

▶ **Theorem 3** (Unicity of Types and Kinds)**.**

1. *If $\Gamma \vdash_\Sigma \Delta : \sigma$ and $\Gamma \vdash_\Sigma \Delta : \tau$, then $\sigma =_\Delta \tau$.*
2. *If $\Gamma \vdash_\Sigma \sigma : K$ and $\Gamma \vdash_\Sigma \sigma : K'$, then $K =_\Delta K'$.*

In order to prove strong normalization we follow the pattern used for pure LF. Namely, we map LF_Δ-terms into terms of the system $\mathcal{B}$ in such a way that redexes in the source language are mapped into redexes in the target language, and then take advantage of Theorem 2. Special care is needed in dealing with redexes occurring in type-dependencies, because these need to be flattened at the level of terms.

▶ **Definition 11.** Let the forgetful mappings $\|\cdot\|$ and $|\cdot|$ be defined as in Figure 11.

The forgetful mappings are extended to contexts and signatures in the obvious way. The clauses for strong pairs/co-pairs are justified by the following lemma:

▶ **Lemma 12.** *If $\Gamma \vdash_\Sigma \langle \Delta_1 , \Delta_2 \rangle : \sigma$ or $\Gamma \vdash_\Sigma [\Delta_1 , \Delta_2] : \sigma$, then $|\Delta_1| =_\beta |\Delta_2|$.*

The following lemmas are proved by straightforward structural induction.

Let $\Gamma \stackrel{def}{=} \{x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n\}$ ($i \neq j$ implies $x_i \not\equiv x_j$), and $\Gamma, x{:}\sigma \stackrel{def}{=} \Gamma \cup \{x{:}\sigma\}$

Let $\Sigma \stackrel{def}{=} \{c_1{:}\sigma_1, \ldots, c_n{:}\sigma_n\}$, and $\Sigma, c{:}\sigma \stackrel{def}{=} \Sigma \cup \{c{:}\sigma\}$

Valid Signatures

$$\frac{}{\langle\rangle \text{ sig}}\ (\epsilon\Sigma) \qquad \frac{\Sigma \text{ sig} \quad \vdash_\Sigma K \quad a \notin \text{dom}(\Sigma)}{\Sigma, a{:}K \text{ sig}}\ (K\Sigma) \qquad \frac{\Sigma \text{ sig} \quad \vdash_\Sigma \sigma : \mathsf{Type} \quad c \notin \text{dom}(\Sigma)}{\Sigma, c{:}\sigma \text{ sig}}\ (\sigma\Sigma)$$

Valid Contexts

$$\frac{\Sigma \text{ sig}}{\vdash_\Sigma \langle\rangle}\ (\epsilon\Gamma) \qquad \frac{\vdash_\Sigma \Gamma \quad \Gamma \vdash_\Sigma \sigma : \mathsf{Type} \quad x \notin \text{dom}(\Gamma)}{\vdash_\Sigma \Gamma, x{:}\sigma}\ (\sigma\Gamma)$$

Figure 9 Valid Signatures and Contexts.

Valid Kinds

$$\frac{\vdash_\Sigma \Gamma}{\Gamma \vdash_\Sigma \mathsf{Type}}\ (Type) \qquad \frac{\Gamma, x{:}\sigma \vdash_\Sigma K}{\Gamma \vdash_\Sigma \Pi x{:}\sigma.K}\ (\Pi K)$$

Valid Families

$$\frac{\vdash_\Sigma \Gamma \quad a{:}K \in \Sigma}{\Gamma \vdash_\Sigma a : K}\ (Const) \qquad \frac{\Gamma \vdash_\Sigma \sigma : K_1 \quad \Gamma \vdash_\Sigma K_2 \quad K_1 =_\Delta K_2}{\Gamma \vdash_\Sigma \sigma : K_2}\ (Conv)$$

$$\frac{\Gamma, x{:}\sigma \vdash_\Sigma \tau : \mathsf{Type}}{\Gamma \vdash_\Sigma \Pi x{:}\sigma.\tau : \mathsf{Type}}\ (\Pi I) \qquad \frac{\Gamma \vdash_\Sigma \sigma : \Pi x{:}\tau.K \quad \Gamma \vdash_\Sigma \Delta : \tau}{\Gamma \vdash_\Sigma \sigma\, \Delta : K[\Delta/x]}\ (\Pi E)$$

$$\frac{\Gamma \vdash_\Sigma \sigma : \mathsf{Type} \quad \Gamma \vdash_\Sigma \tau : \mathsf{Type}}{\Gamma \vdash_\Sigma \sigma \to^{\mathsf{r}} \tau : \mathsf{Type}}\ (\to^{\mathsf{r}} I)$$

$$\frac{\Gamma \vdash_\Sigma \sigma : \mathsf{Type} \quad \Gamma \vdash_\Sigma \tau : \mathsf{Type}}{\Gamma \vdash_\Sigma \sigma \cap \tau : \mathsf{Type}}\ (\cap I) \qquad \frac{\Gamma \vdash_\Sigma \sigma : \mathsf{Type} \quad \Gamma \vdash_\Sigma \tau : \mathsf{Type}}{\Gamma \vdash_\Sigma \sigma \cup \tau : \mathsf{Type}}\ (\cup I)$$

Figure 10 Valid Kinds and Families.

▶ **Lemma 13.**

1. *If* $\sigma =_\Delta \tau$, *then* $\|\sigma\| =_\beta \|\tau\|$.
2. *If* $K_1 =_\Delta K_2$, *then* $\|K_1\| =_\beta \|K_2\|$.

▶ **Lemma 14.**

1. $|\Delta_1[\Delta_2/x]| =_\beta |\Delta_1|\,[|\Delta_2|\,/x]$.
2. $|\sigma[\Delta/x]| =_\beta |\sigma|\,[|\Delta|\,/x]$.

▶ **Lemma 15.**

1. *If* $\Gamma \vdash_\Sigma \sigma : K$, *then* $\|\Gamma\| \vdash_{\mathcal{B}^+} |\sigma| : \|K\|$.
2. *If* $\Gamma \vdash_\Sigma \Delta : \sigma$, *then* $\|\Gamma\| \vdash_{\mathcal{B}^+} |\Delta| : \|\sigma\|$.

where $\vdash_{\mathcal{B}^+}$ *denotes the type system* $\mathcal{B}$, *augmented by* $c_\times : \top \to \top \to \top$ *and the infinite set of axioms* $c_{\|\sigma\|} : \top \to (\|\sigma\| \to \top) \to \top$, *for each type* σ.

Notice that the function $\wr\ \wr$ and $|\ |$ treat differently relevant implication.

$$
\begin{array}{rcl@{\qquad}rcl}
\|\mathsf{Type}\| & = & \top \quad \text{(a special constant)} & \|\sigma \to^{\mathsf{r}} \tau\| & = & \|\sigma\| \to \|\tau\| \\
\|\Pi x{:}\sigma.K\| & = & \|\sigma\| \to \|K\| & \|\sigma\,\Delta\| & = & \|\sigma\| \\
\|a\| & = & a & \|\sigma \cap \tau\| & = & \|\sigma\| \cap \|\tau\| \\
\|\Pi x{:}\sigma.\tau\| & = & \|\sigma\| \to \|\tau\| & \|\sigma \cup \tau\| & = & \|\sigma\| \cup \|\tau\|
\end{array}
$$

$$
\begin{array}{rcl@{\qquad}rcl}
|a| & = & a & |\sigma \to^{\mathsf{r}} \tau| & = & c_{\times}\ |\sigma|\,|\tau| \\
|c| & = & c & |\sigma \cap \tau| & = & c_{\times}\ |\sigma|\,|\tau| \\
|x| & = & x & |\sigma \cup \tau| & = & c_{\times}\ |\sigma|\,|\tau| \\
|\sigma\,\Delta| & = & |\sigma|\,|\Delta| & |\langle \Delta_1\ ,\ \Delta_2 \rangle| & = & |\Delta_1| \\
|\Delta_1\,\Delta_2| & = & |\Delta_1|\,|\Delta_2| & |[\Delta_1\ ,\ \Delta_2]| & = & |\Delta_1| \\
|\Delta_1\,{}^{\mathsf{r}}\Delta_2| & = & |\Delta_1|\,|\Delta_2| & |pr_l\,\Delta| & = & |\Delta| \\
|\lambda x{:}\sigma.\Delta| & = & (\lambda y.\lambda x.\,|\Delta|)\,|\sigma|\ \ y \notin fv(\Delta) & |pr_r\,\Delta| & = & |\Delta| \\
|\curlywedge x{:}\sigma.\Delta| & = & (\lambda y.\lambda x.\,|\Delta|)\,|\sigma|\ \ y \notin fv(\Delta) & |in_l^{\sigma}\,\Delta| & = & (\lambda x.\,|\Delta|)\,|\sigma|\ \ x \notin fv(\Delta) \\
|\Pi x{:}\sigma.\tau| & = & c_{\|\sigma\|}\ |\sigma|\ (\lambda x.\,|\tau|) & |in_r^{\sigma}\,\Delta| & = & (\lambda x.\,|\Delta|)\,|\sigma|\ \ x \notin fv(\Delta)
\end{array}
$$

Figure 11 The forgetful mappings $\|\cdot\|$ and $|\cdot|$

▶ **Lemma 16.**

1. *If* $\sigma \longrightarrow_\beta \tau$, *then* $|\sigma| \longrightarrow_\beta^+ |\tau|$.
2. *if* $\Delta_1 \longrightarrow_\beta \Delta_2$, *then* $|\Delta_1| \longrightarrow_\beta^+ |\Delta_2|$.

Parallel reduction enjoys the strong normalization property, *i.e.*

▶ **Theorem 4** (Strong normalization)**.**

1. *The* LF_Δ *is strongly normalizing, i.e.,*
   a. *If* $\Gamma \vdash_\Sigma K$, *then* K *is strongly normalizing.*
   b. *If* $\Gamma \vdash_\Sigma \sigma : K$, *then* σ *is strongly normalizing.*
   c. *If* $\Gamma \vdash_\Sigma \Delta : \sigma$, *then* Δ *is strongly normalizing.*
2. *Every strongly normalizing pure* λ*-term can be annotated so as to be the essence of a* Δ*-term.*

Proof. 1) Strong normalization derives directly from Lemmas 15, 16 and Theorem 2.
2) By induction on the specification of strongly normalizing terms which can be inductively defined as i) $\Delta_1 \ldots \Delta_n \in SN \Rightarrow \lambda x_1, \ldots, x_n.x\,\Delta_1 \ldots \Delta_n \in SN$ for x possibly among the x_i's, ii) $\Delta[\Delta'/x]\,\Delta_1 \ldots \Delta_n \in SN$, and iii) $\Delta' \in SN \Rightarrow (\lambda x{:}\sigma.\Delta)\,\Delta'\,\Delta_1\ \ldots\ \Delta_n \in SN$. ◀

Local confluence (Proposition 1) and strong normalization (Theorem 4) entail confluence, so we have

▶ **Theorem 5** (Confluence)**.** LF_Δ *is confluent, i.e.:*

1. *If* $K_1 \longrightarrow_\Delta^* K_2$ *and* $K_1 \longrightarrow_\Delta^* K_3$, *then* $\exists K_4$ *such that* $K_2 \longrightarrow_\Delta^* K_4$ *and* $K_3 \longrightarrow_\Delta^* K_4$.
2. *If* $\sigma_1 \longrightarrow_\Delta^* \sigma_2$ *and* $\sigma_1 \longrightarrow_\Delta^* \sigma_3$, *then* $\exists \sigma_4$ *such that* $\sigma_2 \longrightarrow_\Delta^* \sigma_4$ *and* $\sigma_3 \longrightarrow_\Delta^* \sigma_4$.
3. *If* $\Delta_1 \longrightarrow_\Delta^* \Delta_2$ *and* $\Delta_1 \longrightarrow_\Delta^* \Delta_3$, *then* $\exists \Delta_4$ *such that* $\Delta_2 \longrightarrow_\Delta^* \Delta_4$ *and* $\Delta_3 \longrightarrow_\Delta^* \Delta_4$.

The following lemmas are proved by structural induction.

▶ **Lemma 17** (Inversion properties).

1. *If* $\Pi x{:}\sigma.\tau =_\Delta \tau''$, *then* $\tau'' \equiv \Pi x{:}\sigma'.\tau'$, *for some* σ', τ', *such that* $\sigma' =_\Delta \sigma$, *and* $\tau' =_\Delta \tau$.
2. *If* $\sigma \to^{\mathsf{r}} \tau =_\Delta \tau''$, *then* $\tau'' \equiv \sigma' \to^{\mathsf{r}} \tau'$, *for some* σ', τ', *such that* $\sigma' =_\Delta \sigma$, *and* $\tau' =_\Delta \tau$.
3. *If* $\sigma \cap \tau =_\Delta \rho$, *then* $\rho \equiv \sigma' \cap \tau'$, *for some* σ', τ', *such that* $\sigma' =_\Delta \sigma$, *and* $\tau' =_\Delta \tau$.
4. *If* $\sigma \cup \tau =_\Delta \rho$, *then* $\rho \equiv \sigma' \cup \tau'$, *for some* σ', τ', *such that* $\sigma' =_\Delta \sigma$, *and* $\tau' =_\Delta \tau$.
5. *If* $\Gamma \vdash_\Sigma \lambda x{:}\sigma.\Delta : \Pi x{:}\sigma.\tau$, *then* $\Gamma, x{:}\sigma \vdash_\Sigma \Delta : \tau$.
6. *If* $\Gamma \vdash_\Sigma \lambda^{\mathsf{r}} x{:}\sigma.\Delta : \Pi x{:}\sigma.\tau$, *then* $\Gamma, x{:}\sigma \vdash_\Sigma \Delta : \tau$ *and* $\wr \Delta \wr =_\eta x$.
7. *If* $\Gamma \vdash_\Sigma \langle \Delta_1 \ , \Delta_2 \rangle : \sigma \cap \tau$, *then* $\Gamma \vdash_\Sigma \Delta_1 : \sigma$, $\Gamma \vdash_\Sigma \Delta_2 : \tau$, *and* $\wr \Delta_1 \wr =_\beta \wr \Delta_2 \wr$.
8. *If* $\Gamma \vdash_\Sigma [\Delta_1 \ , \Delta_2] : \Pi x{:}\sigma \cup \tau.\rho$, *then* $\Gamma \vdash_\Sigma \Delta_1 : \Pi y{:}\sigma.\rho\,(in_l^\tau\ y)$, $\Gamma \vdash_\Sigma \Delta_2 : \Pi y{:}\tau.\rho\,(in_r^\sigma\ y)$, *and* $\wr \Delta_1 \wr =_\beta \wr \Delta_2 \wr$.
9. *If* $\Gamma \vdash_\Sigma pr_l\,\Delta : \sigma$, *then* $\Gamma \vdash_\Sigma \Delta{:}\sigma \cap \tau$, *for some* τ.
10. *If* $\Gamma \vdash_\Sigma pr_r\,\Delta : \tau$, *then* $\Gamma \vdash_\Sigma \Delta{:}\sigma \cap \tau$, *for some* σ.
11. *If* $\Gamma \vdash_\Sigma in_l^\tau\,\Delta : \sigma \cup \tau$, *then* $\Gamma \vdash_\Sigma \Delta : \sigma$ *and* $\Gamma \vdash_\Sigma \sigma \cup \tau : \mathsf{Type}$.
12. *If* $\Gamma \vdash_\Sigma in_r^\sigma\,\Delta : \sigma \cup \tau$, *then* $\Gamma \vdash_\Sigma \Delta : \tau$ *and* $\Gamma \vdash_\Sigma \sigma \cup \tau : \mathsf{Type}$.

▶ **Proposition 18** (Subderivation).

1. *A derivation of* $\vdash_\Sigma \langle\rangle$ *has a subderivation of* Σ *sig*.
2. *A derivation of* $\Sigma, a{:}K$ *sig has subderivations of* Σ *sig and* $\vdash_\Sigma K$.
3. *A derivation of* $\Sigma, f{:}\sigma$ *sig has subderivations of* Σ *sig and* $\vdash_\Sigma \sigma{:}\mathsf{Type}$.
4. *A derivation of* $\vdash_\Sigma \Gamma, x{:}\sigma$ *has subderivations of* Σ *sig,* $\vdash_\Sigma \Gamma$, *and* $\Gamma \vdash_\Sigma \sigma{:}\mathsf{Type}$.
5. *A derivation of* $\Gamma \vdash_\Sigma \alpha$ *has subderivations of* Σ *sig and* $\vdash_\Sigma \Gamma$.
6. *Given a derivation of the judgement* $\Gamma \vdash_\Sigma \alpha$, *and a subterm occurring in the subject of this judgement, there exists a derivation of a judgement having this subterm as a subject.*

▶ **Theorem 6** (Subject reduction of LF_Δ).

1. *If* $\Gamma \vdash_\Sigma K$, *and* $K \to_\Delta K'$, *then* $\Gamma \vdash_\Sigma K'$.
2. *If* $\Gamma \vdash_\Sigma \sigma : K$, *and* $\sigma \to_\Delta \sigma'$, *then* $\Gamma \vdash_\Sigma \sigma' : K$.
3. *If* $\Gamma \vdash_\Sigma \Delta : \sigma$, *and* $\Delta \to_\Delta \Delta'$, *then* $\Gamma \vdash_\Sigma \Delta' : \sigma$.

Finally, we define a possible algorithm for checking judgements in LF_Δ by computing a type or a kind for a term, and then testing for *definitional equality*, *i.e.* $=_\Delta$, against the given type or kind. This is achieved by reducing both to their unique normal forms and checking that they are identical up to α-conversion. Therefore we finally have:

▶ **Theorem 7** (Decidability). *All the type judgments of* LF_Δ *are recursively decidable.*

Minimal Relevant Implications and Type Inclusion. Type inclusion and the rules of *subtyping* are related to the notion of minimal relevant implication, see [4, 17]. The insight is quite subtle, but ultimately very simple. This is what makes it appealing. The apparently intricate rules of subtyping and type inclusion, which occur in many systems, and might even appear *ad hoc* at times, can all be explained away in our principled approach, by proving that the relevant implication type is inhabited by a term whose essence is essentially a variable.

The following theorem we show how relevant implication subsumes the type-inclusion rules of the theory Ξ of [3], without rule (10): we call Ξ' the resulting set.

▶ **Theorem 8** (Type Inclusion). *The judgement* $\langle\rangle \vdash_\Sigma \Delta : \sigma \to^{\mathsf{r}} \tau$ *(where both* σ *and* τ *do not contain dependencies or relevant families) holds iff* $\sigma \leq \tau$ *holds in the subtype theory* Ξ' *of* $\mathcal{B}$ *enriched with new axioms of the form* $\sigma_1 \leq \sigma_2$ *for each constant* $c : \sigma_1 \to^{\mathsf{r}} \sigma_2 \in \Sigma$.

Proof.
(if). Follows directly from Lemma 17.

(only if). It is possible to write a Δ-term whose essence is an $\eta-$expansion of the identity $(\lambda x.x)$ corresponding to each of the axioms and rules in Ξ'. The Δ-term is obtained by defining a function $\|\sigma \leq \tau\|_\Delta$, where $\sigma \leq \tau$ is a subtyping derivation tree in the type theory Ξ', which coerce a Δ-term from type σ to type τ:

$$(1) \quad \|\sigma \leqslant \sigma \cap \sigma\|_\Delta \stackrel{def}{=} \langle \Delta \ , \ \Delta \rangle$$

$$(2) \quad \|\sigma \cup \sigma \leqslant \sigma\|_\Delta \stackrel{def}{=} [\lambda x{:}\sigma.x \ , \ \lambda x{:}\sigma.x] \, \Delta$$

$$(3) \quad \|\sigma_1 \cap \sigma_2 \leqslant \sigma_i\|_\Delta \stackrel{def}{=} pr_i \, \Delta$$

$$(4) \quad \|\sigma_i \leqslant \sigma_1 \cup \sigma_2\|_\Delta \stackrel{def}{=} in_i \, \Delta$$

$$(6) \quad \|\sigma \leqslant \sigma\|_\Delta \stackrel{def}{=} \Delta$$

$$(7) \quad \left\|\frac{\sigma_1 \leqslant \sigma_2 \quad \tau_1 \leqslant \tau_2}{\sigma_1 \cap \tau_1 \leqslant \sigma_2 \cap \tau_2}\right\|_\Delta \stackrel{def}{=} \langle \|\sigma_1 \leqslant \sigma_2\|_{(pr_l \, \Delta)} \ , \ \|\tau_1 \leqslant \tau_2\|_{(pr_r \, \Delta)} \rangle$$

$$(8) \quad \left\|\frac{\sigma_1 \leqslant \sigma_2 \quad \tau_1 \leqslant \tau_2}{\sigma_1 \cup \tau_1 \leqslant \sigma_2 \cup \tau_2}\right\|_\Delta \stackrel{def}{=} [\lambda x{:}\sigma_1.in_l^{\tau_2} \ \|\sigma_1 \leqslant \sigma_2\|_x \ , \ \lambda x{:}\tau_1.in_r^{\sigma_2} \ \|\tau_1 \leqslant \tau_2\|_x] \, \Delta$$

$$(9) \quad \left\|\frac{\sigma \leqslant \tau \quad \tau \leqslant \rho}{\sigma \leqslant \rho}\right\|_\Delta \stackrel{def}{=} \|\tau \leqslant \rho\|_{(\|\sigma \leqslant \tau\|_\Delta)}$$

$$(11) \quad \|(\sigma \to \tau) \cap (\sigma \to \rho) \leqslant \sigma \to (\tau \cap \rho)\|_\Delta \stackrel{def}{=} \lambda x{:}\sigma.\langle (pr_l \, \Delta) \, x \ , \ (pr_r \, \Delta) \, x \rangle$$

$$(12) \quad \|(\sigma \to \rho) \cap (\tau \to \rho) \leqslant (\sigma \cup \tau) \to \rho\|_\Delta \stackrel{def}{=} \lambda x{:}\sigma \cup \tau.[\lambda y{:}\sigma.(pr_l \, \Delta) \, y \ , \ \lambda y{:}\tau.(pr_r \, \Delta) \, y] \, x$$

$$(14) \quad \left\|\frac{\sigma_2 \leqslant \sigma_1 \quad \tau_1 \leqslant \tau_2}{\sigma_1 \to \tau_1 \leqslant \sigma_2 \to \tau_2}\right\|_\Delta \stackrel{def}{=} \lambda x{:}\sigma_2. \, \|\tau_1 \leqslant \tau_2\|_{(\Delta \, \|\sigma_2 \leqslant \sigma_1\|_x)}$$

◀

A.1 Typed derivation of Pierce's example of Subsection 2.1

$$\dfrac{\dfrac{\begin{array}{l}\Gamma \vdash_\Sigma \lambda x_1{:}\sigma_1.(pr_l\, x)\, x_1\, x_1 : \Pi x_1{:}\sigma_1.(a\, x_4\, x_4)[in_l^{\sigma_2}\, x_1/x_4] \\ \Gamma \vdash_\Sigma \lambda x_2{:}\sigma_2.(pr_r\, x)\, x_2\, x_2 : \Pi x_2{:}\sigma_2.(a\, x_4\, x_4)[in_r^{\sigma_1}\, x_2/x_4] \\ \Gamma, x_4{:}\sigma_1 \cup \sigma_2 \vdash_\Sigma a\, x_4\, x_4 : \mathsf{Type} \\ \wr\, \lambda x_1{:}\sigma_1.(pr_l\, x)\, x_1\, x_1 \wr =_\eta \wr\, \lambda x_2{:}\sigma_2.(pr_r\, x)\, x_2\, x_2) \wr\end{array}}{\Gamma \vdash_\Sigma [\lambda x_1{:}\sigma_1.(pr_l\, x)\, x_1\, x_1\ ,\ \lambda x_2{:}\sigma_2.(pr_r\, x)\, x_2\, x_2] : \Pi x_4{:}\sigma_1 \cup \sigma_2.a\, x_4\, x_4}\ (\cup E) \qquad \Gamma \vdash_\Sigma (\lambda x_3{:}\rho \to \sigma_1 \cup \sigma_2.x_3)\, y\, z : \sigma_1 \cup \sigma_2}{\Gamma \vdash_\Sigma [(\lambda x_1{:}\sigma_1.(pr_l\, x)\, x_1\, x_1)\ ,\ (\lambda x_2{:}\sigma_2.(pr_r\, x)\, x_2\, x_2)]\, ((\lambda x_3{:}\rho \to \sigma_1 \cup \sigma_2.x_3)\, y\, z) : a\, ((\lambda x_3{:}\rho \to \sigma_1 \cup \sigma_2.x_3)\, y\, z)\, ((\lambda x_3{:}\rho \to \sigma_1 \cup \sigma_2.x_3)\, y\, z)}\ (\Pi E)$$
$$\dfrac{\Gamma \vdash_\Sigma [\lambda x_1{:}\sigma_1.(pr_l\, x)\, x_1\, x_1)\ ,\ \lambda x_2{:}\sigma_2.(pr_r\, x)\, x_2\, x_2)]\, ((\lambda x_3{:}\rho \to \sigma_1 \cup \sigma_2.x_3)\, y\, z) : a\, (y\, z)\, (y\, z)}{}\ (Conv)$$

where

$$\Gamma \stackrel{def}{=} x{:}\Pi x_1{:}\sigma_1.\Pi x_2{:}\sigma_1.a\, (in_l^{\sigma_2}\ x_1)\, (in_l^{\sigma_2}\ x_2) \cap \Pi x_1{:}\sigma_2.\Pi x_2{:}\sigma_2.a\, (in_r^{\sigma_1}\ x_1)\, (in_r^{\sigma_1}\ x_2), y{:}\rho \to \sigma_1 \cup \sigma_2, z{:}\rho$$

and

$$\Sigma \stackrel{def}{=} a{:}\sigma_1 \cup \sigma_2 \to \sigma_1 \cup \sigma_2 \to \mathsf{Type}$$

Extending Finite-Memory Determinacy by Boolean Combination of Winning Conditions

Stéphane Le Roux
LSV, CNRS & ENS Cachan, Université Paris-Saclay, Cachan, France

Arno Pauly
Swansea University, Swansea, United Kingdom

Mickael Randour[1]
F.R.S.-FNRS & UMONS – Université de Mons, Mons, Belgium

Abstract

We study finite-memory (FM) determinacy in games on finite graphs, a central question for applications in controller synthesis, as FM strategies correspond to implementable controllers. We establish general conditions under which FM strategies suffice to play optimally, even in a broad multi-objective setting. We show that our framework encompasses important classes of games from the literature, and permits to go further, using a unified approach. While such an approach cannot match ad-hoc proofs with regard to tightness of memory bounds, it has two advantages: first, it gives a widely-applicable criterion for FM determinacy; second, it helps to understand the cornerstones of FM determinacy, which are often hidden but common in proofs for specific (combinations of) winning conditions.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases games on graphs, finite-memory determinacy, multiple objectives

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.38

Related Version An extended version of this paper is available at [24], `https://arxiv.org/abs/1808.05791`.

Funding Research partially supported by F.R.S.-FNRS under Grant n° F.4520.18 (*ManySynth*).

1 Introduction

Controller synthesis through the game-theoretic metaphor. *Two-player games on graphs* are widely studied, notably for their applications in controller synthesis for reactive systems [18, 22, 5, 1]. In this context, Player 1 is seen as the system to control, Player 2 as its uncontrollable environment, and the game models their interaction. The objective of Player 1 is to enforce a given specification represented as a *winning condition*. The goal of synthesis is thus to decide if Player 1 has a *winning strategy*, i.e., one that guarantees victory against all possible strategies of Player 2, and to build such a strategy efficiently if it exists.

Winning strategies are essentially *formal blueprints* for controllers that one may want to implement in practical applications. With that in mind, the complexity of such strategies is of tremendous importance: the simpler the strategy, the easier and cheaper it will be to build the corresponding controller and maintain it. That is why a lot of research effort is put into pinpointing the complexity (in terms of memory and/or randomness) of strategies needed to play optimally for each specific class of games and winning conditions.

[1] F.R.S.-FNRS Research Associate.

Memoryless determinacy. An elegant result by Gimbert and Zielonka established more than ten years ago characterizes the winning conditions that enjoy *memoryless determinacy*[2] in two-player turn-based zero-sum games on finite graphs [17]. These include conditions such as *parity*, *mean-payoff*, or *energy*, all in the single-objective case.

The need for memory. Over the last decade, the increasing need to model complex specifications has shifted research toward games where multiple quantitative and qualitative objectives co-exist and interact, requiring the analysis of *interplay* and *trade-offs* between several objectives. Consider a computer server responding to requests. Decreasing its response-time (modeled using a mean-payoff condition) would usually require to increase its computational power and energy consumption (modeled by an energy condition). Hence, we need to consider games where winning conditions are actually conjunctions of conditions, or even richer Boolean combinations. In this context, memoryless strategies do not suffice, and one has to use an amount of memory which can quickly become an obstacle to implementation (e.g., exponential memory) or which can prevent it completely (infinite memory).

See for example [9, 13, 19] for combinations of energy and parity, [26] for combinations of mean-payoff, [4, 3] for combinations of energy and *average-energy* conditions, [10] for combinations of *total-payoff*, or [10, 6] for combinations of *window* objectives. Establishing precise complexity bounds for such general combinations of winning conditions is tricky and sometimes counterintuitive. For example, while energy games and mean-payoff games are inter-reducible in the single-objective setting, exponential-memory strategies are both sufficient and necessary for conjunctions of energy conditions while *infinite-memory* strategies are required for conjunctions of mean-payoff ones.

Our contribution. Our goal is to provide a *general and abstract theorem* that lets us draw conclusions over games with complex objectives, provided that the primitive objectives used in the construction of the complex objectives fulfill some criteria. Such an abstract approach provides results for many concrete types of objectives at once, and can be applied to new types of objectives in which the community may become interested later. Another advantage of an abstract approach is that it reveals partially the *core features* determining whether the results hold or not. Admittedly, a downside of the abstract approach is that the concrete bounds that we obtain for the required memory will often be far worse than those established in concrete instances (as we depend on the most general upper bound).

Let us sketch our approach intuitively. First, we define the notion of *regularly-predictable* winning condition: a winning condition is regularly-predictable if for every game using it, a finite automaton suffices to recognize histories from which Player 1 has a winning strategy. As we will show, many well-behaved winning conditions fall in this category (including all prefix-independent ones). Second, we consider *regular languages* which subsume many simple winning conditions (e.g., fully bounded energy, window objectives). Third, we introduce the notion of *hypothetical subgame-perfect equilibrium* (hSPE) that is a key technical tool for the theorem to come.[3] Finally, we prove our main result: given a class $\mathcal{W}$ of regularly-predictable winning conditions admitting FM hSPE and closed under Boolean combination, any winning condition obtained by Boolean combination of elements of $\mathcal{W}$ and regular languages is itself a regularly-predictable condition admitting FM hSPE.

---

[2] The existence, in each vertex of the game, of a winning strategy that requires no memory at all, for one of the two players.

[3] Morally, we consider regularly-predictable winning conditions for which FM strategies suffice, but for technical reasons, we need the slightly stronger assumption of existence of FM hSPE.

Discussion. The sketch depicted above may seem unsurprising to the expert reader. Indeed, regular languages are morally equivalent to *safety* conditions, and combining reasonably well-behaved objectives with safety objectives should preserve FM determinacy (with a blow-up due the possible compact encoding of the safety-like conditions). This is perfectly true but in our opinion the interest of our contribution is elsewhere. First, while it is *morally* easy to believe that such an extension of FM determinacy will hold, it is actually quite tedious to manage to prove it for general classes of games (e.g., see all aforementioned articles on specific combinations of objectives). Here, we provide a *general proof* of this scheme of extension, based on (in our humble opinion) *elegant and natural concepts* such as regularly-predictable objectives and hSPE. Second, we provide a thorough discussion of the hypotheses needed for our main theorem, and we show that they are relatively *tight* in the sense that weakening any condition almost immediately leads to falsification of FM determinacy for the resulting complex objective. Thus, we give a *rather complete picture of the frontiers of FM determinacy for combination of objectives* that, hopefully, will help in understanding its cornerstones and drive further research toward such amenable objectives.

Related work. We already mentioned several papers studying specific (combinations of) winning conditions. Here, we highlight works where similar approaches have been considered to establish "meta-theorems" applying to general classes of games. Arguably the most important result in this direction is the determinacy theorem by Martin that guarantees determinacy (without considering the complexity of strategies) for Borel winning conditions [21]. We already discussed Gimbert and Zielonka's focus on memoryless strategies [17]. Kopczynski studied games where memoryless strategies suffice for only one of the players whereas his opponent requires memory [20]. Finally, a similar approach has been pursued in [23], where abstract criteria were identified that enable moving from FM determinacy of two-player win/lose games to the existence of FM Nash equilibria in multi-player multi-outcome games. An extended version of our article is available online [24].

Outline. In Sect. 2, we define the core concepts, discuss known results from the literature, and present an illustrative example that requires Boolean combinations, not only conjunctions. In Sect. 3, we present the main theorem sketched above. In Sect. 4, we discuss classical (combinations of) objectives from the literature and how they fit (or not) in our framework. Then, in Sect. 5, we come back to the theorem to discuss its requirements and its relative tightness. Finally, in Sect. 6, we present several more precise results that can be obtained from our approach when restricting combinations of objectives to conjunctions and disjunctions only. Due to space constraints, some technical details are presented in Appendix A.

2 Preliminaries and example

2.1 Definitions

▶ **Definition 1.** We fix a set C, calling its elements *colors*. A C-colored *arena* O is a tuple $O = \langle V_1, V_2, E, \Gamma \rangle$, where $(V_1 \cup V_2, E)$ is a directed graph such that each vertex has at least one outgoing edge, and $\Gamma \colon V_1 \cup V_2 \to C$ is the coloring function. We assume *finite* arenas.

As we generally consider the set C of colors fixed, we suppress C-colored in the following. A *winning condition* is a set $W \subseteq C^\omega$ of infinite sequences of colors. An arena O together with a winning condition W constitutes a *game* $g = (O, W)$.

A *strategy* for Player i is a function $\sigma_i \colon C^* \times V_i \to (V_1 \cup V_2)$ satisfying that for all $w \in C^*$ and $v \in V_i$ we have $(v, \sigma_i(w, v)) \in E$. This means that a strategy selects a successor vertex based on the current vertex *owned* by the relevant player, and a finite sequence of colors.

▶ Remark. Let us comment on our use of colors. If we consider a fixed game only, we could w.l.o.g. consider each vertex to be colored by itself. However, separating vertices and colors is necessary to speak about using the same winning condition together with different arenas. Then one might expect that a strategy can take into account the history of vertices leading up to the current position, not merely the sequence of colors. However, it is straightforward to verify that players have no incentive to take these specifics into account. Moreover, the notion of hSPE defined below takes into account also histories not actually realizable in the given arena. Defining strategies on color histories is thus the most convenient approach.

A *strategy profile* is a pair of strategies (σ_1, σ_2), which we identify with the function $\sigma = \sigma_1 \cup \sigma_2 \colon C^* \times (V_1 \cup V_2) \to (V_1 \cup V_2)$. We say that Player i can deviate from (σ_1, σ_2) to (σ_1', σ_2') if $\sigma_{3-i} = \sigma'_{3-i}$. A strategy profile σ induces a function $\Sigma_\sigma \colon C^* \times (V_1 \cup V_2) \to C^\omega$ coinductively as follows: $\Sigma_\sigma(w, v)(i) = w(i)$ (where $w(i)$ represents the ith symbol of w) for $i \leq |w|$ and $\Sigma_\sigma(w, v)(k) = \Sigma_\sigma(w\Gamma(v), \sigma(w, v))(k)$ for $k = |w| + 1$. The infinite color sequence $\Sigma_\sigma(w, v)$ is the *play* starting at v with history w. For a play ρ, let $Pref(\rho)$ be the set of finite prefixes of ρ.

▶ **Definition 2.** A *hypothetical subgame-perfect equilibrium* (hSPE) is a strategy profile σ such that for all $w \in C^*$, $v \in V_1 \cup V_2$ the following implications hold: If $\Sigma_\sigma(w, v) \in W$, then $\Sigma_{\sigma'}(w, v) \in W$ for each σ' that Player 2 can deviate to from σ. If $\Sigma_\sigma(w, v) \notin W$, then $\Sigma_{\sigma''}(w, v) \notin W$ for each σ'' that Player 1 can deviate to from σ.

Informally, in an hSPE a player is winning from exactly these combinations of history and current vertex that she can win from at all. What differentiates the notion of hSPE from the usual subgame-perfect equilibria is that we take into account not just histories realizable in the arena, but also the *hypothetical* histories which could not actually have happened. Clearly, every hSPE can be domain-restricted to an SPE. Moreover, if for a winning condition W there are SPE for all arenas, there are also hSPE for all arenas. The subtlety of this notion only becomes relevant when working with restricted classes of arenas.

We focus on the existence of *finite-memory* (FM) hSPE for classes of games. An hSPE is finite-memory when the strategies σ_i, $i \in \{1, 2\}$, can be implemented as *Moore machines*, i.e., finite automata with outputs. The size of an FM strategy (and by extension of an FM hSPE) is taken to be the number of memory states of the Moore machine(s).

2.2 State of the art

We will briefly review a selection of results about combinations of classical objectives in two-player turn-based zero-sum games. As mentioned in Sect. 1, results in this area are too numerous to provide an exhaustive list here. We thus focus on prominent winning conditions and memory requirements only, and do not delve too deep into technical details.

Simple objectives. We first present the simple winning conditions.

- *Parity.* Vertices are labeled with integer priorities, and a play is winning for Player 1 iff, among the priorities seen infinitely often, the maximal one is even. This condition subsumes many simple ones such as *reachability* or *safety* with regard to a given set.
- *Muller.* Let $U_1, \ldots, U_p$ be subsets of vertices, a play is winning iff the set of vertices seen infinitely often is equal to some U_i.

- *Energy.* Vertices are labeled with integer weights representing energy consumption and/or gain. The running sum of weights along a play must stay non-negative at all times. An upper bound may also be fixed. We consider two variants of upper bounds: *battery-like* and *spill-over-like* energy conditions. In the former, any energy in excess is simply lost, while in the latter, the play is lost if the upper bound is exceeded.
- *Mean-payoff.* Vertices are labeled with integer weights, and we look at the limit of the averages over all prefixes of a play. Since this limit need not exist in general, two variants are studied, for lim sup and lim inf. We require the limit to be above a given threshold.
- *Total-payoff.* Vertices are labeled with integer weights, and we look at the limit of partial sums over prefixes. Again, two variants exist for lim sup and for lim inf. We require the limit to be above a given threshold.
- *Average-energy.* Vertices are labeled with integer weights, and we look at the limit of the averages of partial sums (i.e., the average energy level) over prefixes. Again, two variants exist for lim sup and for lim inf. We require the limit to be above a given threshold.
- *Window objectives.* Variations of mean-payoff and parity where instead of looking at the limit over an infinite play, we fix a finite window sliding over the play and we require the appropriate behavior to happen within the window at each step along the play.

Observe that all these winning conditions, along with arbitrary combinations, can be expressed in our formalism, i.e., as subsets of C^ω for an appropriate set of colors C.

Single-objective case. The situation is as follows: parity [27], energy with only a lower bound [8], mean-payoff [15], total-payoff [16] and average-energy [4] games are memoryless determined; Muller games require exponential-memory strategies for both players [14]; lower and upper bounded energy games (both variants) require pseudo-polynomial memory (in the upper bound) for both players [4]; window objectives require polynomial memory (in the window size) for both players [10, 6]. Hence, *all these winning conditions are FM determined.*

Combinations of objectives. Despite this common trait, *these objectives behave in very different ways when used in combinations*, even for very restricted ones. First, let us note that most results in the literature deal with the simplest case of conjunctions (or disjunctions) of objectives. In this setting, parity [12], Muller, energy [13, 19], and window objective [10, 6] games remain FM determined (often, with an exponential blow-up in the number of conjuncts). The same holds for conjunctions of (possibly multiple) lower-bounded energy and parity conditions [9, 13] or of a single average-energy condition with an energy one [4, 3].

On the contrary, conjunctions of mean-payoff conditions [26] or of a single mean-payoff and a single parity condition [11] require infinite-memory strategies (for Player 1). Furthermore, it follows from the undecidability of multi-dimension average-energy games [3] and multi-dimension total-payoff games [10] that unbounded memory is required for both players in these two settings. Finally, let us mention that *Boolean combinations* were studied by Velner for mean-payoff conditions, and they were proved to be undecidable and requiring infinite-memory strategies [25]. For Boolean combinations of window objectives and a strict subset of parity conditions, Bruyère et al. proved that games are exponential-memory determined [7].

The *take-home message* is that winning conditions that are similar in the single-objective case may lead to contrasting behaviors when considering combinations, even for the simple case of conjunctions. Despite all these related works, little is known about the inherent mechanisms underlying FM determinacy, and why some objectives preserve it in combinations while others do not. Our main theorem, in Sect. 3, formulates an answer to that question.

2.3 A motivating example

We define a general class of games combining Muller conditions and instances of the two variants of bounded energy games presented in Sect. 2.2.

▶ **Definition 3** (Multi-dimension bounded-energy Muller games)**.** Let $U_1, \dots, U_p$ be subsets of vertices of a finite arena, let every vertex of this arena be labeled with a tuple in $\mathbb{Z}^{n+m}$, let $b \in \mathbb{N}^{n+m}$, and let φ be a proposition from propositional logic with $p+n+m$ free variables. Player 1 wins the game iff $\varphi(x_1, \dots, x_p, y_1, \dots, y_n, z_1, \dots, z_m)$ holds, where

- x_i holds if the Muller condition induced by U_i is satisfied,
- y_i holds if the the i-th battery-like energy condition holds, using b_i as a battery-like bound and the i-th components of the labels as energy deltas,
- z_i holds if the the i-th spill-over-like energy condition holds, using b_{i+n} as a spill-over-like bound and the $(i+n)$-th components of the labels as energy deltas.

▶ **Example 4.** A reporter has two options to travel across a country and make a documentary. Either by car and by sending pictures instantly to colleagues from her newspaper, or on foot to get a deeper understanding of the country. In the latter case she would not need to send pictures instantly, but to report on her location everyday for safety reasons. She finally decides to go by car, but if at some point she ran out of gas, she would continue on foot. In these specifications we can identify one energy condition, relating to the gas for the car, and two Muller conditions: sending the pictures and reporting the location. Note that such *conditional Muller* conditions could not be simulated by one Muller condition alone.

Whereas all the ingredients of such a game, i.e., Muller, battery-like and spill-over-like energy conditions, are rather well understood, little is known about arbitrary combinations thereof. To prove that they are all FM determined we shall prove a more general result implying that combining a well-behaved winning condition (e.g., Muller) with conditions expressible by finite automata (e.g., bounded energy) yields a well-behaved condition again.

3 Main theorem

3.1 The class of arenas

We will formulate our main theorem for games built with arenas from some class $\mathcal{O}$ that is closed under certain operations. The class of all arenas is trivially closed under these operations, and in most applications it will be the most reasonable choice for $\mathcal{O}$. Our theorem allows for cases, however, where to ensure the premise we exploit some interactions between the specific structure of the arenas considered and the winning condition. The two operations we consider are restrictions of the arena, and products with finite automata.

▶ **Definition 5** (Restriction of an arena)**.** Let $O = \langle V_1, V_2, E, \Gamma \rangle$ be an arena, and let $S \subseteq V_1 \cup V_2$ be such that $vE \cap S \neq \emptyset$ for all $v \in S$, where $vE := \{u \in V_1 \cup V_2 \mid (v,u) \in E\}$. The arena $O\restriction_S := \langle V_1 \cap S, V_2 \cap S, E \cap S^2, \Gamma\restriction_S \rangle$ is called the restriction of O to S.

▶ **Definition 6** (Product arena-automaton)**.** Let $O = \langle V_1, V_2, E, \Gamma \rangle$ be an arena and let $\mathcal{A} = (C, Q, q_0, F, \Delta)$ be a finite automaton over the alphabet C of colors. The product $O \times \mathcal{A}$ is the arena $\langle V_1 \times Q, V_2 \times Q, E', \Gamma' \rangle$ where $\Gamma'(v,q) := \Gamma(v)$, and where $((v,q),(v',q')) \in E'$ iff $(v,v') \in E$ and $(q, \Gamma(v), q') \in \Delta$.

▶ **Lemma 7.** *Let $O = \langle V_1, V_2, E, \Gamma \rangle$ be an arena and let $\mathcal{A} = (C, Q, q_0, F, \Delta)$ and $\mathcal{A}' = (C, Q', q_0', F', \Delta')$ be finite automata.*

1. *(Idempotency)* $(O\restriction_S)\restriction_{S'} = O\restriction_{(S\cap S')}$
2. *(Associativity)* $(O \times \mathcal{A}) \times \mathcal{A}' \cong O \times (\mathcal{A} \times \mathcal{A}')$.
3. *(Restricted commutativity)* $O\restriction_S \times \mathcal{A} = (O \times \mathcal{A})\restriction_{S\times Q}$

▶ **Corollary 8.** *Let the arena O' be obtainable from the arena O by finitely many operations chosen from products with automata and restrictions. Then there is an automaton $\mathcal{A}$ and a suitable set $S \subseteq V \times Q$ such that $O' = (O \times \mathcal{A})\restriction_S$, where Q are the states of $\mathcal{A}$.*

So if we start with some class of arenas, and then wish to close it under products with automata and restriction, we can just first consider all products of the original arenas with finite automata, and then the restrictions of these; and we obtain the desired class.

3.2 Regularly-predictable games

Our theorem requires, informally spoken, that Player 1 always knows whether winning is still possible for him given the history so far and given the current vertex. *Knowing* here means (since we want FM strategies) the existence of a finite automaton producing the answer.

▶ **Definition 9.** A game g is *regularly-predictable* if for all vertices $v \in V$ of g there exists a finite automaton $\mathcal{A}_v$ that reads an initial color sequence and accept it iff Player 1 has a winning strategy from v after this sequence. A winning condition is regularly-predictable if all games using it are regularly-predictable.

Many popular winning conditions are prefix-independent, in which case it does not depend on the history at all whether a player can win from some vertex. Hence, these are trivially regularly-predictable. Reachability and safety conditions, on the other hand, are not prefix-independent, but still regularly-predictable.

At first glance, one may think that FM determinacy implies regular-predictability. This is false, as witnessed by the following proposition, where K_2 is the two-clique with self-loops.

▶ **Proposition 10.** *Energy games with only a lower bound are not regularly-predictable.*

Proof. Consider a one-player game on K_2, with one vertex giving $+1$ energy, and the other giving -1. The winning condition is that the current energy level stays non-negative. The player can win from every history where the energy level had not already been negative before. However, deciding this amounts to deciding the language of all binary words containing at least as many 1s and 0s, a typical example of a non-regular language. Hence such sequences cannot be recognized by a finite automaton. ◀

The proposition above together with the example below shows that regular-predictability and existence of FM hSPE are of incomparable strength.

▶ **Example 11.** Let W be the winning condition for Player 1 that consists of the non-regular sequences in $\{0,1\}^\omega$: it is prefix-independent, so every game is regularly-predictable. Yet, the game over K_2 (with self-loops) involving colors 0 and 1 and where only Player 1 plays is winnable but not via FM strategies (as it would contradict non-regularity of the sequences).

3.3 Boolean combination

The following definition suggests how we will combine the well-behaved regularly-predictable winning conditions to conditions definable by regular languages.

▶ **Definition 12** (Regular combination of winning conditions)**.** Let $\mathcal{W}$ be a class of subsets of C^ω that is closed under Boolean combination. The combination of $\mathcal{W}$ with l regular languages, denoted $R_l(\mathcal{W})$, is defined as follows. Let $\varphi(w_1, \dots, w_k, r_1, \dots, r_l)$ be a propositional-logic formula with $k+l$ variables. Let $W_1, \dots, W_k \in \mathcal{W}$. For $1 \leq i \leq l$ let L_i be a regular language over C. Then the winning condition $\{\rho \in C^\omega \mid \varphi(\rho \in W_1, \dots, \rho \in W_k, Pref(\rho) \cap L_1 = \emptyset, \dots, Pref(\rho) \cap L_l = \emptyset)\}$ is in $R_l(\mathcal{W})$.

Intuitively, we allow for any Boolean combination between variables that either represent satisfaction of an objective from $\mathcal{W}$ (which we will require to be regularly-predictable) or represent the existence of a prefix of the play within a given regular language. Note that it is not restrictive for $\mathcal{W}$ to be closed under Boolean combination: we present our framework as such to allow for very general combinations (e.g., in the toy example from Section 2.3), but, for less amenable winning conditions, one can still take $\mathcal{W}$ to have four elements: a winning condition of interest, its complement, and the empty/universal winning conditions.

3.4 Regular combinations preserve FM determinacy

We are now able to state our main result. We only sketch its proof due to space constraints: full details are provided in Appendix A.

▶ **Theorem 13.**

- *Let $\mathcal{O}$ be a class of arenas that is closed under product with finite automata and under simple restriction.*
- *Let $\mathcal{W}$ be a class of winning conditions that is closed under Boolean combination.*
- *Let all games in $\mathcal{O} \times \mathcal{W}$ be regularly-predictable and have FM hSPE.*

Then all games in $\mathcal{O} \times R_l(\mathcal{W})$ are regularly-predictable and have FM hSPE. (for all $l \in \mathbb{N}$)

Proof idea. The proof of Theorem 13 proceeds by induction on l from Definition 12. The basic idea of the induction step is to do the product of the arena of the game by a finite automaton accepting the language L_l, and then to partition the vertices of the new game into three regions: a region where $Pref(\rho) \cap L_l = \emptyset$ has been falsified, a region where the players can force the play[4] to reach the part of the first region that they like, and a third region restricted to the remaining vertices, where $Pref(\rho) \cap L_l = \emptyset$ holds, and no player has an incentive to leave the third region.

For the first and third regions, a suitable strategy profile will be provided by induction hypothesis, for the second region reachability analysis will do. These profiles together will be translated back into one single profile for the original game. A problem arises: a possible history in the original game may correspond in the product game to a history starting in the third region, going down the second region and coming back to the third. Thus the induced color sequence may fall out of the domain of the profile for the third region. Therefore we need to cope with "impossible histories", and use hSPE rather than the more familiar SPE.

A second problem arises because of the aforementioned reachability analysis. To show this, let us slightly detail the basic idea of the proof. Let $\mathcal{A}$ be a finite automaton recognizing the words with a prefix in L_l. The product $O \times \mathcal{A}$ looks like Fig. 1, where T refers to the vertices where $Pref(\rho) \cap L_l = \emptyset$ has not (yet) been falsified. The bottom part corresponds to the product of O with the final states of $\mathcal{A}$. Fig. 2 invokes the induction hypothesis to get a suitable profile $\sigma_\perp$ for this part of the arena. Then in Fig. 3 we would like to split T

[4] Slight abuse of notation as plays are formally defined as sequences of colors only: for the sake of readability, we sometimes use a play to refer to the corresponding sequence of vertices in the graph.

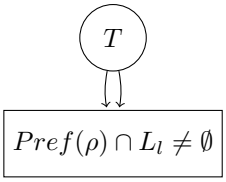


Figure 1 The product arena $O \times \mathcal{A}$.

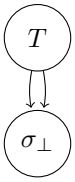


Figure 2 Profile $\sigma_\perp$ provided by induction.

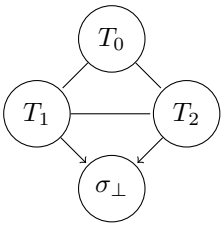


Figure 3 Splitting T via simplistic reachability analysis.

into two regions: first, a region $T_1 \cup T_2$, where Player i can force (as suggested by the arrow tips) every play starting in T_i to reach the region she likes in the bottom component; and second, a region T_0 that no player wants to leave. The problem is that whether a player can win from a vertex in the bottom component depends on how the vertex was reached, so T_1 and T_2 cannot be simply defined as subsets of vertices in $O \times \mathcal{A}$. It is possible to reduce the problem to classical reachability, though: the proof in appendix considers the product of the original arena O with *all* the automata provided by the regular-predictability assumption (as well as with other automata derived from L_l). ◀

Prefix-independence. Many popular winning conditions are prefix-independent, in which case regular-predictability comes for free, and the notions of SPE and hSPE coincide. We state this special case explicitly, and also include an upper bound for the required memory.

▶ **Corollary 14.** *Let $\mathcal{O}$ be a class of arenas that is closed under product with finite automata and under simple restriction. Let $\mathcal{W}$ be a class of prefix-independent winning conditions that is closed under Boolean combination. If all games in $\mathcal{O} \times \mathcal{W}$ have FM SPE, so do all games in $\mathcal{O} \times R_l(\mathcal{W})$.*

Furthermore, let $f(l, n)$ be the memory size required to implement the optimal strategies for n-vertex games in $R_l(\mathcal{W})$ if the regular languages are recognized by automata with m states each. Then $f(l, n) \leq m^n \cdot f(l-1, n) \cdot f(l-1, nm^n)$.

Note that while Corollary 14 provides bounds on the memory requirements, these bounds are non-elementary. Such horrible bounds are unfortunately inevitable given the generality of the framework. In Sect. 6, we provide results for specific Boolean combinations that yield much more reasonable memory bounds.

Let us mention that very simple cases of combinations already lead to large lower bounds in terms of memory. For example, exponential memory is required for conjunctions of fully-bounded energy conditions, while they are actually all covered by the regular languages part of $R_l(\mathcal{W})$ [3]. The same holds for conjunctions of window objectives [10, 6].

Back to the example. We can now apply our theorem to the example of Sect. 2.3.

▶ **Corollary 15.** *Multi-dimension bounded-energy Muller games have FM SPE (aka combination of optimal strategies).*

Proof. Let $\mathcal{O}$ be the class of the finite arenas, which is clearly closed under simple restriction and product by finite automaton. Let $\mathcal{W}$ be the class of the (prefix-independent) Muller winning conditions, which is closed under Boolean combination. All games in $\mathcal{O} \times \mathcal{W}$ have FM SPE [14], and so do all games in $\mathcal{O} \times R_l(\mathcal{W})$ by Corollary 14.

To show that this applies to multi-dimension bounded-energy Muller games, it suffices to see that fully-bounded energy conditions can be expressed as regular languages, which is trivial since the sum of weights must be constrained within the two bounds at all times, hence can only take a bounded number of integer values. ◀

4 Applications

Let us compare the hypotheses and concepts used in Theorem 13 with the classical (combinations of) winning conditions from the literature, already discussed in Sect. 2.2, to get a better grasp of its applicability. Interesting winning conditions for our theorem fall in two categories: regular languages and regularly-predictable conditions. First, observe that any condition that can be defined as a regular language is trivially regularly-predictable. (The latter is a more general notion.) Hence, we first discuss the regular languages.

Regular languages. Among the simplest conditions that can be recognized through finite automata (hence expressed as regular languages) lie *reachability* and *safety* conditions. Indeed, as mentioned in Sect. 1, regular languages are essentially a compact way to represent *safety-like* winning conditions. For example, fully-bounded energy conditions (both battery-like and spill-over-like) can also be encoded through finite automata whose size depends on the upper bound. Window objectives (both for mean-payoff and parity), thanks to their finite window mechanism, can also be represented as regular languages. All the other objectives discussed in Sect. 2.2 cannot.

Regularly-predictable conditions. Recall that for Theorem 13, we require the class $\mathcal{W}$ to be closed under Boolean combination and such that winning conditions are regularly-predictable and admit FM hSPE. Let us review the classical objectives.

- *Regular languages.* As stated above, conditions expressed as regular languages can be used. Hence, this easily permits to rediscover FM determinacy results for multi-dimension fully-bounded energy games [2, 4, 3] or conjunctions of window objectives [10, 6], and even extend them to full Boolean combinations.
- *Parity and Muller.* Any combination of such conditions can be expressed in the closed class. Furthermore they are trivially regularly-predictable (because they are prefix-independent) and admit FM hSPE. Hence, these conditions can be mixed in any Boolean combination with regular languages and retain FM determinacy. This lets us rediscover FM determinacy results for generalized parity games [12], or combinations of parity conditions with window conditions [7], and extend them to full Boolean combinations.
- *Mean-payoff.* The mean-payoff condition is regularly-predictable (as it is prefix-independent) and admits FM hSPE. Unfortunately, this does not hold for Boolean combinations of mean-payoff [26, 25]. Still, one can take $\mathcal{W}$ as the trivial class containing one mean-payoff condition and its complement, and use it in Boolean combinations with regular languages.
- *Average-energy, total-payoff and energy with no upper bound.* These three conditions are not regularly-predictable as one needs to be able to store an arbitrarily large sum of weights in memory to decide if Player 1 can win from a given prefix. Hence our theorem cannot be applied to these conditions.

Theorem applicability. Observe that our theorem cannot be applied to Boolean combinations of mean-payoff, average-energy and total-payoff, or to combinations of mean-payoff and parity (two regularly-predictable conditions but which cannot be put in the same closed class

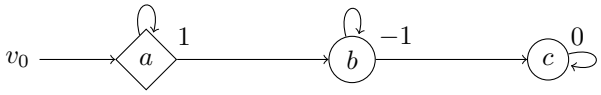


Figure 4 Average-energy games with lower-bounded energy are FM determined but do not admit FM SPE. Vertices are colored by integer weights.

$\mathcal{W}$), four cases in which *indeed FM determinacy is not preserved* [26, 25, 4, 10, 11]. On the other hand, we recover many known results from the literature [2, 4, 3, 10, 6, 12, 7] and are able to extend them to more general combinations (or to completely novel ones).

It remains to discuss *corner cases*: combinations that are known to preserve FM determinacy but are not covered by our theorem. We are aware of three cases from the literature, all involving the energy condition without an upper bound: (a) conjunctions of energy conditions [13, 19], (b) conjunctions of energy and parity conditions [9, 13], (c) conjunctions of energy and a single average-energy condition [3]. It is very interesting to spend a moment on these corner cases. Indeed, the ad-hoc techniques used to prove FM determinacy in all three cases intuitively rely on proving equivalence with games where the energy condition can be bounded *both* from below *and from above*, for a sufficiently large bound. Yet, we know that such fully-bounded energy conditions define regular languages. Hence, for cases (a) and (b) we actually retrieve applicability of our theorem, leaving case (c) as the only case, to our knowledge, of preservation of FM determinacy in the literature which is not covered by Theorem 13. This is because the average-energy condition is not regularly-predictable (as one can see in [4, 3], it behaves rather oddly in comparison to all other classical objectives).

FM determinacy vs. FM hSPE. We can say more on case (c), and for that we want to highlight once again that the result we obtain in Theorem 13 deals with a stronger concept than FM determinacy, namely the existence of FM (h)SPE. As seen above, these two notions do coincide in virtually all cases studied in the literature. Now, case (c) is actually the only setting, to our knowledge, where they do not, as proved in the following example.

▶ **Example 16.** Consider the arena in Fig. 4, colored by integers: vertex a has weight 1, b weight -1 and c weight 0. The objective of Player 1 (circle) is to have the average-energy (AE, limit of the average energy level) less than or equal to zero while keeping the energy level (EL) non-negative at all time. Such games are FM determined [3], and in this case, Player 2 (diamond) has a trivial strategy to win from a: looping forever. He also wins from b (the energy directly drops below zero), and Player 1 wins from c.

Now, consider SPE. Observe that when in vertex c, the EL does not change anymore so the AE is actually equal to the EL when you reach c. Thus, Player 1 can win if he reaches c with an EL of zero (and it is the only way). In an SPE, we need to consider all possible histories. Imagine an history $w = a^n$ for some $n \geq 1$: Player 1 wins by looping exactly n times in b before reaching c. This defines a subgame-perfect strategy for Player 1 which consists in looping in b for as many times as Player 2 looped in a. Clearly, this SPE requires infinite memory. We thus have an FM determined game where no FM SPE exists.

5 Discussion of the requirements

We discuss the requirements of Theorem 13 and explore whether improvement seems plausible.

Arenas. First, we inspect the conditions pertaining to the arenas. Typically all arenas are considered, and the class of all arenas trivially satisfies the closure properties we demand. These properties are only relevant if we need to use specific subsets of the arenas to prove the required properties of the games. To ask that the class of arenas we consider is closed under products with finite automata is a very mild condition. Apart from arenas of bounded size, we would expect all naturally occurring classes of arenas to have this property.

Requiring closure under simple restriction is more restrictive (e.g., the games we consider in the upcoming Example 17 are not closed under simple restriction). To see that this is necessary, let O' be a simple restriction of an arena O such that the game played on O is FM determined, but the one played on O' is not. W.l.o.g., assume that O' has no deadlock. We can obtain a new winning condition for a game played on O by combining the original condition with a regular language expressing informally that, if the game ever reaches a vertex in $O \setminus O'$, the last player to move loses, and otherwise, the winner is the winner of the original winning condition. The resulting game is not FM determined, thus witnessing the need for our requirement of closure under simple restriction.

Furthermore, as soon as we are demanding that our arenas are closed under products with automata and simple restriction, we see that we need to require the existence of FM SPE rather than merely FM determinacy (up to uniformity). The reason is that by taking a suitable combination of products and restriction, we generate an arena that first produces a predetermined finite color sequence before starting the original game.

Regular languages. Theorem 13 is tight in the sense that it fails if making the conjunction of the universal winning condition and a condition derived from any irregular language. Indeed, Example 11 already showed that FM hSPE may not exist for such conditions.

FM determinacy vs. FM hSPE. Being able to win using bounded finite memory, and being able to decide who wins from a given history with a finite automaton (i.e., regular-predictability) together do not suffice to imply the existence of an FM subgame-perfect strategy, as observed in the next example.

▶ **Example 17.** Consider an arena with colors $0, 1, \alpha, \beta$, where a vertex has an α-successor iff it has a β-successor, and each such vertex is controlled by Player 1 (let us call them $\alpha\beta$-vertices). Player 1 wins any play ρ that has a prefix of the form $w\alpha$ with $w \in \{0,1\}^*$ and w has the same number of 0 and 1s, and any play ρ that has a prefix of the form $w\beta$ with $w \in \{0,1\}^*$ and w has different numbers of 0 and 1s. Hence, for Player 1 to win, it suffices to reach an $\alpha\beta$-vertex and pick the α (resp. β) successor if the play contains an equal (resp. a different) number of 0 and 1s. So there is a simple finite automaton deciding from which histories Player 1 can win, and linear memory suffices for Player 1 to win from any winnable history, but Player 1 has no FM subgame-perfect strategy.

In Sect. 4, we also discussed the case of conjunctions of energy and a single average-energy condition [3], which is also FM determined, but do not have FM hSPE, for similar reasons.

Note that the games above are not closed by simple restriction. As a further example, we show that the condition on subgame-perfect strategies is not dispensable entirely.

▶ **Example 18.** Consider the arena in Fig. 5 where Player 1 controls circle vertices and Player 2 controls diamond vertices. Let W be defined such that Player 1 wins iff either the play ends with c^ω, or b occurs an even number of times and the play ends with $deedeeedeeee\ldots$, or b occurs an odd number of times and the play ends with e^ω. Player 1 has an FM winning strategy: play towards b first; if the game returns to a, play d and then loop at e. Player 1

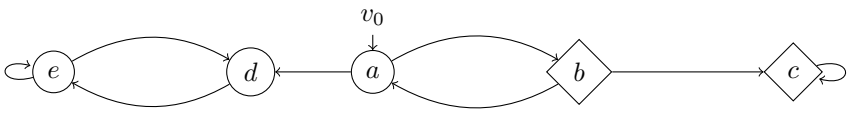


Figure 5 FM SPEs are needed for transfer in combinations, not only FM determinacy.

also has a subgame-perfect strategy, but no FM subgame-perfect strategy (as it needs to react to all histories, hence also the ones where b happens an even number of times).

If we take the conjunction of W with avoiding the regular language of words containing c, then in the resulting game, Player 1 still wins, but he has no FM winning strategy.

6 Specific results for conjunctions and disjunctions

The ability of our main theorem to deal with arbitrary Boolean formulae is often not needed. In fact, as discussed in Sect. 2.2, in the literature on multi-dimension games, it is very common to work only with conjunctions on the winning conditions of Player 1 (which dually means using disjunctions for Player 2). In this section, we give some direct constructions for these cases, which have different requirements or conclusions from our main theorem – and in particular, come with much more reasonable bounds than provided by Corollary 14.

As an outline, we give a brief account of the main constructions here. (i) For simple disjunctions for Player 2, we obtain FM determinacy results without the regularity assumption on the combined language-based condition. (ii) For simple conjunctions for Player 1, we obtain FM determinacy results and better memory bounds without requiring regular-predictability (but requiring regularity of the language). As corollary, we regain known bounds on fully-bounded energy games. (iii) For the case of subgame-perfect strategies (not required in (i) and (ii)), we obtain better memory bounds. The interest of these side results, apart from improved bounds for still general classes of games, is to illustrate how the different hypotheses of Theorem 13 interact and how restrictions on some dimensions of the problem permits to be more general on other dimensions.

The reason why we cannot obtain a general statement as in our main theorem by combining the lemmata in this section is that their conclusions do not match their requirements. We can thus not apply them in an iterative fashion.

To formulate some of the results, we recall the notion of *future game* from [23].

▶ **Definition 19** (Future games). Let g be a game with winning condition W, and let $w \in C^*$. The future game g^w is derived from g by replacing W with $\{\rho \in C^\omega \mid w \cdot \rho \in W\}$. If σ_i is a strategy in g, let $\sigma_i^w(w', v) \coloneqq \sigma_i(ww', v)$ define another strategy for the same player.

Our first lemma consider the case of disjunctions for Player 2, dropping the regularity assumption with regard to the language-based condition.

▶ **Lemma 20.** *Consider a game such that if Player 2 can win g or its future games, he can win by using finite memory. Let L be a language over the alphabet C, and let us derive g_L from g by replacing W with W_L such that $\rho \in W_L$ iff $\rho \in W \vee Pref(\rho) \cap L = \emptyset$. If Player 2 wins g_L, he has an FM winning strategy.*

Proof. Among the histories h in g whose colors are in L, let $\mathcal{H}_L$ contain the minimal ones for the prefix relation. Let $\mathcal{H}_2$ be the histories h in $\mathcal{H}_L$ from where Player 2 wins the future game of g (and therefore also of g_L) after h. For a strategy σ, let $\mathcal{H}(\sigma)$ denote the set of histories compatible with σ, and let $[\mathcal{H}(\sigma)]$ denote the set of plays compatible with σ.

Let σ_2 be a winning strategy for Player 2, so every play in $[\mathcal{H}(\sigma_2)]$ has a prefix in $\mathcal{H}_2$, and we can define T as the subtree (a subset that is a tree) of $\mathcal{H}(\sigma_2)$ whose maximal paths are all in $\mathcal{H}_2$. By König's Lemma T is finite. For each $h \in \mathcal{H}_2 \cap T$ let σ_2^h be an FM strategy making Player 2 win the future game of g after h. The following FM strategy makes Player 2 win g_L: if $h \in T \setminus \mathcal{H}_2$ go to some vertex $v \in V$ such that $hv \in T$; if h has $h_1 \in \mathcal{H}_2$ as a prefix, play as prescribed by $\sigma_2^{h_1}$. ◀

We now turn to conjunctions for Player 1, assuming regularity of the language, but dropping the regular-predictability assumption for the winning condition. For every finite automaton $\mathcal{A}$ let $L_{\mathcal{A}}$ be the words accepted by $\mathcal{A}$.

▶ **Lemma 21.** *Let $g = (O, W)$ be a game such that whenever O' is a simple restriction of a product $O \times \mathcal{A}$ for some finite automaton $\mathcal{A}$, then (O', W) is determined via strategies using finite memory (of size $m(n)$, where n is the number of vertices in O'). Let $\mathcal{A}$ be a finite automaton over the alphabet C, and let us derive $g_{\mathcal{A}}$ from g by replacing W with $W_{\mathcal{A}}$ such that $\rho \in W_{\mathcal{A}}$ iff $\rho \in W \wedge Pref(\rho) \cap L_{\mathcal{A}} = \emptyset$. Then $g_{\mathcal{A}}$ is determined via strategies using finite memory (of size $|Q| \cdot m(|V| \cdot |Q|)$, where Q are the states of $\mathcal{A}$).*

Proof. Let $\mathcal{A} = (E, Q, q^0, F, \Delta)$ and let $g' \coloneqq ((O \times \mathcal{A}), W)$. Let $S \subseteq V \times Q$ be the vertices of g' from where Player 2 cannot force the play to reach $V \times F$. Let us make a case distinction. First case, (v_0, q^0) is not in S. So in g' Player 2 can win by playing positionally. Such a strategy yields a strategy in g ($g_{\mathcal{A}}$) that only needs memory to run $\mathcal{A}$, in order to know the current state in Q. For this a memory of size $|Q|$ suffices.

Second case, $(v_0, q^0) \in S$. By assumption, the simple restriction (S, W) is determined via strategies using finite memory (of size $m(|V| \cdot |Q|)$). If Player 1 has a winning strategy for (S, W), he can use it together with $\mathcal{A}$ to play in $g_{\mathcal{A}}$. If Player 2 has a winning strategy, he can do the same until the play, seen as a play in g', leaves S; and then he can play as in the first case. ◀

The next example shows that the regularity assumption of $L_{\mathcal{A}}$ in Lemma 21 is not dispensable. We will use the language made of the histories with positive energy levels at every prefix, which is not regular.

▶ **Example 22.** We consider games where the vertices are colored by $\{a, b\}$ and where Player 1 wins the plays with colors wa^ω for all $w \in \{a, b\}^*$ and the plays with colors $babbaa \dots b^n a^n \dots$. Such games are FM determined, but their conjunction with a lower-bounded energy condition is not.

Proof. In order to win, Player 1 needs to be able to force arbitrarily long color sequences of a's. But if he can do that, he can force eventually constant a even by a positional strategy. Likewise, for Player 2 it suffices to play a positional strategy preventing eventually constant a to win all games he can win. Hence, the base game is memoryless determined.

To see that the energy version is not FM determined, we consider a one-player game on the 2-clique K_2 (with added self-loops), such that both states s and t belong to Player 1. Let s, the initial vertex, have color b and weight 1, and t have color a and weight -1. Player 1 can win by playing according to the colors $babbaabbb \dots$, which keeps the energy non-negative, but requires infinite memory to do so. However, any FM strategy winning the underlying game has to produce a color sequence of the form wa^ω, which will cause the energy to diverge to $-\infty$, thus falsifying the energy condition. ◀

Next, we consider simple disjunctions for Player 1 and obtain improved bounds.

▶ **Lemma 23.** *Let $g = (O, W)$ be a regularly-predictable game such that whenever O' is a simple restriction of a product $O \times \mathcal{A}$ for some finite automaton $\mathcal{A}$, then (O', W) is determined, and whenever Player 1 can win, he can do so via strategies using finite memory (of size $m(n)$, where n is the number of vertices in O').*

Let $\mathcal{A}$ be a finite automaton over the alphabet C, and let us derive $g_{\mathcal{A}}$ from g by replacing W with $W_{\mathcal{A}}$ such that $\rho \in W_{\mathcal{A}}$ iff $\rho \in W \vee Pref(\rho) \cap L_{\mathcal{A}} = \emptyset$. If Player 1 has a winning strategy in $g_{\mathcal{A}}$, he has one using finite memory (of size $|Q| \cdot |Q_g| \cdot m(|V| \cdot |Q| \cdot |Q_g|)$), where Q and Q_g are the states of $\mathcal{A}$ and of a finite automaton witnessing regular-predictability of g.

Proof. Let a finite automaton $\mathcal{A}_g$ accept exactly the histories that correspond to the future games won by Player 2. Let $\mathcal{A} = (E, Q, q^0, F, \Delta)$, let $\mathcal{A}_g = (E, Q_g, q_g^0, F_g, \Delta_g)$, and let $g' \coloneqq g \times \mathcal{A} \times \mathcal{A}_g$. Let $S \subseteq V \times Q \times Q_g$ be the vertices of g' from where Player 2 cannot force the play to reach $V \times F \times F_g$. Let us make a case distinction.

First case, (v_0, q^0, q_g^0) is not in S, so in g' Player 2 can reach $V \times F \times F_g$ and win, by playing in some way.

Second case, $(v_0, q^0, q_g^0) \in S$. If a finite history h in $g' \mid_S$ reaches $V \times F \times Q_g$, it must be in $V \times F \times (Q_g \setminus F_g)$, by definition of S. So Player 1 can win after h in g', and therefore in the simple restriction $g' \mid_S$ too. By assumption he can do so *via* a strategy using finite memory (of size $m(|V| \cdot |Q| \cdot |Q_g|)$). He can use the same strategy to win $g_{\mathcal{A}}$, but he needs more memory to run $\mathcal{A}$ and $\mathcal{A}_g$ in parallel. ◀

Since the assumptions from Lemma 23 are stronger than those from Lemma 20, we also find that Player 2 has an FM winning strategy if he can win. However, even under the assumptions from Lemma 23, the memory required by Player 2 might not be uniformly bounded. We can obtain bounds for the memory required by Player 2 (Lemma 25) by considering subgame-perfect strategies.

▶ **Lemma 24.** *Let Player 1 be able to win g and all its future games that he wins by some FM strategy (using memory of size k). Moreover, for each FM strategy of size k, let there be a finite automaton (of size l) accepting exactly those histories won by that strategy. Then Player 1 has an FM subgame-perfect strategy of size $(2l \cdot k)^{((k|V|)^{k|V|})}$.*

Proof. There are $(k|V|)^{k|V|}$ FM strategies of size k. For each of them we keep track of its current state, and run the automaton deciding whether it is currently winning (using l bits). We always play according to some strategy, and we change it only once it is no longer winning, and another strategy is identified as winning from the current history. ◀

▶ **Lemma 25.** *Let g be a regularly-predictable game such that Player 2 has a subgame-perfect strategy of size m. Let $\mathcal{A}$ be a finite automaton over the alphabet C. Let us derive $g_{\mathcal{A}}$ from g by replacing W with $W_{\mathcal{A}}$ such that $\rho \in W_{\mathcal{A}}$ iff $\rho \in W \vee Pref(\rho) \cap L_{\mathcal{A}} = \emptyset$. If Player 2 wins $g_{\mathcal{A}}$, he has a winning strategy of size $l|\mathcal{A}| + m$, where some automaton of size l witnesses regular-predictability of g.*

Proof. We can combine the automaton deciding who wins a history and $\mathcal{A}$ into one automaton of size $l|\mathcal{A}|$that accepts the intersection of these two languages. By considering the expansion by this automaton, we see that simulating it suffices for Player 2 to force an accepted history, if he can do so. Now reaching such a history and then switching to the subgame-perfect FM strategy of size m wins $g_{\mathcal{A}}$ if this is possible at all. ◀

We briefly return to the two versions of fully-bounded energy conditions from [2] discussed in Sect. 2.2: battery-like and spill-over-like variants. Let $E_{\max}$ be the energy upper bound

in both cases. Since both conditions can be simulated by a finite automaton with $\mathcal{O}(E_{\max})$ states, Lemma 21 implies the following corollary. Contrast this to the requirement of memory size $|V| \cdot d \cdot W$ (where d is the number of priorities and W the largest energy weight) for unbounded-energy parity games from [9].

▶ **Corollary 26.** *Battery-like energy parity games and spill-over-like energy parity games are determined via strategies using $\mathcal{O}(E_{\max})$ memory states, where $E_{\max}$ is the energy upper bound.*

References

1 Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. Graph Games and Reactive Synthesis. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking.*, pages 921–962. Springer, 2018. doi:10.1007/978-3-319-10575-8_27.

2 Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, Nicolas Markey, and Jiří Srba. Infinite Runs in Weighted Timed Automata with Energy Constraints. In Franck Cassez and Claude Jard, editors, *Formal Modeling and Analysis of Timed Systems*, volume 5215 of *Lecture Notes in Computer Science*, pages 33–47. Springer Berlin Heidelberg, 2008. doi:10.1007/978-3-540-85778-5_4.

3 Patricia Bouyer, Piotr Hofman, Nicolas Markey, Mickael Randour, and Martin Zimmermann. Bounding Average-Energy Games. In Javier Esparza and Andrzej S. Murawski, editors, *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10203 of *Lecture Notes in Computer Science*, pages 179–195, 2017. doi:10.1007/978-3-662-54458-7_11.

4 Patricia Bouyer, Nicolas Markey, Mickael Randour, Kim G. Larsen, and Simon Laursen. Average-energy games. *Acta Inf.*, 55(2):91–127, 2018. doi:10.1007/s00236-016-0274-1.

5 Romain Brenguier, Lorenzo Clemente, Paul Hunter, Guillermo A. Pérez, Mickael Randour, Jean-François Raskin, Ocan Sankur, and Mathieu Sassolas. Non-Zero Sum Games for Reactive Synthesis. In Adrian-Horia Dediu, Jan Janousek, Carlos Martín-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications - 10th International Conference, LATA 2016, Prague, Czech Republic, March 14-18, 2016, Proceedings*, volume 9618 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2016. doi:10.1007/978-3-319-30000-9_1.

6 Véronique Bruyère, Quentin Hautem, and Mickael Randour. Window parity games: an alternative approach toward parity games with time bounds. In Domenico Cantone and Giorgio Delzanno, editors, *Proceedings of the Seventh International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2016, Catania, Italy, 14-16 September 2016.*, volume 226 of *EPTCS*, pages 135–148, 2016. doi:10.4204/EPTCS.226.10.

7 Véronique Bruyère, Quentin Hautem, and Jean-François Raskin. On the Complexity of Heterogeneous Multidimensional Games. In Josée Desharnais and Radha Jagadeesan, editors, *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, volume 59 of *LIPIcs*, pages 11:1–11:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.CONCUR.2016.11.

8 Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Mariëlle Stoelinga. Resource Interfaces. In Rajeev Alur and Insup Lee, editors, *EMSOFT*, volume 2855 of *Lecture Notes in Computer Science*, pages 117–133. Springer, 2003. doi:10.1007/978-3-540-45212-6_9.

9 Krishnendu Chatterjee and Laurent Doyen. Energy parity games. *Theor. Comput. Sci.*, 458:49–60, 2012. doi:10.1016/j.tcs.2012.07.038.

10 Krishnendu Chatterjee, Laurent Doyen, Mickael Randour, and Jean-François Raskin. Looking at mean-payoff and total-payoff through windows. *Inf. Comput.*, 242:25–52, 2015. doi:10.1016/j.ic.2015.03.010.

11 Krishnendu Chatterjee, Thomas A. Henzinger, and Marcin Jurdzinski. Mean-Payoff Parity Games. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 178–187. IEEE Computer Society, 2005. doi:10.1109/LICS.2005.26.

12 Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Generalized Parity Games. In Helmut Seidl, editor, *Foundations of Software Science and Computational Structures, 10th International Conference, FOSSACS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007, Braga, Portugal, March 24-April 1, 2007, Proceedings*, volume 4423 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2007. doi:10.1007/978-3-540-71389-0_12.

13 Krishnendu Chatterjee, Mickael Randour, and Jean-François Raskin. Strategy synthesis for multi-dimensional quantitative objectives. *Acta Inf.*, 51(3-4):129–163, 2014. doi:10.1007/s00236-013-0182-6.

14 Stefan Dziembowski, Marcin Jurdzinski, and Igor Walukiewicz. How Much Memory is Needed to Win Infinite Games? In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29 - July 2, 1997*, pages 99–110. IEEE Computer Society, 1997. doi:10.1109/LICS.1997.614939.

15 Andrzej Ehrenfeucht and Jan Mycielski. Positional Strategies for Mean Payoff Games. *Int. Journal of Game Theory*, 8(2):109–113, 1979.

16 Hugo Gimbert and Wieslaw Zielonka. When Can You Play Positionally? In Jirí Fiala, Václav Koubek, and Jan Kratochvíl, editors, *Mathematical Foundations of Computer Science 2004, 29th International Symposium, MFCS 2004, Prague, Czech Republic, August 22-27, 2004, Proceedings*, volume 3153 of *Lecture Notes in Computer Science*, pages 686–697. Springer, 2004. doi:10.1007/978-3-540-28629-5_53.

17 Hugo Gimbert and Wieslaw Zielonka. Games Where You Can Play Optimally Without Any Memory. In Martín Abadi and Luca de Alfaro, editors, *CONCUR 2005 - Concurrency Theory, 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005, Proceedings*, volume 3653 of *Lecture Notes in Computer Science*, pages 428–442. Springer, 2005. doi:10.1007/11539452_33.

18 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-36387-4.

19 Marcin Jurdzinski, Ranko Lazic, and Sylvain Schmitz. Fixed-Dimensional Energy Games are in Pseudo-Polynomial Time. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 260–272. Springer, 2015. doi:10.1007/978-3-662-47666-6_21.

20 Eryk Kopczynski. Half-Positional Determinacy of Infinite Games. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 336–347. Springer, 2006. doi:10.1007/11787006_29.

21 Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.

22 Mickael Randour. Automated Synthesis of Reliable and Efficient Systems Through Game Theory: A Case Study. In *Proc. of ECCS 2012*, Springer Proceedings in Complexity XVII, pages 731–738. Springer, 2013. doi:10.1007/978-3-319-00395-5_90.

23 Stéphane Le Roux and Arno Pauly. Extending finite-memory determinacy to multi-player games. *Inf. Comput.*, 261(Part):676–694, 2018. doi:10.1016/j.ic.2018.02.024.

24 Stéphane Le Roux, Arno Pauly, and Mickael Randour. Extending finite-memory determinacy by Boolean combination of winning conditions. *CoRR*, abs/1808.05791, 2018. arXiv:1808.05791.

25 Yaron Velner. Robust Multidimensional Mean-Payoff Games are Undecidable. In Andrew M. Pitts, editor, *Foundations of Software Science and Computation Structures - 18th International Conference, FoSSaCS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9034 of *Lecture Notes in Computer Science*, pages 312–327. Springer, 2015. doi:10.1007/978-3-662-46678-0_20.

26 Yaron Velner, Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, Alexander Moshe Rabinovich, and Jean-François Raskin. The complexity of multi-mean-payoff and multi-energy games. *Inf. Comput.*, 241:177–196, 2015. doi:10.1016/j.ic.2015.03.001.

27 Wieslaw Zielonka. Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998. doi:10.1016/S0304-3975(98)00009-7.

A Technical details

Let us first come back to the notion of *FM strategy*. Recall that a strategy for Player i is a function $\sigma_i \colon C^* \times V_i \to (V_1 \cup V_2)$. It is *finite-memory* if it can be encoded by a deterministic *Moore machine* $(M, m_0, \alpha_u, \alpha_n)$ where M is a finite set of states (the memory of the strategy), $m_0 \in M$ is the initial memory state, $\alpha_u \colon M \times C \to M$ is the update function, and $\alpha_n \colon M \times V_i \to (V_1 \cup V_2)$ is the next-action function. The Moore machine defines a strategy σ_i such that $\sigma_i(wv) = \alpha_n(\widehat{\alpha}_u(m_0, w), v)$ for all history $w \in C^*$ and vertex $v \in V_i$, where $\widehat{\alpha}_u$ extends α_u to sequences of colors as expected. The *size* of the strategy is the size $|M|$ of its Moore machine. Note that a strategy is *memoryless* when $|M| = 1$.

We need some additional notation: If I is some finite index set, and $(\mathcal{A}_i)_{i \in I}$ is an I-indexed family of automata, we denote their product by $\otimes_{i \in I} \mathcal{A}_i$.

Proof of Theorem 13. By induction on l. The claim holds for $l = 0$ by assumption since $R_0(\mathcal{W}) = \mathcal{W}$, so let us assume that $l > 0$. Let $g = \langle V_1, V_2, E, \Gamma, W \rangle \in \mathcal{O} \times R_l(\mathcal{W})$, let φ, $W_1, \dots, W_k$, and $L_1, \dots, L_l$ witness that $W \in R_l(\mathcal{W})$.

Let us fix $\varphi_\perp(w_1, \dots, w_k, r_1, \dots, r_{l-1}) \coloneqq \varphi(w_1, \dots, w_k, r_1, \dots, r_{l-1}, \perp)$, and let $g_\perp = \langle V_1, V_2, E, \Gamma, W_\perp \rangle$ where $\rho \in W_\perp$ if and only if $\varphi_\perp(\rho \in W_1, \dots, \rho \in W_k, Pref(\rho) \cap L_1 = \emptyset, \dots, Pref(\rho) \cap L_{l-1} = \emptyset)$. Since $W_\perp \in R_{l-1}(\mathcal{W})$, by induction hypothesis let $\sigma_\perp$ be an hSPE for $g_\perp$ and for all vertices $v \in V$ let $\mathcal{A}_v^\perp = \langle C, Q_v^\perp, q_{0,v}^\perp, F_v^\perp, \Delta_v^\perp \rangle$ witness the regular-predictability of $g_\perp$.

In g, if and once the current color history and the current vertex have falsified $Pref(\rho) \cap L_l = \emptyset$, two issues are simplified: first, playing in g according to $\sigma_\perp$ is optimal for both players; second, for each vertex $v \in V \coloneqq V_1 \cup V_2$, the automaton $\mathcal{A}_v^\perp$ accounts faithfully for who has a winning strategy when reaching v after a given color history.

When $Pref(\rho) \cap L_l = \emptyset$ has not (yet) been falsified, it is more difficult for players both to know how to play optimally and to know who is the potential winner. A special case arises when a player can force the play to eventually falsify $Pref(\rho) \cap L_l = \emptyset$ while

ensuring victory regardless of how falsification happens. Given arbitrary color history and vertex, two types of object are relevant for the players to know whether the special case obtains: for all $v \in V$, the $\mathcal{A}_v^\perp$ and a finite automaton $\mathcal{B}_v^l = (C, Q_v^l, q_{0,v}^l, F_v^l, \Delta_v^l)$ accepting the finite words α such that $\alpha\Gamma(v)$ has a prefix in L_l. (The $\mathcal{B}_v^l$ exist since L_l is regular.) Let $g' \coloneqq \langle O \times \otimes_{u\in V}\mathcal{B}_u^l \times \otimes_{u\in V}\mathcal{A}_u^\perp, W\rangle$, where O is the arena of g. Note that to play using finite memory in g' is like to play in g using the same memory used in g', plus some finite memory keeping track of the $\mathcal{B}_v^l$ and the $\mathcal{A}_v^\perp$.

Let S_1 (S_2) be the vertices of g', i.e., of the form $(v, (q_u^l)_{u\in V}, (q_u^\perp)_{u\in V})$ such that $q_v^l \in F_v^l$ and $q_v^\perp \in F_v^\perp$ ($q_v^\perp \notin F_v^\perp$). We observe, merely by rephrasing, that given a color history α and a vertex v, the special case obtains for Player 1 (2) iff Player 1 (2) has a winning strategy in g' to reach S_1 (S_2) from $(\mathrm{v}, (q_u^l)_{u\in V}, (q_u^\perp)_{u\in V})$ where the states q_u^l and $q_u^\perp$ correspond to the states of $\mathcal{B}_u^l$ and the $\mathcal{A}_u^\perp$, respectively, after reading α. Recognizing the special case in g therefore amounts to reachability analysis in g': let S_1' (S_2') be the vertices of $g' \setminus (S_1 \cup S_2)$ from where Player 1 (2) can reach S_1 (S_2) regardless of how the opponent is playing. So Player 1 (2) has a partial positional strategy in g' that ensures uniformly that S_1 (S_2) is reached from S_1' (S_2'). The combination of the two partial strategies translates back to g into an FM partial strategy profile $\sigma_\perp'$ that uses only the $\mathcal{B}_v^l$ and the $\mathcal{A}_v^\perp$ as auxiliary memory, and such that playing according to $\sigma_\perp \cup \sigma_\perp'$ is optimal for both players when the vertex in g corresponds to a vertex in $S_1 \cup S_2 \cup S_1' \cup S_2'$.

Furthermore, deciding who is the potential winner in g at a vertex after some color history is easy when the corresponding vertex in g' is in $S_1' \cup S_2'$: if it is in S_1', Player 1 is the potential winner; otherwise it is Player 2. So the automaton $\otimes_{u\in V}\mathcal{B}_u^l \times \otimes_{u\in V}\mathcal{A}_u^\perp$ will help us witness regular-predictability of g in the next paragraphs.

Let us now consider the remaining case, where $Pref(\rho) \cap L_l = \emptyset$ has not been falsified, and no player is able to force falsification to his own benefit. The corresponding color histories and vertices in g translate in g' to the set $V_\top \coloneqq (V \times \otimes_{u\in V}Q_u^l \times \otimes_{u\in V}Q_u^\perp) \setminus (S_1 \cup S_2 \cup S_1' \cup S_2')$. By definition of reachability, every vertex in $V_\top$ with an edge towards $S_1 \cup S_2 \cup S_1' \cup S_2'$ has also an edge staying in $V_\top$. So $O' \mid_{V_\top}$ is also an arena, where O' is the arena of g'. Moreover, by definition of S_1', if Player 1 controls a vertex in $V_\top$ with an edge towards $S_1 \cup S_2 \cup S_1' \cup S_2'$, it is an edge towards $S_2 \cup S_2'$, so never taking this edge is optimal. (Likewise for Player 2.) This implies that every (FM) hSPE in "$g_\top \coloneqq g' \mid_{V_\top}$" (precisely defined below) translates back to g into (FM) partial hSPE, and for all color histories and vertices the potential winner in the restriction is the same as in g'. Let $\varphi_\top(w_1, \dots, w_k, r_1, \dots, r_{l-1}) \coloneqq \varphi(w_1, \dots, w_k, r_1, \dots, r_{l-1}, \top)$, and let $g_\top \coloneqq \langle O' \mid_{V_\top}, W_\top\rangle$ where $\rho \in W_\top$ iff $\varphi_\top(\rho \in W_1, \dots, \rho \in W_k, Pref(\rho) \cap L_1 = \emptyset, \dots, Pref(\rho) \cap L_{l-1} = \emptyset)$. Since $W_\top \in R_{l-1}(\mathcal{W})$, by induction hypothesis let $\sigma_\top$ be an hSPE for $g_\top$ and for all vertices $v \in V$ let $\mathcal{C}_v^\top = \langle C, Q_v^\top, q_{0,v}^\top, F_v^\top, \Delta_v^\top\rangle$ witness the regular-predictability of $g_\top$.

To show the regular-predictability of g, let $v \in V$ and let $\mathcal{A}_v$ be the modification of $\otimes_{u\in V}\mathcal{B}_u^l \times \otimes_{u\in V}\mathcal{A}_u^\perp \times \mathcal{C}_v^\top$ where the final states F_v are the $((q_u^l)_{u\in V}, (q_u^\perp)_{u\in V}, q_v^\top)$ such that either $(v, (q_u^l)_{u\in V}, (q_u^\perp)_{u\in V}) \in S_1$ (falsification happened and Player 1 is the potential winner), or $(v, (q_u^l)_{u\in V}, (q_u^\perp)_{u\in V}) \in S_1'$ (Player 1 can force falsification to his own benefit), or $(v, (q_u^l)_{u\in V}, (q_u^\perp)_{u\in V}) \notin S_2 \cup S_2' \wedge q_v^\top \in F_v^\top$ (falsification benefiting Player 2 has not happened and is not going to, and Player 1 is the potential winner of restricted game). The automaton $\mathcal{A}_v$ accepts exactly the color histories after which Player 1 has a winning strategy when starting from v.

A strategy profile σ for g (which is meant to be an hSPE) is build by case disjunction. Informally, given a color history and a vertex,

- if the corresponding vertex in g' is in $S_1 \cup S_2$, follow $\sigma_\perp$,

- if it is in $S_1' \cup S_2'$, follow $\sigma'_\bot$,
- otherwise follow $\sigma_\top$.

More formally, let $\alpha \in C^*$, let $v \in V$, let $((q_u^l)_{u\in V}, (q_u^\bot)_{u\in V})$ be the state of $\otimes_{u\in V}\mathcal{B}_u^l \times \otimes_{u\in V}\mathcal{A}_u^\bot$ after reading $\alpha\Gamma(v)$, and let us make a case disjunction.

- If $(v, (q_u^l)_{u\in V}, (q_u^\bot)_{u\in V}) \in S_1 \cup S_2$ let $\sigma(\alpha, v) \coloneqq \sigma_\bot(\alpha, v)$.
- If $(v, (q_u^l)_{u\in V}, (q_u^\bot)_{u\in V}) \in S_1' \cup S_2'$, let $\sigma(\alpha, v) \coloneqq \pi_1 \circ \sigma'_\bot(\alpha, v, (q_u^l)_{u\in V}, (q_u^\bot)_{u\in V})$
- If $(v, (q_u^l)_{u\in V}, (q_u^\bot)_{u\in V}) \in V_\top$, let $\sigma(\alpha, v) \coloneqq \pi_1 \circ \sigma_\top(\alpha, (v, (q_u^l)_{u\in V}, (q_u^\bot)_{u\in V}))$.

(Above, projection π_1 is used twice to retrieve vertex v from vertex $(v, (q_u^l)_{u\in V}, (q_u^\bot)_{u\in V})$.)
To implement the above strategy, we use the automaton $\otimes_{u\in V}\mathcal{B}_u^l \times \otimes_{u\in V}\mathcal{A}_u^\bot$ to keep track of the current vertex in g', and we also use automata implementing $\sigma_\bot$, $\sigma'_\bot$, and $\sigma_\top$. ◀

Proof of Corollary 14. Let us use the notation from the proof of Theorem 13. Thanks to prefix-independence of the winning condition, we do not need to worry about regular-predictability and the corresponding automata. Since $\sigma_\bot$ is an hSPE on a n-vertex game, $f(l-1, n)$ states suffice to implement it. The $\sigma'_\bot$ for reachability can be chosen memoryless. As $\sigma_\top$ is an hSPE on a game with at most nm^n vertices, $f(l-1, nm^n)$ states suffice to implement it. Additionally, we keep track of the current vertex in g', adding a factor of m^n. Therefore $f(l, n) \leq m^n \cdot f(l-1, n) \cdot f(l-1, nm^n)$. ◀

Proof of Example 17. Player 1 wins from some not-yet-determined history iff he can force an $\alpha\beta$-vertex. Linear memory suffices to keep track of how many 0 and 1's have been encountered along the way (as a simple path suffices), and enables Player 1 to choose correctly. However, which choice is correct depends on the pre-history in a way that a finite automaton cannot keep track of (as this history is unbounded). Thus, there is no FM subgame-perfect strategy. ◀

Deterministic Algorithms for Maximum Matching on General Graphs in the Semi-Streaming Model

Sumedh Tirodkar[1]
IDSIA, USI-SUPSI, Manno, Switzerland
sumedh.tirodkar@idsia.ch

Abstract

We present an improved deterministic algorithm for Maximum Cardinality Matching on general graphs in the Semi-Streaming Model. In the *Semi-Streaming* Model, a graph is presented as a sequence of edges, and an algorithm must access the edges in the given sequence. It can only use $O(n \operatorname{polylog} n)$ space to perform computations, where n is the number of vertices of the graph. If the algorithm goes over the stream k times, it is called a k-pass algorithm. In this model, McGregor [28] gave the currently best known randomized $(1+\varepsilon)$-approximation algorithm for maximum cardinality matching on general graphs, that uses $(1/\varepsilon)^{O(1/\varepsilon)}$ passes. Ahn and Guha [1] later gave the currently best known deterministic $(1+\varepsilon)$-approximation algorithms for maximum cardinality matching: one on bipartite graphs that uses $O\left(\log\log(1/\varepsilon)/\varepsilon^2\right)$ passes, and the other on general graphs that uses $O(\log n \cdot \operatorname{poly}(1/\varepsilon))$ passes (note that, for general graphs, the number of passes is dependent on the size of the input). We present the first deterministic algorithm that achieves a $(1+\varepsilon)$-approximation on general graphs in only a constant number $\left((1/\varepsilon)^{O(1/\varepsilon)}\right)$ of passes.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Semi Streaming, Maximum Matching

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.39

Acknowledgements The author thanks Ashish Chiplunkar, Sagar Kale, Sundar Vishwanathan, and anonymous reviewers for their helpful feedback on the writeup.

1 Introduction

Matching is one of the most well-studied problems in combinatorial optimization. See Schrijver's book [31] and references therein for a comprehensive overview of the classical work. There are polynomial time algorithms known for both weighted and unweighted maximum matching on general graphs [29]. With the advancement of internet and social networks, large amount of data is generated, and often the input graph is so huge that the entire graph may not fit even inside large size RAMs. One way to tackle this problem is to provide the input to the algorithm in pieces. For instance, edges can be provided to the algorithm one by one, or vertices can be provided one by one along with all the edges from that vertex to the previously revealed vertices. The maximum matching problem has been studied in various models in which the input is provided piecewise, for instance, the online preemptive/non-preemptive model (vertex/edge arrival) [15, 11, 23, 7], the dynamic graph model [5, 6, 4, 32], the streaming model [18, 28, 33, 14], etc. In these models, random order [25, 26] arrivals have also been considered.

[1] This work was done when the author was a Post Doctoral Fellow in the School of Technology and Computer Science at TIFR, Mumbai, India.

We study the maximum matching[2] problem in the semi-streaming model. Feigenbaum et al. [18] were the first to consider this problem in the semi-streaming model. A graph stream is an (adversarial) sequence of the edges of a graph, and a semi-streaming algorithm must access the edges in the given order. A semi-streaming algorithm can use $O(n \operatorname{polylog} n)$ space only, where n is the number of vertices in the input graph. Note that a matching can have size $\Omega(n)$, so $\Omega(n \log n)$ space is necessary. If a semi-streaming algorithm goes over the stream k times, it is called a k-pass algorithm. We say that an algorithm is an α-approximation algorithm if the size of the matching output by the algorithm is at least $\frac{1}{\alpha}$ times the size of an optimum matching, and we say that the matching is α-approximate.

McGregor [28] gave the first $(1+\varepsilon)$-approximation algorithm for maximum matching in this model. This algorithm is randomized, and uses a constant number of passes $\left((1/\varepsilon)^{O(1/\varepsilon)}\right)$. Eggert et al. [13] later improved this result on bipartite graphs by giving a $(1+\varepsilon)$-approximation deterministic algorithm that uses $O(1/\varepsilon^5)$ passes. Ahn and Guha [1] also gave linear programming based $(1+\varepsilon)$-approximation deterministic algorithms. For bipartite graphs, they further improved the results. Their algorithm uses only $O(\log\log(1/\varepsilon)/\varepsilon^2)$ passes. But for general graphs, their algorithm uses $O(\log n \cdot \operatorname{poly}(1/\varepsilon))$ passes. The number of passes is dependent on the size of the input. There are no known $(1+\varepsilon)$-approximation deterministic algorithms on general graphs that use a constant number of passes, and it is not clear how to extend the deterministic algorithms on bipartite graphs [13, 1], to deterministic algorithms on general graphs that use only a constant number of passes. In this paper, we present the first $(1+\varepsilon)$-approximation deterministic algorithm for maximum matching on general graphs that uses only a constant number $\left((1/\varepsilon)^{O(1/\varepsilon)}\right)$ of passes.

We first present a $(1+\varepsilon)$-approximation deterministic algorithm for maximum matching on bipartite graphs, that uses $(1/\varepsilon)^{O(1/\varepsilon)}$-passes, which will help in understanding the main techniques behind the algorithm on general graphs. Note that the algorithm on bipartite graphs in itself does not hold much value, as we have already mentioned earlier that there exist $\operatorname{poly}(1/\varepsilon)$-pass $(1+\varepsilon)$-approximation algorithms on bipartite graphs.

These algorithms build on the techniques used in the two-pass algorithm for maximum matching presented in [20]. In the first pass, the algorithm [20] finds a maximal matching, and in the second pass, it finds a semi-matching between matched and unmatched vertices from the first pass. A (λ_X, λ_Y) *Semi-Matching* is defined as a set of edges such that at most λ_X edges are incident on any vertex in X, and at most λ_Y edges are incident on any vertex in Y, when $X \cap Y = \emptyset$. The algorithm uses this semi-matching to find length 3-augmenting paths in the maximal matching. In the following paragraph, we give a brief overview of how these techniques can be used to find longer augmenting paths in bipartite graphs, and some of the difficulties in extending them to general graphs.

Technical Overview

Let M^* denote some optimal matching in the given graph, and let M denote some maximal matching. For any integer $\ell \geq 1$, a connected component of $M \cup M^*$ that has a path of length $(2\ell+1)$ is called a length $(2\ell+1)$-*augmenting* path (non-augmenting otherwise) with respect to M^*. In general, a length $(2\ell+1)$-augmenting path contains ℓ edges in M, and $(\ell+1)$ edges not in M. We call an edge in M $(2\ell+1)$-*augmentable*, if it belongs to a length $(2\ell+1)$-augmenting path. A length $(2\ell+1)$-*augmentation* is a replacement of edges in M, by edges not in M, from a length $(2\ell+1)$-augmenting path.

[2] Unless explicitly mentioned, Maximum Matching means Maximum Cardinality Matching.

It is widely known, and can also be easily proved that in a maximal matching M, if there are no augmenting paths of length less than $(2/\varepsilon+1)$ with respect to some optimum matching M^*, then the matching M is $(1+\varepsilon)$-approximate. We present deterministic algorithms which output a matching M, that contains a negligible number of augmenting paths of length less than $(1/\varepsilon+1)$ with respect to some optimum matching M^*. We use Lemma 1 to prove that this matching is an $(1+O(\varepsilon))$-approximation to the optimum matching. Informally, the lemma states that if there exist a negligible number of augmenting paths of length less that $(1/\varepsilon+1)$ in a maximal matching M, then M is a good approximation to any optimum matching.

The algorithms for bipartite graphs and general graphs have a common high level idea. Each algorithm begins by finding a maximal matching M in the first pass. Then it first carries out length 3-augmentations of some edges in M, such that after a constant number of passes, the number of remaining 3-augmentable edges in M with respect to any fixed optimal matching M^* is negligible. Then it similarly carries out length 5-augmentations, followed by length 7-augmentations, and so on, up to length $(1/\varepsilon)$-augmentations.

Suppose there are no augmenting paths in M of length less than $(2\ell+1)$. Then, a length $(2\ell+1)$-augmenting path is found as follows. Suppose $au_1v_1u_2v_2u_3\dots v_{\ell-1}u_\ell v_\ell b$ denotes some length $(2\ell+1)$-augmenting path in M with respect to M^*. (Note that edges $u_iv_i \in M$, for all $1 \le i \le \ell$.) In one pass, (a constant fraction of) the outermost edges of any length $(2\ell+1)$-augmenting path (for instance au_1 and $v_\ell b$) are added to a semi-matching S. Now that the algorithm already knows the outermost four edges (for instance $au_1, u_1v_1, u_\ell v_\ell, v_\ell b$) in the augmenting path, it tries to find the length $(2\ell-3)$-augmenting path $v_1u_2v_2u_3\dots v_{\ell-1}u_\ell$. Using this length $(2\ell-3)$-augmenting path $v_1u_2v_2u_3\dots v_{\ell-1}u_\ell$, and edges in M and S, the algorithm finds a length $(2\ell+1)$-augmenting path.

For general graphs, suppose such a procedure is successful in finding a middle length $(2\ell'-3)$ path of some length $(2\ell+1)$-augmenting path. Then how to ensure that the edges stored in the semi-matchings, along with the edges in M, do not form an odd length cycle with this middle length $(2\ell'-3)$ path? We do not face this issue for bipartite graphs as they do not contain any odd length cycles, and so we require new ideas (Directed Semi-Matchings – defined in Section 2) to extend the described techniques to find longer augmenting paths in general graphs.

A Comparison with McGregor's Randomized Algorithm

McGregor's algorithm [28] begins by finding a maximal matching M. To find length $(2\ell+1)$-augmenting paths (for all $\ell \le (1/(2\varepsilon))$), the algorithm randomly partitions the vertices in G into $(\ell+2)$ layers $L_0, L_1, \dots, L_\ell, L_{\ell+1}$. The unmatched vertices in G are added to either L_0 or $L_{\ell+1}$, and the matched vertices are added at random to the layers $L_1, \dots, L_\ell$. The algorithm only stores the edges between two consecutive layers. This makes the graph bipartite, and removes the possibility of any odd length cycle. The algorithm works by finding maximal matchings, first between the first and second levels and then between the vertices in the second that were matched in the first matching, and the third level, and then between the nodes in the third level that were matched in the second matching and the fourth level and so on. This gives node-disjoint $(\ell+1)$-paths, which correspond to length $(2\ell+1)$-augmenting paths in G. The random partitioning operation keeps roughly a $(1/\ell)^{O(\ell)}$ fraction of the length $(2\ell+1)$-augmenting paths, as each such augmenting path "survives" the partitioning by this probability. By repeating this process by a factor of $\ell^{O(\ell)}$, almost all length $(2\ell+1)$-augmenting paths are found. The algorithm finds a $(1+\varepsilon)$-approximate matching with probability $(1-f)$ by running $O(\log(1/f))$ copies of this procedure in parallel.

Like McGregor's algorithm, our algorithm also ensures that a negligible number of $(2\ell+1)$-augmentable edges remain in M (for all $\ell \leq (1/(2\varepsilon))$). As pointed out earlier, the main difficulty in finding augmenting paths in general graphs is due to the existence of odd length cycles (this challenge is present in nearly all variants of the matching problem in non-bipartite graphs, including in the polynomial time algorithms for this problem in the offline setting [29]). McGregor's algorithm handles this issue by considering a random bipartite subgraph of the original graph in each augmentation phase. We rely on the use of directed semi-matchings. The major technical challenge lies in finding which edges need to be stored in the directed semi-matchings. The algorithm needs to recognize and ensure that it does not store edges (in the directed semi-matchings) that form an odd length cycle. This turns out to be non-trivial. Section 4 gives more details.

Note. The random bipartitioning operation (from McGregor's algorithm [28]) in general graphs, keeps roughly a $(1/\ell)^{O(\ell)}$ fraction of the length $(2\ell+1)$-augmenting paths. And hence, this process has to be repeated $\ell^{O(\ell)}$ times to find almost all length $(2\ell+1)$-augmenting paths. So, it is unlikely that this idea can be extended to get a poly$(1/\varepsilon)$-pass $(1+\varepsilon)$-approximation algorithm on general graphs. Our deterministic algorithm explicitly bypasses the barrier of "blossoms" (hereafter, we refer to an odd length alternating (edges not in M and edges in M) cycle that starts and ends at an unmatched vertex as a *blossom*), and hence can be viewed as a step towards achieving a poly$(1/\varepsilon)$-pass $(1+\varepsilon)$-approximation algorithm for general graphs.

Related Work

The algorithm that finds a maximal matching (in which an edge is added to the matching M if there are no edges in M incident on any of its endpoints) is a trivial one-pass 2-approximation algorithm, and no (randomized/deterministic) one-pass algorithm with approximation ratio better than 2 is known for maximum matching although this model was introduced over a decade ago. It remains as one of the major open problems in the streaming community [27]. Goel, Kapralov, and Khanna [19], using connection between streaming complexity and communication complexity, proved that for any $\varepsilon > 0$, a one-pass semi-streaming $(3/2-\varepsilon)$-approximation algorithm does not exist. Later, Kapralov [21], building on those techniques, showed the non-existence of a one-pass semi-streaming $(e/(e-1)-\varepsilon)$-approximation algorithms for any $\varepsilon > 0$. Konrad et al. [25] showed that if the algorithm is allowed one extra pass, then a better than 2-approximate matching can be obtained, by giving a two-pass algorithm. Later, Esfandiari et al. [16] improved these results for bipartite graphs, and Kale and Tirodkar [20] gave improved results on triangle-free as well as general graphs.

Feigenbaum et al. [18] gave a $(3/2+\varepsilon)$-approximation deterministic algorithm on bipartite graphs that uses $O(\log(1/\varepsilon)/\varepsilon)$ passes. Later, Ahn and Guha [1] gave a $(3/2+\varepsilon)$-approximation deterministic algorithm on general graphs that uses $O(\log(1/\varepsilon)/\varepsilon^2)$ passes. Kale and Tirodkar [20] improved both these results by giving a $(3/2+\varepsilon)$-approximation deterministic algorithm on general graphs that use only $O(1/\varepsilon)$ passes.

Feigenbaum et al. [18] gave the first one-pass algorithm for maximum weight matching, with an approximation ratio 6. Subsequent results improved this approximation ratio. Recently in a breakthrough, Paz and Schwartzman [30] gave a $(2+\varepsilon)$-approximation algorithm. The multi-pass version of the problem was considered first by McGregor [28], then by Ahn and Guha [1]. Chakrabarti and Kale [9] and Chekuri et al. [10] consider a more general version of the matching problem where a submodular function is defined on the edges of the input graph.

The problem of estimating the *size* of a maximum matching (instead of outputting the actual matching) has been well studied [22, 17, 8, 2]. The Maximum Matching problem has also been studied on dynamic streams (in which edges can be added as well as deleted) [24, 3, 12].

1.1 Organization of the paper

After setting up notation in Section 2, we see a $(1+\varepsilon)$-approximation deterministic algorithm for maximum matching on bipartite graphs in Section 3. Then in Section 4, we see our main result – a $(1+\varepsilon)$-approximation deterministic algorithm on general graphs that uses a constant number of passes.

2 Preliminaries

Let M^* denote some optimal matching in the given graph, and let M denote some maximal matching. We define the following terms which are used in the subsequent sections.

▶ **Definition 2.1.** A (λ_X, λ_Y) *Directed Semi-Matching* is defined as a set of directed edges such that at most λ_X incoming edges are incident on any vertex in X, and at most λ_Y outgoing edges are incident on any vertex in Y.

For a (λ_X, λ_Y) Semi-Matching S, let $\deg_S(x)$ represent the number of edges in S incident on vertex x. For a directed semi-matching S, let $\deg^O_S(x)$ and $\deg^I_S(x)$ represent the number of outgoing and incoming edges, respectively, in S incident on vertex x. For an edge xy in the input stream, both the directed edges xy and yx are considered while finding the directed semi-matchings.

▶ **Definition 2.2.** While finding the middle length $(2\ell'+1)$ path of the length $(2\ell+1)$-augmenting paths, the end points of M which have a length $(\ell-\ell')$ alternating path (with edges in M and edges in the semi-matchings) to unmatched vertices, are called *free vertices*.

(For instance, the green vertices in Figure 1 are free vertices while finding the middle length $(2\ell-3)$-augmenting path.) Suppose $au_1v_1u_2v_2u_3\ldots v_{\ell-1}u_\ell v_\ell b$ denotes some augmenting path of length $(2\ell+1)$ in M with respect to M^*. Then, $v_{(\ell-\ell')/2}u_{(\ell-\ell')/2+1}v_{(\ell-\ell')/2+1}\cdots u_{(\ell+\ell')/2}\ v_{(\ell+\ell')/2}u_{(\ell+\ell')/2+1}$ denotes the middle length $(2\ell'+1)$ path of the augmenting path. Vertex $u_{(\ell-\ell')/2}$ (or $v_{(\ell+\ell')/2+1}$) is a free vertex if there exists a length $(\ell-\ell')$ path to unmatched vertices starting from $u_{(\ell-\ell')/2}$ (or $v_{(\ell+\ell')/2+1}$) via alternate edges in M and the semi-matchings maintained by the algorithm. When $\ell'=\ell$, $V\setminus V(M)$ are the free vertices.

We use the following lemma (similar to Lemma 1 in [28]) in analyzing the performance of the algorithms described in this paper.

▶ **Lemma 1.** *Let M^* be a maximum matching in G. If there are at most $\varepsilon^2|M|$ $(2\ell+1)$-augmentable edges in a maximal matching M with respect to M^*, for any $\ell \le (1/\varepsilon-1)/2$, then M is a $(1+3\varepsilon)$-approximate maximum matching.*

Proof. Let k_ℓ denote the number of $(2\ell+1)$-augmentable edges in M. In any length $(2\ell+1)$-augmenting path, the ratio of number of edges in M^* to the number of edges in M is $\frac{l+1}{l}$.

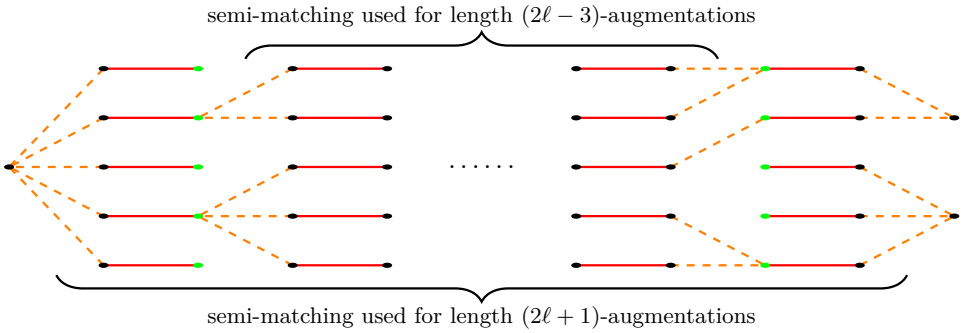


Figure 1 Length $(2\ell+1)$-augmentations. Solid red lines represent edges in M, and dashed orange lines represent edges that belong to S during the recursive steps.

Then,

$$\begin{aligned}
|M^*| &\leq 2k_1 + \frac{3}{2}k_2 + \frac{4}{3}k_3 + \cdots + \frac{(1/\varepsilon-1)/2+1}{(1/\varepsilon-1)/2}k_{(1/\varepsilon-1)/2} \\
&\quad + \frac{(1/\varepsilon+1)/2+1}{(1/\varepsilon+1)/2}(|M| - k_1 - k_2 - \cdots - k_{(1/\varepsilon-1)/2}) \\
&= \left(1 + \frac{2}{1/\varepsilon+1}\right)|M| + \left(2 - \frac{1/\varepsilon+3}{1/\varepsilon+1}\right)k_1 + \left(\frac{4}{3} - \frac{1/\varepsilon+3}{1/\varepsilon+1}\right)k_3 + \ldots \\
&\quad + \left(\frac{1/\varepsilon+1}{1/\varepsilon-1} - \frac{1/\varepsilon+3}{1/\varepsilon+1}\right)k_{(1/\varepsilon-1)/2} \\
&\leq (1+2\varepsilon)|M| + \varepsilon^2 \cdot |M| + \varepsilon^2 \cdot |M| + \cdots + \varepsilon^2 \cdot |M| \\
&\leq (1+3\varepsilon)|M|.
\end{aligned}$$

◀

For the sake of exposition, we ignore floors and ceilings during the analyses of the algorithms presented in Sections 3 and 4.

3 Warming Up: An Algorithm on Bipartite Graphs

In this section, we present a $(1/\varepsilon)^{O(1/\varepsilon)}$-pass $(1+\varepsilon)$-approximation deterministic algorithm for maximum matching on bipartite graphs.

The high level idea for the algorithm is already described in Section 1. What remains to be described is how the length $(2\ell+1)$-augmentations are performed. The function call for length $(2\ell+1)$-augmentations in the matching M assumes that there are no shorter augmenting paths (although there is a small number of them). For each function call, the algorithm finds a $(1/\varepsilon^4, 1)$ semi-matching S in one pass, between the free and matched vertices. For an edge $e \in M$, if there are no edges in S, incident on any of the endpoints of e, add e to M'. For an edge $e \in M$, if there is an edge in S, incident on one of the end points of e, add the other endpoint to the set V_1. Then, the function recursively calls AUGMENT$(\ell-2, V_1, M')$ to find length $(2\ell-3)$-augmenting paths for the matching M' in the graph induced on $V_1 \cup V(M')$. For the recursive call, vertices in V_1 are the free vertices, and vertices in $V(M')$ are the matched vertices. Using these length $(2\ell-3)$-augmenting paths, and edges in S, the algorithm finds length $(2\ell+1)$-augmenting paths in M. Figure 1 gives an illustration of the semi-matchings stored by the algorithm which are used for length $(2\ell+1)$-augmentations.

Algorithm 1 gives a formal description.

Algorithm 1 Deterministic Algorithm on Bipartite Graphs.

Find Maximal Matching M in the first pass.
for $\ell = 1$ `to` $1/(2\varepsilon)$ **do** ▷ To Remove almost all $(2\ell+1)$-augmenting paths
  for `Phase` $p = 1$ `to` $1/\varepsilon^{5\ell}$ **do**
    $M \leftarrow \text{AUGMENT}(\ell, V, M)$.
Output M.
function AUGMENT(ℓ, V, M) ▷ Function of length $(2\ell+1)$-augmentations.
  if $\ell = 0$ **then**
    Find Maximal Matching M' on vertex set V in one pass.
    $M \leftarrow M'$.
  else
    $S \leftarrow \emptyset$, $M' \leftarrow \emptyset$.
    $V_1 \leftarrow V \setminus V(M)$. ▷ Set of free vertices.
    $S \leftarrow \text{SEMI}(1, V(M), 1/\varepsilon^4, V_1)$.
    if $\ell > 1$ **then** ▷ For longer than length 3-augmentations
      $V_1 \leftarrow \emptyset$
      for all edges $uv \in M$ **do**
        if $\exists au \in S$ **then**
          $V_1 \leftarrow V_1 \cup \{v\}$. ▷ Free Vertices in the recursive call.
        if $\nexists$ edge in S on either u or v **then**
          $M' \leftarrow M' \cup \{uv\}$.
      $M' \leftarrow \text{AUGMENT}(\ell - 2, V_1, M')$. ▷ Find length $(2\ell-3)$-augmentations.
    $(2\ell+1)$-augment M greedily using edges in S and M'.
  return M.
function SEMI$(\lambda_X, X, \lambda_Y, Y)$ ▷ Finds a semi-matching in one pass.
  $S \leftarrow \emptyset$.
  for all edges xy such that $x \in X$, $y \in Y$ **do**
    if $\deg_S(x) < \lambda_X$ and $\deg_S(y) < \lambda_Y$ **then**
      $S \leftarrow S \cup \{xy\}$
  return S.

We begin the analysis for any length $(2\ell+1)$-augmentations for the matching M. The function assumes that there are no shorter length augmenting paths in M with respect to any optimal matching. Let E_ℓ denote the set of $(2\ell+1)$-augmentable edges in M with respect to an optimal matching M^*, such that $|E_\ell|$ is maximum.

Bad Edges

Suppose $au_1v_1u_2v_2u_3 \dots v_{\ell-1}u_\ell v_\ell b$ denotes any length $(2\ell+1)$-augmenting path in M with respect to M^*. Then, $v_{(\ell-\ell')/2}u_{(\ell-\ell')/2+1}v_{(\ell-\ell')/2+1} \dots u_{(\ell+\ell')/2}v_{(\ell+\ell')/2}\ u_{(\ell+\ell')/2+1}$ denotes the middle length $(2\ell'+1)$ path of the augmenting path. Edge $u_{(\ell-\ell')/2+1}v_{(\ell-\ell')/2+1} \in M'$ (or $u_{(\ell+\ell')/2}v_{(\ell+\ell')/2} \in M'$) is called a *b*ad edge, if there is no edge in S from $u_{(\ell-\ell')/2+1}$ (or $v_{(\ell+\ell')/2}$) to a free vertex, during the recursive call AUGMENT(ℓ', V_1, M'). Note that V_1 is the set of free vertices during a function call AUGMENT(ℓ', V_1, M').

Let $E^\ell_{B_1}$ denote a set of bad edges u_1v_1 (or $u_\ell v_\ell$) if no edge is added to S during a function call AUGMENT(ℓ, V, M) that is incident from the vertex u_1 (or v_ℓ) to some free vertex (in this case $V_1 = V \setminus V(M)$ are the free vertices).

▶ **Lemma 2.** $|E^{\ell}_{B_1}| \leq 2\varepsilon^4|M|$.

Proof. Suppose there is no edge in S incident from u_1 (or v_ℓ) to any unmatched vertex in V_1. This means that for all the edges incident from u_1 (or v_ℓ) to unmatched vertices, there already were $1/\varepsilon^4$ edges in S incident on the respective unmatched vertices. For $\ell > 1$, each edge in M can have at most one edge in S incident on its endpoints (as there are no length 3-augmentable edges in M). Hence, $|E^{\ell}_{B_1}|/\varepsilon^4 \leq |M|$.

For $\ell = 1$, each edge in M can have at most one edge in S incident on each its endpoints. Hence, $|E^{\ell}_{B_1}|/\varepsilon^4 \leq 2|M|$. ◀

After finding a $(1/\varepsilon^4, 1)$ semi-matching S from unmatched to matched vertices, the function call $\textsc{Augment}(\ell, V, M)$ makes a recursive call $\textsc{Augment}(\ell-2, V_1, M')$. Inside the recursive call, V_1 is the set of free vertices, and $V(M')$ is the set of matched vertices. Let $E^{\ell}_{B_2}$ denote the bad edges formed while finding a $(1/\varepsilon^4, 1)$ semi-matching inside the recursive call. By Lemma 2, $|E^{\ell}_{B_2}| \leq 2\varepsilon^4|M'| \leq 2\varepsilon^4|M|$.

If $au_1v_1u_2v_2u_3 \ldots v_{\ell-1}u_\ell v_\ell b$ denotes any length $(2\ell+1)$-augmenting path in M with respect to M^*, let E^{ℓ}_B denote the set of all the bad edges $u_iv_i \in E_\ell$ formed during one function call $\textsc{Augment}(\ell, V, M)$, and the subsequent recursive calls inside the function. The following lemma bounds the total number of bad edges E^{ℓ}_B in E_ℓ.

▶ **Lemma 3.** $|E^{\ell}_B| \leq \varepsilon^3|M|$.

Proof. We write a recurrence relation to find the total number of bad edges E^{ℓ}_B in E_ℓ.

$$
\begin{aligned}
&\textsc{BadEdges}(\ell) = |E^{\ell}_{B_1}| + \textsc{BadEdges}(\ell-2) \\
&\quad\implies |E^{\ell}_B| \leq 2\varepsilon^4|M| + \textsc{BadEdges}(\ell-2) \qquad \ldots \text{by Lemma 2} \\
&\quad\implies |E^{\ell}_B| \leq 2\varepsilon^4|M| + 2\varepsilon^4|M| + \cdots + 2\varepsilon^4|M| \qquad \ldots \text{at most } (1/(4\varepsilon)) \text{ times.} \\
&\quad\implies |E^{\ell}_B| \leq \varepsilon^3|M|.
\end{aligned}
$$

The function $\textsc{BadEdges}(\ell')$ gives the total number of bad edges in E_ℓ from one function call $\textsc{Augment}(\ell', V_1, M')$. ◀

Now, we give a bound on the total number of augmentations during one function call $\textsc{Augment}(\ell, V, M)$. We say that an augmenting path is "good" if none of the edges in that augmenting path belong to E^{ℓ}_B.

▶ **Lemma 4.** *One function call* $\textsc{Augment}(\ell, V, M)$ *augments at least* $\varepsilon^{4\ell}/2$ *fraction of the total number of good* $(2\ell+1)$*-augmenting paths in* M*.*

Proof. Consider a length $(2\ell+1)$-augmentation of an augmenting path P (for instance, $P := au_1v_1u_2v_2u_3 \ldots v_{\ell-1}u_\ell v_\ell b$). There are at most $(1/\varepsilon^4)^\ell$ length $(2\ell+1)$-augmenting paths (considering edges in M and all edges in S during the recursive calls) with endpoint a (, and at most $(1/\varepsilon^4)^\ell$ length $(2\ell+1)$-augmenting paths with endpoint b). This is because, on any of the free vertices, at most $1/\varepsilon^4$ edges are stored in S during the function call $\textsc{Augment}(\ell, V, M)$, and the subsequent recursive calls. (See Figure 1 for instance.)

After the length $(2\ell+1)$-augmentation of P, none of the other $2/(\varepsilon^4)^\ell - 1$ length $(2\ell+1)$-augmenting paths can be augmented. (Figure 2 gives an illustration.) Thus, the total number of augmentations during one call $\textsc{Augment}(\ell, V, M)$ is at least $\frac{1}{2/(\varepsilon^4)^\ell}$ fraction of the total number of good $(2\ell+1)$-augmenting paths in M. ◀

▶ **Lemma 5.** *After* $\left(1/\varepsilon^{5\ell}\right)$ *function calls* $\textsc{Augment}(\ell, V, M)$*, there are at most* $\varepsilon^2|M|$ *edges remaining in* M *that are* $(2\ell+1)$*-augmentable.*

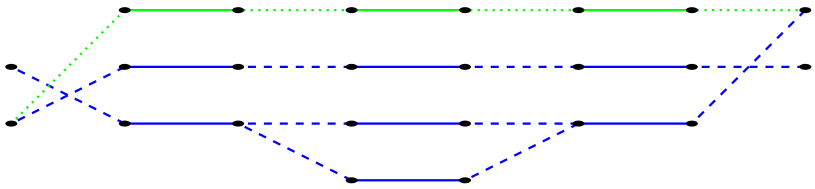


Figure 2 Suppose $\ell = 3$. Green solid lines and dotted lines represent a path P that was augmented, and blue solid lines and dashed lines represent the paths that can no longer be augmented due to that augmentation of path P.

Proof. Lemma 4 shows that, if there are k good $(2\ell+1)$-augmenting paths in M with respect to some M^*, then in each call $\textsc{Augment}(\ell, V, M)$, at least $\varepsilon^{4\ell}$ fraction of these good paths are augmented (ignoring the 1/2 factor). So, after $\log(1/\varepsilon^{4\ell})/\varepsilon^{4\ell}$ passes, the number of good $(2\ell+1)$-augmenting paths remaining in M are at most

$$k(1-\varepsilon^{4\ell})^{(\log(1/\varepsilon^{4\ell})/\varepsilon^{4\ell})} \leq \varepsilon^{4\ell} \cdot k \leq \varepsilon^{4\ell} \cdot |M| \leq \varepsilon^3 |M|.$$

Suppose that all the bad edges E_B^ℓ from the last function call $\textsc{Augment}(\ell, V, M)$ belong to distinct length $(2\ell+1)$-augmenting paths. Then, by Lemma 3, and by the bound on the number of good length $(2\ell+1)$-augmenting paths remaining, at most $2\varepsilon^3|M|$ length $(2\ell+1)$-augmenting paths remain in M.

Since we only consider augmenting paths of length at most $(1/\varepsilon)$, and each augmenting path of length $(1/\varepsilon)$ contains at most $(1/(2\varepsilon))$ edges in M, the total number of $(2\ell+1)$-augmentable edges remaining in M is at most $\varepsilon^2|M|$. ◀

As mentioned earlier, the function call $\textsc{Augment}(\ell, V, M)$ assumes that there are no shorter than length $(2\ell+1)$-augmenting paths. But the analysis does not require this assumption. This assumption is used in the proof of Lemma 2. But Lemma 2 anyways gives an overestimate on the upper bound for $\ell > 1$. Also, during the function call $\textsc{Augment}(\ell, V, M)$, if the algorithm comes across a shorter augmenting path, it is ignored. Thus, using Lemmas 1 and 5, we claim the following result.

▶ **Theorem 6.** *Algorithm 1 is a $(1/\varepsilon)^{O(1/\varepsilon)}$-pass $(1+\varepsilon)$-approximation deterministic algorithm for maximum matching on bipartite graphs.*

Note that during the function call $\textsc{Augment}(\ell, V, M)$, at most one edge is stored in S from a matched vertex to a free vertex. So, there are at most $2|M|$ edges stored in the semi-matchings at any stage. Thus, the algorithm uses $O(n \log n)$ space.

4 Algorithm on General Graphs

In this section, we present a $(1/\varepsilon)^{O(1/\varepsilon)}$-pass $(1+\varepsilon)$-approximation deterministic algorithm for maximum matching on general graphs. The high level idea for the algorithm on general graphs is same as the algorithm on bipartite graphs.

As already mentioned earlier, there do not exist "blossoms" in the bipartite graphs, and the existence of blossoms in the general graphs, makes it difficult to find augmenting paths. We address this issue in the following manner.

While finding the semi-matchings, instead of storing at most one edge on any matched vertex, the algorithm stores at most two edges, and the second edge is chosen carefully. The algorithm needs access to all prior semi-matchings it has stored during the function call

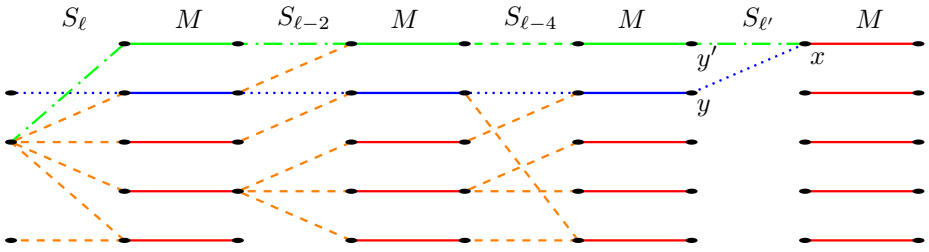


Figure 3 Blue solid and dotted lines represent the path p_{xy}, and green solid and dash-dot lines represent the path $p_{xy'}$. As these paths do not intersect in any vertex other than x, xy is added to $S_{\ell'}$.

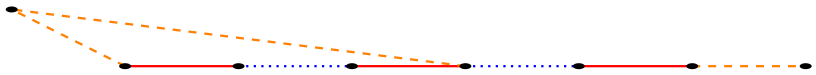


Figure 4 Consider the iteration to find length 7-augmentations in general graphs. All the edges represented by the dashed orange lines will be added to the semi-matching S_3, and in the subsequent iteration if we only consider those edges in M on which there are no edges stored in S_3 (there are none), then we wont be able to find the length 7-augmenting path. So we need to consider all the edges in M in the subsequent iterations. But then $V_1 \cap V(M) \neq \emptyset$.

AUGMENT(ℓ, V, M) to add the second edge. So, rather than using recursive calls inside the function call AUGMENT (ℓ, V, M), we iteratively find semi-matchings. Let $S_{\ell'}$ denote the semi-matching stored during the iteration to find the middle length $(2\ell'+1)$ path of any length $(2\ell+1)$-augmenting path. In this iteration, when the edge xy is read during a pass, such that $x \in V(M)$ and $y \in V_1$, suppose $\exists xy' \in S_{\ell'}$. Let P_{xy} (or $P_{xy'}$) denote the set of paths starting with xy (or xy'), alternating between some edge in M and some edge in $S_{\ell'+2i}$, for i going from 1 to $(\ell-\ell')/2$, ending in some edge in S_ℓ. If $\exists p_{xy} \in P_{xy}$, and $\exists p_{xy'} \in P_{xy'}$, such that p_{xy} and $p_{xy'}$ do not intersect in any vertex except x, then we add xy to $S_{\ell'}$. This ensures that when the middle length $(2\ell'+1)$ path of any length $(2\ell+1)$-augmenting path is found, there exist two non-intersecting paths of length $(\ell-\ell')$, such that each path contains one edge each from $M, S_{\ell'+2}, M, S_{\ell'+4}, \ldots, M, S_\ell$ (in that sequence), and they form length $(2\ell+1)$-augmenting path along with the middle length $(2\ell'+1)$ path. See Figure 3 for an illustration of the process of adding a second edge, incident on vertex x, to $S_{\ell'}$.

For general graphs, unlike bipartite graphs, we require that $V(M)$ and V_1 are not disjoint (see Figure 4 for a reason).

So, the algorithm needs to store directed semi-matchings instead of semi-matchings. Because $V(M)$ and V_1 are not disjoint, while adding an edge xy to any directed semi-matching, it is ensured that there exists a directed path starting with xy containing alternate edges in M and the directed semi-matchings previously stored, such that it does not visit x, thus avoiding formation of a cycle. (See Figures 5 and 6 for an illustration.)

Algorithm 2 gives a formal description.

Bad Edges

In the function call AUGMENT(ℓ, V, M), suppose we are in an iteration to find the middle length $(2\ell'+1)$ path of the length $(2\ell+1)$-augmenting paths in the matching matching M. Let $au_1v_1u_2v_2u_3 \ldots v_{\ell-1}u_\ell v_\ell b$ denote any length $(2\ell+1)$-augmenting path in M with respect to M^*. Then, $v_{(\ell-\ell')/2}u_{(\ell-\ell')/2+1}v_{(\ell-\ell')/2+1} \cdots\ u_{(\ell+\ell')/2}v_{(\ell+\ell')/2}\ u_{(\ell+\ell')/2+1}$ denotes

Figure 5 Red solid lines represent edges in M. Edge xy is not added to a Directed Semi-Matching because there does not exist a directed path starting from xy and not visiting x. The other orange dotted lines represent edges stored in Directed Semi-Matchings in previous iterations.

Figure 6 Red solid lines represent edges in M. Edge xy is added to a Directed Semi-Matching because there exists a directed path starting from xy and not visiting x. The other orange dotted lines represent edges stored in Directed Semi-Matchings in previous iterations.

the middle length $(2\ell' + 1)$ path of the augmenting path. Edge $u_{(\ell-\ell')/2+1}v_{(\ell-\ell')/2+1} \in M$ (or $u_{(\ell+\ell')/2}v_{(\ell+\ell')/2} \in M$) is called a bad edge under one of the following two conditions. (Note that in any iteration, V_1 denotes the set of free vertices.)

1. If there is no directed edge added to $S_{\ell'}$ which is incident from the vertex $u_{(\ell-\ell')/2+1}$ (or $v_{(\ell+\ell')/2}$) to some vertex in V_1, because there already were $(1/\varepsilon^4)$ incoming edges in $S_{\ell'}$ incident on the vertices in V_1 for all the edges incident from $u_{(\ell-\ell')/2+1}$ (or $v_{(\ell+\ell')/2}$).
2. If there is only one directed edge (not in M^*) added to $S_{\ell'}$ that is incident from the vertex $u_{(\ell-\ell')/2+1}$ (or $v_{(\ell+\ell')/2}$) to some vertex in V_1, and a second directed edge is not added to $S_{\ell'}$, because there already were $(1/\varepsilon^4)$ incoming edges in $S_{\ell'}$ incident on the vertices in V_1 for all other edges incident from $u_{(\ell-\ell')/2+1}$ (or $v_{(\ell+\ell')/2}$).

Note. A directed edge which is incident from the vertex $u_{(\ell-\ell')/2+1}$ (or $v_{(\ell+\ell')/2}$) to some vertex in V_1 may not be added to $S_{\ell'}$ also for one of the following two other reasons. First, if the addition of such an edge is going to form a cycle, and second, if the addition of such an edge does not produce two non-intersecting paths from the vertex $u_{(\ell-\ell')/2+1}$ to the unmatched vertices. We argue that the edge $u_{(\ell-\ell')/2+1}v_{(\ell-\ell')/2+1} \in M$ (or $u_{(\ell+\ell')/2}v_{(\ell+\ell')/2} \in M$) is not bad due to either of the above mentioned reasons.

In the first case, such an edge can be ignored, as there already exists a shorter directed path from the vertex $u_{(\ell-\ell')/2+1}$ to some unmatched vertex. The second case needs careful attention. Let $u_{(\ell-\ell')/2+1}v_{(\ell-\ell')/2} \in M^*$ be the edge not added to $S_{\ell'}$, and let $u_{(\ell-\ell')/2+1}v$ be the only edge that is added to $S_{\ell'}$. Either $v_{(\ell-\ell')/2}u_{(\ell-\ell')/2} \in M^*$ is a bad edge or not. If it is a bad edge, we can ignore the edge $u_{(\ell-\ell')/2+1}v_{(\ell-\ell')/2+1} \in M$. Otherwise, there are two sub cases.

1. There are two directed edges from $u_{(\ell-\ell')/2}$ in $S_{\ell'+2}$, which means there are two non-intersecting directed paths p_1 and p_2, from $u_{(\ell-\ell')/2}$ to unmatched vertices. If there exists a directed path from v which intersects p_1 or p_2 or neither of them, then the algorithm should be able to add $u_{(\ell-\ell')/2+1}v_{(\ell-\ell')/2}$ to $S_{\ell'}$. If there exists a directed path from v which intersects both p_1 and p_2, then the edge $u_{(\ell-\ell')/2+1}v_{(\ell-\ell')/2+1}$ is not a bad edge, as it can be used in the length $(2\ell + 1)$-augmentation.
2. There is only one directed edge from $u_{(\ell-\ell')/2}$ in $S_{\ell'+2}$. Suppose there exists a directed path from $u_{(\ell-\ell')/2}$ to unmatched vertices, which does not contain any bad edges (If there does not exist such a path, then we can ignore the edge $u_{(\ell-\ell')/2+1}v_{(\ell-\ell')/2+1} \in M$.)

Algorithm 2 Deterministic Algorithm on General Graphs.

Find Maximal Matching M in the first pass.
for $\ell = 1$ `to` $1/(2\varepsilon)$ **do** ▷ To Remove almost all $(2\ell+1)$-augmenting paths
  for `Phase` $p = 1$ `to` $1/\varepsilon^{5\ell}$ **do**
    $M \leftarrow \text{AUGMENT}(\ell, V, M)$.
Output M.
function AUGMENT(ℓ, V, M) ▷ Function of length $(2\ell+1)$-augmentations.
  $\ell' := \ell$, $V_1 \leftarrow V \setminus V(M)$.
  while $\ell' \geq 0$ **do**
    $S_{\ell'} \leftarrow \emptyset$.
    for all edges ab **do** ▷ One pass to find a directed semi-matching
      for all $xy \in \{ab, ba\}$ **do** ▷ Directed edges ab and ba.
        if $x \in V(M)$, $y \in V_1$, and $\exists$ a directed path starting with xy, with one edge each from $M, S_{\ell'+2}, M, S_{\ell'+4}, \ldots, M, S_\ell$ (in that sequence) that does not visit x **then**
          if $\deg^O_{S_{\ell'}}(x) = 0$ and $\deg^I_{S_{\ell'}}(y) < 1/\varepsilon^4$ **then**
            $S_{\ell'} \leftarrow S_{\ell'} \cup \{xy\}$ ▷ Add directed edge xy.
          else if $\deg^O_{S_{\ell'}}(x) = 1$ and $\deg^I_{S_{\ell'}}(y) < 1/\varepsilon^4$ **then**
            Suppose $\exists xy' \in S_{\ell'}$.
            Let P_{xy} (or $P_{xy'}$) denote the set of directed paths starting with xy (or xy'), with one edge each from $M, S_{\ell'+2}, M, S_{\ell'+4}, \ldots, M, S_\ell$ (in that sequence).
            if $\exists p_{xy} \in P_{xy}$, and $\exists p_{xy'} \in P_{xy'}$, such that p_{xy} and $p_{xy'}$ do not intersect in any vertex except x **then**
              $S_{\ell'} \leftarrow S_{\ell'} \cup \{xy\}$ ▷ Add directed edge xy.
    $V_1 \leftarrow \emptyset$.
    for all edges $uv \in M$ **do**
      if $\exists ua \in S_{\ell'}$ **then** ▷ Directed edge ua.
        $V_1 \leftarrow V_1 \cup \{v\}$. ▷ Set of free vertices for the next iteration.
    $\ell' := \ell' - 2$ ▷ End of While Loop
  $(2\ell+1)$-augment M greedily using edges in $S_\ell, S_{\ell-2}, \ldots$.
return M.

This implies that there is a directed path from v that intersects this directed path, and hence the edge $u_{(\ell-\ell')/2+1}v_{(\ell-\ell')/2+1}$ is not a bad edge, as it can be used in the length $(2\ell+1)$-augmentation (along with the directed edge $u_{(\ell-\ell')/2+1}v \in S_{\ell'}$).

4.1 Analysis

We begin the analysis for any length $(2\ell+1)$-augmentations for the matching M. Let E_ℓ denote the set of $(2\ell+1)$-augmentable edges in M with respect to an optimal matching M^*, such that $|E_\ell|$ is maximum. Also, let $E^\ell_{B_1}$ denote the outermost bad edges of any length $(2\ell+1)$-augmenting path. The following lemma gives an upper bound on $E^\ell_{B_1}$.

▶ **Lemma 7.** $|E^\ell_{B_1}| \leq 4\varepsilon^4|M|$.

Proof. Without loss of generality, lets consider the edge u_1v_1 in a length $(2\ell+1)$-augmenting path $au_1v_1u_2v_2u_3 \ldots v_{\ell-1}u_\ell v_\ell b$. If u_1v_1 is a bad edge, then by the definition of a bad edge, there are $(1/\varepsilon^4)$ edges in S_ℓ responsible. So, $(1/\varepsilon^4)|E^\ell_{B_1}| \leq |S_\ell|$. For $\ell > 1$, each edge in M

can have at most two outgoing edges in S_ℓ incident on its endpoints (as there are no length 3-augmentable edges in M), i.e. $|S_\ell| \leq 2|M|$. Hence, $|E^\ell_{B_1}|/\varepsilon^4 \leq 2|M|$.

For $\ell = 1$, each edge in M can have at most two outgoing edges in S_ℓ incident from each its of endpoints, i.e. $|S_\ell| \leq 4|M|$. Hence, $|E^\ell_{B_1}|/\varepsilon^4 \leq 4|M|$. ◀

Let E^ℓ_B denote the set of the bad edges, formed during the iterations inside the function call $\textsc{Augment}(\ell, V, M)$, in all the length $(2\ell+1)$-augmenting paths. The following lemma bounds the total number of bad edges E^ℓ_B in E_ℓ.

▶ **Lemma 8.** $|E^\ell_B| \leq \varepsilon^3|M|$.

Proof. We write a recurrence relation to find the total number of bad edges E^ℓ_B in E_ℓ.

$$\begin{aligned}
&\textsc{BadEdges}(\ell) = |E^\ell_{B_1}| + \textsc{BadEdges}(\ell-2)\\
&\quad\implies |E^\ell_B| \leq 2\varepsilon^4|M| + \textsc{BadEdges}(\ell-2) \qquad \dots \text{by Lemma 7}\\
&\quad\implies |E^\ell_B| \leq 2\varepsilon^4|M| + 2\varepsilon^4|M| + \dots + 2\varepsilon^4|M| \qquad \dots \text{at most } (1/(4\varepsilon)) \text{ times.}\\
&\quad\implies |E^\ell_B| \leq \varepsilon^3|M|.
\end{aligned}$$

The function $\textsc{BadEdges}(\ell')$ gives the total number of bad edges in E_ℓ formed during the iteration to find middle length $(2\ell'+1)$ path of any length $(2\ell+1)$-augmenting path. ◀

Now, we give a bound on the total number of augmentations during one function call $\textsc{Augment}(\ell, V, M)$. We say that an augmenting path is "good" if none of the edges in that augmenting path belong to E^ℓ_B.

▶ **Lemma 9.** *One function call* $\textsc{Augment}(\ell, V, M)$ *augments at least* $\varepsilon^{4\ell}/2$ *fraction of the total number of good* $(2\ell+1)$*-augmenting paths in* M*.*

Proof. Consider a length $(2\ell+1)$-augmentation of an augmenting path P (for instance, $P := au_1v_1u_2v_2u_3 \dots v_{\ell-1}u_\ell v_\ell b$). There are at most $(1/\varepsilon^4)^\ell$ length $(2\ell+1)$-augmenting paths (considering edges in $M, S_\ell, S_{\ell-2}, \dots$) with the endpoint a (, and at most $(1/\varepsilon^4)^\ell$ length $(2\ell+1)$-augmenting paths with the endpoint b). This is because, on any of the free vertices, at most $1/\varepsilon^4$ edges are stored in any S_i during the function call $\textsc{Augment}(\ell, V, M)$.

After the length $(2\ell+1)$-augmentation of P, none of the other $2/(\varepsilon^4)^\ell - 1$ length $(2\ell+1)$-augmenting paths can be augmented. Thus, the total number of augmentations during one call $\textsc{Augment}(\ell, V, M)$ is at least $\frac{1}{2/(\varepsilon^4)^\ell}$ fraction of the total number of good $(2\ell+1)$-augmenting paths in M. ◀

▶ **Lemma 10.** *After* $(1/\varepsilon^{5\ell})$ *function calls* $\textsc{Augment}(\ell, V, M)$*, the number of* $(2\ell+1)$*-augmentable edges remaining in* M *is at most* $\varepsilon^2|M|$*.*

Proof. Lemma 9 shows that, if there are k good $(2\ell+1)$-augmenting paths in M with respect to some M^*, then in each call $\textsc{Augment}(\ell, V, M)$, at least $\varepsilon^{4\ell}$ fraction of these good paths are augmented (ignoring the 1/2 factor). So, after $\log(1/\varepsilon^{4\ell})/\varepsilon^{4\ell}$ passes, the number of good $(2\ell+1)$-augmenting paths remaining in M are at most

$$k(1-\varepsilon^{4\ell})^{(\log(1/\varepsilon^{4\ell})/\varepsilon^{4\ell})} \leq \varepsilon^{4\ell} \cdot k \leq \varepsilon^{4\ell} \cdot |M| \leq \varepsilon^3|M|.$$

Suppose all the bad edges E^ℓ_B from the last function call $\textsc{Augment}(\ell, V, M)$ belong to distinct length $(2\ell+1)$-augmenting paths. Then, by Lemma 8, and by the bound on the number of good length $(2\ell+1)$-augmenting paths remaining, at most $2\varepsilon^3|M|$ length $(2\ell+1)$-augmenting paths remain in M.

Since we only consider augmenting paths of length at most $(1/\varepsilon)$, and each augmenting path of length $(1/\varepsilon)$ contains at most $(1/(2\varepsilon))$ edges in M, the total number of $(2\ell+1)$-augmentable edges remaining in M is at most $\varepsilon^2|M|$. ◀

As mentioned earlier, the function call $\textsc{Augment}(\ell, V, M)$ assumes that there are no shorter than length $(2\ell+1)$-augmenting paths. But the analysis does not require this assumption. This assumption is used in the proof of Lemma 7. But Lemma 7 anyways gives an overestimate on the upper bound for $\ell > 1$. Also, during the function call $\textsc{Augment}(\ell, V, M)$, if the algorithm comes across a shorter augmenting path, it is ignored. Thus, using Lemmas 1 and 10, we claim the following result.

▶ **Theorem 11.** *Algorithm 2 is a $(1/\varepsilon)^{O(1/\varepsilon)}$-pass $(1+\varepsilon)$-approximation deterministic algorithm for maximum matching on general graphs.*

Note that during the function call $\textsc{Augment}(\ell, V, M)$, at most two edges are stored in any S_ℓ from a matched vertex to free vertices. So, there are at most $2\ell|M|$ edges stored in the directed semi-matchings at any stage. Thus, the algorithm uses $O\left(\frac{1}{\varepsilon} \cdot n \log n\right) = O(n \log n)$ space.

References

1. Kook Jin Ahn and Sudipto Guha. Linear Programming in the Semi-streaming Model with Application to the Maximum Matching Problem. *Inf. Comput.*, 222:59–79, January 2013. doi:10.1016/j.ic.2012.10.006.
2. Sepehr Assadi, Sanjeev Khanna, and Yang Li. On Estimating Maximum Matching Size in Graph Streams. In *Proc. 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1723–1742, 2017. doi:10.1137/1.9781611974782.113.
3. Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Maximum Matchings in Dynamic Graph Streams and the Simultaneous Communication Model. In *Proc. 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1345–1364, 2016. URL: http://dl.acm.org/citation.cfm?id=2884435.2884528.
4. Aaron Bernstein and Cliff Stein. Faster Fully Dynamic Matchings with Small Approximation Ratios. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 692–711, 2016. doi:10.1137/1.9781611974331.ch50.
5. Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Deterministic Fully Dynamic Data Structures for Vertex Cover and Matching. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 785–804, 2015. doi:10.1137/1.9781611973730.54.
6. Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. Fully Dynamic Approximate Maximum Matching and Minimum Vertex Cover in $O(\log^3 n)$ Worst Case Update Time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 470–489, 2017. doi:10.1137/1.9781611974782.30.
7. Niv Buchbinder, Danny Segev, and Yevgeny Tkach. Online Algorithms for Maximum Cardinality Matching with Edge Arrivals. In *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, pages 22:1–22:14, 2017. doi:10.4230/LIPIcs.ESA.2017.22.
8. Marc Bury and Chris Schwiegelshohn. Sublinear Estimation of Weighted Matchings in Dynamic Data Streams. In *Proc. 23rd Annual European Symposium on Algorithms*, pages 263–274, 2015. doi:10.1007/978-3-662-48350-3_23.

9 Amit Chakrabarti and Sagar Kale. Submodular maximization meets streaming: matchings, matroids, and more. *Mathematical Programming*, 154(1):225–247, 2015. doi:10.1007/s10107-015-0900-7.

10 Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming Algorithms for Submodular Function Maximization. In *Proc. 42nd International Colloquium on Automata, Languages and Programming*, pages 318–330, 2015. doi:10.1007/978-3-662-47672-7_26.

11 Ashish Chiplunkar, Sumedh Tirodkar, and Sundar Vishwanathan. On Randomized Algorithms for Matching in the Online Preemptive Model. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 325–336, 2015. doi:10.1007/978-3-662-48350-3_28.

12 Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via Sampling with Applications to Finding Matchings and Related Problems in Dynamic Graph Streams. In *Proc. 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1326–1344, 2016. URL: http://dl.acm.org/citation.cfm?id=2884435.2884527.

13 Sebastian Eggert, Lasse Kliemann, Peter Munstermann, and Anand Srivastav. Bipartite Matching in the Semi-streaming Model. *Algorithmica*, 63(1):490–508, 2012. doi:10.1007/s00453-011-9556-8.

14 Leah Epstein, Asaf Levin, Julian Mestre, and Danny Segev. Improved Approximation Guarantees for Weighted Matching in the Semi-streaming Model. *SIAM Journal on Discrete Mathematics*, 25(3):1251–1265, 2011. doi:10.1137/100801901.

15 Leah Epstein, Asaf Levin, Danny Segev, and Oren Weimann. Improved Bounds for Online Preemptive Matching. In *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, Kiel, Germany*, pages 389–399, 2013. doi:10.4230/LIPIcs.STACS.2013.389.

16 H. Esfandiari, M. Hajiaghayi, and M. Monemizadeh. Finding Large Matchings in Semi-Streaming. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 608–614, December 2016. doi:10.1109/ICDMW.2016.0092.

17 Hossein Esfandiari, Mohammad T. Hajiaghayi, Vahid Liaghat, Morteza Monemizadeh, and Krzysztof Onak. Streaming Algorithms for Estimating the Matching Size in Planar Graphs and Beyond. In *Proc. 26th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1217–1233, 2015. URL: http://dl.acm.org/citation.cfm?id=2722129.2722210.

18 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2):207–216, December 2005. doi:10.1016/j.tcs.2005.09.013.

19 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In *Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 468–485, 2012. URL: http://dl.acm.org/citation.cfm?id=2095116.2095157.

20 Sagar Kale and Sumedh Tirodkar. Maximum Matching in Two, Three, and a Few More Passes Over Graph Streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, pages 15:1–15:21, 2017. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.15.

21 Michael Kapralov. Better bounds for matchings in the streaming model. In *Proc. 24th Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2013.

22 Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Approximating Matching Size from Random Streams. In *Proc. 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 734–751, 2014. URL: http://dl.acm.org/citation.cfm?id=2634074.2634129.

23 R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An Optimal Algorithm for On-line Bipartite Matching. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*, STOC '90, pages 352–358, New York, NY, USA, 1990. ACM.

24 Christian Konrad. Maximum Matching in Turnstile Streams. In *Proc. 23rd Annual European Symposium on Algorithms*, pages 840–852, 2015. doi:10.1007/978-3-662-48350-3_70.

25 Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum Matching in Semi-Streaming with Few Passes. *CoRR*, abs/1112.0184, 2014. URL: http://arxiv.org/abs/1112.0184.

26 Mohammad Mahdian and Qiqi Yan. Online Bipartite Matching with Random Arrivals: An Approach Based on Strongly Factor-revealing LPs. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 597–606, New York, NY, USA, 2011. ACM. doi:10.1145/1993636.1993716.

27 Andrew McGregor. Problem 60: Single-Pass Unweighted Matchings. http://sublinear.info/index.php?title=Open_Problems:60. Accessed: 2018-02-10.

28 Andrew McGregor. Finding graph matchings in data streams. In *Proc. 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 170–181, 2005. doi:10.1007/11538462_15.

29 Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{|V|}|E|)$ Algorithm for Finding Maximum Matching in General Graphs. In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science*, SFCS '80, pages 17–27, Washington, DC, USA, 1980. IEEE Computer Society. doi:10.1109/SFCS.1980.12.

30 Ami Paz and Gregory Schwartzman. A $(2+\epsilon)$-Approximation for Maximum Weight Matching in the Semi-Streaming Model. In *Proc. 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2153–2161, 2017. doi:10.1137/1.9781611974782.140.

31 A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer, 2003.

32 Shay Solomon. Fully Dynamic Maximal Matching in Constant Update Time. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 325–334, 2016. doi:10.1109/FOCS.2016.43.

33 Mariano Zelke. Weighted matching in the semi-streaming model. In *Proc. 25th International Symposium on Theoretical Aspects of Computer Science*, pages 669–680, 2008.

Sketching, Streaming, and Fine-Grained Complexity of (Weighted) LCS

Karl Bringmann
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany
kbringma@mpi-inf.mpg.de

Bhaskar Ray Chaudhury
Max Planck Institute for Informatics, Saarland Informatics Campus, Graduate School of Computer Science, Saarbrücken, Germany
braycha@mpi-inf.mpg.de

Abstract

We study sketching and streaming algorithms for the Longest Common Subsequence problem (LCS) on strings of small alphabet size $|\Sigma|$. For the problem of deciding whether the LCS of strings x, y has length at least L, we obtain a sketch size and streaming space usage of $\mathcal{O}(L^{|\Sigma|-1} \log L)$. We also prove matching unconditional lower bounds.

As an application, we study a variant of LCS where each alphabet symbol is equipped with a weight that is given as input, and the task is to compute a common subsequence of maximum total weight. Using our sketching algorithm, we obtain an $\mathcal{O}(\min\{nm, n+m^{|\Sigma|}\})$-time algorithm for this problem, on strings x, y of length n, m, with $n \geq m$. We prove optimality of this running time up to lower order factors, assuming the Strong Exponential Time Hypothesis.

2012 ACM Subject Classification Theory of computation → Communication complexity, Theory of computation → Design and analysis of algorithms

Keywords and phrases algorithms, SETH, communication complexity, run-length encoding

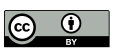
Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.40

1 Introduction

1.1 Sketching and Streaming LCS

In the Longest Common Subsequence problem (LCS) we are given strings x and y and the task is to compute a longest string z that is a subsequence of both x and y. This problem has been studied extensively, since it has numerous applications in bioinformatics (e.g. comparison of DNA sequences [5]), natural language processing (e.g. spelling correction [40, 49]), file comparison (e.g. the UNIX `diff` utility [23, 38]), etc. Motivated by big data applications, in the first part of this paper we consider space-restricted settings as follows:

- *LCS Sketching:* Alice is given x and Bob is given y. Both also are given a number L. Alice and Bob compute sketches $\mathrm{sk}_L(x)$ and $\mathrm{sk}_L(y)$ and send them to a third person, the referee, who decides whether the LCS of x and y is at least L. The task is to minimize the size of the sketch (i.e., its number of bits) as well as the running time of Alice and Bob (encoding) and of the referee (decoding).
- *LCS Streaming:* We are given L, and we scan the string x from left to right once, and then the string y from left to right once. After that, we need to decide whether the LCS of x and y is at least L. We want to minimize the space usage as well as running time.



Analogous problem settings for the related edit distance have found surprisingly good solutions after a long line of work [11, 29, 46, 16]. For LCS, however, strong unconditional lower bounds are known for sketching and streaming: Even for $L = 4$ the sketch size and streaming memory must be $\Omega(n)$ bits, since the randomized communication complexity of this problem is $\Omega(n)$ [47]. Similarly strong results hold even for *approximating* the LCS length [47], see also [35]. However, these impossibility results construct strings over alphabet size $\Theta(n)$.

In contrast, in this paper we focus on strings x, y defined over a fixed alphabet Σ (of constant size). This is well motivated, e.g., for binary files ($\Sigma = \{0, 1\}$), DNA sequences ($\Sigma = \{A, G, C, T\}$), or English text ($\Sigma = \{a, \ldots, z, A, \ldots, Z\}$ plus punctuation marks). We therefore *suppress factors depending only on* $|\Sigma|$ in $\mathcal{O}$-notation throughout the whole paper. Surprisingly, this setting was ignored in the sketching and streaming literature so far; the only known upper bounds also work in the case of large alphabet and are thus $\Omega(n)$.

Before stating our first main result we define a run in a string as the non extendable repetition of a character. For example the string *baaabc* has a run of character a of length 3. Our first main result is the following deterministic sketch.

▶ **Theorem 1.** *Given a string x of length n over alphabet Σ and an integer L, we can compute a subsequence $C_L(x)$ of x such that (1) $|C_L(x)| = \mathcal{O}(L^{|\Sigma|})$, (2) $C_L(x)$ consists of $\mathcal{O}(L^{|\Sigma|-1})$ runs of length at most L, and (3) any string y of length at most L is a subsequence of x if and only if it is a subsequence of $C_L(x)$. Moreover, $C_L(x)$ is computed by a one-pass streaming algorithm with memory $\mathcal{O}(L^{|\Sigma|-1} \log L)$ and running time $\mathcal{O}(1)$ per symbol of x.*

Note that we can store $C_L(x)$ using $\mathcal{O}(L^{|\Sigma|-1} \log L)$ bits, since each run can be encoded using $\mathcal{O}(\log L)$ bits. This directly yields a solution for *LCS sketching*, where Alice and Bob compute the sketches $\mathrm{sk}_L(x) = C_L(x)$ and $\mathrm{sk}_L(y) = C_L(y)$ and the referee computes an LCS of $C_L(x)$ and $C_L(y)$. If this has length at least L then also x, y have LCS length at least L. Similarly, if x, y have an LCS z of length at least L, then z is also a subsequence of $C_L(x)$ and $C_L(y)$, and thus their LCS length is at least L, showing correctness. The sketch size is $\mathcal{O}(L^{|\Sigma|-1} \log L)$ bits, the encoding time is $\mathcal{O}(n)$, and the decoding time is $\mathcal{O}(L^{2|\Sigma|})$, as LCS can be computed in quadratic time in the string length $\mathcal{O}(L^{|\Sigma|})$.

We similarly obtain an algorithm for *LCS streaming* by computing $C_L(x)$ and then $C_L(y)$ and finally computing an LCS of $C_L(x)$ and $C_L(y)$. The space usage of this streaming algorithm is $\mathcal{O}(L^{|\Sigma|-1} \log L)$, and the running time is $\mathcal{O}(1)$ per symbol of x and y, plus $\mathcal{O}(L^{2|\Sigma|})$ for the last step.

These size, space, and time bounds are surprisingly good for $|\Sigma| = 2$, but quickly deteriorate with larger alphabet size. For very large alphabet size, this deterioration was to be expected due to the $\Omega(n)$ lower bound for $|\Sigma| = \Theta(n)$ from [47]. We further show that this deterioration is necessary by proving optimality of our sketch in several senses:

- We show that for any L, Σ there exists a string x (of length $\mathcal{O}(L^{|\Sigma|})$) such that no string x' of length $o(L^{|\Sigma|})$ has the same set of subsequences of length at most L. Similarly, this string x cannot be replaced by any string consisting of $o(L^{|\Sigma|-1})$ runs without affecting the set of subsequences of length at most L. This shows optimality of Theorem 1 among sketches that replace x by another string x' (not necessarily a subsequence of x) and then compute an LCS of x' and y. See Theorem 4.
- More generally, we study the *Subsequence Sketching* problem: Alice is given a string x and number L and computes $\mathrm{sk}_L(x)$. Bob is then given $\mathrm{sk}_L(x)$ and a string y of length L and decides whether y is a subsequence of x. Observe that any solution for LCS sketching or streaming with size/memory $S = S(L, \Sigma)$ yields a solution for subsequence sketching

with sketch size S.[1] Hence, any lower bound for subsequence sketching yields a lower bound for LCS sketching and streaming. We show that any deterministic subsequence sketch has size $\Omega(L^{|\Sigma|-1}\log L)$ in the worst case over all strings x. This matches the run-length encoding of $C_L(x)$ even up to the $\log L$-factor. If we restrict to strings of length $\Theta(L^{|\Sigma|-1})$, we still obtain a sketch size lower bound of $\Omega(L^{|\Sigma|-1})$. See Theorem 7.

- Finally, randomization does not help either: We show that any randomized subsequence sketch, where Bob may err in deciding whether y is a subsequence of x with small constant probability, has size $\Omega(L^{|\Sigma|-1})$, even restricted to strings x of length $\Theta(L^{|\Sigma|-1})$. See Theorem 10.

We remark that Theorem 1 only makes sense if $L \ll n$. Although this is not the best motivated regime of LCS in practice, it corresponds to testing whether x and y are "very different" or "not very different". This setting naturally occurs, e.g., if one string is much longer than the other, since then $L \leq m \ll n$. We therefore think that studying this regime is justified for the fundamental problem LCS.

1.2 WLCS: In between min-quadratic and rectangular time

As an application of our sketch, we determine the (classic, offline) time complexity of a weighted variant of LCS, which we discuss in the following.

A textbook dynamic programming algorithm computes the LCS of given strings x, y of length n in time $\mathcal{O}(n^2)$. A major result in fine-grained complexity shows that further improvements by polynomial factors would refute the Strong Exponential Time Hypothesis (SETH) [1, 13] (see Section 5 for a definition). In case x and y have different lengths n and m, with $n \geq m$, Hirschberg's algorithm computes their LCS in time $\mathcal{O}((n+m^2)\log n)$ [22], and this is again near-optimal under SETH. This running time could be described as "min-quadratic", as it is quadratic in the minimum of the two string lengths. In contrast, many other dynamic programming type problems have "rectangular" running time[2] $\widetilde{\mathcal{O}}(nm)$, with a matching lower bound of $(nm)^{1-o(1)}$ under SETH, e.g., Fréchet distance [4, 12], dynamic time warping [1, 13], and regular expression pattern matching [43, 10].

Part of this paper is motivated by the intriguing question whether there are problems with *intermediate* running time, between "min-quadratic" and "rectangular". Natural candidates are generalizations of LCS, such as the weighted variant *WLCS* as defined in [1]: Here we have an additional *weight function* $W\colon \Sigma \to \mathbb{N}$, and the task is to compute a common subsequence of x and y with maximum total weight. This problem is a natural variant of LCS that, e.g., came up in a SETH-hardness proof of LCS [1]. It is not to be confused with other weighted variants of LCS that have been studied in the literature, such as a statistical distance measure where given the probability of every symbol's occurrence at every text location the task is to find a long and likely subsequence [6, 18], a variant of LCS that favors consecutive matches [36], or edit distance with given operation costs [13].

Clearly, WLCS inherits the hardness of LCS and thus requires time $(n+m^2)^{1-o(1)}$. However, the matching upper bound $\widetilde{\mathcal{O}}(n+m^2)$ given by Hirschberg's algorithm only works as long as the function W is fixed (then the hidden constant depends on the largest weight). Here, we focus on the variant where the weight function W is part of the input. In this case, the basic $\mathcal{O}(nm)$-time dynamic programming algorithm is the best known.

[1] For LCS sketching this argument only uses that we can check whether y is a subsequence of x by testing whether the LCS length of x and y is $|y|$. For LCS streaming we use the memory state right after reading x as the sketch $\mathrm{sk}_L(x)$ and then use the same argument.

[2] By $\widetilde{\mathcal{O}}$-notation we ignore factors of the form $\mathrm{polylog}(n)$.

Our second main result is to settle the time complexity of WLCS in terms of n and m for any fixed constant alphabet Σ, up to lower order factors $n^{o(1)}$ and assuming SETH.

▶ **Theorem 2.** *WLCS can be solved in time* $\mathcal{O}(\min\{nm, n+m^{|\Sigma|}\})$. *Assuming SETH, WLCS requires time* $\min\{nm, n+m^{|\Sigma|}\}^{1-o(1)}$, *even restricted to* $n = \Theta(m^\alpha)$ *and* $|\Sigma| = \sigma$ *for any constants* $\alpha \in \mathbb{R}, \alpha \geq 1$ *and* $\sigma \in \mathbb{N}, \sigma \geq 2$.

In particular, for $|\Sigma| > 2$ the time complexity of WLCS is indeed "intermediate", in between "min-quadratic" and "rectangular"! To the best of our knowledge, this is the first result of fine-grained complexity establishing such an intermediate running time.

To prove Theorem 2 we first observe that the usual $\mathcal{O}(nm)$ dynamic programming algorithm also works for WLCS. For the other term $n + m^{|\Sigma|}$, we compress x by running the sketching algorithm from Theorem 1 with $L = m$. This yields a string $x' = C_m(x)$ of length $\mathcal{O}(m^{|\Sigma|})$ such that WLCS has the same value on (x, y) and (x', y), since every subsequence of length at most m of x is also a subsequence of x', and vice versa. Running the $\mathcal{O}(nm)$-time algorithm on (x', y) would yield total time $\mathcal{O}(n + m^{|\Sigma|+1})$, which is too slow by a factor m. To obtain an improved running time, we use the fact that x' consists of $\mathcal{O}(m^{|\Sigma|-1})$ runs. We design an algorithm for WLCS on a run-length encoded string x' consisting of r runs and an uncompressed string y of length m running time $\mathcal{O}(rm)$. This generalizes algorithms for LCS with one run-length encoded string [7, 20, 37]. Together, we obtain time $\mathcal{O}(\min\{nm, n + m^{|\Sigma|}\})$. We then show a matching SETH-based lower bound by combining our construction of incompressible strings from our sketching lower bounds (Theorem 4) with the by-now classic SETH-hardness proof of LCS [1, 13].

1.3 Further Related Work

Analyzing the running time in terms of multiple parameters like n, m, L has a long history for LCS [8, 9, 19, 22, 24, 26, 42, 44, 51]. Recently tight SETH-based lower bounds have been shown for all these algorithms [14]. In the second part of this paper, we perform a similar complexity analysis on a weighted variant of LCS. This follows the majority of recent work on LCS, which focused on transferring the early successes and techniques to more complicated problems, such as longest common increasing subsequence [39, 33, 52, 17], tree LCS [41], and many more generalizations and variants of LCS, see, e.g., [32, 15, 48, 28, 3, 34, 30, 21, 45, 25]. For brevity, here we ignore the equally vast literature on the closely related edit distance.

1.4 Notation

For a string x of length n over alphabet Σ, we write $x[i]$ for its i-th symbol, $x[i \dots j]$ for the substring from the i-th to j-th symbol, and $|x|$ for its length. For $c \in \Sigma$ we write $|x|_c := |\{i \mid x_i = c\}|$. For strings x, y we write $x \circ y$ for their concatenation, and for $k \in \mathbb{N}$ we write x^k for the k-fold repetition $x \circ \dots \circ x$. A subsequence of x is any string of the form $y = x[i_1] \circ x[i_2] \circ \dots \circ x[i_\ell]$ with $1 \leq i_1 < i_2 < \dots < i_\ell \leq |x|$; in this case we write $y \preceq x$. A *run* in x is a maximal substring $x[i \dots j] = c^{j-i+1}$, consisting of a single alphabet letter $c \in \Sigma$. Recall that we suppress factors depending only on $|\Sigma|$ in $\mathcal{O}$-notation.

2 Sketching LCS

In this section design a sketch for LCS, proving Theorem 1. Consider any string z defined over alphabet $S \subseteq \Sigma$. We call z a (q, S)-*permutation string* if we can partition $z = z^{(1)} \circ z^{(2)} \circ \dots \circ z^{(q)}$ such that each $z^{(i)}$ contains each symbol in S at least once. Observe that a (q, S) permutation string contains any string y of length at most q over the alphabet S as a subsequence.

Algorithm 1 Outline for computing $C_L(x)$ given a string x and an integer L.

initialize $C_L(x)$ as the empty string
for all i from 1 to $|x|$ **do**
  if for all $S \subseteq \Sigma$ with $x[i] \in S$, no suffix of $C_L(x)$ is an (L,S)-permutation string **then**
    set $C_L(x) \leftarrow C_L(x) \circ x[i]$
return $C_L(x)$

▶ **Claim 3.** *Consider any string $x = x' \circ c \circ x''$, where x', x'' are strings over alphabet Σ and $c \in \Sigma$. Let $S \subseteq \Sigma$. If some suffix of x' is an (L,S)-permutation string and $c \in S$, then for all strings y of length at most L we have $y \preceq x$ if and only if $y \preceq x' \circ x''$.*

Proof. The "if"-direction is immediate. To prove the "only if", consider any subsequence y of x of length $d \leq L$ and let $y = x[i_1] \circ x[i_2] \circ \ldots \circ x[i_d]$. Let ℓ and r be the length of x' and x'', respectively. If $i_k \neq \ell + 1$ for all $1 \leq k \leq d$, then clearly $y \preceq x' \circ x''$. Thus, assume that $i_k = \ell + 1$ for some k. Let a be minimal such that $x[a \ldots \ell]$ only contains symbols in S. By assumption, $x[a \ldots \ell]$ is an (L,S)-permutation string, and $c = x[\ell + 1] \in S$. Let $j \geq 1$ be the minimum index such that $x[i_j] \ldots x[i_k]$ only contains symbols in S. Since j is minimal, $x[i_{j-1}] \notin S$ and thus $i_b < a$ for all $b < j$. Therefore $x[i_1] \circ x[i_2] \circ \ldots \circ x[i_{j-1}] \preceq x[0 \ldots a-1]$. Since $x[a \ldots \ell]$ is an (L,S)-permutation string and $|x[i_j] \circ \ldots \circ x[i_k]| \leq d \leq L$, it follows that $x[i_j] \circ \ldots \circ x[i_k]$ is a subsequence of $x[a \ldots \ell]$. Hence, $x[i_1] \circ \ldots \circ x[i_k] \preceq x'$ and $x[i_{k+1}] \circ \ldots \circ x[i_d] \preceq x''$, and thus $y \preceq x' \circ x''$. ◀

The above claim immediately gives rise to the following one-pass streaming algorithm.

By Claim 3, the string $C_L(x)$ returned by this algorithm satisfies the subsequence property (3) of Theorem 1. Note that any run in $C_L(x)$ has length at most L, since otherwise for $S = \{c\}$ we would obtain an (L,S)-permutation string followed by another symbol c, so that Claim 3 would apply. We now show the upper bounds on the length and the number of runs. Consider a substring $z = C_L(x)[i \ldots j]$ of $C_L(x)$, containing symbols only from $S \subseteq \Sigma$. We claim that z consists of at most $r_L(|S|) := 2(L+1)^{|S|-1} - 1$ runs. We prove our claim by induction on $|S|$. For $|S| = 1$, the claim holds trivially. For $|S| > 1$ and any $k \geq 1$, let i_k be the minimal index such that $z[1 \ldots i_k]$ is a (k,S)-permutation string, or $i_k = \infty$ if no such prefix of z exists. Note that $i_L \geq |z|$, since otherwise a proper prefix of z would be an (L,S)-permutation string, in which case we would have deleted the last symbol of z. The string $z[i_{k-1} + 1 \ldots i_k - 1]$ contains symbols only from $S \setminus \{z[i_k]\}$ and thus by induction hypothesis consists of at most $r_L(|S| - 1)$ runs. Since $i_L \geq |z|$, we conclude that the number of runs in z is at most $L \cdot (r_L(|S| - 1) + 1) \leq L \cdot 2(L+1)^{|S|-2} \leq 2(L+1)^{|S|-1} - 1 = r_L(|S|)$. Thus the number of runs of $C_L(x)$ is at most $r_L(|\Sigma|) \in \mathcal{O}(L^{|\Sigma|-1})$, and since each run has length at most L we obtain $|C_L(x)| \in \mathcal{O}(L^{|\Sigma|})$.

Algorithm 2 shows how to efficiently implement Algorithm 1 in time $\mathcal{O}(1)$ per symbol of x. We maintain a counter t_S (initialized to 0) and a set Q_S (initialized to $\emptyset$) for every $S \subseteq \Sigma$ with the following meaning. After reading $x[1 \ldots i]$, let j be minimal such that $x[j \ldots i]$ consists of symbols in S. Then t_S is the maximum number t such that $x[j \ldots i]$ is a (t,S)-permutation string. Moreover, let k be minimal such that $x[j \ldots k]$ still is a (t_S,S)-permutation string. Then $Q_S \subseteq S$ is the set of symbols that appear in $x[k+1 \ldots i]$. In other words, in the future we only need to read the symbols in $S \setminus Q_S$ to complete a $(t_S + 1, S)$-permutation string. In particular, when reading the next symbol $x[i+1]$, in order to check whether Claim 3 applies we only need to test whether for any $S \subseteq \Sigma$ with $x[i+1] \in S$ we have $t_S \geq L$. Updating t_S and Q_S is straightforward, and shown in Algorithm 2.

Algorithm 2 Computing $C_L(x)$ in time $\mathcal{O}(1)$ per symbol of x.

```
1: set t_s ← 0, Q_S ← ∅ for all S ⊆ Σ
2: set C_L(x) to the empty string
3: for all i from 1 to |x| do
4:   if t_S < L for all S ⊆ Σ with x[i] ∈ S then
5:     set C_L(x) ← C_L(x) ∘ x[i]
6:     for all S such that x[i] ∈ S do
7:       set Q_S ← Q_S ∪ {x[i]}
8:       if Q_S = S then
9:         set Q_S ← ∅
10:        set t_S ← t_S + 1
11:    for all S such that x[i] ∉ S do
12:      set t_S ← 0
13:      set Q_S ← ∅
```

Since we assume $|\Sigma|$ to be constant, each iteration of the loop runs in time $\mathcal{O}(1)$, and thus the algorithm determines $C_L(x)$ in time $\mathcal{O}(n)$. This finishes the proof of Theorem 1.

3 Optimality of the Sketch

In this section we show that the sketch $C_L(x)$ is optimal in many ways. First, we show that the length and the number of runs are optimal for any sketch that replaces x by any other string z with the same set of subsequences of length at most L.

▶ **Theorem 4.** *For any L and Σ there exists a string x such that for any string z with $\{y \mid y \preceq x, |y| \leq L\} = \{y \mid y \preceq z, |y| \leq L\}$ we have $|z| = \Omega(L^{|\Sigma|})$ and z consists of $\Omega(L^{|\Sigma|-1})$ runs.*

Let $\Sigma = \{0, 1, \ldots, \sigma - 1\}$ and $\Sigma_k = \{0, 1, \ldots, k-1\}$. We construct a family of strings $x^{(k)}$ recursively as follows, where $m := L/|\Sigma|$:

$$x^{(0)} = 0^m$$
$$x^{(k)} = (x^{(k-1)} \circ k)^m \circ x^{(k-1)} \quad \text{for } 1 \leq k \leq \sigma - 1.$$

Theorem 4 now follows from the following inductive claim, for $k = \sigma - 1$.

▶ **Claim 5.** *For any string z with $\{y \mid y \preceq x^{(k)}, |y| \leq m(k+1)\} = \{y \mid y \preceq z, |y| \leq m(k+1)\}$ we have $|z| \geq m^{k+1}$ and the number of runs in z is at least m^k.*

Proof. We use induction on k. For $k = 0$, since $y = 0^m \preceq x^{(0)}$ we have $z = 0^{m'}$ with $m' \geq m$ and the number of runs in z is exactly 1. For any $k > 0$, if $|x^{(k)}|_k > |z|_k$ then $k^m \preceq x^k$ but $k^m \not\preceq z$, and similarly if $|x^{(k)}|_k < |z|_k$ then $k^{m+1} \preceq z$ but $k^{m+1} \not\preceq x^{(k)}$ (note that $m(k+1) \geq m+1$ since $k \geq 1$, and thus y can be k^{m+1}). This implies $|z|_k = m$ and thus we have $z = z^{(0)} \circ k \circ z^{(1)} \circ k \circ \ldots \circ k \circ z^{(m)}$, where each $z^{(i)}$ is a string on alphabet Σ_{k-1}. Hence, for any $0 \leq i \leq m$ and string y' of length at most mk, we have $y = k^i y' k^{m-i} \preceq z$ if and only if $y' \preceq z^{(i)}$. Similarly, $y \preceq x^{(k)}$ holds if and only if $y' \preceq x^{(k-1)}$. Since $y \preceq z$ is equivalent to $y \preceq x$ by assumption, we obtain that $y' \preceq z^{(i)}$ is equivalent to $y' \preceq x^{(k-1)}$. By induction hypothesis, $z^{(i)}$ has length at least m^k and consists of at least m^{k-1} runs. Summing over all i, string z has length at least m^{k+1} and consists of at least m^k runs. ◀

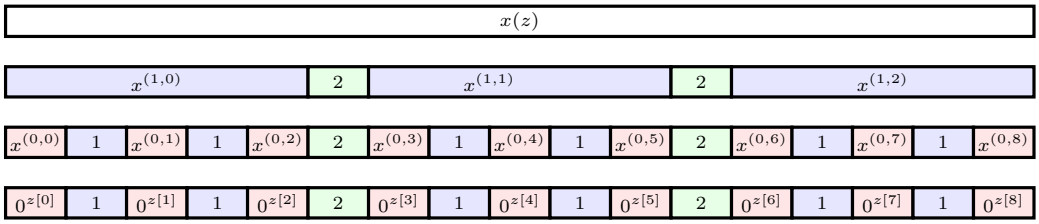


Figure 1 Illustration of constructing $x(z)$ from z. Let $m = \sigma = 3$. Consider a string z of length $m^{\sigma-1} = 9$. The figure shows the construction of $x(z)$ from z.

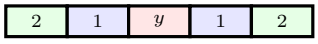


Figure 2 Illustration of the construction of $pat(i, y)$. Let $m = \sigma = 3$. Consider $i = 4 = 1 \cdot 3^1 + 1 \cdot 3^0$. Therefore $\mathrm{pat}(i, y) = 21y12$.

Note that the run-length encoding of $C_L(x)$ has bit length $\mathcal{O}(L^{|\Sigma|-1} \log L)$, since $C_L(x)$ consists of $\mathcal{O}(L^{|\Sigma|-1})$ runs, each of which can be encoded using $\mathcal{O}(\log L)$ bits. We now show that this sketch has optimal size, even in the setting of *Subsequence Sketching*: Alice is given a string x of length n over alphabet Σ and a number L and computes $\mathrm{sk}_L(x)$. Bob is then given $\mathrm{sk}_L(x)$ and a string y of length at most[3] L and decides whether y is a subsequence of x.

We construct the following hard strings for this setting, similarly to the previous construction. Let $\Sigma = \{0, 1, 2, \ldots, \sigma - 1\}$ and $m \in \mathbb{N}$. Consider any vector $z \in \{0, \ldots, m-1\}^k$, where $k := m^{\sigma-1}$. We define the string $x = x(z)$ recursively as follows; see Figure 1 for an illustration:

$$
\begin{aligned}
x(z) &= x^{(\sigma-1,0)} \\
x^{(c,i)} &= \Big(\bigcirc_{j=0}^{m-2} x^{(c-1,m\cdot i+j)} \circ c \Big) \circ x^{(c-1,m\cdot i+m-1)} \quad \text{for } 1 \le c \le \sigma - 1 \\
x^{(0,i)} &= 0^{z[i]}
\end{aligned}
$$

A straightforward induction shows that $|x(z)| \le m^\sigma - 1$. Moreover, for any $0 \le i < m^{\sigma-1}$ with base-m representation $i = \sum_{j=0}^{\sigma-2} i_j \cdot m^j$, where $0 \le i_j < m$, we define the following string; see Figure 2 for an illustration:

$$\mathrm{pat}(i, y) := \big(\bigcirc_{j=1}^{\sigma-1} (\sigma - j)^{i_{\sigma-1-j}} \big) \circ y \circ \big(\bigcirc_{j=1}^{\sigma-1} j^{m-1-i_{j-1}} \big).$$

The following claim shows that testing whether $\mathrm{pat}(i, y)$ is a subsequence of $x(z)$ allows to infer the entries of z.

▶ **Claim 6.** *We have* $\mathrm{pat}(i, y) \preceq x(z)$ *if and only if* $y \preceq 0^{z[i]}$.

Proof. See Figure 3 for illustration. Given i and y, let $z^{(c)} = c^{i_{c-1}} \circ z^{(c-1)} \circ c^{m-1-i_{c-1}}$ for all $1 \le c \le \sigma - 1$, and $z^{(0)} = y$. Note that $z^{(\sigma-1)} = \mathrm{pat}(i, y)$. Set $j_c := \sum_{l=c}^{\sigma-2} i_l \cdot m^{l-c}$, so in particular we have $j_c = m \cdot j_{c-1} + i_c$. Observe that $z^{(c)} \preceq x^{(c,j_c)}$ if and only if

[3] In the introduction, we used a slightly different definition where Bob is given a string of length *exactly* L. This might seem slightly weaker, but in fact the two formulations are equivalent (up to increasing L by 1), as can be seen by replacing x by $x' = 0^L 1 x$ and y by $y' = 0^{L-|y|} 1 y$. Then $y \preceq x$ if and only if $y' \preceq x'$, and y' has fixed length $L + 1$.

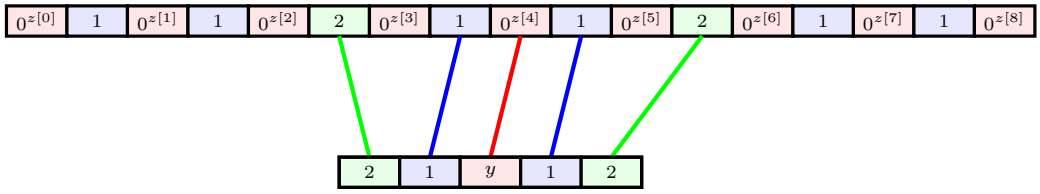


Figure 3 Illustration of Claim 6. Let $m = \sigma = 3$ and $i = 4$. Then $\mathrm{pat}(i, y) = 21y12$. Now observe that $\mathrm{pat}(i, y) \preceq x(z)$ if and only if $y \preceq x^{(0,i)} = 0^{z[i]}$.

$z^{(c-1)} \preceq x^{(c-1,j_{c-1})}$, which follows immediately after matching all c's in $z^{(c)}$ and $x^{(c,j_c)}$. Therefore, $\mathrm{pat}(i, y) = z^{(\sigma-1)} \preceq x^{(\sigma-1,0)} = x(z)$ holds if and only if $z^{(c)} \preceq x^{(c,j_c)}$ for any $c \leq \sigma - 2$. Substituting $c = 0$ we obtain that $\mathrm{pat}(i, y) \preceq x(z)$ holds if and only if $y = z^{(0)} \preceq x^{(0,j_0)} = x^{(0,i)} = 0^{z[i]}$. ◀

▶ **Theorem 7.** *Any deterministic subsequence sketch has size $\Omega(L^{|\Sigma|-1} \log L)$ in the worst case. Restricted to strings of length $\Theta(L^{|\Sigma|-1})$, the sketch size is $\Omega(L^{|\Sigma|-1})$.*

Proof. Let $m := L/|\Sigma|$. Let $z \in \{0, \ldots, m-1\}^k$ with $k = m^{|\Sigma|-1}$ and let $x = x(z)$ as above. Alice is given x, L as input. Notice that there are m^k distinct inputs for Alice. Assume for contradiction that the sketch size is less that $k \cdot \log m$ for every x. Then the total number of distinct possible sketches is strictly less than m^k. Therefore, at least two strings, say $x(z_1)$ and $x(z_2)$, have the same encoding, for some $z_1, z_2 \in \{0, \ldots, m-1\}^k$ with $z_1 \neq z_2$. Let i be such that $z_1[i] \neq z_2[i]$, and without loss of generality $z_1[i] < z_2[i]$. Now set Bob's input to $y = \mathrm{pat}(i, z_2[i])$, which is a valid subsequence of $x(z_2)$, but not of $x(z_1)$. However, since the encoding for both $x(z_2)$ and $x(z_1)$ is the same, Bob's output will be incorrect for at least one of the strings. Finally, note that $|y| \leq m\sigma = L$. Hence, we obtain a sketch size lower bound of $\Omega(k \log m) = \Omega(L^{|\Sigma|-1} \log L)$.

If we instead choose z from $\{0,1\}^k$, then the constructed string $x(z)$ has length $\mathcal{O}(k) = \mathcal{O}(L^{|\Sigma|-1})$, and the same argument as above yields a sketch lower bound of $\Omega(L^{|\Sigma|-1})$. ◀

We now discuss the complexity of randomized subsequence sketching where Bob is allowed to err with probability $1/3$. To this end, we will reduce from the *Index* problem.

▶ **Definition 8.** In the *Index* problem, Alice is given an n-bit string $z \in \{0,1\}^n$ and sends a message to Bob. Bob is given Alices's message and an integer $i \in [n]$ and outputs $z[i]$.

Intuitively, since the communication is one-sided, Alice cannot infer i and therefore has to send the whole string z. This intuition also holds for randomized protocols, as follows.

▶ **Fact 9** ([31]). *The randomized one-way communication complexity of Index is $\Omega(n)$.*

Claim 6 shows that subsequence sketching allows us to infer the bits of an arbitrary string z, and thus the hardness of Index carries over to subsequence sketching.

▶ **Theorem 10.** *In a* randomized *subsequence sketch, Bob is allowed to err with probability $1/3$. Any randomized subsequence sketch has size $\Omega(L^{|\Sigma|-1})$ in the worst case. This holds even restricted to strings of length $\Theta(L^{|\Sigma|-1})$.*

Proof. We reduce the Index problem to subsequence sketching. Let $z \in \{0,1\}^k$ be the input to Alice in the Index problem, where $k = m^{|\Sigma|-1}$. As above, we construct the corresponding input $x(z)$ to Alice in subsequence sketching. Observe that $|x(z)| = \mathcal{O}(m^{|\Sigma|-1})$. For any input i to Bob in the Index problem, we construct the corresponding input $\mathrm{pat}(i, 0)$ for Bob in subsequence sketching. We have $\mathrm{pat}(i, 0) \preceq x(z)$ if and only if $z[i] = 1$ (by Claim 6). This yields a lower bound of $\Omega(k) = \Omega(m^{|\Sigma|-1}) = \Omega(L^{|\Sigma|-1})$ on the sketch size (by Fact 9). ◀

4 Weighted LCS

▶ **Definition 11.** In the WLCS problem we are given strings x, y of lengths n, m over alphabet Σ and given a function $W\colon \Sigma \to \mathbb{N}$. A weighted longest common subsequence (WLCS) of x and y is any string z with $z \preceq x$ and $z \preceq y$ maximizing $W(z) = \sum_{i=1}^{|z|} W(z[i])$. The task is to compute this maximum weight, which we abbreviate as $\mathrm{WLCS}(x, y)$.

In the remainder of this section we will design an algorithm for computing $\mathrm{WLCS}(x, y)$ in time $\mathcal{O}(\min\{nm, n + m^{|\Sigma|}\})$. This yields the upper bound of Theorem 2. Note that here we focus on computing the maximum weight $\mathrm{WLCS}(x, y)$; standard methods can be applied to reconstruct a subsequence attaining this value. We prove a matching conditional lower bound of $\min\{nm, n + m^{|\Sigma|}\}^{1-o(1)}$ in the next section.

Let x, y, W be given. The standard dynamic programming algorithm for determining $\mathrm{LCS}(x, y)$ in time $\mathcal{O}(nm)$ trivially generalizes to $\mathrm{WLCS}(x, y)$ as well. Alternatively, we can first compress x to $x' := C_m(x)$ in time $\mathcal{O}(n)$ and then compute the $\mathrm{WLCS}(x', y)$, which is equal to $\mathrm{WLCS}(x, y)$ since all subsequences of length at most m of x are also subsequences of $C_m(x)$. We show below in Theorem 12 how to compute WLCS of a run-length encoded string x' with r runs and a string y of length m in time $\mathcal{O}(rm)$. Since $x' = C_m(x)$ consists of $O(m^{|\Sigma|-1})$ runs and the length of y is m, we can compute $\mathrm{WLCS}(x, y) = \mathrm{WLCS}(C_m(x), y)$ in time $\mathcal{O}(m^{|\Sigma|})$. In total, we obtain time $\mathcal{O}(\min\{nm, n + m^{|\Sigma|}\})$.

It remains to solve WLCS on a run-length encoded string x with r runs and a string y of length m in time $\mathcal{O}(rm)$. For (unweighted) LCS a dynamic programming algorithm with this running time was presented by Liu et al. [37]. We first give a brief intuitive explanation as to why their algorithm does not generalize to WLCS. Let $x = c_1^{\ell_1} c_2^{\ell_2} \dots c_r^{\ell_r}$ be the run-length encoded string, where $c_i \in \Sigma$, and let $L_i = \sum_{j=1}^{i} \ell_j$. Let $D(i, j) := \mathrm{WLCS}(x[1 \dots L_i], y[1 \dots j])$. Liu et al.'s algorithm relies on a recurrence for $D(i, j)$ in terms of $D(i, j-1)$. Consider an input like $x = ba_1a_2 \cdots a_k b$ and $y = a_1a_2 \cdots a_k bb$ with $W(b) > \sum_{\ell \in [k]} W(a_\ell)$. Note that $D(k+2, k+1) = \sum_{\ell \in [k]} W(a_\ell) + W(b)$, but $D(k+2, k+2) = 2W(b)$. Thus $D(k+2, k+2) = D(k+2, k+1) - \sum_{\ell \in [k]} W(a_\ell) + W(b)$. Therefore, in the weighted setting $D(i, j)$ and $D(i, j-1)$ can differ by complicated terms that seem hard to figure out locally. Our algorithm that we develop below instead relies on a recurrence for $D(i, j)$ in terms of $D(i-1, j')$.

▶ **Theorem 12.** *Given a run-length encoded string x consisting of r runs, a string y of length m, and a weight function $W\colon \Sigma \to \mathbb{N}$ we can determine $\mathrm{WLCS}(x, y)$ in time $\mathcal{O}(rm)$.*

Proof. We write the run-length encoded string x as $c_1^{\ell_1} c_2^{\ell_2} \dots c_r^{\ell_r}$ with $c_i \in \Sigma$ and $\ell_i \geq 1$. Let $L_i = \sum_{j=1}^{i} \ell_j$. We will build a dynamic programming table D where $D(i, j)$ stores the value $\mathrm{WLCS}(x[1 \dots L_i], y[1 \dots j])$. In particular, $D(0, j) = D(i, 0) = 0$ for all i, j. We will show how to compute this table in $\mathcal{O}(1)$ (amortized) time per entry in the following. Since we can split $\mathrm{WLCS}(x[1 \dots L_i], y[1 \dots j]) = \max_{0 \leq k \leq j} \mathrm{WLCS}(x[1 \dots L_{i-1}], y[1 \dots k]) + \mathrm{WLCS}(c_i^{\ell_i}, y[k+1 \dots j])$, we obtain the recurrence $D(i, j) = \max_{0 \leq k \leq j} D(i-1, k) + W(c_i) \cdot \min\{\ell_i, |y[k+1 \dots j]|_{c_i}\}$. Since $D(i, j)$ is monotonically non-decreasing in i and j, we may rewrite the same recurrence as

$$\begin{aligned} D(i, j) &= \max_{0 \leq k \leq j \,:\, |y[k+1 \dots j]|_{c_i} \leq \ell_i} D(i-1, k) + W(c_i) \cdot |y[k+1 \dots j]|_{c_i}. \\ &= W(c_i) \cdot |y[1 \dots j]|_{c_i} + \max_{0 \leq k \leq j \,:\, |y[k+1 \dots j]|_{c_i} \leq \ell_i} D(i-1, k) - W(c_i) \cdot |y[1 \dots k]|_{c_i} \end{aligned}$$

Algorithm 3 Computing RTLM($K_{i,j}$) from RTLM($K_{i,j-1}$).

1: Initialize RTLM($K_{i,j}$) = RTLM($K_{i,j-1}$)
2: **while** the smallest (=leftmost) element k of RTLM($K_{i,j}$) satisfies $|y[k+1\dots j]|_{c_i} > \ell_i$ **do**
3: Remove k from RTLM($K_{i,j}$)
4: **while** the largest (=rightmost) element k of RTLM($K_{i,j}$) satisfies $h_i(k) \le h_i(j)$ **do**
5: Remove k from RTLM($K_{i,j}$)
6: Append j to RTLM($K_{i,j}$)

Let $b_{i,j}$ be the minimum value of $0 \le k \le j$ such that $|y[k+1\dots j]|_{c_i} \le \ell_i$. Note that $b_{i,j}$ is well-defined, since for $k = j$ we always have $|y[k+1\dots j]|_{c_i} = 0 \le \ell_i$, and note that $b_{i,j}$ is monotonically non-decreasing in j. We define the *active k-window* $K_{i,j}$ as the interval $\{b_{i,j}, b_{i,j}+1, \dots, j\}$. Note that $K_{i,j}$ is non-empty and both its left and right boundary are monotonic in j. Let $h_i(k) := D(i-1,k) - W(c_i)\cdot|y[1\dots k]|_{c_i}$ be the *height* of k. We define $\mathit{highest}(K_{i,j})$ as $\max_{k\in K_{i,j}} h_i(k)$. With this notation, we can rewrite the above recurrence as

$$D(i,j) = W(c_i)\cdot|y[1\dots j]|_{c_i} + \mathit{highest}(K_{i,j}).$$

We can precompute all values $|y[1\dots j]|_c$ in $\mathcal{O}(m)$ time. Hence, in order to determine $D(i,j)$ in amortized time $\mathcal{O}(1)$ it remains to compute $\mathit{highest}(K_{i,j})$ in amortized time $\mathcal{O}(1)$. To this end, we maintain the *right to left maximum sequence* of the active window $K_{i,j}$. Specifically, we consider the sequence RTLM($K_{i,j}$) = $\langle k_s, k_{s-1}, \dots, k_1\rangle$ where $k_1 = j$ and for any $p > 1$, k_p is is the largest number in $K_{i,j}$ with $k_p < k_{p-1}$ and $h_i(k_p) > h_i(k_{p-1})$. In particular, k_s is the largest number in $K_{i,j}$ attaining $h_i(k_s) = \mathit{highest}(K_{i,j})$. Hence, from this sequence RTLM($K_{i,j}$) we can determine $\mathit{highest}(K_{i,j})$ and thus $D(i,j)$ in time $\mathcal{O}(1)$. It remains to argue that we can maintain RTLM($K_{i,j}$) in amortized time $\mathcal{O}(1)$ per table entry. We sketch an algorithm to obtain RTLM($K_{i,j}$) from RTLM($K_{i,j-1}$).

It is easy to see correctness, since the first while loop removes right to left maxima that no longer lie in the active window, the second while loop removes right to left maxima that are dominated by the new element j, and the last line adds j. Note that $|y[k+1\dots j]|_c = |y[1\dots j]|_c - |y[1\dots k]|_c$ can be computed in time $\mathcal{O}(1)$ from the precomputed values $|y[1\dots j]|_c$, and thus the while conditions can be checked in time $\mathcal{O}(1)$. A call of Algorithm 3 can necessitate multiple removal operations, but only one insertion. By charging removals to the insertion of the removed element, we see that Algorithm 3 runs in amortized time $\mathcal{O}(1)$. We therefore can compute each table entry $D(i,j)$ in amortized time $\mathcal{O}(1)$ and obtain total time $\mathcal{O}(rm)$. Pseudocode for the complete algorithm is given below. ◀

5 Conditional lower bound for Weighted LCS

In this section, we prove a conditional lower bound for Weighted LCS, based on the standard hypothesis SETH, which was introduced by Impagliazzo, Paturi, and Zane [27] and asserts that satisfiability has no algorithms that are much faster than exhaustive search.

Strong Exponential Time Hypothesis (SETH). *For any $\varepsilon > 0$ there is a $k \ge 3$ such that k-SAT on n variables cannot be solved in time $\mathcal{O}((2-\varepsilon)^n)$.*

Algorithm 4 Computing $\mathrm{WLCS}(x, y)$ in time $\mathcal{O}(r \cdot m)$.

precompute $|y[1 \dots i]|_c$ for all $i \in [m]$ and $c \in \Sigma$.
set $D(i, 0) = D(0, j) = 0$ for any $0 \le i \le r$ and $0 \le j \le m$.
for $i = 1, \dots, r$ **do**
  $\mathrm{RTLM}(K_{i,0}) \leftarrow \langle 0 \rangle$.
  for $j = 1, \dots, m$ **do**
    Update $\mathrm{RTLM}(K_{i,j})$ as in Algorithm 3
    Let k be the smallest (=leftmost) element of $\mathrm{RTLM}(K_{i,j})$
    Compute $highest(K_{i,j}) = h_i(k) = D(i-1, k) - W(c_i) \cdot |y[1 \dots k]|_{c_i}$
    $D(i, j) \leftarrow W(c_i) \cdot |y[1 \dots j]|_{c_i} + highest(K_{i,j})$.
return $D(r, m)$

Essentially all known SETH-based lower bounds for polynomial-time problems (e.g. [1, 10, 12, 13, 14]) use reductions via the *Orthogonal Vectors problem* (OV): Given sets A, $B \subseteq \{0,1\}^D$ of size $|A| = N$, $|B| = M$, determine whether there are $a \in A, b \in B$ that are orthogonal, i.e., $\sum_{i=1}^{D} a[i] \cdot b[i] = 0$, where the sum is over the integers. Simple algorithms solve OV in time $\mathcal{O}(2^D(N+M))$ and $\mathcal{O}(NMD)$. The fastest known algorithm for $D = c(N) \log N$ runs in time $N^{2-1/\mathcal{O}(\log c(N))}$ (when $N = M$) [2], which is only slightly subquadratic for $D \gg \log N$. This has led to the following reasonable hypothesis.

(Unbalanced) Orthogonal Vectors Hypothesis (OVH). *For any $\gamma > 0$, OV restricted to $M = \Theta(N^\gamma)$ and $D = N^{o(1)}$ requires time $(NM)^{1-o(1)}$.*

A well-known reduction by Williams [50] shows that SETH implies OVH in case $\gamma = 1$. Moreover, an observation in [14] shows that if OVH holds for some $\gamma > 0$ then it holds for all $\gamma > 0$. Thus, OVH is a weaker assumption than SETH, and any OVH-based lower bound also implies a SETH-based lower bound. The conditional lower bound in this section does not only hold assuming SETH, but even assuming the weaker OVH.

We use the following construction from the OVH-based lower bound for LCS [1, 13]. For binary alphabet, such a construction was given in [13].

▶ **Theorem 13.** *Given $A, B \subseteq \{0,1\}^D$ of size N, in time $\mathcal{O}(DN)$ we can compute strings x_A and y_B on alphabet $\{0,1\}$ of length $\Theta(DN)$ as well as a number τ such that $\mathrm{LCS}(x_A, y_B) \ge \tau$ holds if and only if there is an orthogonal pair of vectors in A and B. In this construction, x_A and y_B depend only on A and B, respectively, and $|x_A|, |y_B|, \tau$ depend only on N, D.*

We now prove a conditional lower bound for WLCS, i.e., the lower bound of Theorem 2.

▶ **Theorem 14.** *Given strings x, y of lengths n, m with $n \ge m$ over alphabet Σ, computing $\mathrm{WLCS}(x, y)$ requires time $\min\{nm, n+m^{|\Sigma|}\}^{1-o(1)}$, assuming OVH. This holds even restricted to $n = m^{\alpha \pm o(1)}$ and $|\Sigma| = \sigma$ for any fixed constants $\alpha \in \mathbb{R}, \alpha \ge 1$ and $\sigma \in \mathbb{N}, \sigma \ge 2$.*

Proof. Let $\Sigma = \{0, 1, \dots, \sigma - 1\}$ and $\alpha = \alpha_I + \alpha_F$, where $\alpha_I = \lfloor \alpha \rfloor$ and $\alpha_F = \alpha - \alpha_I$ are the integral and fractional parts. Let $M \in \mathbb{N}$ and set $N = \min\{M^{\alpha_I} \cdot \lceil M^{\alpha_F} \rceil, M^{\sigma-1}\}$. Note that M divides N. Consider any instance $A = \{a_0, a_1, \dots, a_{N-1}\} \subseteq \{0,1\}^d$ and $B = \{b_0, b_1, \dots, b_{M-1}\} \subseteq \{0,1\}^D$ of the Orthogonal Vectors problem. Partition A into $A^0, A^1, \dots, A^{N/M-1}$, where $|A^i| = M$. Then by Theorem 13 we can construct strings $x_A^{(i)}$ and y_B on alphabet $\{0,1\}$ of length $\Theta(DM)$ and $\tau \in \mathbb{N}$ in time $\mathcal{O}(DM)$ such that A^i and B contain an orthogonal pair of vectors if and only if $\mathrm{LCS}(x_A^{(i)}, y_B) \ge \tau$. Note that A and B contain an orthogonal pair of vectors if and only if for some $0 \le i < \frac{N}{M}$, A^i and B

contain an orthogonal pair of vectors. Hence, A and B contain an orthogonal pair if and only if $\max_{0\le i<\frac{N}{M}} \mathrm{LCS}(x_A^{(i)}, y_B) \ge \tau$. In the following, we encode the latter inequality into an instance of WLCS.

For simplicity we only give the proof for integral α and $\alpha < \sigma$ (the remaining cases are omitted and can be found in the appendix). In this case, $N = M^\alpha$ and the running time lower bound that we will prove is $(nm)^{1-o(1)}$.

We set λ to any value such that $\lambda > |y_B|/M$, and note that $\lambda \in \Theta(D)$ suffices. Set $W(k) = \lambda \cdot M^{k-1}$ for $k \ge 2$, and $W(1) = W(0) = 1$. Let $\Sigma_k = \{0, 1, \ldots, k-1\}$. We construct strings x and y as follows:

$$\begin{aligned}
x &= x^{(\alpha,0)} \\
x^{(k,i)} &= \Big(\bigcirc_{j=0}^{M-2} x^{(k-1,M\cdot i+j)} \circ k \Big) \circ x^{(k-1,M\cdot i+(M-1))} \quad \text{for } 2 \le k \le \alpha \\
x^{(1,i)} &= x_A^i \quad \text{for } 0 \le i < N/M \\
y &= y^{(\alpha)} \\
y^{(k)} &= k^{M-1} \circ y^{(k-1)} \circ k^{M-1} \quad \text{for } 2 \le k \le \alpha \\
y^{(1)} &= y_B.
\end{aligned}$$

Observe that for all k, $x^{(k,i)}$ and $y^{(k)}$ are defined on Σ_k. In particular, since $\alpha \le \sigma - 1$ we only use symbols from Σ. Let $\ell(k)$ denote the length of $x^{(k,i)}$ for any i. Observe that $\ell(k) = M \cdot \ell(k-1) + (M-1)$ and $\ell(1) \in \Theta(DM)$. Thus, $\ell(k) \in \Theta(DM^k)$ and $n := |x| \in \Theta(DM^\alpha)$. It is straightforward to see that $m := |y| = \Theta((k+D)M) = \Theta(DM)$, since $k \le |\Sigma| = \mathcal{O}(1)$. Recall that for any string z, $W(z)$ is its total weight.

▶ **Claim 15.** *For any integer $2 \le k \le \alpha$, we have (1) $(M-1) \cdot \sum_{\ell=2}^{k} W(\ell) = \lambda(M^k - M)$ and (2) $W(y^{(k)}) < W(k+1) + \lambda \cdot (M^k - M)$.*

Proof. For (1), we calculate $(M-1) \cdot \sum_{\ell=2}^{k} W(\ell) = (M-1) \cdot \sum_{\ell=1}^{k-1} \lambda M^\ell = \lambda(M^k - M)$. For (2), by definition of $y^{(k)}$ and λ we have

$$W(y^{(k)}) < \lambda M + 2(M-1) \cdot \sum_{\ell=2}^{k-1} W(l) \overset{(1)}{=} \lambda M + 2\lambda(M^k - M) = W(k+1) + \lambda(M^k - M). \blacktriangleleft$$

We now can perform the core step of our correctness argument.

▶ **Lemma 16.** *For any $2 \le k \le \alpha$ and $0 \le i < M^{k-1}$, we have (1) $\mathrm{WLCS}(x^{(k,i)}, y^{(k)}) \ge \lambda(M^k - M)$, and (2) $\mathrm{WLCS}(x^{(k,i)}, y^{(k)}) = (M-1) \cdot W(k) + \mathrm{WLCS}(x^{(k-1,j)}, y^{(k-1)})$ for some $M \cdot i \le j < M \cdot (i+1)$.*

Proof. For (1), clearly $\bigcirc_{j=2}^{k} j^{M-1}$ is a common subsequence of $x^{(k,i)}$ and $y^{(k)}$. Together with Claim 15.(1), we obtain $\mathrm{WLCS}(x^{(k,i)}, y^{(k)}) \ge \sum_{j=2}^{k} (M-1) \cdot W(j) = \lambda(M^k - M)$.

For (2), we claim that k^{M-1} is a subsequence of any WLCS of $x^{(k,i)}$ and $y^{(k)}$. Assuming otherwise, the WLCS can contain at most $M-2$ symbols k and all of $y^{(k-1)}$. Therefore,

$$\begin{aligned}
\mathrm{WLCS}(x^{(k,i)}, y^{(k)}) &\le (M-2) \cdot W(k) + W(y^{(k-1)}) \\
&< (M-2) \cdot W(k) + W(k) + \lambda(M^{k-1} - M) \quad \text{by Claim 15.(2)} \\
&= (M-1) \cdot \lambda M^{k-1} + \lambda(M^{k-1} - M) = \lambda \cdot (M^k - M).
\end{aligned}$$

This contradicts $\mathrm{WLCS}(x^{(k,i)}, y^{(k)}) \ge \lambda(M^k - M)$. It follows that k^{M-1} is a subsequence of the WLCS of $x^{(k,i)}$ and $y^{(k)}$. Hence, $\mathrm{WLCS}(x^{(k,i)}, y^{(k)}) = (M-1) \cdot W(k) + \mathrm{WLCS}(x^{(k-1,j)}, y^{(k-1)})$ for some j with $M \cdot i \le j < M \cdot (i+1)$. ◀

Recursively applying the above lemma and substituting $x^{(1,j)}$ by x_A^j, we conclude that $\mathrm{WLCS}(x,y) = \lambda \cdot (M^\alpha - M) + \max_{0 \le j < M^{\alpha-1}} \mathrm{LCS}(x_A^j, y_B)$. Using $M^\alpha = N$ and the construction of x_A^j, y_B, we obtain that $\mathrm{WLCS}(x,y) \ge \lambda(N-M)+\tau$ holds if and only if there is an orthogonal pair of vectors in A and B. Since OVH asserts that solving the OV instance (A,B) in the worst case requires time $(NM)^{1-o(1)}$, even for $D = N^{o(1)}$, we obtain that determining $\mathrm{WLCS}(x,y)$ requires time $(NM)^{1-o(1)} = (nm/D^2)^{1-o(1)} = (nm)^{1-o(1)}$. This completes the proof for all instances where $\alpha < \sigma$ is integral. Note that if $\alpha \ge \sigma$, the claimed lower bound trivially holds as it matches the input size. Now we consider the two remaining cases, where $\sigma - 1 < \alpha < \sigma$ and $\alpha < \sigma - 1$.

Case $\sigma - 1 < \alpha < \sigma$. Then $N = M^{\alpha_I} = M^{\sigma-1}$. We construct strings x and y as follows:

$$x = x^{(\alpha_I,0)} \circ \alpha_I \circ 0^{DM^\alpha}$$
$$y = y^{(\alpha_I)} \circ \alpha_I.$$

Again, since $\alpha_I \le \sigma - 1$ the strings x and y only use symbols in Σ. We now have $n := |x| \in \Theta(DM^\alpha)$ and $m := |y| \in \Theta(DM)$. Clearly, $\mathrm{WLCS}(x,y) \ge \mathrm{WLCS}(x^{(\alpha_I,0)}, y^{(\alpha_I)}) + W(\alpha_I) \ge W(\alpha_I) + \lambda(M^{\alpha_I} - M)$. Similar to the proof for integral α, we claim that α_I^M is a subsequence of the WLCS of x and y. Assuming otherwise, the WLCS of x and y contains at most $M-1$ symbols α_I and all of $y^{(\alpha_I-1)}$. Therefore,

$$\begin{aligned}\mathrm{WLCS}(x,y) &\le (M-1)\cdot W(\alpha_I) + W(y^{(\alpha_I-1)}) \\ &< (M-1)\cdot W(\alpha_I) + W(\alpha_I) + \lambda(M^{\alpha_I-1} - M) \quad \text{by Claim 15.(2)} \\ &= W(\alpha_I) + \lambda\cdot(M^{\alpha_I} - M^{\alpha_I-1} + M^{\alpha_I-1} - M) = W(\alpha_I) + \lambda(M^{\alpha_I} - M).\end{aligned}$$

This contradicts $\mathrm{WLCS}(x,y) \ge W(\alpha_I) + \lambda(M^{\alpha_I} - M)$. Hence, α_I^M is a subsequence of the WLCS of x and y, and $\mathrm{WLCS}(x,y) = W(\alpha_I) + \mathrm{WLCS}(x^{(\alpha_I,0)}, y^{(\alpha_I)})$. It follows that $\mathrm{WLCS}(x,y) \ge \lambda M^{\alpha_I-1} + \lambda(M^{\alpha_I} - M) + \tau$ holds if and only if there exists an orthogonal pair of vectors in A and B. OVH asserts that solving the OV instance (A,B) in the worst case requires time $(NM)^{1-o(1)}$, even for $D = N^{o(1)}$. Using $N = \Theta(M^{\alpha_I}) = \Theta(M^{\sigma-1})$, we obtain that determining $\mathrm{WLCS}(x,y)$ requires time $(NM)^{1-o(1)} = (M^\sigma)^{1-o(1)} = ((m/D)^\sigma)^{1-o(1)} = (m^{|\Sigma|})^{1-o(1)}$. This completes the proof in the case $\sigma - 1 < \alpha < \sigma$.

Case $\alpha < \sigma - 1$. In this case $\alpha_I \le \sigma - 2$ and $N = M^{\alpha_I} \cdot \lceil M^{\alpha_F} \rceil$. Let $f = \lceil M^{\alpha_F} \rceil$ as shorthand. We construct x and y as follows:

$$x = \Big(\bigcirc_{j=0}^{f-2} x^{(\alpha_I,j)} \circ (\alpha_I + 1) \Big) \circ x^{(\alpha_I, f-1)}$$
$$y = (\alpha_I + 1)^f \circ y^{(\alpha_I)} \circ (\alpha_I + 1)^f$$

Once again x and y consist of symbols in Σ, since $\alpha_I \le \sigma - 2$. Since $|x^{(\alpha_I,i)}| \in \Theta(DM^{\alpha_I})$, we have $n := |x| \in \Theta(DM^{\alpha_I+\alpha_F}) = \Theta(DM^\alpha)$, and $m := |y| \in \Theta(DM)$. The same argument as before, now with f instead of M parts, shows that $\mathrm{WLCS}(x,y) = (f-1)W(\alpha_I+1) + \mathrm{WLCS}(x^{(\alpha_I,j)}, y^{(\alpha_I)})$ holds for some $0 \le j < f$. Plugging in $\mathrm{WLCS}(x^{(\alpha_I,j)}, y^{(\alpha_I)})$, we see that

$$\mathrm{WLCS}(x,y) = \lambda(f-1)M^{\alpha_I} + \lambda(M^{\alpha_I} - M) + \max_{0 \le j \le \frac{N}{M}-1} \mathrm{LCS}(x_A^j, y_B).$$

Hence, $\mathrm{WLCS}(x,y) \ge \lambda(f-1)M^{\alpha_I} + \lambda(M^{\alpha_I} - M) + \tau$ holds if and only if there is an orthogonal pair of vectors in A and B. OVH asserts that solving the OV instance (A,B) in the worst

case requires time $(NM)^{1-o(1)}$, even for $D = N^{o(1)}$. Using $N = \Theta(M^{\alpha_I} \cdot f) = \Theta(M^{\alpha})$, we obtain that determining $\mathrm{WLCS}(x, y)$ requires time $(NM)^{1-o(1)} = (M^{\alpha+1})^{1-o(1)} = (nm/D^2)^{1-o(1)} = (nm)^{1-o(1)}$. This completes the proof of the last case $\alpha < \sigma - 1$.

Finally, note that in all cases we constructed strings over alphabet size σ of length $n = M^{\alpha \pm o(1)}$ and $m = M^{1 \pm o(1)}$, and thus $n = m^{\alpha \pm o(1)}$. ◀

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Quadratic-Time Hardness of LCS and other Sequence Similarity Measures. In *Proc. 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS'15)*, pages 59–78, 2015.

2 Amir Abboud, Ryan Williams, and Huacheng Yu. More Applications of the Polynomial Method to Algorithm Design. In *Proc. 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'15)*, pages 218–230, 2015.

3 Jochen Alber, Jens Gramm, Jiong Guo, and Rolf Niedermeier. Towards optimally solving the longest common subsequence problem for sequences with nested arc annotations in linear time. In *Proc. 13th Annual Symposium on Combinatorial Pattern Matching (CPM'02)*, pages 99–114, 2002.

4 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *Internat. J. Comput. Geom. Appl.*, 5(1–2):78–99, 1995.

5 Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

6 Amihood Amir, Zvi Gotthilf, and B. Riva Shalom. Weighted LCS. *Journal of Discrete Algorithms*, 8(3):273–281, 2010.

7 Hsing-Yen Ann, Chang-Biau Yang, Chiou-Ting Tseng, and Chiou-Yi Hor. A fast and simple algorithm for computing the longest common subsequence of run-length encoded strings. *Information Processing Letters*, 108(6):360–364, 2008.

8 Alberto Apostolico. Improving the Worst-Case Performance of the Hunt-Szymanski Strategy for the Longest Common Subsequence of Two Strings. *Inf. Process. Lett.*, 23(2):63–69, 1986. doi:10.1016/0020-0190(86)90044-X.

9 Alberto Apostolico and Concettina Guerra. The Longest Common Subsequence Problem Revisited. *Algorithmica*, 2:316–336, 1987. doi:10.1007/BF01840365.

10 Arturs Backurs and Piotr Indyk. Which Regular Expression Patterns are Hard to Match? In *Proc. 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS'16)*, pages 457–466, 2016.

11 Djamal Belazzougui and Qin Zhang. Edit Distance: Sketching, Streaming, and Document Exchange. In *Proc. 57th Annual Symposium on Foundations of Computer Science*, pages 51–60, 2016.

12 Karl Bringmann. Why Walking the Dog Takes Time: Frechet Distance Has No Strongly Subquadratic Algorithms Unless SETH Fails. In *Proc. 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS'14)*, pages 661–670, 2014. doi:10.1109/FOCS.2014.76.

13 Karl Bringmann and Marvin Künnemann. Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping. In *Proc. 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS'15)*, pages 79–97, 2015.

14 Karl Bringmann and Marvin Künnemann. Multivariate Fine-Grained Complexity of Longest Common Subsequence. In *Proc. 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'18)*, 2018. To appear.

15 Mauro Castelli, Riccardo Dondi, Giancarlo Mauri, and Italo Zoppis. The Longest Filled Common Subsequence Problem. In *Proc. 28th Annual Symposium on Combinatorial Pattern Matching (CPM'17)*, 2017. To appear.

16 Diptarka Chakraborty, Elazar Goldenberg, and Michal Kouckỳ. Streaming algorithms for computing edit distance without exploiting suffix trees. *ArXiv:1607.03718*, 2016.

17 Wun-Tat Chan, Yong Zhang, Stanley P.Y. Fung, Deshi Ye, and Hong Zhu. Efficient algorithms for finding a longest common increasing subsequence. *Journal of Combinatorial Optimization*, 13(3):277–288, 2007.

18 Marek Cygan, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Polynomial-time approximation algorithms for weighted LCS problem. *Discrete Applied Mathematics*, 204:38–48, 2016.

19 David Eppstein, Zvi Galil, Raffaele Giancarlo, and Giuseppe F. Italiano. Sparse Dynamic Programming I: Linear Cost Functions. *J. ACM*, 39(3):519–545, July 1992. doi:10.1145/146637.146650.

20 Valerio Freschi and Alessandro Bogliolo. Longest common subsequence between run-length-encoded strings: a new algorithm with improved parallelism. *Information Processing Letters*, 90(4):167–173, 2004.

21 Zvi Gotthilf, Danny Hermelin, Gad M. Landau, and Moshe Lewenstein. Restricted LCS. In *Proc. 17th International Conference on String Processing and Information Retrieval (SPIRE'10)*, pages 250–257, 2010.

22 Daniel S. Hirschberg. Algorithms for the Longest Common Subsequence Problem. *J. ACM*, 24(4):664–675, 1977. doi:10.1145/322033.322044.

23 J. W. Hunt and M. D. McIlroy. An algorithm for differential file comparison. Computing Science Technical Report 41, Bell Laboratories, 1975.

24 James W. Hunt and Thomas G. Szymanski. A Fast Algorithm for Computing Longest Subsequences. *Commun. ACM*, 20(5):350–353, 1977. doi:10.1145/359581.359603.

25 Costas S Iliopoulos, Marcin Kubica, M Sohel Rahman, and Tomasz Waleń. Algorithms for computing the longest parameterized common subsequence. In *Proc. 18th Annual Conference on Combinatorial Pattern Matching (CPM'07)*, pages 265–273, 2007.

26 Costas S. Iliopoulos and M. Sohel Rahman. A New Efficient Algorithm for Computing the Longest Common Subsequence. *Theory of Computing Systems*, 45(2):355–371, 2009. doi:10.1007/s00224-008-9101-6.

27 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which Problems Have Strongly Exponential Complexity? *J. Computer and System Sciences*, 63(4):512–530, 2001.

28 Tao Jiang, Guohui Lin, Bin Ma, and Kaizhong Zhang. The longest common subsequence problem for arc-annotated sequences. *Journal of Discrete Algorithms*, 2(2):257–270, 2004.

29 Hossein Jowhari. Efficient communication protocols for deciding edit distance. In *Proc. European Symposium on Algorithms*, pages 648–658, 2012.

30 Orgad Keller, Tsvi Kopelowitz, and Moshe Lewenstein. On the longest common parameterized subsequence. *Theoretical Computer Science*, 410(51):5347–5353, 2009.

31 Ilan Kremer, Noam Nisan, and Dana Ron. On Randomized One-Round Communication Complexity. *Computational Complexity*, 8(1):21–49, 1999.

32 Keita Kuboi, Yuta Fujishige, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Faster STR-IC-LCS computation via RLE. In *Proc. 28th Annual Symposium on Combinatorial Pattern Matching (CPM'17)*, 2017. To appear, arXiv:1703.04954.

33 Martin Kutz, Gerth Stølting Brodal, Kanela Kaligosi, and Irit Katriel. Faster algorithms for computing longest common increasing subsequences. *Journal of Discrete Algorithms*, 9(4):314–325, 2011.

34 Gad M. Landau, Baruch Schieber, and Michal Ziv-Ukelson. Sparse LCS common substring alignment. *Information Processing Letters*, 88(6):259–270, 2003.

35 David Liben-Nowell, Erik Vee, and An Zhu. Finding longest increasing and common subsequences in streaming data. *Journal of Combinatorial Optimization*, 11(2):155–175, 2006.

36 Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out*, 2004.

37 Jia Jie Liu, Yue-Li Wang, and Richard CT Lee. Finding a longest common subsequence between a run-length-encoded string and an uncompressed string. *Journal of Complexity*, 24(2):173–184, 2008.

38 Webb Miller and Eugene W. Myers. A File Comparison Program. *Softw., Pract. Exper.*, 15(11):1025–1040, 1985. doi:10.1002/spe.4380151102.

39 Johra Muhammad Moosa, M. Sohel Rahman, and Fatema Tuz Zohora. Computing a longest common subsequence that is almost increasing on sequences having no repeated elements. *Journal of Discrete Algorithms*, 20:12–20, 2013.

40 Howard L. Morgan. Spelling correction in systems programs. *Communications of the ACM*, 13(2):90–94, 1970.

41 Shay Mozes, Dekel Tsur, Oren Weimann, and Michal Ziv-Ukelson. Fast algorithms for computing tree LCS. *Theoretical Computer Science*, 410(43):4303–4314, 2009.

42 Eugene W. Myers. An $O(ND)$ Difference Algorithm and Its Variations. *Algorithmica*, 1(2):251–266, 1986. doi:10.1007/BF01840446.

43 Gene Myers. A four russians algorithm for regular expression pattern matching. *Journal of the ACM (JACM)*, 39(2):432–448, 1992.

44 Narao Nakatsu, Yahiko Kambayashi, and Shuzo Yajima. A Longest Common Subsequence Algorithm Suitable for Similar Text Strings. *Acta Inf.*, 18:171–179, 1982. doi:10.1007/BF00264437.

45 Pavel Pevzner and Michael Waterman. Matrix longest common subsequence problem, duality and Hilbert bases. In *Proc. 3th Annual Symposium on Combinatorial Pattern Matching (CPM'92)*, pages 79–89, 1992.

46 Michael Saks and C. Seshadhri. Space efficient streaming algorithms for the distance to monotonicity and asymmetric edit distance. In *Proc. 24th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1698–1709, 2013.

47 Xiaoming Sun and David P. Woodruff. The communication and streaming complexity of computing the longest common and increasing subsequences. In *Proc. 18th Annual ACM-SIAM Aymposium on Discrete Algorithms*, pages 336–345, 2007.

48 Alexander Tiskin. Longest common subsequences in permutations and maximum cliques in circle graphs. In *Proc. 17th Annual Symposium on Combinatorial Pattern Matching (CPM'06)*, pages 270–281, 2006.

49 Robert A. Wagner and Michael J. Fischer. The String-to-String Correction Problem. *J. ACM*, 21(1):168–173, 1974. doi:10.1145/321796.321811.

50 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365, 2005.

51 Sun Wu, Udi Manber, Gene Myers, and Webb Miller. An $O(NP)$ Sequence Comparison Algorithm. *Inf. Process. Lett.*, 35(6):317–323, 1990. doi:10.1016/0020-0190(90)90035-V.

52 I-Hsuan Yang, Chien-Pin Huang, and Kun-Mao Chao. A fast algorithm for computing a longest common increasing subsequence. *Information Processing Letters*, 93(5):249–253, 2005.

On the Inner Product Predicate and a Generalization of Matching Vector Families

Balthazar Bauer
ENS, 45 Rue d'Ulm, 75005 Paris, France
balthazar.bauer@ens.fr

Jevgēnijs Vihrovs[1]
Centre for Quantum Computer Science, University of Latvia, Raiņa 19, LV-1586 Riga, Latvia
jevgenijs.vihrovs@lu.lv

Hoeteck Wee[2]
CNRS and ENS, 45 Rue d'Ulm, 75005 Paris, France
wee@di.ens.fr

Abstract

Motivated by cryptographic applications such as predicate encryption, we consider the problem of representing an arbitrary predicate as the inner product predicate on two vectors. Concretely, fix a Boolean function P and some modulus q. We are interested in encoding x to $\vec{\mathbf{x}}$ and y to $\vec{\mathbf{y}}$ so that

$$P(x, y) = 1 \iff \langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle = 0 \bmod q,$$

where the vectors should be as short as possible. This problem can also be viewed as a generalization of matching vector families, which corresponds to the equality predicate. Matching vector families have been used in the constructions of Ramsey graphs, private information retrieval (PIR) protocols, and more recently, secret sharing.

Our main result is a simple lower bound that allows us to show that known encodings for many predicates considered in the cryptographic literature such as greater than and threshold are essentially optimal for prime modulus q. Using this approach, we also prove lower bounds on encodings for composite q, and then show tight upper bounds for such predicates as greater than, index and disjointness.

2012 ACM Subject Classification Security and privacy → Public key encryption

Keywords and phrases Predicate Encryption, Inner Product Encoding, Matching Vector Families

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.41

Acknowledgements We thank Srijita Kundu, Swagato Sanyal and Alexander Belov for helpful discussions, and Krišjānis Prūsis for suggestions on the presentation. We greatly thank Miklos Santha for hospitality during our stay at CQT, and Andris Ambainis for support. We are also grateful to the anonymous reviewers for suggestions and useful feedback.

[1] Work done in part while interning at Centre for Quantum Technologies at NUS. Supported by the ERC Advanced Grant MQC.
[2] Work done in part while visiting Centre for Quantum Technologies at NUS. Supported in part by ERC Project aSCEND (H2020 639554).

38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018).
Editors: Sumit Ganguly and Paritosh Pandya; Article No. 41; pp. 41:1–41:13

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

1 Introduction

There are many situations in cryptography where one is interested in computing some function F of a sensitive input x but the computational model is restricted so that only "simple" functions F can be directly computed. For instance, the entries of x may be encrypted so that only *affine* functions can be computed, or distributed between multiple non-interacting parties so that only *local* functions can be computed, or simply that we only know how to construct schemes for handling simple functions.

For all of these reasons, it is useful to be able to "encode" complex functions as simple functions. An extremely influential example of an "encoding" in the cryptographic literature is that of garbling schemes (or randomized encodings), which have found applications in many areas of cryptography and elsewhere (see [20, 11, 14, 3, 2, 4, 19] and references therein).

In this work, we consider the problem of *inner product encoding*, namely, representing an arbitrary predicate as the inner product predicate on two vectors. Concretely, fix a Boolean function P (a predicate) and some modulus q (may be composite as well as prime). We are interested in mappings $x \mapsto \vec{\mathbf{x}}, y \mapsto \vec{\mathbf{y}}$ that map to vectors in $\mathbb{Z}_q^\ell$ such that for all x, y:

$$P(x,y) = 1 \iff \langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle = 0 \bmod q,$$

and ℓ is as small as possible. This notion is motivated by the study of predicate encryption in [15], where q is typically very large, for instance, as large as the domains of P, and can also be viewed as a natural generalization of matching vector families to arbitrary predicates.

As an example, consider the equality predicate over $[n]$. Here, if $q = 2$, then it is not difficult to show that the vectors must have length $\Omega(n)$. On the other hand, if $q > n$, then it is sufficient to use vectors of length 2: the inner product of $(1, x)$ and $(y, -1)$ is 0 mod q iff $x = y$. More generally, for any predicate $P : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$ and any prime $q \geq 2$, the "truth table" construction achieves vectors of length $\min\{|\mathcal{X}|, |\mathcal{Y}|\}$.

Interestingly, inner product predicate encoding for the equality predicate have been studied in combinatorics and complexity theory, where they are known as matching vector families. Moreover, matching vector families have found many applications, including the construction of Ramsey graphs, private information retrieval (PIR) protocols [13, 21, 10, 7, 8], and more recently, secret-sharing schemes [17, 18, 16]. Here, prior works showed that if q is a prime, then we must use vectors of length $\Omega(n^{\frac{1}{q-1}})$ [7].

1.1 Our results

Our main results are nearly tight bounds for many predicates considered in the cryptographic literature such as greater than and threshold, for both prime and composite modulus q. In particular, we have the following results for prime modulus q:

- Greater than predicate for numbers in $[n]$ requires vectors of length n. This rules out the possibility of deriving the predicate encryption for range queries with $O(\sqrt{n})$ ciphertext and secret key sizes in [6] as a special case of inner product predicate encryption.
- Threshold for n-bit strings and threshold t requires vectors of length 2^{n-t+1}. This rules out the possibility of constructing full-fledged functional encryption schemes by carrying out FHE decryption in the lattice-based predicate encryption of Gorbunov, Vaikuntanathan and Wee [12] using a pairing-based functional encryption scheme for the inner product predicate.

We then investigate encodings for composite q, specifically when q is a product of k distinct primes. In many cases, a lower bound of ℓ/k for composite q follows naturally if our method gives lower bound ℓ for prime q. For predicates such as greater than, index and

Table 1 Summary of upper and lower bounds.

predicate	q prime		q product of k primes	
	upper	lower	upper	lower
$\mathbf{EQ}_n$ [3]	$O(qn^{\frac{1}{q-1}})$	$\Omega(n^{\frac{1}{q-1}})$	$2^{\tilde{O}((\log n)^{1/k})}$	$\Omega(\log n)$
$\mathbf{GT}_n$	n	n	n/k	n/k
$\mathbf{DISJ}_n$ [4], $\mathbf{INDEX}_n, \mathbf{NEQ}_n$	n	n	n/k	n/k
$\mathbf{ETHR}_n^t$ [4] [5]	$n+1$	$n/2$	$n+1$	$n/2k$
$\mathbf{MPOLY}_n^{d,q}$	n^d	n^d	n^d	n^d/k
$\mathbf{THR}_n^t$	n^{n-t+1}	2^{n-t+1}	n^{n-t+1}	$2^{n-t+1}/k$
$\mathbf{OR}\text{–}\mathbf{EQ}_n^q$	2^n	2^n	2^n	$2^n/k$

disjointness, we are able to show tight lower and upper bounds for both prime and composite q. The full summary of upper and lower bounds is shown in Table 1, and the listed predicates are described in Section 3.

Finally, we also consider probabilistic inner product predicate encoding. For example, there is a probabilistic encoding of length $O((\log n)^2)$ for the greater than predicate for numbers in $[n]$, while any deterministic encoding must have length $\Omega(n)$, if q is prime.

Our lower bound technique

Our lower bound technique is remarkably simple. Suppose that q is prime and we can represent a predicate $P : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$ as an inner product predicate on vectors of length r corresponding to mappings $x \mapsto \vec{\mathbf{x}}, y \mapsto \vec{\mathbf{y}}$. Following [5], we consider a matrix F of dimensions $|\mathcal{X}| \times |\mathcal{Y}|$ over $\mathbb{Z}_q$ whose (x, y)'th entry is $\langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle \bmod q$. Then the matrix F has rank at most r, because we can write F as the product of two matrices of dimensions $|\mathcal{X}| \times r$ and $r \times |\mathcal{Y}|$. Concretely, $F = UV$ where the x'th row of U is $\vec{\mathbf{x}}^{\mathrm{T}}$ and the y'th column of V is $\vec{\mathbf{y}}$. This means that to show a lower bound on r, it suffices to show that F has large rank, e.g. by exhibiting a full rank submatrix.

As an example, consider the greater than predicate on $[n]$ for any prime modulus q. Then, the matrix F is an $n \times n$ upper triangular matrix where all the entries on and above the diagonal are non-zero. This matrix has rank n, hence any correct construction must have dimension at least n. Note that the above lower bound argument breaks down when q is composite. In fact, if $q = 2^n$, there is an encoding for greater than with dimension 1: take $x \mapsto 2^x, y \mapsto 2^{n-y}$. Correctness follows from the fact that $2^x \cdot 2^{n-y} = 0 \bmod 2^n \Leftrightarrow x \geq y$, and the construction extends also to the setting where q is a product of n distinct primes.

In order to extend our lower bounds to composite q that is the product of k distinct primes, we observe that if $F \bmod q$ contains a triangular submatrix of dimensions $\ell \times \ell$, then there exists some prime factor p of q such that $F \bmod p$ contains a triangular submatrix of dimensions $\ell/k \times \ell/k$; this follows from looking at the CRT decomposition of q and a pigeonhole argument. This simple observation allows us to translate many of our lower bounds to the composite modulus setting, which we prove to be essentially optimal via new upper bounds.

[3] Bounds from previous works, see Section 4.1 for references.
[4] For sufficiently large q.
[5] Assuming $t \leq n - 2$, see Section 4.6.

For instance, for the "greater than" predicate, we obtain a tight bound of n/k when q is a product of k distinct primes; this is sharp contrast to standard matching vector families (i.e., the equality predicate), where we have constructions of length $2^{\tilde{O}((\log n)^{1/k})}$ when q is a product of k distinct primes. For the upper bound, we begin with a construction of length 1 for $k = n$ and then derive the more general construction by treating the inputs as vectors of length n and then dividing that into n/k blocks each of length k.

Finally, we extend our results to the randomized setting. Here, we use a similar argument to show that the minimum size of a probabilistic inner product encoding is upper bounded by the probabilistic rank introduced by Alman and Williams [1].

2 Main Theorem

In this section we describe our lower bound technique. Let $P : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$ be a predicate, and $q \geq 2$ be the integer modulus. We say that a matrix $F : \mathcal{X} \times \mathcal{Y}$ *represents* P modulo q if for all $x \in \mathcal{X}, y \in \mathcal{Y}$, we have $F_{x,y} = 0 \bmod q$ iff $P(x, y) = 1$.

An *inner product encoding* of P of length ℓ is a pair of mappings from $\mathcal{X}, \mathcal{Y}$ to Z_q^ℓ that map x, y to $\vec{\mathbf{x}}, \vec{\mathbf{y}}$ in a way that the matrix $F : \mathcal{X} \times \mathcal{Y}$ defined by $F_{x,y} = \langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle \bmod q = (\sum_{i=1}^{\ell} \vec{\mathbf{x}}_i \cdot \vec{\mathbf{y}}_i) \bmod q$ represents P. Denote the length of the shortest reduction from P to inner product modulo q by $\mathrm{DI}(P, q)$ (Deterministic Inner product). Then we have the following simple and effective lower bound method.

▶ **Theorem 1.** *For any predicate P and any prime $q \geq 2$, we have $\mathrm{DI}(P, q) = \min_F \mathrm{rank}(F)$, where F is any matrix that represents P modulo q.*

Proof. We show that if P can be represented by a matrix F modulo q, then the necessary and sufficient length of the encoding from P to F is exactly $\mathrm{rank}(F)$. The decomposition rank definition states that the rank of an $m \times n$ matrix F is the smallest integer r such that F can be factored as $F = UV$, where U is an $m \times r$ matrix and V is a $r \times n$ matrix. Let $U_{x,*}$ be the row vector of U that corresponds to $x \in \mathcal{X}$ and $V_{*,y}$ be the column vector of V that corresponds to $y \in \mathcal{Y}$. Then the pair of mappings $x \mapsto U_{x,*}^{\mathrm{T}}$ and $y \mapsto V_{*,y}$ is a correct encoding of P, which is also the shortest possible for F. ◀

Therefore, to show a lower bound on the length of an encoding for P, it is sufficient to exhibit a set of rows R and a set of columns C such that for any matrix F that represents P, the submatrix $F[C, R]$ is a full rank submatrix. Typically we find a large full rank upper triangular submatrix and apply Theorem 1. Other times, we prove a lower bound for some predicate Q, and then prove that the same lower bound holds for P by showing a predicate reduction from Q to P (see Section 3 for details).

For composite q, we have the following lower bound:

▶ **Theorem 2.** *Let $q = p_1 \cdots p_k$ be a product of k distinct primes. Let P be a predicate such that every matrix F that represents P modulo q is a triangular $n \times n$ matrix such that all numbers on the main diagonal are non-zero modulo q. Then $\mathrm{DI}(P, q) \geq n/k$.*

Proof. Let F represent P modulo q. Let $F^{(i)} = F \bmod p_i$ (all entries taken modulo p_i). Since all entries on the main diagonal of F are non-zero, there exists $i \in [k]$ such that there at least n/k non-zero entries on the main diagonal of $F^{(i)}$ by pigeonhole principle. As $F^{(i)}$ is also a triangular matrix, the rank of $F^{(i)}$ modulo p_i is at least n/k. By Theorem 1, the length of any encoding from P to $F^{(i)}$ modulo p_i must be at least n/k, hence also $\mathrm{DI}(P, q) \geq n/k$. ◀

3 Definitions and Predicates

In this section, first we describe some of the notation used throughout the paper. Then we define the predicates examined in the paper, and define the predicate reduction.

Notation

We denote the set of all subsets of $[n]$ by $2^{[n]}$. For a set $S \subseteq [n]$, define the characteristic vector $\chi(S) \in \{0,1\}^n$ by

$$\chi(S)_i = \begin{cases} 1, & \text{if } i \in S, \\ 0, & \text{otherwise.} \end{cases}$$

Conversely, for a vector $x \in \{0,1\}^n$, let $\chi^{-1}(x)$ be the characteristic set of x.

For simplicity, denote the characteristic vector of $\{i\}$ by $\mathbf{e}_i$ (the length is usually inferred from the context). The characteristic vectors of $\varnothing$ and $[n]$ are denoted by 0^n and 1^n. We denote the identity matrix of dimension n by I_n, and all ones matrix by J_n.

For a truth expression T, we define $[T]$ to be 1 if T is true, and 0 if T is false. For example, $[x = y] = 1$ iff $x = y$.

For a number $x \in [2^n]$, let $\mathrm{bin}(x) \in \{0,1\}^n$ be the binary representation of $x - 1$.

Predicates

We consider the predicates listed below.

- Equality: $\mathcal{X} = \mathcal{Y} = [n]$ and $\mathbf{EQ}_n(x,y) = [x = y]$.
- Greater than: $\mathcal{X} = \mathcal{Y} = [n]$ and $\mathbf{GT}_n(x,y) = [x > y]$.
- Inequality: $\mathcal{X} = \mathcal{Y} = [n]$ and $\mathbf{NEQ}_n(x,y) = [x \neq y]$.
- Index: $\mathcal{X} = \{0,1\}^n, \mathcal{Y} = [n]$ and $\mathbf{INDEX}_n(x,i) = [x_i = 0]$. Here, x_i denotes the i'th coordinate of x. Note that we can also interpret x as the characteristic vector of a subset of $[n]$. Because in our model 0 mod q corresponds to "true", we have defined the index to be true if the bit value in the corresponding position is 0.
- Disjointness: $\mathcal{X} = \mathcal{Y} = 2^{[n]}$ and $\mathbf{DISJ}_n(S,T) = [S \cap T = \varnothing]$.
- Exact threshold: $\mathcal{X} = \mathcal{Y} = 2^{[n]}$ and $\mathbf{ETHR}^t_n(S,T) = [|S \cap T| = t]$, where $t \in [n]$ is the threshold parameter.
- Threshold: $\mathcal{X} = \mathcal{Y} = 2^{[n]}$ and $\mathbf{THR}^t_n(S,T) = [|S \cap T| \geq t]$, where $t \in [n]$ is the threshold parameter.
- Multilinear polynomials: $\mathcal{X} = \mathbb{Z}_q^n$, $\mathcal{Y} \subseteq \{p \mid p \in \mathbb{Z}_q[x_1, \ldots, x_n], \deg(p) \leq d\}$, the latter is the set of all multilinear polynomials of degree at most d. Then $\mathbf{MPOLY}^{d,q}_n(x,p) = [p(x_1, \ldots, x_n) = 0 \bmod q]$.
- Disjunction of equality tests: $\mathcal{X} = \mathcal{Y} = \mathbb{Z}_q^n$ and $\mathbf{OR}{-}\mathbf{EQ}^q_n(x,y) = [\bigvee_{i=1}^n x_i = y_i]$.

Reductions

We say that a predicate $P_1 : \mathcal{X}_1 \times \mathcal{Y}_1 \to \{0,1\}$ can be *reduced* to a predicate $P_2 : \mathcal{X}_2 \times \mathcal{Y}_2 \to \{0,1\}$ if there exist two mappings $f : \mathcal{X}_1 \to \mathcal{X}_2$ and $g : \mathcal{Y}_1 \to \mathcal{Y}_2$ such that $P_2(f(x), g(y)) = P_1(x,y)$ for all $x \in \mathcal{X}_1, y \in \mathcal{Y}_1$ (or mappings $f : \mathcal{X}_1 \to \mathcal{Y}_2$ and $g : \mathcal{Y}_1 \to \mathcal{X}_2$). In that case we write $P_2 \Rightarrow P_1$.

For example, consider the following reductions:

- $\mathbf{DISJ}_n \Rightarrow \mathbf{INDEX}_n \Rightarrow \mathbf{NEQ}_n$.
  The reduction $\mathbf{DISJ}_n \Rightarrow \mathbf{INDEX}_n$ holds since $\mathbf{INDEX}_n(x,i) = \mathbf{DISJ}_n(\chi^{-1}(x), \{i\})$. On the other hand, $\mathbf{INDEX}_n \Rightarrow \mathbf{NEQ}_n$, as $\mathbf{NEQ}_n(i,j) = \mathbf{INDEX}_n(\mathbf{e}_i, j)$.
- $\mathbf{INDEX}_n \Rightarrow \mathbf{GT}_n$.
  As $\mathbf{GT}_n(x,y) = \mathbf{INDEX}_n(\chi([y]), x)$, the reduction follows.
- Let $P : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ be any predicate. Then $\mathbf{INDEX}_{\min\{|\mathcal{X}|,|\mathcal{Y}|\}} \Rightarrow P$.
  Let T be the $\mathcal{X} \times \mathcal{Y}$ truth table of P defined by $T_{x,y} = P(x,y)$. Then we have $P(x,y) = \mathbf{INDEX}_{|\mathcal{X}|}(T_x, y)$ and $\mathbf{INDEX}_{|\mathcal{X}|} \Rightarrow P$. Similarly, we also have $\mathbf{INDEX}_{|\mathcal{Y}|} \Rightarrow P$.

Effectively, then an inner product encoding for P_2 implies an encoding for P_1 and a lower bound for P_1 implies a lower bound for P_2. This makes it easier to prove upper and lower bounds. For example, as later we prove that $\mathrm{DI}(\mathbf{INDEX}_n, q) = n$ for prime q (see Section 4.2), the last reduction implies that $\mathrm{DI}(P,q) \leq \min\{|\mathcal{X}|, |\mathcal{Y}|\}$ for all predicates P.

If q is a product of k distinct primes, then $\mathrm{DI}(P,q) \leq \min\{|\mathcal{X}|, |\mathcal{Y}|\}/k$ for the same reason. Therefore, for any predicate, if $k = \min\{|\mathcal{X}|, |\mathcal{Y}|\}$, there is an encoding of $\mathcal{X}$ and $\mathcal{Y}$ simply to numbers modulo q.

4 Deterministic Encodings

In this section, we apply our technique to provide lower bounds on deterministic inner product encodings for many well-known predicates. For each of them, first we discuss the encodings and then proceed to prove lower bounds.

4.1 Equality

An encoding for $\mathbf{EQ}_n$ over q is a matching family of vectors modulo q [7]. The maximum size of a matching family of vectors of length ℓ modulo q is denoted by $\mathrm{MV}(q,\ell)$ and has been studied extensively. Lower and upper bounds on $\mathrm{MV}(q,\ell)$ give upper and lower bounds on $\mathrm{DI}(\mathbf{EQ}_n, q)$, respectively (in the relevant literature, usually q and ℓ are denoted by m and n, respectively).

For prime q, a tight $\mathrm{DI}(\mathbf{EQ}_n, q) = \Theta(qn^{\frac{1}{q-1}})$ bound is known [7]. If q is a product of k primes, we have a $2^{\tilde{O}\left((\log n)^{1/k}\right)}$ upper bound from [13]. For any composite q, we also have an $\Omega(\log n)$ lower bound from [9].

Here, first we show two simple upper bounds for $q = 2$ and $q \geq n$. Then we reprove the optimal lower bound for $q = 2$ using our rank lower bound.

Upper bounds

For $q = 2$, we construct an encoding of length n. Let $\vec{\mathbf{x}} = \mathbf{e}_x$ and $\vec{\mathbf{y}} = 1^n - \mathbf{e}_y$. Then $\langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle = \langle \mathbf{e}_x, 1^n \rangle - \langle \mathbf{e}_x, \mathbf{e}_y \rangle = 1 - [x = y]$, thus it is a correct inner product encoding and $\mathrm{DI}(\mathbf{EQ}_n, 2) \leq n$.

Let q be any integer such that $q \geq n$. Let $\vec{\mathbf{x}} = (1, x)$ and $\vec{\mathbf{y}} = (y, -1)$. Then $\langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle = y - x$, so it is 0 iff $x = y$. Therefore, $\mathrm{DI}(\mathbf{EQ}_n, q) \leq 2$.

Lower bound

We show a matching lower bound for case $q = 2$. There is a unique matrix F over $\mathbb{Z}_2$ that represents $\mathbf{EQ}_n$, namely $F_{x,y} = 0 \bmod q \Leftrightarrow x = y$. Express $F = J_n - I_n$. By sub-additivity of rank, we have $\mathrm{rank}(F) \geq \mathrm{rank}(I_n) - \mathrm{rank}(J_n) = n - 1$. Hence, by Theorem 1, any inner product encoding of $\mathbf{EQ}_n$ modulo 2 requires vectors of length at least $n - 1$, that is, $\mathrm{DI}(\mathbf{EQ}_n, 2) \geq n - 1$.

4.2 Index

We prove that $\mathrm{DI}(\mathbf{INDEX}_n, q) = \lceil n/k \rceil$, for every q that is a product of k distinct primes.

For some q, the upper bound follows from $\mathbf{DISJ}_n \Rightarrow \mathbf{INDEX}_n$ (see Section 4.5). However, there is a much simpler encoding, which we present below. Moreover, this upper bound holds for every q that is the product of k distinct primes.

Upper bound

We begin with the warm-up for the special case $k = n$. Here, consider

$$\vec{\mathbf{x}} = \prod_{i=1}^{n} p_i^{1-x_i}, \qquad \vec{\mathbf{y}} = q/p_y.$$

Then $\langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle = 0 \bmod q$ iff $x_y = 0$.

Next, we consider general k, n. Since $\mathbf{INDEX}_{\lceil n/k \rceil \cdot k} \Rightarrow \mathbf{INDEX}_n$, it is enough to construct an encoding for the case $k \mid n$. The data is the string $x \in \{0,1\}^n$, and the index is given by $y \in [n]$. Encode x as an $n/k \times k$ binary matrix $X_{i,j} = x_{(i-1)\cdot k + j}$, and y as an $n/k \times k$ binary matrix $Y_{i,j} = [y = (i-1)\cdot k + j]$.

Now we construct the encoding.

$$\vec{\mathbf{x}}_i = \prod_{j=1}^{k} p_j^{X_{i,j}}, \qquad \vec{\mathbf{y}}_i = \begin{cases} q/p_j, & \text{if } Y_{i,j} = 1, \\ 0, & \text{otherwise.} \end{cases}$$

Now we analyze the correctness of the protocol. Let i, j be such that $Y_{i,j} = 1$. Then $\langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle = \prod_{l=1}^{k} p_l^{X_{i,l}} \cdot (q/p_j)$.

- If $X_{i,j} = 1$, then $\langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle = 0 \bmod q$.
- If $X_{i,j} = 0$, then $p_j \nmid \langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle$, hence $\langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle \neq 0 \bmod q$.

Lower bound

The lower bound follows from $\mathbf{INDEX}_n \Rightarrow \mathbf{NEQ}_n$ (see Section 4.3).

4.3 Inequality

We show that $\mathrm{DI}(\mathbf{NEQ}_n, q) = \lceil n/k \rceil$, for every q that is the product of k distinct primes.

Upper bound

The upper bound follows from $\mathbf{INDEX}_n \Rightarrow \mathbf{NEQ}_n$ (see Section 4.2).

Lower bound

Any matrix that represents $\mathbf{NEQ}_n$ is a diagonal matrix with non-zero entries on the main diagonal. By Theorem 2, it follows that $\mathrm{DI}(\mathbf{NEQ}_n, q) \geq n/k$.

4.4 Greater Than

We show that $\mathrm{DI}(\mathbf{GT}_n, q) = \lceil n/k \rceil$, for every q that is the product of k distinct primes.

Upper bound

The upper bound follows from $\mathbf{INDEX}_n \Rightarrow \mathbf{GT}_n$ (see Section 4.2).

If q is prime, the encoding simplifies to $\vec{\mathbf{x}} = \mathbf{e}_x$ and $\vec{\mathbf{y}} = \sum_{i=1}^{y} \mathbf{e}_i$. If $k = n$, a different simple encoding is $\vec{\mathbf{x}} = \prod_{i=1}^{x-1} p_i$ and $\vec{\mathbf{y}} = \prod_{i=y+1}^{n} p_i$.

Lower bound

Let F be any matrix that represents $\mathbf{GT}_n$ modulo q. Then all entries below the main diagonal are 0, while all entries on and above the main diagonal are non-zero, hence F is a triangular matrix. By Theorem 2, we conclude that $\mathrm{DI}(\mathbf{GT}_n, q) \geq n/k$.

4.5 Disjointness

We prove that $\mathrm{DI}(\mathbf{DISJ}_n, q) = \lceil n/k \rceil$ for an appropriate choice of q that depends on n, and that $\mathrm{DI}(\mathbf{DISJ}_n, q) \geq n/k$ if q is any product of k distinct primes.

Upper bound

We start with a simple encoding for $k = n$ that works for any product of n distinct primes q. Recall that the sets S and T are the input to disjointness. Let

$$\vec{\mathbf{x}} = \prod_{i=1}^{n} p_i^{1-\chi(S)_i}, \qquad \vec{\mathbf{y}} = \prod_{i=1}^{n} p_i^{1-\chi(T)_i}.$$

Then $\langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle = \prod_{i=1}^{n} p_i^{2-\chi(S)_i-\chi(T)_i}$ is 0 mod q iff S and T are disjoint. If $k < n$, then for any p_i it is possible that although some of the products $\vec{\mathbf{x}}_i \cdot \vec{\mathbf{y}}_i$ are not divisible by p_i, their sum might be divisible by p_i, hence the encoding doesn't work for any q.

For the general case, the following variation of Dirichlet's theorem will be useful for us.

▶ **Theorem 3** (Dirichlet). *For any integer $q \geq 2$, there are infinitely many primes p such that $p = 1 \bmod q$.*

Let $q = p_1 \cdots p_k$ be a product of k distinct primes $p_1, \ldots, p_k$ to be defined later. We construct an encoding of length n/k for the case $k \mid n$. Encode $S \subseteq [n]$ as an $n/k \times k$ binary matrix $X_{i,j} = \chi(S)_{(i-1)\cdot k+j}$. Similarly encode T as Y. Let

$$\vec{\mathbf{x}}_i = \prod_{j=1}^{k} p_j^{1-X_{i,j}}, \qquad \vec{\mathbf{y}}_i = \prod_{j=1}^{k} p_j^{1-Y_{i,j}}.$$

Now we find the appropriate primes $p_1, \ldots, p_k$ for the general case. We construct them and prove the correctness by induction on k.

Base case. If $k = 1$, then q is a prime itself. Pick any prime q such that $q > n$. We have $\langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle = \sum_{i=1}^{n} q^{2-\chi(S)_i-\chi(T)_i}$. If x and y are disjoint, then $q \mid \langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle$. Suppose that S and T are not disjoint. Let $b = |S \cap T|$. Then $\langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle = (\sum_{i \in S \cap T} 1) \bmod q = b \bmod q$. As $b \leq n$, we have $b < q$, therefore $\langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle \neq 0 \bmod q$.

Inductive step. Assume that there exists a correct encoding for some q such that it is a product of $k-1$ distinct primes $p_1, \dots, p_{k-1}$. Let p_k be a prime such that $p_k > (n/k) \cdot (p_1 \cdots p_{k-1})^2$ and $p_k = 1 \bmod (p_1 \cdots p_{k-1})$ (such exist by Theorem 3).

Suppose that $\langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle = p_k \cdot a + b$, where $b \in \{0, \dots, p_k - 1\}$. Examine the sets $S^{(k)} = \{i \in [n/k] \mid ik \in S\}$ and $T^{(k)} = \{i \in [n/k] \mid ik \in T\}$.

- Suppose that $S^{(k)}$ and $T^{(k)}$ are not disjoint. Then the set $I = S^{(k)} \cap T^{(k)}$ is non-empty. If $i \notin I$, then at least one of $X_{i,k}$ and $Y_{i,k}$ is 0, thus $\vec{\mathbf{x}}_i \cdot \vec{\mathbf{y}}_i = \prod_{j=1}^{k} p_j^{2-X_{i,j}-Y_{i,j}}$ is divisible by p_k. Thus, we have that $b = \sum_{i \in I} \prod_{j=1}^{k-1} p_j^{2-X_{i,j}-Y_{i,j}} < (n/k) \cdot (p_1 \cdots p_{k-1})^2 < p_k$. Therefore, $\langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle = b \bmod p_k \neq 0 \bmod p_k$.
- Suppose that $S^{(k)}$ and $T^{(k)}$ are disjoint. Then for all $i \in [n/k]$, we have that $p_k \mid \vec{\mathbf{x}}_i \vec{\mathbf{y}}_i$. Therefore, $p_k \mid \langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle$.
  Moreover, since $p_k = 1 \bmod (p_1 \cdots p_{k-1})$, we have that $\vec{\mathbf{x}}_i \bmod (p_1 \cdots p_{k-1}) = \prod_{j=1}^{k-1} p_j^{1-X_{i,j}}$ and $\vec{\mathbf{y}}_i \bmod (p_1 \cdots p_{k-1}) = \prod_{j=1}^{k-1} p_j^{1-Y_{i,j}}$. Therefore, $\langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle \bmod (p_1 \cdots p_{k-1})$ is equal to 0 iff the sets $S \setminus S^{(k)}$ and $T \setminus T^{(k)}$ are disjoint by the inductive hypothesis.

Lower bound

The lower bound follows from $\mathbf{DISJ}_n \Rightarrow \mathbf{INDEX}_n$ (see Section 4.2).

4.6 Exact Threshold

Upper bound

The following encoding modulo $q \geq n$ of length $n+1$ is due to Katz, Sahai and Waters [15]. For all $1 \leq i \leq n$, let $\vec{\mathbf{x}}_i = \chi(S)_i$, and let $\vec{\mathbf{x}}_{n+1} = 1$. For all $1 \leq i \leq n$, let $\vec{\mathbf{y}}_i = \chi(T)_i$, and let $\vec{\mathbf{y}}_{n+1} = -t$. Then $\langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle$ is equal to 0 iff $|S \cap T| = t$. Therefore, $\mathrm{DI}(\mathbf{ETHR}_n^t, q) \leq n+1$.
Surprisingly, if $t \geq n-1$, there exist constant size encodings.

- If $t = n$, there is an encoding of length 2. The encoding is as follows: $\vec{\mathbf{x}} = (1, [S = [n]])$ and $\vec{\mathbf{y}} = (1, -[T = [n]])$. Then we have $\langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle = 1 - [S = [n]] \cdot [T = [n]]$, which is 0 iff $S = T = [n]$.
- If $t = n-1$, there is an encoding of length 3. The encoding for S and T is as follows:

$$\vec{\mathbf{x}} = \begin{cases} (1,0,0), & \text{if } |S| = n, \\ (0,i,1), & \text{if } |S| = [n] \setminus \{i\}, \\ (1,-1,1), & \text{otherwise.} \end{cases} \qquad \vec{\mathbf{y}} = \begin{cases} (1,0,0), & \text{if } |T| = n, \\ (0,1,-i), & \text{if } |T| = [n] \setminus \{i\}. \\ (1,1,1), & \text{otherwise.} \end{cases}$$

It is easy to check by hand that $\langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle = 0$ iff $|S \cap T| = n-1$. Note that we require $q \geq n+2$.

Lower bound

We show that for $1 \leq t \leq n-2$, we have $\mathrm{DI}(P, q) \geq \max\{n-t+2, t+2\}/k \geq (n/2+2)/k$.

(a) First we prove that if $t \geq 1$, the length of any encoding must be at least $(n-t+2)/k$. We show that by using two reductions.
Firstly, we have $\mathbf{ETHR}_n^t \Rightarrow \mathbf{ETHR}_{n-t+1}^1$, because we can map $S \mapsto S \cup \{n-t+2, \dots, n\}$.
Secondly, we prove that $\mathbf{ETHR}_m^1 \Rightarrow \mathbf{GT}_{m+1}$. Consider the following mappings:

$$f = \begin{cases} 1 \mapsto \varnothing, \\ i \mapsto [i-1], \end{cases} \qquad g = \begin{cases} j \mapsto \{j\}, \\ m+1 \mapsto \varnothing. \end{cases} \tag{1}$$

Consider a pair of numbers $x, y \in [m+1]$. If $x = 1$, then $\mathbf{GT}_{m+1}(x, y) = 0$ and also $\mathbf{ETHR}^1_m(f(x), g(y)) = \mathbf{ETHR}^1_m(\varnothing, g(y)) = 0$. If $y = m+1$, then $\mathbf{GT}_{m+1}(x, y) = 0$ and $\mathbf{ETHR}^1_m(f(x), g(y)) = \mathbf{ETHR}^1_m(f(x), \varnothing) = 0$. Otherwise, $\mathbf{ETHR}^1_m(f(x), g(y)) = \mathbf{ETHR}^1_m([x-1], \{y\}) = \mathbf{GT}_{m+1}(x, y)$. Hence the reduction is correct.
Therefore, we conclude that

$$\mathrm{DI}(\mathbf{ETHR}^t_n, q) \geq \mathrm{DI}(\mathbf{ETHR}^1_{n-t+1}, q) \geq \mathrm{DI}(\mathbf{GT}_{n-t+2}, q) \geq (n-t+2)/k$$

by the lower bound on greater than of Section 4.4.

(b) Now we prove that if $t \leq n-2$, the length of any encoding is at least $(t+2)/k$. Again, we exhibit two reductions.
Firstly, $\mathbf{ETHR}^t_n \Rightarrow \mathbf{ETHR}^t_{t+2}$ simply mapping any set to itself. Secondly, $\mathbf{ETHR}^{m-2}_m \Rightarrow \mathbf{NEQ}_m$. This is because we can map $x \mapsto [m] \setminus \{x\}$ for any $x \in [m]$. Then the size of the intersection $|([m] \setminus \{x\}) \cap ([m] \setminus \{y\})|$ is equal to $m-2$ if $x \neq y$, and $m-1$, if $x = y$. Therefore, it follows that

$$\mathrm{DI}(\mathbf{ETHR}^t_n, q) \geq \mathrm{DI}(\mathbf{ETHR}^t_{t+2}, q) \geq \mathrm{DI}(\mathbf{NEQ}_{t+2}, q) \geq (t+2)/k$$

by the lower bound on inequality of Section 4.3.

Therefore, for any $1 \leq t \leq n-2$, any encoding must have length at least $\max\{n-t+2, t+2\}/k$ and we have that $\mathrm{DI}(\mathbf{ETHR}^t_n, q) = \Omega(n)$.

4.7 Multilinear Polynomials

First we show a known encoding that gives $\mathrm{DI}(\mathbf{MPOLY}^{d,q}_n, q) \leq \binom{n}{\leq d} = O(n^d)$. Then we show a lower bound of $\mathrm{DI}(\mathbf{MPOLY}^{d,q}_n, q) \geq \binom{n}{d}/k = \Omega(n^d/k)$. For prime q, there is an optimal lower bound $\mathrm{DI}(\mathbf{MPOLY}^{d,q}_n, q) \geq \binom{n}{\leq d}$.[6]

Upper bound

The following is a simple construction by [15]. For $S \subseteq [n]$, let $X_S = \prod_{i \in S} x_i$ and let $p = \sum_{S \subseteq [n], |S| \leq d} a_S X_S$ be a multilinear polynomial of degree at most d. For each subset $S \subseteq [n]$ such that $|S| \leq d$, let $\vec{\mathbf{x}}_S = X_S$ and $\vec{\mathbf{y}}_S = a_S$; then $\langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle$ is precisely equal to $p(x)$. Since a multilinear polynomial of degree at most d on n variables has at most $\binom{n}{\leq d} = \sum_{i=0}^{d} \binom{n}{i} \leq (n+1)^d$ monomials, it follows that $\mathrm{DI}(\mathbf{MPOLY}^{d,q}_n, q) = O(n^d)$.

Lower bound

We show a reduction $\mathbf{MPOLY}^{d,q}_n \Rightarrow \mathbf{NEQ}_{\binom{n}{d}}$. Let S be the bijection from the numbers in $\left[\binom{n}{d}\right]$ to subsets of $[n]$ of size d. For a pair of inputs $x, y \in \left[\binom{n}{d}\right]$, consider mappings $x \mapsto \chi(S(x))$ and $y \mapsto X_{S(y)}$. Since $\mathbf{MPOLY}^{d,q}_n(\chi(S(x)), X_{S(y)}) = 0$ iff $x \neq y$, it is a correct reduction. Thus, $\mathrm{DI}(\mathbf{MPOLY}^{d,q}_n, q) \geq \binom{n}{d}/k = \Omega(n^d/k)$ by the lower bound from Section 4.3.

Note that if q is prime, we can get a tight lower bound of $\binom{n}{\leq k}$. Let $\ell = \mathrm{DI}(\mathbf{MPOLY}^{d,q}_n, q)$. Since any two distinct polynomials disagree on some inputs, each polynomial must be mapped to a different vector. Therefore, the number of possible vectors must be at least the number of possible polynomials, $|\mathbb{Z}_q^\ell| \geq |\mathcal{Y}|$. The total number of possible monomials of degree at most d is $\binom{n}{\leq d}$. Each monomial can have any coefficient in $\mathbb{Z}_q$. Hence, $q^\ell \geq q^{\binom{n}{\leq d}}$ and $\ell \geq \binom{n}{\leq d}$.

[6] We thank an anonymous reviewer for pointing out this lower bound.

4.8 Threshold

First we show an upper bound of $\mathrm{DI}(\mathbf{THR}^t_n, q) = O(n^{n-t+1})$ for $q > n!$ ($q > n$ if q is prime), and then a lower bound of $\mathrm{DI}(\mathbf{THR}^t_n, q) \geq 2^{n-t+1}/k$ if q is any product of k primes.

Upper Bound

The idea is to encode the threshold into multilinear polynomial evaluation. Let $x = \chi(S)$ and $y = \chi(T)$. Examine the following polynomial:

$$p_y(x) = \left(\sum_{i=1}^{n} x_i y_i - t\right) \cdot \left(\sum_{i=1}^{n} x_i y_i - (t+1)\right) \cdot \ldots \cdot \left(\sum_{i=1}^{n} x_i y_i - n\right).$$

Firstly, $\sum_{i=1}^{n} x_i y_i = |S \cap T|$, thus $p_y(x) = 0$ iff $|S \cap T| \geq t$ (assuming $q > n!$). Secondly, the degree of each factor is 1, hence $\deg(p_y) = n - t + 1$. Note that the polynomial p_y is still multilinear, as all the variables are 0 or 1. Therefore, we have a reduction $\mathbf{MPOLY}^{n-t+1,q}_n \Rightarrow \mathbf{THR}^t_n$. The upper bound from Section 4.7 implies that $\mathrm{DI}(\mathbf{THR}^t_n, q) \leq \mathrm{DI}(\mathbf{MPOLY}^{n-t+1,q}_n, q) \leq \binom{n}{\leq n-t+1} = O(n^{n-t+1})$.

Lower Bound

First of all, we have $\mathbf{THR}^t_n \Rightarrow \mathbf{THR}^1_{n-t+1}$, as we can map a set $S \subseteq [n-t+1]$ to $S \cup \{n-t+2, \ldots, n\}$. Next we prove that $\mathrm{DI}(\mathbf{THR}^1_m, q) \geq 2^m/k$.

Let F be any matrix representing $\mathbf{THR}^1_m$. We show that F is a triangular matrix with all entries on the main diagonal being non-zero. Then the claim follows by Theorem 2.

Order the rows of F by the increasing order of the size of the sets they correspond to. Then order the columns of F in such a way that the sets corresponding to the i-th row and the i-th column are the complements of each other.

As the complements don't overlap, the numbers on the main diagonal of F are non-zero. Now examine any entry on the i-th row and j-th column such that $i \geq j$. Let S correspond to the set of the i-th row and T correspond to the set of the j-th column. Since the columns are ordered by the decreasing size of the sets, we have that $|S| \geq m - |T|$, or equivalently $|S| + |T| \geq m$.

If $|S| + |T| > m$, then the sets must overlap and the value of $F_{i,j}$ is 0. If $|S| + |T| = m$, then the only way S and T do not overlap is if T is the complement of S. In any case all the numbers below the main diagonal are 0, and non-zero on the main diagonal.

4.9 Disjunctions of Equality Tests

We show that for prime q, we have $\mathrm{DI}(\mathbf{OR}{-}\mathbf{EQ}^q_n, q) \leq 2^n$ and if q is a product of k distinct primes, then $\mathrm{DI}(\mathbf{OR}{-}\mathbf{EQ}^q_n, q) \geq 2^n/k$.

Upper bound

We prove that $\mathbf{MPOLY}^{n,q}_n \Rightarrow \mathbf{OR}{-}\mathbf{EQ}^q_n$. Examine a multilinear polynomial $p_y(x) = \prod_{i=1}^{n}(x_i - y_i)$. Clearly, $p_y(x) = 0 \bmod q$ iff at least one equality holds. Therefore, if we map $x \mapsto x$ and $y \mapsto p_y$, then we have a correct reduction to multilinear polynomial evaluation. By the upper bound from Section 4.7, we have $\mathrm{DI}(\mathbf{OR}{-}\mathbf{EQ}^q_n, q) \leq \mathrm{DI}(\mathbf{MPOLY}^{n,q}_n, q) \leq \sum_{i=0}^{n} \binom{n}{i} = 2^n$.

Lower bound

We prove that $\mathbf{OR}{-}\mathbf{EQ}_n^q \Rightarrow \mathbf{NEQ}_{2^n}$. For the input $x, y \in [2^n]$ to $\mathbf{NEQ}_n$, map $x \mapsto \mathrm{bin}(x)$ and $y \mapsto \mathrm{bin}(y) \oplus 1^n$. As $x \neq y$ iff there exists an i such that $\mathrm{bin}(x)_i \neq \mathrm{bin}(y)_i$, we have that $x \neq y$ iff $\mathbf{OR}{-}\mathbf{EQ}_n^q(\mathrm{bin}(x), \mathrm{bin}(y) \oplus 1^n) = 1$. The lower bound follows by Section 4.3.

5 Randomized Constructions

We can formulate the problem in the randomized setting as follows. Let $P : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ be a predicate. Consider all pairs of mappings $\mathcal{U} = \{(x \mapsto \vec{\mathbf{x}}, y \mapsto \vec{\mathbf{y}}) \mid \vec{\mathbf{x}}, \vec{\mathbf{y}} \in \mathbb{Z}_q^\ell \text{ for some } \ell\}$. These also include mappings that are incorrect inner product encodings of P. Let μ be a probability distribution over $\mathcal{U}$. Then μ is a *probabilistic inner product encoding* modulo q with error ϵ, if $\Pr[P(x,y) \neq [\langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle = 0 \bmod q] \mid (x \mapsto \vec{\mathbf{x}}, y \mapsto \vec{\mathbf{y}}) \sim \mu] \leq \epsilon$.

We consider the length of the longest encoding under μ to be the length of μ and denote it by $\mathrm{RI}^\mu(P,q)$ (Randomized Inner product). Then define $\mathrm{RI}_\epsilon(P,q) = \min_\mu \mathrm{RI}^\mu(P,q)$, where μ ranges over all probabilistic inner product encodings of P modulo q with error ϵ.

Next is the definition of the probabilistic rank (over $\mathbb{Z}_q$) by Alman and Williams [1]:

▶ **Definition 4** (Probabilistic Matrix)**.** For $n, m \in \mathbb{N}$, define a *probabilistic matrix* over $\mathbb{Z}_q$ to be a distribution of matrices $\mathcal{M} \subset \mathbb{Z}_q^{n \times m}$. A probabilistic matrix $\mathcal{M}$ *computes* a matrix $A \in \mathbb{Z}_q^{n \times m}$ with error $\epsilon > 0$ if for every entry $(i,j) \in [n] \times [m]$, $\Pr_{M \sim \mathcal{M}}[A_{i,j} \neq M_{i,j}] \leq \epsilon$.

▶ **Definition 5** (Probabilistic Rank)**.** Let q be prime. Then a probabilistic matrix $\mathcal{M}$ has *rank* r if the maximum rank of an M in support of $\mathcal{M}$ is r. Define the *ϵ-probabilistic rank* of a matrix $A \in \mathbb{Z}_q^{n \times m}$ to be the minimum rank of a probabilistic matrix computing M with error ϵ. Denote it by $\mathrm{rank}_\epsilon(A)$.

As we can see, the probabilistic choice of a distribution μ corresponds to a matrix M sampled from $\mathcal{M}$. By a similar reasoning as in Theorem 1, we have the following theorem:

▶ **Theorem 6.** *For any predicate P, prime $q \geq 2$ and error ϵ, $\mathrm{RI}_\epsilon(P,q) \leq \min_F \mathrm{rank}_\epsilon(F)$, where F is any matrix that represents P modulo q.*

Proof. Let F be any matrix that represents P modulo q. Suppose that $\mathcal{M}$ is a probabilistic matrix that computes F. Then any M in support of $\mathcal{M}$ defines an encoding of length $\mathrm{rank}(M)$ by the decomposition rank. Therefore, there is a probability distribution over the encodings such that the maximum length is $\mathrm{rank}_\epsilon(F)$. ◀

For some predicates, the probabilistic rank can be much smaller than the deterministic rank. Let $T(P)$ be a truth table of a predicate P (defined by $T(P)_{x,y} = P(x,y)$). The same authors prove that $\mathrm{rank}_\epsilon(T(\mathbf{EQ}_n)) = O(1/\epsilon)$ and $\mathrm{rank}_\epsilon(T(\mathbf{LEQ}_n)) = O((\log n)^2/\epsilon)$ (see Lemmas D.1 and D.2 in [1]). Since the matrix $T(P)$ represents the predicate $\neg P$ (in our setting), these results imply that for any prime q:

1. $\mathrm{RI}_\epsilon(\mathbf{NEQ}_n, q) = O(1/\epsilon)$,
2. $\mathrm{RI}_\epsilon(\mathbf{GT}_n, q) = O((\log n)^2/\epsilon)$.

We conclude by showing that these results immediately imply a constant length probabilistic encoding for $\mathbf{EQ}_n$ modulo any prime:

▶ **Corollary 7.** *For any prime q, we have $\mathrm{RI}_\epsilon(\mathbf{EQ}_n, q) = O(1/\epsilon)$.*

Proof. Let $\mathcal{M}$ be a probabilistic matrix that computes $T(\mathbf{EQ}_n)$ with error ϵ. The matrix $F(\mathbf{EQ}_n) = J_n - T(\mathbf{EQ}_n)$ represents $\mathbf{EQ}_n$. Therefore, the probabilistic matrix $J_n - \mathcal{M}$ computes $F(\mathbf{EQ}_n)$ with error ϵ. Since $\mathrm{rank}(F(\mathbf{EQ}_n)) \leq 1 + \mathrm{rank}(T(\mathbf{EQ}_n))$, we have that $\mathrm{RI}(\mathbf{EQ}_n, q) = O(1/\epsilon)$. ◀

References

1 Josh Alman and Ryan Williams. Probabilistic Rank and Matrix Rigidity. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 641–652, New York, NY, USA, 2017. ACM.

2 Benny Applebaum. Randomly Encoding Functions: A New Cryptographic Paradigm - (Invited Talk). In *ICITS*, pages 25–31, 2011.

3 Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. *SIAM J. Comput.*, 36(4):845–888, 2006.

4 Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of Garbled Circuits. In *ACM CCS*, 2012. Also Cryptology ePrint Archive, Report 2012/265.

5 Abhishek Bhowmick, Zeev Dvir, and Shachar Lovett. New bounds for matching vector families. In *STOC*, pages 823–832, 2013.

6 Dan Boneh and Brent Waters. Conjunctive, Subset, and Range Queries on Encrypted Data. In *TCC*, pages 535–554, 2007.

7 Zeev Dvir, Parikshit Gopalan, and Sergey Yekhanin. Matching Vector Codes. *SIAM J. Comput.*, 40(4):1154–1178, 2011.

8 Zeev Dvir and Sivakanth Gopi. 2-Server PIR with Sub-Polynomial Communication. In *STOC*, pages 577–584, 2015.

9 Zeev Dvir and Guangda Hu. Matching-Vector Families and LDCs over Large Modulo. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 513–526, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

10 Klim Efremenko. 3-Query Locally Decodable Codes of Subexponential Length. *SIAM J. Comput.*, 41 (6):1694–1703, 2012.

11 Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation. In *STOC*, pages 554–563, 1994.

12 Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate Encryption for Circuits from LWE. In *CRYPTO (2)*, pages 503–523, 2015. Also, Cryptology ePrint Archive, Report 2015/029.

13 Vince Grolmusz. Superpolynomial Size Set-systems with Restricted Intersections mod 6 and Explicit Ramsey Graphs. *Combinatorica*, 20(1):71–86, 2000.

14 Yuval Ishai and Eyal Kushilevitz. Randomizing Polynomials: A New Representation with Applications to Round-Efficient Secure Computation. In *FOCS*, pages 294–304, 2000.

15 Jonathan Katz, Amit Sahai, and Brent Waters. Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products. In *EUROCRYPT*, pages 146–162, 2008.

16 Tianren Liu and Vinod Vaikuntanathan. Breaking the Circuit-Size Barrier in Secret Sharing. STOC 2018. Cryptology ePrint Archive, Report 2018/333, 2018.

17 Tianren Liu, Vinod Vaikuntanathan, and Hoeteck Wee. Conditional Disclosure of Secrets via Non-linear Reconstruction. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, pages 758–790, 2017.

18 Tianren Liu, Vinod Vaikuntanathan, and Hoeteck Wee. Towards Breaking the Exponential Barrier for General Secret Sharing. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*, pages 567–596, 2018.

19 Manoj Prabhakaran and Amit Sahai. *Secure Multi-Party Computation*. IOS Press, 2003.

20 Andrew Chi-Chih Yao. Theory and Applications of Trapdoor Functions. In *FOCS*, pages 80–91, 1982.

21 Sergey Yekhanin. Towards 3-query locally decodable codes of subexponential length. *J. ACM*, 55(1):1:1–1:16, 2008.

Extending Propositional Separation Logic for Robustness Properties

Alessio Mansutti
LSV, CNRS, ENS Paris-Saclay, Université Paris-Saclay, Cachan, France

Abstract

We study an extension of *propositional separation logic* that can specify *robustness properties*, such as acyclicity and garbage freedom, for automatic verification of stateful programs with singly-linked lists. We show that its satisfiability problem is PSPACE-complete, whereas modest extensions of the logic are shown to be TOWER-hard. As separating implication, reachability predicates (under some syntactical restrictions) and a unique quantified variable are allowed, this logic subsumes several PSPACE-complete separation logics considered in previous works.

2012 ACM Subject Classification Theory of computation → Separation logic

Keywords and phrases Separation logic, decision problems, reachability, logics on trees, interval temporal logic, adjuncts and quantifiers elimination

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.42

Acknowledgements I would like to thank S. Demri and E. Lozes for their feedback.

1 Introduction

Separation Logic [26] is a well-known assertion logic providing a scalable solution for Hoare-style verification of imperative, heap-manipulating programs [7, 16, 29]. To achieve scalability, separation logic relies in two spatial connectives to represent memory regions: the *separating conjunction* ($*$) and the *separating implication* ($-\!\!*$). These operators allow to express complex properties of stateful programs, making this logic the core assertion language of many tools [2, 3, 6, 14, 15, 18, 20, 23]. To achieve automation, the underlying Hoare-style proof system requires these tools to check for the classical decision problems of satisfiability, validity and entailment. The complexity of these problems have been quite studied:

- PTIME algorithms for satisfiability and entailment have been defined for the *symbolic heap fragment* (the core logic of many tools) [9]. This complexity is achieved by removing the separating implication from separation logic and heavily restricting the use of Boolean connectives. Decidability results (the general lower-bound is EXPTIME) for these problems are also known when the fragment is enriched with inductive predicates [1, 17, 19, 21].
- PSPACE-completeness has been shown for propositional separation logic [8]. Its extension with one quantified variable, denoted with $\mathsf{1SL}(*, -\!\!*)$, was also found to be in PSPACE [12]. However, adding a second quantified variable causes the logic to become undecidable [11].
- In absence of the separating implication, PSPACE-completeness has also been proved for $\mathsf{SL}(*, \mathtt{ls})$, i.e. the propositional fragment enriched with the list-segment predicate $\mathtt{ls}$. Here, adding the separating implication again leads to undecidability [13].

Besides these decision problems, in program analysis it is crucial to be able to check for *robustness properties* such as *garbage freedom* and *acyclicity* (see Section 2 for precise definitions). A recent work [19] tackles these problems for the symbolic heap fragment with user-defined inductive predicates by introducing the framework of heap automata. Within this framework, both garbage freedom and acyclicity are shown to be EXPTIME-complete.

A natural question is how to check the satisfaction of robustness properties for propositional separation logic as they are not expressible in $\mathsf{1SL}(*, -\!\!*)$ nor in $\mathsf{SL}(*, \mathtt{ls})$. Indeed, it would be nice to capture these problems directly in the logic, without introducing any external framework, to then solve them using procedures for the classical decision problems.

Our contribution. In this paper we address this question by studying an extension of propositional separation logic that captures both $\mathsf{1SL}(*, -\!\!*)$ and $\mathsf{SL}(*, \mathtt{ls})$ and whose expressive power allows to directly reduce the robustness properties to entailment. This logic, herein called $\mathsf{1SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$, is defined from $\mathsf{1SL}(*, -\!\!*)$ by adding reachability predicates under some syntactical restrictions (the formal definition is given in Section 2). In Section 4 we show that the satisfiability problem (and hence entailment, validity and robustness properties) of this logic can be decided in PSpace. As far as we know, this makes $\mathsf{1SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$ the largest decidable fragment of separation logic including full Boolean connectives, spatial connectives and reachability predicates and the first one where these predicates can be used in the scope of $-\!\!*$, albeit in a controlled way to retain decidability [13]. To show the PSpace upper-bound we extend the widely used proof technique of test formulæ introduced in [22].

This complexity result is rather surprising as, besides subsuming the results in [12] and [13], slightly extending the logic entails Tower-hardness (the complexity class Tower has been introduced in [28] and sits between the class of elementary problems and the class of primitive-recursive problems). Indeed, in Section 3 we show how weakening the syntactic restrictions on reachability predicates allows the logic to capture a Tower-complete fragment of Moszkowski's propositional interval temporal logic (PITL) [25]. To better formalise this result we first introduce an alternative semantics for PITL and reduce this logic to an intermediate logic interpreted on trees (ALT). We then consider a modest extension of $\mathsf{1SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$ and show that it captures ALT, proving its non-elementary complexity.

2 The separation logic $\mathsf{1SL}(*, -\!\!*, \mathtt{reach}^+)$

Let VAR be a countably infinite set of program variables and let LOC be a countably infinite set of locations. A *memory state* is a pair (s, h) consisting of a variable valuation function (the *store*) $s : \mathsf{VAR} \to \mathsf{LOC}$ and a partial function with finite domain (the *heap*) $h : \mathsf{LOC} \to_{\mathtt{fin}} \mathsf{LOC}$. We denote with $\mathrm{dom}(h)$ the domain of definition of a heap h and with $\mathrm{ran}(h)$ its range. Each element in $\mathrm{dom}(h)$ is understood as a *memory cell* of h. With h^δ we denote $\delta \geq 0$ functional composition(s) of h. Two heaps h_1 and h_2 are said to be disjoint, written $h_1 \perp h_2$, whenever $\mathrm{dom}(h_1) \cap \mathrm{dom}(h_2) = \emptyset$. We define the union $h_1 + h_2$ of h_1 and h_2 as the standard sum of two functions $(h_1 + h_2)(\ell) \stackrel{\text{def}}{=} \textbf{if } \ell \in \mathrm{dom}(h_1) \textbf{ : } h_1(\ell) \textbf{ else } h_2(\ell)$, defined only whenever $h_1 \perp h_2$.

We extend propositional separation logic with reachability predicates and one quantified variable denoted by $\mathtt{u} \notin \mathsf{VAR}$. We call this logic $\mathsf{1SL}(*, -\!\!*, \mathtt{reach}^+)$. Its formulæ φ are from

$$\varphi := \mathtt{emp} \mid e_1 = e_2 \mid e_1 \hookrightarrow e_2 \mid \mathtt{reach}^+(e_1, e_2) \mid \varphi \wedge \varphi \mid \neg\varphi \mid \exists \mathtt{u}\, \varphi \mid \varphi * \varphi \mid \varphi -\!\!* \varphi$$

where $e_1, e_2 \in \mathsf{VAR} \cup \{\mathtt{u}\}$. We denote with $\mathsf{fv}(\varphi)$ the set of free variables in φ. $\mathsf{1SL}(*, -\!\!*, \mathtt{reach}^+)$ is interpreted on triples (s, h, l), where (s, h) is a memory state and $l \in \mathsf{LOC}$ is the current assignment of the only quantified variable $\mathtt{u}$. The satisfaction relation $\models$ is defined as follows (standard clauses for $\neg$ and $\wedge$ are omitted throughout the paper)

- $(s, h, l) \models \mathtt{emp}$ if and only if $\mathrm{dom}(h) = \emptyset$.
- $(s, h, l) \models e_1 = e_2$ if and only if $[\![e_1]\!] = [\![e_2]\!]$, with $[\![\mathtt{u}]\!] \stackrel{\text{def}}{=} l$ and $[\![\mathtt{x}]\!] \stackrel{\text{def}}{=} s(\mathtt{x})$ for every $\mathtt{x} \in \mathsf{VAR}$.
- $(s, h, l) \models e_1 \hookrightarrow e_2$ if and only if $s([\![e_1]\!]) = [\![e_2]\!]$.
- $(s, h, l) \models \mathtt{reach}^+(e_1, e_2)$ if and only if there is $\delta \geq 1$ such that $h^\delta([\![e_1]\!]) = [\![e_2]\!]$.

- $(s,h,l) \models \exists \mathtt{u}\ \varphi$ if and only if there is $l' \in \mathsf{LOC}$ such that $(s,h,l') \models \varphi$.
- $(s,h,l) \models \varphi_1 * \varphi_2$ iff $h_1 + h_2 = h$, $(s,h_1,l) \models \varphi_1$ and $(s,h_2,l) \models \varphi_2$, for some h_1 and h_2.
- $(s,h,l) \models \varphi_1 \mathrel{-\!\!*} \varphi_2$ iff for all h', if $h' \perp h$ and $(s,h',l) \models \varphi_1$ then $(s,h+h',l) \models \varphi_2$.

Standard connectives $\Rightarrow$, $\Leftrightarrow$, $\vee$ and the universal quantification $\forall$ are derived as usual. We denote with $\top$ the tautology $\mathtt{emp} \vee \neg\mathtt{emp}$ and with $\bot$ its negation. $\mathtt{alloc}(e)$ stands for $e \hookrightarrow e \mathrel{-\!\!*} \bot$, the formula satisfied if and only if $[\![e]\!] \in \mathrm{dom}(h)$. We recursively define the formula $\mathtt{size} \geq \beta$ as $\mathtt{size} \geq 0 \stackrel{\text{def}}{=} \top$ and $\mathtt{size} \geq \beta{+}1 \stackrel{\text{def}}{=} \neg\mathtt{emp} * \mathtt{size} \geq \beta$. $\mathtt{size} \geq \beta$ is satisfied if and only if $\mathrm{card}(\mathrm{dom}(h)) \geq \beta$ (we write $\mathrm{card}(.)$ to denote the cardinality of a set). We write $\mathtt{size} = \beta$ for the formula $\mathtt{size} \geq \beta \wedge \neg\mathtt{size} \geq \beta{+}1$. For a complete description of separation logic we refer the reader to the classical paper by Reynolds [26]. Note that the *heap-precise* predicates defined in [26] can be retrieved in our logic. Indeed, the *points-to* relation $e_1 \mapsto e_2$ can be expressed as $e_1 \hookrightarrow e_2 \wedge \mathtt{size} = 1$ whereas the *list-segment* relation $\mathtt{ls}(e_1,e_2)$ can be defined as $(e_1 = e_2 \wedge \mathtt{emp}) \vee (e_1 \neq e_2 \wedge \mathtt{reach}^+(e_1,e_2) \wedge \neg(\mathtt{size} = 1 * \mathtt{reach}^+(e_1,e_2)))$.

Decision problems and robustness properties. The *satisfiability problem* takes as input a formula φ and asks whether there is a model (s,h,l) such that $(s,h,l) \models \varphi$. The *validity problem* asks whether φ is satisfied by every memory state. Given a second formula ψ, the *entailment problem* $\varphi \models \psi$ asks whether each memory state satisfying φ also satisfies ψ.

As advocated in [19], besides these decision problems, in program analysis we are also interested in the *robustness properties* of *acyclicity* and *garbage freedom*. The acyclicity property asks every model satisfying φ to be acyclic. Instead, garbage freedom holds whenever in every model satisfying φ, each memory cell is reachable from a program variable of $\mathsf{fv}(\varphi)$. In $\mathsf{1SL}(*, \mathrel{-\!\!*}, \mathtt{reach}^+)$ both problems can be reduced to entailment:

- acyclicity requires us to be able to solve $\varphi \models \forall \mathtt{u}\ \neg\mathtt{reach}^+(\mathtt{u},\mathtt{u})$;
- garbage freedom can be expressed as $\varphi \models \forall \mathtt{u}\ (\mathtt{alloc}(\mathtt{u}) \Rightarrow \bigvee_{\mathtt{x} \in \mathsf{fv}(\varphi)} (\mathtt{reach}^+(\mathtt{x},\mathtt{u}) \vee \mathtt{x} = \mathtt{u}))$.

As our logic is closed under Boolean connectives, validity and entailment reduce to satisfiability. The main purpose of this paper is then to study the complexity status of the latter problem for fragments of $\mathsf{1SL}(*, \mathrel{-\!\!*}, \mathtt{reach}^+)$ that can still express both robustness properties.

Decidability status and restrictions. As shown in [13], extending propositional separation logic so that it can express bounded reachability up to distance three leads to undecidability. Unfortunately this makes the satisfiability problem of $\mathsf{1SL}(*, \mathrel{-\!\!*}, \mathtt{reach}^+)$ undecidable. Indeed, two-steps reachability between two program variables $\mathtt{x}$ and $\mathtt{y}$ can be expressed as $\exists \mathtt{u}\ (\mathtt{x} \hookrightarrow \mathtt{u} \wedge \mathtt{u} \hookrightarrow \mathtt{y} \wedge \mathtt{u} \neq \mathtt{x} \wedge \mathtt{u} \neq \mathtt{y})$ whereas three-steps reachability is captured with

$$(\mathtt{reach}^+(\mathtt{x},\mathtt{y}) \wedge \mathtt{size} = 3 \wedge \underbrace{\neg(\mathtt{size} = 1 * \mathtt{reach}^+(\mathtt{x},\mathtt{y}))}_{\text{isolating any memory cell makes impossible for } s(\mathtt{x}) \text{ to reach } s(\mathtt{y})}) * \top$$

To retain decidability we propose to restrict the logic to those formulæ where each occurrence of $\mathtt{reach}^+(e_1,e_2)$ is constrained so that

(R1) it is not on the right side of its first $\mathrel{-\!\!*}$ ancestor (seeing the formula as a tree), and

(R2) if $e_1 = \mathtt{u}$ then $e_2 = \mathtt{u}$.

For instance, given two formulæ φ and ψ satisfying these conditions, the formula $\mathtt{reach}^+(\mathtt{u},\mathtt{x}) * (\varphi \mathrel{-\!\!*} \psi)$ only satisfies R1, both conditions are satisfied in $\varphi \mathrel{-\!\!*} (\mathtt{reach}^+(\mathtt{x},\mathtt{y}) \mathrel{-\!\!*} \psi)$ whereas the formula $\varphi \mathrel{-\!\!*} (\psi * \mathtt{reach}^+(\mathtt{u},\mathtt{u}))$ only satisfies R2. A grammar of this logic is given in Section 4, where we show that under these conditions the satisfiability problem of $\mathsf{1SL}(*, \mathrel{-\!\!*}, \mathtt{reach}^+)$ can be decided in PSpace.

From the results in [13], weakening even slightly R1 seems to be a challenge. Indeed, after enforcing R1 the logic can still freely express two-steps reachability between program variables and it is unable to do the same for paths of length three only in positions of the formula where

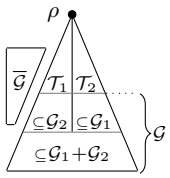
$\texttt{reach}^+$ cannot occur. On these positions, even the modest addition of a second quantified variable causes then undecidability (coherently with the results in [11]). We could hope for the satisfiability problem to still be in PSPACE without the R2 condition. In the next section we show that this is not the case: without R2 the problem is TOWER-hard. Nevertheless, under these conditions the logic is still able to express both robustness properties.

3 Tower-hardness of 1SL($*, -\!\!*, \texttt{reach}^+$) under R1

In this section we show that $\mathsf{1SL}_{\texttt{R1}}(*, -\!\!*, \texttt{reach}^+)$, i.e. the fragment of $\mathsf{1SL}(*, -\!\!*, \texttt{reach}^+)$ formulæ satisfying the condition R1, is TOWER-hard. To do so, we first introduce an auxiliary logic (ALT) interpreted on trees. At its core, ALT is a simple logic whose formulæ can only split a tree and check whether the only (quantified) variable points to a node in the tree or not. In a way, ALT represents a small subset of the properties expressible in $\mathsf{1SL}_{\texttt{R1}}(*, -\!\!*, \texttt{reach}^+)$ that are sufficient to reach TOWER-hardness. Indeed, despite its simplicity, we show that ALT is TOWER-complete. In particular, the hardness proof is achieved by reduction of Moszkowski's propositional interval temporal logic [25] with locality principle (PITL). This reduction is done by first defining an alternative semantics for PITL based on marked words.

3.1 An auxiliary logic on trees ALT

To easily relate ALT with separation logic, finite trees are here defined using heaps encoding the parent relation. We assume a location $\rho \in \mathsf{LOC}$ as the root of all trees. A heap $\mathcal{T}$ is a *tree* whenever $\rho \notin \mathrm{dom}(\mathcal{T})$ and for each $\ell \in \mathrm{dom}(\mathcal{T})$ there is $\delta \geq 1$ such that $\mathcal{T}^\delta(\ell) = \rho$. Then, ℓ is a *descendant* (resp. *child*) of $\ell' \in \mathrm{ran}(\mathcal{T})$ whenever there is $\delta \geq 1$ (resp. $\delta = 1$) such that $\mathcal{T}^\delta(\ell) = \ell'$. It is straightforward to see that these definitions are equivalent to the classical ones. As formally introduced below, ALT formulæ are able to chop a tree, preventing some memory cells to reach ρ. These locations form a heap $\mathcal{G}$, called *garbage*, such that $\rho \notin \mathrm{dom}(\mathcal{G}) \cup \mathrm{ran}(\mathcal{G})$.



We denote with $\mathbb{T}$ the domain of pairs $(\mathcal{T}, \mathcal{G})$ where $\mathcal{T}$ and $\mathcal{G}$ are respectively a tree and a garbage such that $\mathrm{dom}(\mathcal{T}) \cap (\mathrm{dom}(\mathcal{G}) \cup \mathrm{ran}(\mathcal{G})) = \emptyset$. The notions of disjointness ($\perp$) and composition on disjoint heaps ($+$) are naturally extended to elements of $\mathbb{T}$. $(\mathcal{T}_1, \mathcal{G}_1)$ and $(\mathcal{T}_2, \mathcal{G}_2)$ are disjoint whenever $(\mathcal{T}_1 + \mathcal{G}_1) \perp (\mathcal{T}_2 + \mathcal{G}_2)$. If they are disjoint then their composition $(\mathcal{T}_1, \mathcal{G}_1) + (\mathcal{T}_2, \mathcal{G}_2)$ (see picture on the left) is the pair $(\mathcal{T}_1 + \mathcal{T}_2 + \mathcal{G}, \overline{\mathcal{G}}) \in \mathbb{T}$ such that $\mathcal{G} + \overline{\mathcal{G}} = \mathcal{G}_1 + \mathcal{G}_2$ and $\mathrm{dom}(\mathcal{G}) = \{\ell \in \mathrm{dom}(\mathcal{G}_1 + \mathcal{G}_2) \mid (\mathcal{G}_1 + \mathcal{G}_2)^\delta(\ell) \in \mathrm{dom}(\mathcal{T}_1 + \mathcal{T}_2) \text{ for some } \delta \geq 1\}$. ALT-formulæ φ are built from

$$\varphi := \mathsf{T}(\mathtt{u}) \mid \mathsf{G}(\mathtt{u}) \mid \varphi \wedge \varphi \mid \neg\varphi \mid \exists \mathtt{u}\ \varphi \mid \varphi * \varphi$$

and interpreted on *states* $(\mathcal{T}, \mathcal{G}, l)$ where $(\mathcal{T}, \mathcal{G}) \in \mathbb{T}$ and $l \in \mathsf{LOC} \setminus \{\rho\}$ is the current assignment of u. The satisfaction relation $\models$ is defined as follows:

- $(\mathcal{T}, \mathcal{G}, l) \models \mathsf{T}(\mathtt{u})$ if only if $l \in \mathrm{dom}(\mathcal{T})$.
- $(\mathcal{T}, \mathcal{G}, l) \models \mathsf{G}(\mathtt{u})$ if and only if $l \in \mathrm{dom}(\mathcal{G})$.

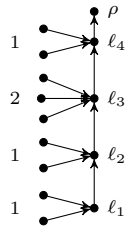
- $(\mathcal{T},\mathcal{G},l) \models \exists \mathtt{u}\ \varphi$ if and only if there is $l' \in \mathsf{LOC} \setminus \{\rho\}$ such that $(\mathcal{T},\mathcal{G},l') \models \varphi$.
- $(\mathcal{T},\mathcal{G},l) \models \varphi_1 * \varphi_2$ iff $(\mathcal{T}_1,\mathcal{G}_1,l) \models \varphi_1$ and $(\mathcal{T}_2,\mathcal{G}_2,l) \models \varphi_2$ for some $(\mathcal{T}_1,\mathcal{G}_1) + (\mathcal{T}_2,\mathcal{G}_2) = (\mathcal{T},\mathcal{G})$.

The tautology $\top$ is defined as $\mathsf{T}(\mathtt{u}) \vee \neg\mathsf{T}(\mathtt{u})$. $\mathtt{alloc}(\mathtt{u}) \stackrel{\text{def}}{=} \mathsf{T}(\mathtt{u}) \vee \mathsf{G}(\mathtt{u})$ is the formula satisfied if and only if $l \in \mathrm{dom}(\mathcal{T}+\mathcal{G})$. The size $|\varphi|$ of a formula φ is defined as: 1 for the atomic predicates $\mathsf{T}(\mathtt{u})$ and $\mathsf{G}(\mathtt{u})$, $|\varphi_1 \wedge \varphi_2| \stackrel{\text{def}}{=} \max(|\varphi_1|, |\varphi_2|)$, $|\neg\varphi| \stackrel{\text{def}}{=} |\exists \mathtt{u}\ \varphi| \stackrel{\text{def}}{=} |\varphi|$ and $|\varphi_1 * \varphi_2| \stackrel{\text{def}}{=} |\varphi_1| + |\varphi_2|$.

Notice how the $*$ operator splits the model similarly to the separating conjunction in separation logic. In Section 3.3 we explore the similarities between these two logics by providing the formal translation from ALT to $\mathsf{1SL}_{\mathtt{R1}}(*, \mathrel{-\!\!*}, \mathtt{reach}^+)$. ALT is also reminiscent of static ambient logic [5, 22], where the *composition* operator $\varphi \mid \psi$ cuts the tree into two parts. However, differently from the $*$ operator of ALT, this operation preserves the parent relation. Then, $\varphi \mid \psi$ holds on trees that can be divided into a tree satisfying φ, one satisfying ψ and no garbage locations are generated by the split. Given a model $(\mathcal{T},\mathcal{G}) \in \mathbb{T}$ where $\mathcal{G} = \emptyset$, this semantics can be retrieved in ALT with the formula $(\varphi \wedge \neg\exists \mathtt{u}\ \mathsf{G}(\mathtt{u})) * (\psi \wedge \neg\exists \mathtt{u}\ \mathsf{G}(\mathtt{u}))$.

Expressive power: encoding words in ALT. Despite using one single variable, the ability of splitting the model with a operator having the semantics of the separating conjunction greatly increases the expressive power of ALT. In particular, we show that ALT is able to characterise finite words. We first establish a correspondence between words and trees of a particular shape. Let $\Sigma = [1,n]$ be the alphabet of natural numbers between 1 and n. Let $\mathfrak{w} = \mathtt{a}_1 \cdots \mathtt{a}_k$ be a k-letters word in Σ^* and $\{\ell_1,\cdots,\ell_k\}$ be a set of k locations. For every $i \in [1,k]$, let $\mathsf{L}(i)$ be a set of $\mathtt{a}_i + 1$ locations different from $\ell_1,\cdots,\ell_k$ and so that for each distinct $i,j \in [1,k]$, $\mathsf{L}(i) \cap \mathsf{L}(j) = \emptyset$. An encoding of $\mathfrak{w}$ is a tree $\mathcal{T}$ defined on these sets as

$$\mathcal{T}(\ell_k) \stackrel{\text{def}}{=} \rho; \quad \mathcal{T}(\ell_i) \stackrel{\text{def}}{=} \ell_{i+1} \text{ for each } i \in [1,k-1]; \quad \mathcal{T}(\ell) \stackrel{\text{def}}{=} \ell_i \text{ for each } i \in [1,k] \text{ and } \ell \in \mathsf{L}(i)$$



The locations $\ell_1,\cdots,\ell_k$ are the *main path* of $\mathcal{T}$, where $\mathcal{T}(\ell_i) = \ell_{i+1}$ for $i \in [1,k-1]$, and $\mathcal{T}(\ell_k) = \rho$. These are the only locations in $\mathrm{dom}(\mathcal{T})$ with at least one child, with ℓ_1 being the only location with the same number of descendants and children. We say that a location $\ell \in \mathrm{dom}(\mathcal{T})$ encodes the symbol $\mathtt{a} \in \Sigma$ if it has exactly $\mathtt{a}+1$ children that are not in the main path. Then the locations of the main path of $\mathcal{T}$ are the only ones encoding symbols, where ℓ_i encodes $\mathtt{a}_i$ for any $i \in [1,k]$. The tree on the left encodes the word 1121. We now show how to capture these trees with a logical formula. As symbols in Σ are represented using the number of children of elements in $\mathcal{T}$, we need a formula that expresses this number for the location assigned to $\mathtt{u}$. We first define $\mathtt{size}_{\mathsf{T}+\mathsf{G}} \geq \beta$ and $\mathtt{size}_{\mathsf{G}} \geq \beta$, two formulæ respectively stating that $\mathrm{dom}(\mathcal{T}+\mathcal{G})$ and $\mathrm{dom}(\mathcal{G})$ have size at least $\beta \in \mathbb{N}$. They are $\top$ for $\beta = 0$ and otherwise

$$\begin{aligned} \mathtt{size}_{\mathsf{T}+\mathsf{G}} \geq \beta &\stackrel{\text{def}}{=} \exists \mathtt{u}\ \mathtt{alloc}(\mathtt{u}) * \mathtt{size}_{\mathsf{T}+\mathsf{G}} \geq \beta - 1 \\ \mathtt{size}_{\mathsf{G}} \geq \beta &\stackrel{\text{def}}{=} \exists \mathtt{u}\ (\mathsf{G}(\mathtt{u}) \wedge \underbrace{((\mathtt{alloc}(\mathtt{u}) \wedge \mathtt{size}_{\mathsf{T}+\mathsf{G}} = 1) * \mathtt{size}_{\mathsf{G}} \geq \beta - 1)}_{\text{by excluding a location in } \mathcal{G}\text{, at least other } \beta-1 \text{ such locations can be found}}) \end{aligned}$$

where $\mathtt{size}_{\mathsf{T}+\mathsf{G}} = \beta$ is $\mathtt{size}_{\mathsf{T}+\mathsf{G}} \geq \beta \wedge \neg\mathtt{size}_{\mathsf{T}+\mathsf{G}} \geq \beta + 1$, the formula that checks if $\mathcal{T}+\mathcal{G}$ has exactly β elements. In $\mathtt{size}_{\mathsf{G}} \geq \beta$, notice how the $*$ operator is used to isolate the

memory cell of $\mathcal{G}$ corresponding to $\mathtt{u}$ from the remaining part of the model and then search for other $\beta - 1$ elements of $\mathcal{G}$. This "trick" is often used in our formulæ, including the one that checks the number of descendants of a location $l \in \mathrm{dom}(\mathcal{T})$ corresponding to $\mathtt{u}$:

$$\#\mathtt{desc}(\mathtt{u}) \geq \beta \stackrel{\text{def}}{=} \top * (\underbrace{(\forall \mathtt{u}\ \neg \mathsf{G}(\mathtt{u}))}_{\mathcal{G} \text{ is empty}} \wedge \underbrace{((\mathtt{alloc}(\mathtt{u}) \wedge \mathtt{size}_{\mathsf{T}+\mathsf{G}} = 1) * \mathtt{size}_{\mathsf{G}} \geq \beta)}_{\text{isolating } l \text{ creates a garbage of at least } \beta \text{ locations}})$$

▶ **Proposition.** *$l \in \mathrm{dom}(\mathcal{T})$ and has at least β descendants $\iff$ $(\mathcal{T}, \mathcal{G}, l) \models \#\mathtt{desc}(\mathtt{u}) \geq \beta$.*

$\#\mathtt{desc}(\mathtt{u}) \geq \beta$ can then be used to define a formula that checks the number of children of the location l corresponding to $\mathtt{u}$: $\#\mathtt{child}(\mathtt{u}) \geq 0 \stackrel{\text{def}}{=} \mathsf{T}(\mathtt{u})$, whereas $\#\mathtt{child}(\mathtt{u}) \geq \beta + 1$ is

$$\top * (\underbrace{(\forall \mathtt{u}\ \neg \mathsf{G}(\mathtt{u}))}_{\mathcal{G} \text{ is empty}} \wedge \overbrace{\#\mathtt{desc}(\mathtt{u}) \geq \beta + 1}^{l \text{ has at least } \beta+1 \text{ descendants}} \wedge \underbrace{\neg(\mathtt{size}_{\mathsf{T}+\mathsf{G}} = \beta * (\mathsf{T}(\mathtt{u}) \wedge \neg \#\mathtt{desc}(\mathtt{u}) \geq 1))}_{\text{isolating } \beta \text{ memory cells makes impossible for } l \text{ to reach } \rho \text{ and have no descendants}})$$

▶ **Proposition.** *$l \in \mathrm{dom}(\mathcal{T})$ and has at least β children $\iff$ $(\mathcal{T}, \mathcal{G}, l) \models \#\mathtt{child}(\mathtt{u}) \geq \beta$.*

Notice that the size of every formula introduced above is linear with respect to β. We denote with $\mathtt{symbol}(\mathtt{u})$ the formula $\#\mathtt{desc}(\mathtt{u}) \geq 1$, which in our encoding is satisfied if and only if $\mathtt{u}$ is interpreted by a location in the main path. We define $\mathtt{1st}_{\mathsf{S}}(\mathtt{u})$ as the formula that check if $\mathtt{u}$ corresponds to the location that encodes the first letter of a word and that symbol is in $\mathsf{S} \subseteq \Sigma$. As stated above, this is the only location of the main path with the same number of descendants and children. Then, $\mathtt{1st}_{\mathsf{S}}(\mathtt{u})$ can be easily defined as follows:

$$\mathtt{1st}_{\mathsf{S}}(\mathtt{u}) \stackrel{\text{def}}{=} \bigvee\nolimits_{\beta \in \mathsf{S}} (\#\mathtt{desc}(\mathtt{u}) = \beta + 1 \wedge \#\mathtt{child}(\mathtt{u}) = \beta + 1)$$

Lastly, we define a formula, linear in the size of $\Sigma = [1, n]$, that characterises the family of trees encoding a word by capturing the properties of the encoding. $\mathtt{word}_n \stackrel{\text{def}}{=} \psi \wedge \chi_n$ where

$$\begin{aligned} \psi &\stackrel{\text{def}}{=} \overbrace{\neg(\exists \mathtt{u}\ \mathsf{T}(\mathtt{u}) * \exists \mathtt{u}\ \mathsf{T}(\mathtt{u}))}^{\rho \text{ has at most 1 child}} \wedge \overbrace{(\exists \mathtt{u}\ \mathtt{symbol}(\mathtt{u}) \vee \forall \mathtt{u}\ \neg \mathsf{T}(\mathtt{u}))}^{\mathcal{T} \text{ is empty or it encodes symbols}} \\ \chi_n &\stackrel{\text{def}}{=} \forall \mathtt{u}\ (\mathtt{symbol}(\mathtt{u}) \implies \mathtt{1st}_{[1,n]}(\mathtt{u}) \vee \underbrace{((\mathtt{size}_{\mathsf{T}+\mathsf{G}} = 1 * \mathtt{1st}_{[1,n]}(\mathtt{u})) \wedge \neg \mathtt{1st}_{[1,n+1]}(\mathtt{u}))}_{\text{the location corresponding to } \mathtt{u} \text{ encodes a symbol in } [1,n] \text{ and exactly one of its children encodes a letter}}) \end{aligned}$$

▶ **Proposition.** *Let $(\mathcal{T}, \mathcal{G}, l)$ be a state. $\mathcal{T}$ encodes a word in $[1, n]^*$ $\iff$ $(\mathcal{T}, \mathcal{G}, l) \models \mathtt{word}_n$.*

3.2 An alternative semantics for PITL

Propositional interval temporal logic (PITL) [25] was introduced for the verification of hardware components. Similarly to separation logic, it contains an operator called *chop* $\mid$ that splits the model into two parts. We refer the reader to [24] for a complete description of PITL and consider here its interpretation under *locality principle*, known to be Tower-complete (the full logic is undecidable). Every formula of PITL is built from

$$\varphi \coloneqq \mathtt{a} \mid \mathtt{pt} \mid \varphi \wedge \varphi \mid \neg\varphi \mid \varphi \mid \varphi$$

and is interpreted on a non-empty finite word $\mathfrak{w} \in \Sigma^+$, where the satisfaction relation $\models$ is

- $\mathfrak{w} \models \mathtt{a}$ if and only if $\mathfrak{w}$ is headed by the symbol $\mathtt{a}$, i.e. there is $\mathfrak{w}' \in \Sigma^*$ such that $\mathfrak{w} = \mathtt{a}\mathfrak{w}'$.
- $\mathfrak{w} \models \mathtt{pt}$ if and only if the length of $\mathfrak{w}$ is 1, i.e. $\mathfrak{w} \in \Sigma$.
- $\mathfrak{w} \models \varphi_1 \mid \varphi_2$ iff there are $\mathtt{a} \in \Sigma$ and $\mathfrak{w}', \mathfrak{w}'' \in \Sigma^*$ s.t. $\mathfrak{w} = \mathfrak{w}'\mathtt{a}\mathfrak{w}''$, $\mathfrak{w}'\mathtt{a} \models \varphi_1$ and $\mathtt{a}\mathfrak{w}'' \models \varphi_2$.

PITL seems a good candidate for a reduction, since finite words and the predicates $\mathtt{a}$, $\mathtt{pt}$ can be easily encoded in it. Capturing $\varphi_1 | \varphi_2$ seems however to be challenging. Let $\mathfrak{w} = \mathtt{a}_1 \cdots \mathtt{a}_k$ be a non-empty word and let $\mathcal{T}$ be a tree encoding $\mathfrak{w}$. Let $\ell_1, \cdots, \ell_k$ be the main path of $\mathcal{T}$. From the semantics of the chop operator, $\mathfrak{w} \models \varphi_1 | \varphi_2$ if and only if there is a position $i \in [1, k]$ so that $\mathtt{a}_1 \cdots \mathtt{a}_i \models \varphi_1$ and $\mathtt{a}_i \cdots \mathtt{a}_k \models \varphi_2$. Alternatively, we would like to split the main path of $\mathcal{T}$ so that we are able to check that its prefix with main path $\ell_1, \cdots, \ell_i$ encodes a word satisfying φ_1 whereas its suffix with main path $\ell_i, \cdots, \ell_k$ encodes a word satisfying φ_2. However, using $*$ to naïvely cutting $\mathcal{T}$ causes the locations $\ell_1, \cdots, \ell_i$ not to encode any word (as they do not reach ρ anymore), making it impossible to check the satisfaction of φ_1. To solve this issue we define an equivalent interpretation of PITL based on marked symbols.

A *marking* of an alphabet Σ is a bijection $\overline{(.)} : \Sigma \to \overline{\Sigma}$, relating each symbol $\mathtt{a} \in \Sigma$ to its *marked representation* $\overline{\mathtt{a}} \in \overline{\Sigma}$. We denote with $\mathfrak{L}$ the extended alphabet $\Sigma \cup \overline{\Sigma}$. A word $\mathfrak{L}^+$ is *marked* if it has some symbols from $\overline{\Sigma}$. The satisfaction relation $\models_\bullet$ on a marked word $\mathfrak{w}$ is

- $\mathfrak{w} \models_\bullet \mathtt{a}$ iff $\mathfrak{w}$ is headed by $\mathtt{a}$ or $\overline{\mathtt{a}}$.
- $\mathfrak{w} \models_\bullet \mathtt{pt}$ iff $\mathfrak{w}$ is headed by a marked symbol.
- $\mathfrak{w} \models_\bullet \varphi_1 | \varphi_2$ iff there are $\overline{\mathtt{a}} \in \overline{\Sigma}$, $\mathtt{b} \in \Sigma$, $\mathfrak{w}', \mathfrak{w}'_1, \mathfrak{w}'_2 \in \Sigma^*$ and $\mathfrak{w}'' \in \mathfrak{L}^*$ s.t. $(\mathfrak{w} = \mathfrak{w}' \overline{\mathtt{a}} \mathfrak{w}''$, $\mathfrak{w}' \overline{\mathtt{a}} \mathfrak{w}'' \models_\bullet \varphi_1$ and $\overline{\mathtt{a}} \mathfrak{w}'' \models_\bullet \varphi_2)$ or $(\mathfrak{w}' = \mathfrak{w}'_1 \mathtt{b} \mathfrak{w}'_2$, $\mathfrak{w}'_1 \overline{\mathtt{b}} \mathfrak{w}'_2 \overline{\mathtt{a}} \mathfrak{w}'' \models_\bullet \varphi_1$ and $\mathtt{b} \mathfrak{w}'_2 \overline{\mathtt{a}} \mathfrak{w}'' \models_\bullet \varphi_2)$.

Here, the satisfaction of a formula is only checked on the prefix $\mathtt{a}_1 \cdots \mathtt{a}_{i-1} \overline{\mathtt{a}}_i$ of $\mathfrak{w}$ that ends with the first marked letter. To check whether $\mathfrak{w} \models_\bullet \varphi_1 | \varphi_2$ we search for a position $j \in [1, i]$ inside this prefix so that φ_1 is satisfied by $\mathfrak{w}$ updated so that its j-th letter is marked, and φ_2 is satisfied by the suffix of $\mathfrak{w}$ starting in j. As shown in the next section, taking the suffix of a word and marking a symbol can be simulated in ALT. As the two semantics of PITL are shown to be equivalent (by induction on the structure of φ), this makes ALT capture PITL.

▶ **Theorem 1** ($\models$ equiv. $\models_\bullet$). *Let* $\mathfrak{w} \in \Sigma^+$. *Let* $\overline{\mathfrak{w}} = \mathfrak{w}' \overline{\mathtt{a}} \mathfrak{w}''$ *with* $\mathfrak{w}' \in \Sigma^*$, $\overline{\mathtt{a}} \in \overline{\Sigma}$ *and* $\mathfrak{w}'' \in \mathfrak{L}^*$. *If* $\mathfrak{w} = \mathfrak{w}' \mathtt{a}$ *then for every* PITL *formula* φ *we have* $\mathfrak{w} \models \varphi \iff \overline{\mathfrak{w}} \models_\bullet \varphi$.

3.3 Tower-completeness of ALT and other complexity results

We reduce the satisfiability problem of PITL on marked words to the satisfiability problem of ALT. Let $\Sigma = [1, n]$, $\mathfrak{L} = \Sigma \cup \overline{\Sigma}$ and let $\mathfrak{f} : \mathfrak{L} \to [1, 2n]$ be the bijection $\mathfrak{f}(\mathtt{a}) \stackrel{\text{def}}{=} 2\mathtt{a}$ for $\mathtt{a} \in \Sigma$ and $\mathfrak{f}(\overline{\mathtt{a}}) \stackrel{\text{def}}{=} 2\mathtt{a} - 1$ for $\overline{\mathtt{a}} \in \overline{\Sigma}$. $\mathfrak{f}(\mathtt{a}_1 \cdots \mathtt{a}_k)$ denotes the translated word $\mathfrak{f}(\mathtt{a}_1) \cdots \mathfrak{f}(\mathtt{a}_k)$. $\mathfrak{f}$ maps $\mathfrak{L}$ into the alphabet $[1, 2n]$, whose words can be encoded into trees (as in Section 3.1). In these trees each symbol $\mathtt{a} \in \Sigma$ (resp. $\overline{\mathtt{a}} \in \overline{\Sigma}$) corresponds to a location ℓ in the main path having $2\mathtt{a} + 1$ (resp. $2\mathtt{a}$) children not in this path. Then, removing exactly one of these children is equal to marking a symbol. We can check if the assignment of $\mathtt{u}$ encodes marked symbols:

$$\mathtt{marked}_n(\mathtt{u}) \stackrel{\text{def}}{=} \bigvee_{i \in [1,n]} ((\#\mathtt{child}(\mathtt{u}) = 2i \wedge \mathtt{1st}_{[1,2n]}(\mathtt{u})) \vee (\underbrace{\#\mathtt{child}(\mathtt{u}) = 2i + 1}_{\text{the location corresponding to } \mathtt{u} \text{ has } 2i \text{ children not in the main path and one in it}} \wedge \neg \mathtt{1st}_{[1,2n]}(\mathtt{u})))$$

As stated in Section 3.2, $\mathfrak{w} \models_\bullet \varphi$ examine the prefix of $\mathfrak{w}$ that ends with the first marked symbol. In trees $\mathcal{T}$ encoding $\mathfrak{w}$, this corresponds to the locations that reach every encoding of marked symbols. The idea is then to track the number of these symbols in $\mathcal{T}$. The formula $\mathtt{marks}_n \geq \beta$, defined as $\top$ for $\beta = 0$, shown below is satisfied only by trees with at least β marked symbols. Then, the formula $\#\mathtt{marks}_n(\mathtt{u}) \geq \beta$ checks if the current assignment of $\mathtt{u}$ encodes a symbol that reaches at least (other) β marked locations.

$$\begin{aligned} \mathtt{marks}_n \geq \beta &\stackrel{\text{def}}{=} \exists \mathtt{u}\, (\mathtt{marked}_n(\mathtt{u}) \wedge \overbrace{((\mathtt{size}_{\mathsf{T}+\mathsf{G}} = 1 \wedge \mathtt{alloc}(\mathtt{u})) * \mathtt{marks}_n \geq \beta - 1)}^{\text{by excluding a marked location, at least other } \beta - 1 \text{ such locations can be found}}) \\ \#\mathtt{marks}_n(\mathtt{u}) \geq \beta &\stackrel{\text{def}}{=} \mathtt{symbol}(\mathtt{u}) \wedge ((\mathtt{size}_{\mathsf{T}+\mathsf{G}} = 1 \wedge \mathtt{alloc}(\mathtt{u})) * \mathtt{marks}_n \geq \beta) \end{aligned}$$

At last, we define the translation $\nabla_\beta(\varphi)$, parametrised on the number $\beta \geq 1$ of marked symbols, of a PITL formula φ. $\nabla_\beta(\varphi)$ is homomorphic for Boolean connectives, $\nabla_\beta(\mathtt{a})$ and $\nabla_\beta(\mathtt{pt})$ are respectively $\exists \mathtt{u}\ \mathtt{1st}_{[2\alpha-1,2\alpha]}(\mathtt{u})$ and $\exists \mathtt{u}\ (\mathtt{1st}_{[1,2n]}(\mathtt{u}) \wedge \mathtt{marked}_n(\mathtt{u}))$, whereas $\nabla_\beta(\varphi_1 \mid \varphi_2)$ is

$$\begin{aligned}\exists \mathtt{u}\ \Big(&\mathtt{symbol}(\mathtt{u}) \wedge \big((\mathtt{1st}_{[1,2n]}(\mathtt{u}) \wedge \mathtt{marked}_n(\mathtt{u}) \wedge \nabla_\beta(\varphi_1) \wedge \nabla_\beta(\varphi_2)) \vee \\ &(\mathtt{1st}_{[1,2n]}(\mathtt{u}) \wedge \neg\mathtt{marked}_n(\mathtt{u}) \wedge (\mathtt{size}_\mathsf{G} = 1 * (\mathtt{marked}_n(\mathtt{u}) \wedge \nabla_{\beta+1}(\varphi_1))) \wedge \nabla_\beta(\varphi_2)) \vee \\ &(\neg\mathtt{1st}_{[1,2n]}(\mathtt{u}) \wedge \mathtt{marked}_n(\mathtt{u}) \wedge \#\mathtt{marks}_n(\mathtt{u}) \geq \beta - 1 \wedge \nabla_\beta(\varphi_1) \wedge (\mathtt{size}_\mathsf{G} = 1 * (\mathtt{1st}_{[1,2n]}(\mathtt{u}) \wedge \nabla_\beta(\varphi_2)))) \vee \\ &(\neg\mathtt{1st}_{[1,2n]}(\mathtt{u}) \wedge \neg\mathtt{marked}_n(\mathtt{u}) \wedge \#\mathtt{marks}_n(\mathtt{u}) \geq \beta \wedge (\mathtt{size}_\mathsf{G} = 1 * (\mathtt{marked}_n(\mathtt{u}) \wedge \nabla_{\beta+1}(\varphi_1))) \\ &\wedge (\mathtt{size}_\mathsf{G} = 1 * (\mathtt{1st}_{[1,2n]}(\mathtt{u}) \wedge \nabla_\beta(\varphi_2))))\big)\Big).\end{aligned}$$

The translation follows closely the relation $\models_\bullet$. The case for $\nabla_\beta(\varphi_1 \mid \varphi_2)$ is split into four disjuncts, depending on whether or not $\mathtt{u}$ points to a location (1) encoding the first letter of the word and (2) encoding the first marked symbol. For example, in the third disjunct $\mathtt{u}$ points to a location l which is not the first in the main path but that encodes the first marked symbol. $\nabla_\beta(\varphi_1)$ needs then to hold on the current state whereas $\nabla_\beta(\varphi_2)$ must be checked with respect to the portion of tree where l encodes the same marked symbol but is now the first location in the main path. To obtain this, the formula cuts the tree by only removing the child of l that is in the main path. Lemma 2 (whose proof is by induction on the structure of φ) ensures that the translation captures the semantics of the formula. The reduction of PITL with the standard semantics (Theorem 3) then stems from Theorem 1.

▶ **Lemma 2.** *Let $\mathfrak{w} \in \mathfrak{T}^+$ be a marked word with $\beta \geq 1$ marked symbols. Let $(\mathcal{T}, \mathcal{G}, l)$ be a state such that $\mathcal{T}$ encodes $\mathfrak{f}(\mathfrak{w})$. For every* PITL *formula φ, $\mathfrak{w} \models_\bullet \varphi \iff (\mathcal{T}, \mathcal{G}, l) \models \nabla_\beta(\varphi)$.*

▶ **Theorem 3** (PITL to ALT)**.** *Every* PITL *formula φ interpreted on $\models$ is equisatisfiable with*

$$\underbrace{\mathtt{word}_{2n} \wedge \exists \mathtt{u}\ \mathsf{T}(\mathtt{u})}_{\mathcal{T}\text{ encodes a non-empty word}} \wedge \forall \mathtt{u}\ (\mathtt{marked}_n(\mathtt{u}) \Leftrightarrow \underbrace{(\mathsf{T}(\mathtt{u}) \wedge \neg(\top * \mathsf{G}(\mathtt{u})))}_{\mathtt{u}\text{ is interpreted a child of }\rho}) \wedge \nabla_1(\varphi).$$

Complexity results. Even though the translation from PITL to ALT is exponential, the Tower-completeness of PITL [24] ensures that ALT is Tower-hard. It remains to show that ALT is captured by $\mathsf{1SL}_{\mathtt{R1}}(*, -\!\!*, \mathtt{reach}^+)$. The translation $\tau_\mathtt{x}(\varphi)$ of an ALT formula φ provided here is quite straightforward, with the only specificity being the role of $\mathtt{x} \in \mathsf{VAR}$ as the root of the tree. $\tau_\mathtt{x}(\varphi)$ is homomorphic for Boolean connectives and $*$ operators, $\tau_\mathtt{x}(\mathsf{T}(\mathtt{u})) \stackrel{\text{def}}{=} \mathtt{reach}^+(\mathtt{u}, \mathtt{x})$, $\tau_\mathtt{x}(\mathsf{G}(\mathtt{u})) \stackrel{\text{def}}{=} \mathtt{alloc}(\mathtt{u}) \wedge \neg\mathtt{reach}^+(\mathtt{u}, \mathtt{x})$ and $\tau_\mathtt{x}(\exists \mathtt{u}\ \varphi) \stackrel{\text{def}}{=} \exists \mathtt{u}\ (\mathtt{u} \neq \mathtt{x} \wedge \tau_\mathtt{x}(\varphi))$. Its soundness is proved by induction on the structure of φ (Lemma 4) and implies Theorem 5.

▶ **Lemma 4.** *Let φ be a* ALT *formula and let $\mathtt{x} \in \mathsf{VAR}$. Let $(\mathcal{T}, \mathcal{G}, l)$ be a state and let s be a store such that $s(\mathtt{x}) = \rho$. Then, $(\mathcal{T}, \mathcal{G}, l) \models \varphi$ if and only if $(s, \mathcal{T} + \mathcal{G}, l) \models \tau_\mathtt{x}(\varphi)$.*

▶ **Theorem 5.** *Let $\mathtt{x} \in \mathsf{VAR}$. Every* ALT *formula φ is equisat. with $\mathtt{u} \neq \mathtt{x} \wedge \neg\mathtt{alloc}(\mathtt{x}) \wedge \tau_\mathtt{x}(\varphi)$.*

As the separating implication only appears inside `alloc` predicates, the formula obtained through the translation satisfies R1. This proves both that $\mathsf{1SL}_{\mathtt{R1}}(*, -\!\!*, \mathtt{reach}^+)$ is Tower-hard and that ALT is Tower-complete as it is captured by $\mathsf{1SL}(*, \mathtt{reach}^+, \mathtt{alloc})$, a fragment of first-order separation logic without $-\!\!*$ which is known to be Tower-complete [4].

4 PSpace-completeness of $\mathsf{1SL}(*, -\!\!*, \mathtt{reach}^+)$ under R1 and R2

We now consider $\mathsf{1SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$, the fragment of $\mathsf{1SL}(*, -\!\!*, \mathtt{reach}^+)$ satisfying both R1 and R2. Formulæ of this fragment are built from the non-terminals of the following grammar.

$$\begin{aligned}\mathcal{C} &:= e_1 = e_2 \mid e_1 \hookrightarrow e_2 \mid \mathtt{emp} \mid \mathcal{C} \wedge \mathcal{C} \mid \neg\mathcal{C} \mid \exists \mathtt{u}\ \mathcal{C} \mid \mathcal{C} * \mathcal{C} \mid \mathcal{A} -\!\!* \mathcal{C} \\ \mathcal{A} &:= \mathcal{C} \mid \mathtt{reach}^+(\mathtt{x}, e_1) \mid \mathtt{reach}^+(\mathtt{u}, \mathtt{u}) \mid \mathcal{A} \wedge \mathcal{A} \mid \neg\mathcal{A} \mid \exists \mathtt{u}\ \mathcal{A} \mid \mathcal{A} * \mathcal{A}\end{aligned}$$

where $\mathtt{x} \in \mathsf{VAR}$, $e_1, e_2 \in \mathsf{VAR} \cup \{\mathtt{u}\}$. Notice how each *antecedent* of a separating implication ($-\!\!*$) is in $\mathcal{A}$ whereas its *consequent* is in $\mathcal{C}$. We say that φ is a $\mathcal{A}$-formula (resp. $\mathcal{C}$-formula) whenever it is in the language generated by $\mathcal{A}$ (resp. $\mathcal{C}$). Every $\mathcal{C}$-formula is thus a $\mathcal{A}$-formula.

For this logic, we show that satisfiability can be solved in PSPACE by proving a *small model property*: every satisfiable formula has a polynomial size model. As far as we know, this is the first fragment with reachability predicates in the scope of $-\!\!*$ that is proved decidable. Moreover, it subsumes the logics studied in [12] and [13] which were also found to be PSPACE-complete. The result is shown by extending the *test formulæ technique* introduced by Lozes in [22]: we design a finite index equivalence relation on memory states based on the satisfaction of atomic predicates (the *test formulæ*). Then, we show that any formula of our logic can be expressed as a Boolean combination of test formulæ, effectively replacing quantifiers and spatial connectives. A proof of small model property for Boolean combination of test formulæ thus extends to $\mathsf{1SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$. To handle the asymmetry of $\mathcal{A} -\!\!* \mathcal{C}$, the technique is here extended by introducing two families of test formulæ, one for the $\mathcal{C}$ fragment and one for the $\mathcal{A}$ fragment (i.e. respectively the set of every $\mathcal{C}$-formula and the set of every $\mathcal{A}$-formula). The two families are then combined together in order to deduce the complexity of $\mathsf{1SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$ (as we show in Section 4.3).

4.1 A family of test formulæ capturing $\mathcal{C}$

In order to define the set of test formulæ for the $\mathcal{C}$ fragment we proceed as follows:

(1) we introduce a set of syntactical terms and a partition of a memory state.

(2) We then highlight properties of these objects by introducing the set of test formulæ.

(3) Lastly, we show that the test formulæ internalise the semantics of separating conjunctions and quantifications.

The same steps are carried out, in Section 4.2, for the test formulæ of the $\mathcal{A}$ fragment.

As we are interested in the satisfiability of a given formula, it is natural to consider only the finite set $\mathsf{X} \subseteq_{\mathtt{fin}} \mathsf{VAR}$ of variables appearing in it. The *syntactical terms* $\mathcal{C}\mathsf{Term}_{\mathsf{X}}$ are defined as $\mathsf{X} \cup \mathcal{C}\mathsf{Next}_{\mathsf{X}}$, where $\mathcal{C}\mathsf{Next}_{\mathsf{X}} \stackrel{\text{def}}{=} \{\mathtt{n}(\mathtt{x}) \mid \mathtt{x} \in \mathsf{X}\}$ is the set of *next-point variables*. Given a memory state (s, h, l), we write $[\![\mathtt{x}]\!]^{\mathsf{X}}_{s,h}$ for $s(\mathtt{x})$ and $[\![\mathtt{n}(\mathtt{x})]\!]^{\mathsf{X}}_{s,h}$ to denote (if it exists) the location $h(s(\mathtt{x}))$. The locations corresponding to terms are said to be *labelled* and their set is denoted with $\mathcal{C}\mathsf{Labels}^{\mathsf{X}}_{s,h}$. Labelled locations correspond to locations for which the logic is able to express particular properties. As such, the test formulæ primarily speak about relationships between these locations, as well as the following subsets of $\mathrm{dom}(h)$:

- $\mathcal{C}\mathsf{Pred}^{\mathsf{X}}_{s,h}(\ell) \stackrel{\text{def}}{=} \{\ell' \in \mathrm{dom}(h) \setminus \mathcal{C}\mathsf{Labels}^{\mathsf{X}}_{s,h} \mid h(\ell') = \ell\}$, for every $\ell \in s(\mathsf{X})$, i.e. the set of unlabelled predecessors of a location corresponding to a program variable.
- $\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h} \stackrel{\text{def}}{=} \{\ell \in \mathrm{dom}(h) \setminus \mathcal{C}\mathsf{Labels}^{\mathsf{X}}_{s,h} \mid h(\ell) = \ell\}$, i.e. the set of unlabelled self-loops.
- $\mathcal{C}\mathsf{Size}^{\mathsf{X}}_{s,h} \stackrel{\text{def}}{=} \mathrm{dom}(h) \setminus (\mathcal{C}\mathsf{Labels}^{\mathsf{X}}_{s,h} \cup \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h} \cup \bigcup_{\ell \in s(\mathsf{X})} \mathcal{C}\mathsf{Pred}^{\mathsf{X}}_{s,h}(\ell))$, i.e. the set of unlabelled locations that do not self-loop and are not predecessors of program variables.

Notice how $\{\mathrm{dom}(h) \cap \mathcal{C}\mathsf{Labels}^{\mathsf{X}}_{s,h}, \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h}, \mathcal{C}\mathsf{Size}^{\mathsf{X}}_{s,h}\} \cup \{\mathcal{C}\mathsf{Pred}^{\mathsf{X}}_{s,h}(\ell) \mid \ell \in s(\mathsf{X})\}$ partitions $\mathrm{dom}(h)$. The test formulæ $\mathcal{C}\mathsf{TEST}(\mathsf{X}, \alpha)$ are parametric on X and $\alpha \in \mathbb{N}^+$. Here, α is a quantity that roughly express upper-bounds on the capabilities of a $\mathcal{C}$-formula φ to check the sizes of the sets of the partition. In Section 4.3 we show how α is connected to the size of φ. $\mathcal{C}\mathsf{TEST}(\mathsf{X}, \alpha)$ is divided into two sets, a *skeleton* $\mathcal{C}\mathsf{SKEL}(\mathsf{X}, \alpha)$ expressing structural properties that do not depend on the assignment of $\mathtt{u}$, and an *observed* set $\mathcal{C}\mathsf{OBS}(\mathsf{X}, \alpha)$ of relationships

between the memory state and the location currently assigned to $\mathtt{u}$.

$$\mathcal{C}\mathsf{SKEL}(\mathsf{X},\alpha) \stackrel{\text{def}}{=} \left\{ \begin{array}{l} \mathtt{t}_1 = \mathtt{t}_2,\ \mathtt{alloc}(\mathtt{t}_1),\ \mathtt{t}_1 \hookrightarrow \mathtt{x},\ \mathtt{t}_1 \hookrightarrow \mathtt{t}_1, \\ \#\mathtt{pred}^{\mathcal{C}}_{\mathsf{X}}(\mathtt{x}) \geq \beta,\ \#\mathtt{loop}^{1}_{\mathsf{X}} \geq \beta,\ \mathtt{size}^{\mathcal{C}}_{\mathsf{X}} \geq \beta \end{array} \middle| \begin{array}{r} \mathtt{x} \in \mathsf{X},\ \beta \in [1,\alpha] \\ \text{and } \mathtt{t}_1, \mathtt{t}_2 \in \mathcal{C}\mathsf{Term}_{\mathsf{X}} \end{array} \right\}$$

$$\mathcal{C}\mathsf{OBS}(\mathsf{X},\alpha) \stackrel{\text{def}}{=} \left\{ \mathtt{u} = \mathtt{t},\ \mathtt{u} \in \mathtt{pred}^{\mathcal{C}}_{\mathsf{X}}(\mathtt{x}),\ \mathtt{u} \in \mathtt{loop}^{1}_{\mathsf{X}},\ \mathtt{u} \in \mathtt{size}^{\mathcal{C}}_{\mathsf{X}} \ \middle|\ \mathtt{x} \in \mathsf{X} \text{ and } \mathtt{t} \in \mathcal{C}\mathsf{Term}_{\mathsf{X}} \right\}$$

The formal semantics of the test formulæ is provided below:

- $(s,h,l) \models \mathtt{t}_1 = \mathtt{t}_2$ if and only if $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$ and $[\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h}$ are defined and $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h} = [\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h}$.
- $(s,h,l) \models \mathtt{alloc}(\mathtt{t}_1)$ if and only if $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$ is defined and $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h} \in \mathrm{dom}(h)$.
- $(s,h,l) \models \mathtt{t}_1 \hookrightarrow \mathtt{x}$ if and only if $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$ is defined and $h([\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}) = s(\mathtt{x})$.
- $(s,h,l) \models \mathtt{t}_1 \hookrightarrow \mathtt{t}_1$ if and only if $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$ is defined and $h([\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}) = [\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$.
- $(s,h,l) \models \#\mathtt{pred}^{\mathcal{C}}_{\mathsf{X}}(\mathtt{x}) \geq \beta$ if and only if $\mathrm{card}(\mathcal{C}\mathsf{Pred}^{\mathsf{X}}_{s,h}(s(\mathtt{x}))) \geq \beta$.
- $(s,h,l) \models \#\mathtt{loop}^{1}_{\mathsf{X}} \geq \beta$ if and only if $\mathrm{card}(\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h}) \geq \beta$.
- $(s,h,l) \models \mathtt{size}^{\mathcal{C}}_{\mathsf{X}} \geq \beta$ if and only if $\mathrm{card}(\mathcal{C}\mathsf{Size}^{\mathsf{X}}_{s,h}) \geq \beta$.
- $(s,h,l) \models \mathtt{u} = \mathtt{t}$ if and only if $[\![\mathtt{t}]\!]^{\mathsf{X}}_{s,h}$ is defined and $l = [\![\mathtt{t}]\!]^{\mathsf{X}}_{s,h}$.
- $(s,h,l) \models \mathtt{u} \in \mathtt{pred}^{\mathcal{C}}_{\mathsf{X}}(\mathtt{x})$ if and only if $l \in \mathcal{C}\mathsf{Pred}^{\mathsf{X}}_{s,h}(s(\mathtt{x}))$.
- $(s,h,l) \models \mathtt{u} \in \mathtt{loop}^{1}_{\mathsf{X}}$ if and only if $l \in \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h}$.
- $(s,h,l) \models \mathtt{u} \in \mathtt{size}^{\mathcal{C}}_{\mathsf{X}}$ if and only if $l \in \mathcal{C}\mathsf{Size}^{\mathsf{X}}_{s,h}$.

In $\mathcal{C}\mathsf{TEST}(\mathsf{X},\alpha)$, the classical predicates $=$, $\hookrightarrow$ and $\mathtt{alloc}$ are extended to terms of $\mathcal{C}\mathsf{Term}_{\mathsf{X}}$. For instance $\mathtt{n}(\mathtt{x}) \hookrightarrow \mathtt{y}$ is satisfied by only those memory states (s,h,l) where $h(h(s(\mathtt{x}))) = s(\mathtt{y})$. There are some restrictions: all program variables are from X and $\mathtt{t}_2 \hookrightarrow \mathtt{t}_1$ is syntactically constrained so that $\mathtt{t}_2$ is equal to $\mathtt{t}_1$ when the latter is a next-point variable (so for instance $\mathtt{n}(\mathtt{x}) \hookrightarrow \mathtt{n}(\mathtt{y})$ is not a test formula). The test formulæ $\#\mathtt{pred}^{\mathcal{C}}_{\mathsf{X}}(\mathtt{x}) \geq \beta$, $\#\mathtt{loop}^{1}_{\mathsf{X}} \geq \beta$ and $\mathtt{size}^{\mathcal{C}}_{\mathsf{X}} \geq \beta$ are respectively satisfied whenever the sets $\mathcal{C}\mathsf{Pred}^{\mathsf{X}}_{s,h}(s(\mathtt{x}))$, $\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h}$ and $\mathcal{C}\mathsf{Size}^{\mathsf{X}}_{s,h}$ have size at least β. As β is bounded by α, memory states having both at least α elements in one of these sets satisfy the same test formulæ related to that set. Lastly, the formulæ $\mathtt{u} \in \mathtt{pred}^{\mathcal{C}}_{\mathsf{X}}(\mathtt{x})$, $\mathtt{u} \in \mathtt{loop}^{1}_{\mathsf{X}}$ and $\mathtt{u} \in \mathtt{size}^{\mathcal{C}}_{\mathsf{X}}$ respectively check whether the location currently assigned to $\mathtt{u}$ is in $\mathcal{C}\mathsf{Pred}^{\mathsf{X}}_{s,h}(s(\mathtt{x}))$, $\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h}$ or $\mathcal{C}\mathsf{Size}^{\mathsf{X}}_{s,h}$.

It is possible to show that each test formula can be expressed with a $\mathcal{C}$-formula. For instance, $\mathtt{u} \in \mathtt{loop}^{1}_{\mathsf{X}}$ is equivalent to $\mathtt{u} \hookrightarrow \mathtt{u} \wedge \bigwedge_{\mathtt{x} \in \mathsf{X}} (\mathtt{u} \neq \mathtt{x} \wedge \neg \mathtt{x} \hookrightarrow \mathtt{u})$ and the formula $\#\mathtt{loop}^{1}_{\mathsf{X}} \geq \beta$ is equivalent to $\exists \mathtt{u}\ (\mathtt{u} \in \mathtt{loop}^{1}_{\mathsf{X}} \wedge ((\mathtt{alloc}(\mathtt{u}) \wedge \mathtt{size} = 1) * \#\mathtt{loop}^{1}_{\mathsf{X}} \geq \beta - 1))$, where $\#\mathtt{loop}^{1}_{\mathsf{X}} \geq 0 \stackrel{\text{def}}{=} \top$. Although this result ensures that the test formulæ are not more expressive than the logic, we need to show the converse. To do so, we start by defining an indistinguishability relation between memory states, denoted with $(s,h,l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha} (s',h',l')$, that holds if and only if for all $\varphi \in \mathcal{C}\mathsf{TEST}(\mathsf{X},\alpha)$ it holds $(s,h,l) \models \varphi \iff (s',h',l') \models \varphi$.

We then use this relation to show that the expressive power of the test formulæ allows to mimic quantifiers and separating conjunctions, as done in [12] for separation logic with one quantified variable. This result should not be surprising, as the equivalence relation $\simeq_\alpha$ defined in [12] (Def. 3.8) can be shown equivalent to $\approx^{\mathcal{C}}_{\mathsf{X},\alpha}$. We first handle the quantifiers.

▶ **Lemma 6** ($\mathcal{C}$:∃ indistinguishability). *Assume* $(s,h,l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha} (s',h',l')$. *Let* $\ell \in \mathsf{LOC} \setminus \mathsf{L}$ *with* $\mathsf{L} \stackrel{\text{def}}{=} \mathrm{dom}(h') \cup \mathrm{ran}(h') \cup s'(\mathsf{X})$. *For every* $l_1 \in \mathsf{LOC}$ *there is* $l'_1 \in \mathsf{L} \cup \{\ell\}$ *s.t.* $(s,h,l_1) \approx^{\mathcal{C}}_{\mathsf{X},\alpha} (s',h',l'_1)$.

This holds as we show that for every assignment l_1 we can find a location l'_1 so that the formulæ in $\mathcal{C}\mathsf{OBS}(\mathsf{X},\alpha)$ are equisatisfied by both memory states. Indeed, formulæ of $\mathcal{C}\mathsf{SKEL}(\mathsf{X},\alpha)$ do not depend on the assignment of $\mathtt{u}$ but they are key to prove the result. For example, suppose that $l_1 \in \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h}$. From the equisatisfaction of $\#\mathtt{loop}^{1}_{\mathsf{X}} \geq 1$, $\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s',h'}$ is not empty and we can choose l'_1 to be in this set. Then, (s,h,l_1) and (s',h',l'_1) satisfy the same test formulæ. Notice how l'_1 is taken from a finite set. This is key to prove that the logic is in PSPACE (Theorem 15). Lemma 6 shows that two indistinguishable memory states cannot be distinguished using quantifiers. We show that the same holds for separating conjunctions.

▶ **Lemma 7** ($\mathcal{C}$:$*$ indistinguishability). *Let* $\mathsf{X} \subseteq_{\mathtt{fin}} \mathsf{VAR}$ *and* $\alpha, \alpha_1, \alpha_2 \in \mathbb{N}^+$ *with* $\alpha = \alpha_1 + \alpha_2$. *Assume* $(s,h,l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha} (s',h',l')$. *For all heaps* h_1, h_2 *such that* $h = h_1 + h_2$ *there are heaps* h'_1, h'_2 *such that* $h' = h'_1 + h'_2$, $(s,h_1,l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha_1} (s',h'_1,l')$ *and* $(s,h_2,l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha_2} (s',h'_2,l')$.

This result can be proved by looking at (s,h_1,l) and (s,h_2,l) in terms of their partitions

$$\{\mathrm{dom}(h_k) \cap \mathcal{C}\mathsf{Labels}^{\mathsf{X}}_{s,h_k}, \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h_k}, \mathcal{C}\mathsf{Size}^{\mathsf{X}}_{s,h_k}\} \cup \{\mathcal{C}\mathsf{Pred}^{\mathsf{X}}_{s,h_k}(\ell) \mid \ell \in s(\mathsf{X})\}, \qquad k \in \{1,2\}$$

and showing that it is possible to construct h'_1 and h'_2 by dividing h' in such a way that $(s,h_1,l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha_1} (s',h'_1,l')$ and $(s,h_2,l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha_2} (s',h'_2,l')$. For instance, let us consider the case of unlabelled self-loops. In the following, the index k stands for 1 or 2. We define $\mathsf{L}_k \stackrel{\text{def}}{=} \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h} \cap \mathrm{dom}(h_k)$, i.e. the set of unlabelled self-loops of h assigned to h_k. We partially construct h'_1 and h'_2 by defining two disjoint sets $\mathsf{L}'_1 \subseteq \mathrm{dom}(h'_1)$ and $\mathsf{L}'_2 \subseteq \mathrm{dom}(h'_2)$ so that:

$$\mathsf{L}'_1 \cup \mathsf{L}'_2 = \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s',h'} \qquad \min(\alpha_k, \mathrm{card}(\mathsf{L}_k)) = \min(\alpha_k, \mathrm{card}(\mathsf{L}'_k)) \qquad l \in \mathsf{L}_k \Leftrightarrow l' \in \mathsf{L}'_k$$

This can be done as, by $(s,h,l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha} (s',h',l')$, it follows that $l \in \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h} \Leftrightarrow l' \in \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s',h'}$ and $\min(\alpha, \mathrm{card}(\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h})) = \min(\alpha, \mathrm{card}(\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s',h'}))$. The construction goes by cases:

- if $\mathrm{card}(\mathsf{L}_1) < \alpha_1$ then select $\mathrm{card}(\mathsf{L}_1)$ locations from $\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s',h'}$. If $l \in \mathsf{L}_1$ then l' is one of the selected locations. These locations constitute L'_1. Then, L'_2 is the set $\mathsf{L}' \setminus \mathsf{L}'_1$;
- else if $\mathrm{card}(\mathsf{L}_2) < \alpha_2$ then select $\mathrm{card}(\mathsf{L}_2)$ locations from $\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s',h'}$. If $l \in \mathsf{L}_2$ then l' is one of the selected locations. These locations constitute L'_2. Then, $\mathsf{L}'_1 \stackrel{\text{def}}{=} \mathsf{L}' \setminus \mathsf{L}'_2$;
- otherwise $\mathrm{card}(\mathsf{L}_1) \geq \alpha_1$ and $\mathrm{card}(\mathsf{L}_2) \geq \alpha_2$. Select α_1 locations from $\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s',h'}$. If $l \in \mathsf{L}_1$ then l' is one of the selected locations. These locations constitute L'_1. Then, $\mathsf{L}'_2 \stackrel{\text{def}}{=} \mathsf{L}' \setminus \mathsf{L}'_1$.

Suppose now that, by considering all the other elements of the partitions of h_1 and h_2, we can complete the construction of h'_1 and h'_2 so that $l \in \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h_k} \setminus \mathsf{L}_k \Leftrightarrow l' \in \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s',h'_k} \setminus \mathsf{L}'_k$ and $\min(\alpha_k, \mathrm{card}(\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h_k} \setminus \mathsf{L}_k)) = \min(\alpha_k, \mathrm{card}(\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s',h'_k} \setminus \mathsf{L}'_k))$ holds. From the properties of L'_k ensured by construction it then holds that

$$l \in \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h} \Leftrightarrow l' \in \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s',h'} \qquad \min(\alpha_k, \mathrm{card}(\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h_k})) = \min(\alpha_k, \mathrm{card}(\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s',h'_k}))$$

By semantics of the test formulæ we then conclude the following equisatisfiability results:

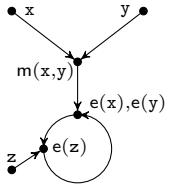
- $(s,h_k,l) \models \mathtt{u} \in \mathtt{loop}^1_{\mathsf{X}}$ if and only if $(s',h'_k,l') \models \mathtt{u} \in \mathtt{loop}^1_{\mathsf{X}}$;
- for each $\beta \in [1,\alpha_k]$, $(s,h_k,l) \models \#\mathtt{loop}^1_{\mathsf{X}} \geq \beta$ if and only if $(s',h'_k,l') \models \#\mathtt{loop}^1_{\mathsf{X}} \geq \beta$.

In order to complete the proof of Lemma 7, a reasoning similar to what we presented here for unlabelled self-loops is applied to each element of the partition and every test formula. This concludes the study of the family of test formulæ for the $\mathcal{C}$ fragment.

4.2 A family of test formulæ capturing $\mathcal{A}$

We now consider the $\mathcal{A}$ fragment and follow the same steps of the last section. Let $\mathsf{X} \subseteq_{\mathtt{fin}} \mathsf{VAR}$. $\mathcal{A}\mathsf{Term}_{\mathsf{X}}$ is the set of *syntactical terms* $\mathsf{X} \cup \mathcal{A}\mathsf{Meet}_{\mathsf{X}} \cup \mathcal{A}\mathsf{End}_{\mathsf{X}}$, where $\mathcal{A}\mathsf{Meet}_{\mathsf{X}} \stackrel{\text{def}}{=} \{\mathsf{m}(\mathtt{x},\mathtt{y}) \mid \mathtt{x},\mathtt{y} \in \mathsf{X}\}$ and $\mathcal{A}\mathsf{End}_{\mathsf{X}} \stackrel{\text{def}}{=} \{\mathsf{e}(\mathtt{x}) \mid \mathtt{x} \in \mathsf{X}\}$ are respectively the set of *meet-point* and *end-point variables*. The interpretation of terms $[\![.]\!]^{\mathsf{X}}_{s,h}$ of the last section is extended on the new terms:

- $[\![\mathsf{m}(\mathtt{x},\mathtt{y})]\!]^{\mathsf{X}}_{s,h} = \ell \stackrel{\text{def}}{\Leftrightarrow} h^{\delta_1}(s(\mathtt{x})) = h^{\delta_2}(s(\mathtt{y})) = \ell$ for some $\delta_1, \delta_2 \geq 1$ and for each $0 \leq \delta'_1 \leq \delta_1$, $0 \leq \delta'_2 \leq \delta_2$, if $\delta'_1 \neq \delta_1$ or $\delta'_2 \neq \delta_2$ then $h^{\delta'_1}(s(\mathtt{x})) \neq h^{\delta'_2}(s(\mathtt{y}))$. For every $\delta' \geq 1$, $h^{\delta'}(\ell) \neq \ell$.
- $[\![\mathsf{e}(\mathtt{x})]\!]^{\mathsf{X}}_{s,h} = \ell \stackrel{\text{def}}{\Leftrightarrow}$ there is $\delta \geq 1$ such that $h^{\delta}(s(\mathtt{x})) = \ell$ and if $\ell \in \mathrm{dom}(h)$ then $h^{\delta'}(\ell) = \ell$ for some $\delta' \geq 1$ and for all $0 \leq \delta_1 < \delta$ there no $\delta_2 \geq 1$ is such that $h^{\delta_2}(h^{\delta_1}(s(\mathtt{x}))) = h^{\delta_1}(s(\mathtt{x}))$.



$[\![\mathsf{m}(\mathtt{x},\mathtt{y})]\!]^{\mathsf{X}}_{s,h}$ is the first location ℓ that is reached by both $s(\mathtt{x})$ and $s(\mathtt{y})$. Moreover, for $[\![\mathsf{m}(\mathtt{x},\mathtt{y})]\!]^{\mathsf{X}}_{s,h}$ to be defined, $s(\mathtt{x})$ and $s(\mathtt{y})$ cannot reach each other and ℓ cannot be inside a cycle. Instead, $[\![\mathsf{e}(\mathtt{x})]\!]^{\mathsf{X}}_{s,h}$ is defined if $s(\mathtt{x})$ is a memory cell that is not inside a loop, as the first location reachable from $s(\mathtt{x})$ that is inside a loop or is not in $\mathrm{dom}(h)$. We extend the notion of *labelled location* to the locations corresponding to terms of $\mathcal{A}\mathsf{Term}_{\mathsf{X}}$ and denote with $\mathcal{A}\mathsf{Labels}^{\mathsf{X}}_{s,h}$ their set. The picture on the right provide an example of the labelled locations in a memory state. As done for $\mathcal{C}$, the test formulæ of the $\mathcal{A}$ fragment express conditions on the labelled locations and on a specific partition of a memory state. Let $\alpha \in \mathbb{N}^+$. We define,

- $\mathcal{A}\mathsf{Pred}^{\mathsf{X}}_{s,h}(\ell) \stackrel{\text{def}}{=} \{\ell' \in \mathrm{dom}(h) \mid h(\ell') = \ell \text{ and } h^{\delta}(s(\mathtt{y})) \neq \ell' \text{ for every } \mathtt{y} \in \mathsf{X} \text{ and } \delta \geq 0\}$ for each $\ell \in s(\mathsf{X})$, i.e. the set of predecessors of a location corresponding to a program variable that are not reached by any location corresponding to a program variable;
- $\mathcal{A}\mathsf{Path}^{\mathsf{X}}_{s,h}(\ell) \stackrel{\text{def}}{=} \{\ell' \in \mathrm{dom}(h) \mid \exists \delta \geq 0\ h^{\delta}(\ell) = \ell' \text{ and } h^{\delta'}(\ell) \notin \mathcal{A}\mathsf{Labels}^{\mathsf{X}}_{s,h} \text{ for every } 0 < \delta' \leq \delta\}$ for each $\ell \in \mathcal{A}\mathsf{Labels}^{\mathsf{X}}_{s,h}$. This is the set of memory cells that are reachable from the labelled location ℓ without passing through any labelled location different from ℓ;
- $\mathcal{A}\mathsf{Loop}^{\mathsf{X}}_{s,h}(\beta) \stackrel{\text{def}}{=} \big\{\{\ell_0,\cdots,\ell_{\beta-1}\} \subseteq \mathrm{dom}(h) \mid h(\ell_i) = \ell_{(i+1 \mod \beta)} \text{ and } \ell_i \notin \mathcal{A}\mathsf{Labels}^{\mathsf{X}}_{s,h}, \text{ for every } i \in [0,\beta-1]\big\}$ for each $\beta \in [1,\alpha]$, i.e. the set of unlabelled cycles of size β;
- $\mathcal{A}\mathsf{Loop}{\Uparrow}^{\mathsf{X}}_{s,h}(\alpha) \stackrel{\text{def}}{=} \big\{\{\ell_0,\cdots,\ell_{\gamma}\} \subseteq \mathrm{dom}(h) \mid (h(\ell_i) = \ell_{(i+1 \mod \gamma)} \text{ and } \ell_i \notin \mathcal{A}\mathsf{Labels}^{\mathsf{X}}_{s,h}), \text{ for each } i \in [0,\gamma-1] \text{ with } \gamma > \alpha\big\}$, i.e. the set of unlabelled cycles of size at least $\alpha+1$;
- $\mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h}(\alpha)$ is the set of locations in $\mathrm{dom}(h)$ that are not in any of the sets defined above.

It is straightforward to see that these sets constitute a partition of $\mathrm{dom}(h)$. Notice that any $\ell \in \mathcal{A}\mathsf{Labels}^{\mathsf{X}}_{s,h}$ is in $\mathrm{dom}(h)$ if and only if $\mathcal{A}\mathsf{Path}^{\mathsf{X}}_{s,h}(\ell) \neq \emptyset$. From the interpretation of terms, if $\mathcal{A}\mathsf{Path}^{\mathsf{X}}_{s,h}(\ell) \neq \emptyset$ then there is exactly one location ℓ' in this set that points to a labelled location. Then, we denote with $\mathcal{A}\mathsf{seen}^{\mathsf{X}}_{s,h}(\ell)$ the location $h(\ell')$, i.e. the first labelled location reachable from ℓ in at least one step. As in the previous section, the set of test formulæ $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha)$ is split into a *skeleton* $\mathcal{A}\mathsf{SKEL}(\mathsf{X},\alpha)$ and an *observed* set $\mathcal{A}\mathsf{OBS}(\mathsf{X},\alpha)$.

$$\mathcal{A}\mathsf{SKEL}(\mathsf{X},\alpha) \stackrel{\text{def}}{=} \left\{ \begin{array}{l} \mathtt{t}_1 = \mathtt{t}_2,\ \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1,\mathtt{t}_2) \geq \beta^{\wr} \\ \#\mathtt{loop}_{\mathsf{X}}(\beta) \geq \beta^{\circlearrowleft},\ \#\mathtt{loop}^{\Uparrow}_{\mathsf{X}} \geq \beta^{\circlearrowleft} \\ \#\mathtt{pred}^{\mathcal{A}}_{\mathsf{X}}(\mathtt{x}) \geq \beta,\ \mathtt{size}^{\mathcal{A}}_{\mathsf{X}} \geq \beta \end{array} \middle| \begin{array}{r} \beta^{\wr} \in [1, \frac{1}{6}(\alpha+1)(\alpha+2)(\alpha+3)] \\ \beta^{\circlearrowleft} \in [1, \frac{1}{2}\alpha(\alpha+3)-1],\ \beta \in [1,\alpha] \\ \mathtt{x} \in \mathsf{X},\ \mathtt{t}_1,\mathtt{t}_2 \in \mathcal{A}\mathsf{Term}_{\mathsf{X}} \end{array} \right\}$$

$$\mathcal{A}\mathsf{OBS}(\mathsf{X},\alpha) \stackrel{\text{def}}{=} \left\{ \begin{array}{l} \mathtt{u} \in \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1,\mathtt{t}_2) \geq (\overleftarrow{\beta}, \overrightarrow{\beta}) \\ \mathtt{u} = \mathtt{t}_1,\ \mathtt{u} \in \mathtt{loop}_{\mathsf{X}}(\beta),\ \mathtt{u} \in \mathtt{loop}^{\Uparrow}_{\mathsf{X}} \\ \mathtt{u} \in \mathtt{pred}^{\mathcal{A}}_{\mathsf{X}}(\mathtt{x}),\ \mathtt{u} \in \mathtt{size}^{\mathcal{A}}_{\mathsf{X}} \end{array} \middle| \begin{array}{r} \overleftarrow{\beta} \in [1, \frac{1}{6}\alpha(\alpha+1)(\alpha+2)+1] \\ \overrightarrow{\beta} \in [1, \frac{1}{2}\alpha(\alpha+3)],\ \beta \in [1,\alpha] \\ \mathtt{x} \in \mathsf{X},\ \mathtt{t}_1,\mathtt{t}_2 \in \mathcal{A}\mathsf{Term}_{\mathsf{X}} \end{array} \right\}$$

The formal semantics of the test formulæ is provided below:

- $(s,h,l) \models \mathtt{t}_1 = \mathtt{t}_2$ if and only if $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$ and $[\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h}$ are both defined, $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h} = [\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h}$.
- $(s,h,l) \models \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1,\mathtt{t}_2) \geq \beta$ iff $\mathrm{card}(\mathcal{A}\mathsf{Path}^{\mathsf{X}}_{s,h}([\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h})) \geq \beta$, $[\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h} = \mathcal{A}\mathsf{seen}^{\mathsf{X}}_{s,h}([\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h})$.
- $(s,h,l) \models \#\mathtt{loop}_{\mathsf{X}}(\beta_1) \geq \beta_2$ if and only if $\mathcal{A}\mathsf{Loop}^{\mathsf{X}}_{s,h}(\beta_1)$ has at least β_2 cycles.
- $(s,h,l) \models \#\mathtt{loop}^{\Uparrow}_{\mathsf{X}} \geq \beta$ if and only if $\mathcal{A}\mathsf{Loop}{\Uparrow}^{\mathsf{X}}_{s,h}(\alpha)$ has at least β cycles.
- $(s,h,l) \models \#\mathtt{pred}^{\mathcal{A}}_{\mathsf{X}}(\mathtt{x}) \geq \beta$ if and only if $\mathcal{A}\mathsf{Pred}^{\mathsf{X}}(s(\mathtt{x}))$ has at least β elements.
- $(s,h,l) \models \mathtt{size}^{\mathcal{A}}_{\mathsf{X}} \geq \beta$ if and only if $\mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h}(\alpha)$ has at least β elements.

- $(s,h,l) \models \mathtt{u} = \mathtt{t}$ if and only if $l = [\![\mathtt{t}]\!]^{\mathsf{X}}_{s,h}$;
- $(s,h,l) \models \mathtt{u} \in \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1,\mathtt{t}_2) \geq (\beta_1,\beta_2)$ if and only if there are $\delta_1 \geq \beta_1$ and $\delta_2 \geq \beta_2$ such that $\delta_1 + \delta_2 = \mathrm{card}(\mathcal{A}\mathsf{Path}^{\mathsf{X}}_{s,h}([\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}))$ and $h^{\delta_1}([\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}) = l$ and $h^{\delta_2}(l) = [\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h}$;
- $(s,h,l) \models \mathtt{u} \in \mathtt{loop}_{\mathsf{X}}(\beta)$ if and only if there is a set $\mathsf{L} \in \mathcal{A}\mathsf{Loop}^{\mathsf{X}}_{s,h}(\beta)$ such that $l \in \mathsf{L}$;
- $(s,h,l) \models \mathtt{u} \in \mathtt{loop}^{\Uparrow}_{\mathsf{X}}$ if and only if there is a set $\mathsf{L} \in \mathcal{A}\mathsf{Loop}{\Uparrow}^{\mathsf{X}}_{s,h}(\alpha)$ such that $l \in \mathsf{L}$;
- $(s,h,l) \models \mathtt{u} \in \mathtt{pred}^{\mathcal{A}}_{\mathsf{X}}(\mathtt{x})$ if and only if $l \in \mathcal{A}\mathsf{Pred}^{\mathsf{X}}_{s,h}(s(\mathtt{x}))$;
- $(s,h,l) \models \mathtt{u} \in \mathtt{size}^{\mathcal{A}}_{\mathsf{X}}$ if and only if $l \in \mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h}(\alpha)$.

As done for the $\mathcal{C}$ fragment, these test formulæ follow closely the partition defined above. For example, $\mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1,\mathtt{t}_2) \geq \beta$ states that $\mathcal{A}\mathsf{Path}^{\mathsf{X}}_{s,h}([\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h})$ describe a path of length at least β from the location corresponding to $\mathtt{t}_1$ to its nearest labelled location $[\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h}$. Then, formula $\mathtt{u} \in \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1,\mathtt{t}_2) \geq (\beta_1,\beta_2)$ state that the current assignment of $\mathtt{u}$ is in this path and is reached from $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$ after at least β_1 steps whereas it reaches $[\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h}$ in at least β_2 steps.

The interesting aspect of $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha)$ lies on the upper-bounds given to the formulæ, e.g. the bound $\frac{1}{2}\alpha(\alpha+3)-1$ on β for $\#\mathtt{loop}^{\Uparrow}_{\mathsf{X}} \geq \beta$ formulaæ. These non-trivial upper-bounds arise as we internalise the separating conjunction so that Lemma 8 (see below) holds. Since these bounds are novelty in the test formulæ proof technique, as an example we informally show how to derive the bound $\frac{1}{2}\alpha(\alpha+3)-1$ on β for $\#\mathtt{loop}^{\Uparrow}_{\mathsf{X}} \geq \beta$ formulæ. Let (s,h,l) be a memory state, $h_1 + h_2 = h$ and $\alpha,\alpha_1,\alpha_2 \in \mathbb{N}^+$ so that $\alpha = \alpha_1 + \alpha_2$ (as in Lemma 8) and (w.l.o.g.) $\alpha_1 \geq \alpha_2$. We study how the satisfaction of the test formulæ changes when the heap h is split into h_1 and h_2 by looking at the possible partitions of these two heaps.

In our simple case, for each loop $\mathsf{P} \in \mathcal{A}\mathsf{Loop}{\Uparrow}^{\mathsf{X}}_{s,h}(\alpha)$ one of the following must hold: $\mathsf{P} \subseteq \mathrm{dom}(h_1)$, $\mathsf{P} \subseteq \mathrm{dom}(h_2)$ or P is divided into two non-empty sets $\mathsf{P}_1 \subseteq \mathrm{dom}(h_1)$ and $\mathsf{P}_2 \subseteq \mathrm{dom}(h_2)$. By definition of the partition, in the first two cases we have $\mathsf{P} \in \mathcal{A}\mathsf{Loop}{\Uparrow}^{\mathsf{X}}_{s,h_k}(\alpha_k)$, for $k \in \{1,2\}$, whereas for the third case we have $\mathsf{P}_1 \subseteq \mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h_1}(\alpha_1)$ and $\mathsf{P}_2 \subseteq \mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h_2}(\alpha_2)$. Then, these locations affects the formulæ $\#\mathtt{loop}^{\Uparrow}_{\mathsf{X}} \geq \beta_1$ and $\mathtt{size}^{\mathcal{A}}_{\mathsf{X}} \geq \beta_2$ of $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha_1)$ and $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha_2)$. For $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha)$, we denote with $\mathcal{L}(\alpha)$ the (desired) upper-bound on β for $\#\mathtt{loop}^{\Uparrow}_{\mathsf{X}} \geq \beta$ formulæ and with $\mathcal{S}(\alpha)$ the one for $\mathtt{size}^{\mathcal{A}}_{\mathsf{X}} \geq \beta'$ formulæ. For the sake of brevity, suppose we derived $\mathcal{S}(\alpha) = \alpha$ with similar arguments to the ones herein described. To effectively internalise the separating conjunct, $\mathcal{L}(\alpha)$ must be at least the sum of the upper-bounds $\mathcal{L}(\alpha_1)$ and $\mathcal{L}(\alpha_2)$ of $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha_1)$ and $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha_2)$ respectively, plus the upper-bound $\mathcal{S}(\alpha_1)$ of $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha_1)$ (as $\alpha_1 \geq \alpha_2$). Then, we need to satisfy the inequality $\mathcal{L}(\alpha) \geq \max_{\alpha=\alpha_1+\alpha_2}(\mathcal{L}(\alpha_1)+\mathcal{L}(\alpha_2)+\mathcal{S}(\alpha_1)+1)$, where the last addend 1 is introduced to handle the quantified variable $\mathtt{u}$. As $\mathcal{L}(\alpha_1)+\mathcal{L}(\alpha_2)+\mathcal{S}(\alpha_1)+1$ is maximal for $\alpha_1 = \alpha - 1$, we solve the recurrence system $\{\mathcal{L}(1) = 1,\ \mathcal{L}(\alpha+1) = \mathcal{L}(\alpha)+\mathcal{L}(1)+\alpha+1\}$ (here, $\mathcal{L}(1) = 1$ refers to the base-case of $\alpha = 1$) and obtain the upper-bound $\mathcal{L}(\alpha) = \frac{1}{2}\alpha(\alpha+3)-1$ that satisfies the inequality above. Similarly, we derive all the upper-bounds of $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha)$ *(Appendix A provides details of these derivations).*

We say that two memory states are indistinguishable, written $(s,h,l) \approx^{\mathcal{A}}_{\mathsf{X},\alpha} (s',h',l')$, for the $\mathcal{A}$ fragment if and only if for all $\varphi \in \mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha)$ $(s,h,l) \models \varphi \iff (s',h',l') \models \varphi$. As for the test formulæ of the $\mathcal{C}$ fragment (Lemma 7), we can show that two indistinguishable memory states cannot be distinguished using separating conjunctions.

▶ **Lemma 8** ($\mathcal{A}$:$*$ indistinguishability). *Let* $\mathsf{X} \subseteq_{\mathtt{fin}} \mathsf{VAR}$ *and* $\alpha,\alpha_1,\alpha_2 \in \mathbb{N}^+$ *with* $\alpha = \alpha_1 + \alpha_2$. *Assume* $(s,h,l) \approx^{\mathcal{A}}_{\mathsf{X},\alpha} (s',h',l')$. *For all heaps* h_1, h_2 *such that* $h = h_1 + h_2$ *there are heaps* h'_1, h'_2 *such that* $h' = h'_1 + h'_2$, $(s,h_1,l) \approx^{\mathcal{A}}_{\mathsf{X},\alpha_1} (s',h'_1,l')$ *and* $(s,h_2,l) \approx^{\mathcal{A}}_{\mathsf{X},\alpha_2} (s',h'_2,l')$.

As done for Lemma 7, this lemma can be proved by looking at (s,h_1,l) and (s,h_2,l) in terms of their partitions and showing that it is possible to construct h'_1 and h'_2 by dividing h' in such a way that $(s,h_1,l) \approx^{\mathcal{A}}_{\mathsf{X},\alpha_1} (s',h'_1,l')$ and $(s,h_2,l) \approx^{\mathcal{A}}_{\mathsf{X},\alpha_2} (s',h'_2,l')$. In order to relate

this result to the observations done for Lemma 7 and show the role of the upper-bounds introduced in this section, let us consider the case of loops of size greater than α. In the following, the index k stands for 1 or 2. For h_1 and h_2, we define the three following sets

$$\mathsf{L}_1 \stackrel{\text{def}}{=} \{\mathsf{P} \in \mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha) \mid \mathsf{P} \subseteq \mathrm{dom}(h_1)\} \qquad \mathsf{L}_2 \stackrel{\text{def}}{=} \{\mathsf{P} \in \mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha) \mid \mathsf{P} \subseteq \mathrm{dom}(h_2)\}$$

$$\mathsf{S} \stackrel{\text{def}}{=} \{(\mathsf{P}_1, \mathsf{P}_2) \mid \mathsf{P}_1 \cup \mathsf{P}_2 \in \mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha),\ \mathsf{P}_1 \neq \emptyset \neq \mathsf{P}_2 \text{ and for each } k \in \{1,2\}\ \mathsf{P}_k \subseteq \mathrm{dom}(h_k)\}$$

Then, L_k is the set of loops in $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha)$ that are completely assigned to h_k whereas S contains the loops of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha)$ that are split between h_1 and h_2. As introduced above, for $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha)$ we denote with $\mathcal{L}(\alpha) \stackrel{\text{def}}{=} \frac{1}{2}\alpha(\alpha+3)-1$ the upper-bound for $\#\mathtt{loop}^{\Uparrow}_{\mathsf{X}} \geq \beta$ formulæ and with $\mathcal{S}(\alpha) \stackrel{\text{def}}{=} \alpha$ the one for $\mathtt{size}^{\mathcal{A}}_{\mathsf{X}} \geq \beta'$ formulæ. Moreover, $\mathcal{P}(.)$ denotes the powerset operator whereas $\pi_k(.)$ denotes the k-th projection of a tuple. We partially construct h'_1 and h'_2 by defining three sets $\mathsf{L}'_1 \subseteq \mathcal{P}(\mathrm{dom}(h'_1))$, $\mathsf{L}'_2 \subseteq \mathcal{P}(\mathrm{dom}(h'_2))$ and $\mathsf{S}' \subseteq \mathcal{P}(\mathrm{dom}(h'_1)) \times \mathcal{P}(\mathrm{dom}(h'_2))$ satisfying the following properties:

- L'_k is the set of loops in $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ that are completely assigned to h'_k
- S' contains the loops of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ that are split between h'_1 and h'_2.
- there is $\mathsf{P} \in \mathsf{L}_k$ with $l \in \mathsf{P}$ if and only if there is $\mathsf{P}' \in \mathsf{L}'_k$ with $l' \in \mathsf{P}'$;
- there is $\mathsf{P} \in \mathsf{S}$ with $l \in \pi_k(\mathsf{P})$ if and only if there is $\mathsf{P}' \in \mathsf{S}'$ with $l' \in \pi_k(\mathsf{P}')$;
- $\min(\mathcal{L}(\alpha_k), \mathrm{card}(\mathsf{L}_k)) = \min(\mathcal{L}(\alpha_k), \mathrm{card}(\mathsf{L}'_k))$;
- $\min(\mathcal{S}(\alpha_k), \mathrm{card}(\bigcup_{\mathsf{P}\in\mathsf{S}} \pi_k(\mathsf{P}))) = \min(\mathcal{S}(\alpha_k), \mathrm{card}(\bigcup_{\mathsf{P}'\in\mathsf{S}'} \pi_k(\mathsf{P}')))$.

By $(s,h,l) \approx^{\mathcal{A}}_{\mathsf{X},\alpha} (s',h',l')$ we are guaranteed to find a construction satisfying all these properties, as it implies that $\min(\mathcal{L}(\alpha), \mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha))) = \min(\mathcal{L}(\alpha), \mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)))$ and $l \in \mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha) \Leftrightarrow l' \in \mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$. Recall that, by definition of the upper-bounds, $\mathcal{L}(\alpha) \geq \mathcal{L}(\alpha_1) + \mathcal{L}(\alpha_2) + \mathcal{S}(\max(\alpha_1,\alpha_2)) + 1$. Then, similarly to the proof of Lemma 7, the construction goes by cases depending on whether or not the cardinalities of L_1, L_2 and S are less than $\mathcal{L}(\alpha_1)$, $\mathcal{L}(\alpha_2)$ and $\mathcal{S}(\max(\alpha_1,\alpha_2))$ respectively and on whether or not l belongs to a set in $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha)$ *(Appendix B provides the details of this step of the construction).*

Suppose now that, by considering all the other elements of the partitions of h_1 and h_2, we can complete the construction of h'_1 and h'_2 so that both heaps enjoy the following properties:

$$l \in \mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h_k}(\alpha_k) \setminus \mathsf{L}_k \Leftrightarrow l' \in \mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'_k}(\alpha_k) \setminus \mathsf{L}'_k$$

$$l \in \mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h}(\alpha_k) \setminus \textstyle\bigcup_{\mathsf{P}\in\mathsf{S}} \pi_k(\mathsf{P}) \Leftrightarrow l' \in \mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s',h'}(\alpha_k) \setminus \bigcup_{\mathsf{P}'\in\mathsf{S}'} \pi_k(\mathsf{P}')$$

$$\min(\mathcal{L}(\alpha_k), \mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h_k}(\alpha_k) \setminus \mathsf{L}_k)) = \min(\mathcal{L}(\alpha_k), \mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'_k}(\alpha_k) \setminus \mathsf{L}'_k))$$

$$\min(\mathcal{S}(\alpha_k), \mathrm{card}(\mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h_k}(\alpha_k) \setminus \bigcup_{\mathsf{P}\in\mathsf{S}} \pi_k(\mathsf{P}))) = \min(\mathcal{S}(\alpha_k), \mathrm{card}(\mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s',h'_k}(\alpha_k) \setminus \bigcup_{\mathsf{P}'\in\mathsf{S}'} \pi_k(\mathsf{P}')))$$

Then, as previously done for Lemma 7, by semantics of the test formulæ and from the properties of L'_k and S' ensured by construction we obtain the following equisatisfiability results:

- $(s,h_k,l) \models \mathtt{u} \in \mathtt{loop}^{\Uparrow}_{\mathsf{X}}$ if and only if $(s',h'_k,l') \models \mathtt{u} \in \mathtt{loop}^{\Uparrow}_{\mathsf{X}}$;
- $(s,h_k,l) \models \mathtt{u} \in \mathtt{size}^{\mathcal{A}}_{\mathsf{X}}$ if and only if $(s',h'_k,l') \models \mathtt{u} \in \mathtt{size}^{\mathcal{A}}_{\mathsf{X}}$;
- for each $\beta \in [1, \mathcal{L}(\alpha_k)]$, $(s,h_k,l) \models \#\mathtt{loop}^{\Uparrow}_{\mathsf{X}} \geq \beta$ if and only if $(s',h'_k,l') \models \#\mathtt{loop}^{\Uparrow}_{\mathsf{X}} \geq \beta$.
- for each $\beta \in [1, \mathcal{S}(\alpha_k)]$, $(s,h_k,l) \models \mathtt{size}^{\mathcal{A}}_{\mathsf{X}} \geq \beta$ if and only if $(s',h'_k,l') \models \mathtt{size}^{\mathcal{A}}_{\mathsf{X}} \geq \beta$.

In order to complete the proof of Lemma 8, a similar reasoning is applied to each element of the partition and every test formula. We conclude this section by showing that $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha)$ also enjoys quantification indistinguishability. The proof goes, as for Lemma 6, by checking how the satisfaction of formulæ in $\mathcal{A}\mathsf{OBS}(\mathsf{X},\alpha)$ changes as the quantified variable is reassigned.

▶ **Lemma 9** ($\mathcal{A}$:∃ indistinguishability). *Assume* $(s,h,l) \approx^{\mathcal{A}}_{\mathsf{X},\alpha} (s',h',l')$. *Let* $\ell \in \mathsf{LOC} \setminus \mathsf{L}$ *with* $\mathsf{L} \stackrel{\text{def}}{=} \mathrm{dom}(h') \cup \mathrm{ran}(h') \cup s'(\mathsf{X})$. *For every* $l_1 \in \mathsf{LOC}$ *there is* $l'_1 \in \mathsf{L} \cup \{\ell\}$ *s.t.* $(s,h,l_1) \approx^{\mathcal{A}}_{\mathsf{X},\alpha} (s',h',l'_1)$.

4.3 Connecting the two families of test formulæ

We are now ready to show that the two indistinguishability relations introduced for the $\mathcal{C}$ and $\mathcal{A}$ fragment allow us to mimic the $\mathcal{A} \mathrel{-\!\!*} \mathcal{C}$ separating implication using the test formulæ.

▶ **Lemma 10** ($\mathcal{A} \mathrel{-\!\!*} \mathcal{C}$ indistinguishability). *Let* $(s,h,l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha+\mathrm{card}(\mathsf{X})} (s',h',l')$ *for some* $\alpha \in \mathbb{N}^+$ *and* $\mathsf{X} \subseteq_{\mathtt{fin}} \mathsf{VAR}$. *For all heaps* h_1 *disjoint from* h *there exists* h'_1 *disjoint from* h' *such that:*

1. $\mathrm{card}(\mathrm{dom}(h'_1))$ *is bounded by a polynomial* $\mathfrak{P}(\mathrm{card}(\mathsf{X}),\alpha)$ *in* $\mathcal{O}(\mathrm{card}(\mathsf{X})^3\alpha^4)$,
2. $(s,h_1,l) \approx^{\mathcal{A}}_{\mathsf{X},\alpha} (s',h'_1,l')$ *and*
3. $(s,h+h_1,l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha} (s',h'+h'_1,l')$.

The proof is achieved by defining a "small" heap h'_1 so that the first two points of the lemma are guaranteed by construction. In doing so, we need to carefully handle the labelled locations so that the third point also holds. For instance, suppose that in h_1 there is a path from $s(\mathtt{x})$ to a location corresponding to the term $\mathtt{t}$. Moreover, in this path the location $h_1(s(\mathtt{x}))$ corresponds to $\mathsf{n}(\mathtt{y})$ in h, as represented below (on the left). Then, $(s,h+h_1,l) \models \mathsf{n}(\mathtt{x}) = \mathsf{n}(\mathtt{y})$.

We then construct (see the memory state on the right) h'_1 so that there is a path from $s'(\mathtt{x})$ to a location corresponding to the term $\mathtt{t}$ where $h'_1(s'(\mathtt{x}))$ is the location corresponding to the therm $\mathsf{n}(\mathtt{y})$ in h'. The hypothesis $(s,h,l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha+\mathrm{card}(\mathsf{X})} (s',h',l')$ guarantees that this can always be done correctly. The third point of the lemma is then achieved by noticing that the second point implies $(s,h_1,l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha} (s',h',l')$. Indeed this holds from the following result.

▶ **Lemma 11.** *Let* $\mathsf{X} \subseteq_{\mathtt{fin}} \mathsf{VAR}$ *and* $\alpha \in \mathbb{N}^+$. *It holds that* $\approx^{\mathcal{A}}_{\mathsf{X},\alpha} \subseteq \approx^{\mathcal{C}}_{\mathsf{X},\alpha}$.

To relate the equisatisfaction of φ in $\mathsf{1SL}^{\mathtt{R1}}_{\mathtt{R2}}(*,\mathrel{-\!\!*},\mathtt{reach}^+)$ to the indistinguishability relations, we define its *memory size* $|\varphi|_{\mathsf{m}}$ as: 1 for atomic formulæ, $|\varphi_1 \wedge \varphi_2|_{\mathsf{m}} \stackrel{\text{def}}{=} \max(|\varphi_1|_{\mathsf{m}}, |\varphi_2|_{\mathsf{m}})$, $|\neg\varphi|_{\mathsf{m}} \stackrel{\text{def}}{=} |\varphi|_{\mathsf{m}}$, $|\varphi_1 * \varphi_2|_{\mathsf{m}} \stackrel{\text{def}}{=} |\varphi_1|_{\mathsf{m}} + |\varphi_2|_{\mathsf{m}}$ and $|\varphi_1 \mathrel{-\!\!*} \varphi_2|_{\mathsf{m}} \stackrel{\text{def}}{=} \mathsf{fv}(\varphi_1 \mathrel{-\!\!*} \varphi_2) + \max(|\varphi_1|_{\mathsf{m}}, |\varphi_2|_{\mathsf{m}})$.

▶ **Lemma 12.** *Let* φ *be a* $\mathcal{A}$*-formula such that* $\mathsf{fv}(\varphi) \subseteq \mathsf{X} \subseteq_{\mathtt{fin}} \mathsf{VAR}$ *and* $|\varphi|_{\mathsf{m}} \leq \alpha \in \mathbb{N}^+$. *Let* (s,h,l), (s',h',l') *be two memory states.* $(s,h,l) \models \varphi \iff (s',h',l') \models \varphi$ *whenever*

1. $(s,h,l) \approx^{\mathcal{A}}_{\mathsf{X},\alpha} (s',h',l')$, *or*
2. $(s,h,l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha} (s',h',l')$ *and* φ *is a* $\mathcal{C}$*-formula.*

The proof is by structural induction on φ. The basic cases require to translate every atomic $\mathcal{C}$-formula and $\mathtt{reach}^+(e_1,e_2)$ into a boolean combination of respectively $\mathcal{C}\mathsf{TEST}(\mathsf{X},1)$ and $\mathcal{A}\mathsf{TEST}(\mathsf{X},1)$ formulæ. The inductive cases for Boolean connectives are immediate, whereas for $*$, $\mathrel{-\!\!*}$, $\exists$ we use the various indistinguishability lemmata. Then, the following result holds.

▶ **Theorem 13** ($\mathcal{A}$ captures the logic). *Every* $\mathsf{1SL}^{\mathtt{R1}}_{\mathtt{R2}}(*,\mathrel{-\!\!*},\mathtt{reach}^+)$ *formula* φ *is logically equivalent to a Boolean combination of test formulæ from* $\mathcal{A}\mathsf{TEST}(\mathsf{fv}(\varphi),|\varphi|_{\mathsf{m}})$.

For the proof, we first show that $(s,h,l) \approx^{\mathcal{A}}_{\mathsf{fv}(\varphi),|\varphi|_{\mathsf{m}}} (s',h',l')$ and $(s',h',l') \models \bigwedge_{\psi \in \mathsf{LIT}(s,h,l)} \psi$ are equivalent, where $\mathsf{LIT}(s,h,l)$ is the finite set of literals

$$\{\psi \in \mathcal{A}\mathsf{TEST}(\mathsf{fv}(\varphi),|\varphi|_{\mathsf{m}}) \mid (s,h,l) \models \psi\} \cup \{\neg\psi \mid (s,h,l) \not\models \psi \text{ and } \psi \in \mathcal{A}\mathsf{TEST}(\mathsf{fv}(\varphi),|\varphi|_{\mathsf{m}})\}$$

Then, the expression $\bigvee_{(s,h,l)\models\varphi} \bigwedge_{\psi\in\mathsf{LIT}(s,h,l)} \psi$ is equivalent to a Boolean combination χ of $\mathcal{A}\mathsf{TEST}(\mathsf{fv}(\varphi),|\varphi|_{\mathsf{m}})$ formulæ. Using Lemma 12 we conclude that φ is logically equivalent to χ.

Complexity upper bounds. Analogously to what is done in [12, 13], following Theorem 13 we can establish a small model property for $\mathsf{1SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$, whose proof relies on the upper-bounds on $\mathcal{A}\mathsf{TEST}(\mathsf{X}, \alpha)$ formulæ discussed in the previous section. Let $|\mathsf{X}_\varphi| \stackrel{\text{def}}{=} \mathrm{card}(\mathsf{fv}(\varphi))$.

▶ **Theorem 14** (small model). *Every satisfiable φ in $\mathsf{1SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$ is satisfied by a state (s, h, l) such that $\mathrm{card}(\mathrm{dom}(h))$ is bounded by a polynomial $\mathcal{Q}(|\mathsf{X}_\varphi|, |\varphi|_{\mathtt{m}})$ in $\mathcal{O}(|\mathsf{X}_\varphi|^3 \, |\varphi|^4_{\mathtt{m}})$.*

The satisfiability problem of $\mathsf{1SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$ is then PSPACE-complete (the hardness is inherited from propositional separation logic [9]). Indeed, as $|\varphi|_{\mathtt{m}}$ is at most $|\mathsf{X}_\varphi| \times |\varphi|$, where $|\varphi|$ is the number of symbols in φ, Theorem 14 ensures a polynomial bound $\mathcal{Q}(|\mathsf{X}_\varphi|, |\mathsf{X}_\varphi| \times |\varphi|)$ on the heap of the smallest memory state that satisfies φ. Then, the non-deterministic PSPACE algorithm (leading to a PSPACE upper-bound by Savitch Theorem [27]) first guess a heap h, a store s restricted to $\mathsf{fv}(\varphi)$ and $l \in \mathsf{LOC}$ such that $\mathrm{dom}(h) \cup \mathrm{ran}(h) \cup \mathrm{ran}(s) \cup \{l\}$ has at most $2 \times \mathcal{Q}(|\mathsf{X}_\varphi|, |\mathsf{X}_\varphi| \times |\varphi|) + |\mathsf{X}_\varphi| + 1$ locations (in the worst case all these sets are disjoint). It then checks that $(s, h, l) \models \varphi$ by using a linear-depth recursive algorithm that internalises the semantics of φ (see e.g. [8]). The various indistinguishability lemmata ensure that only a polynomial amount of locations ever needs to be considered. For instance, $(s, h, l) \not\models \varphi_1 -\!\!* \varphi_2$ if and only if we can guess h_1 so that $(s, h_1, l) \models \varphi_1$, $(s, h + h_1, l) \not\models \varphi_2$ and (by Lemma 10) $\mathrm{dom}(h_1) \cup \mathrm{ran}(h_2)$ has at most $2 \times \mathfrak{P}(|\mathsf{X}_\varphi|, |\mathsf{X}_\varphi| \times |\varphi|)$ locations.

▶ **Theorem 15.** *The satisfiability problem of $\mathsf{1SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$ can be decided in* PSPACE.

5 Conclusions

We studied $\mathsf{1SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$, an extension of propositional separation logic involving one quantified variable and reachability predicates whose satisfiability problem is PSPACE-complete. We discussed how modest extensions of the logic entail TOWER-hardness.

As far as we know, $\mathsf{1SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$ is the largest decidable fragment in which the separating implication cohabits with reachability predicates and quantified variables, subsuming the logics introduced in [12] and [13] which were also found to be PSPACE-complete. Moreover, crucial robustness properties lying outside the expressive power of many fragments of separation logic can be directly expressed in $\mathsf{1SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$ and checked in PSPACE. Then, a procedure to decide robustness properties for this logic could perhaps be implemented inside tools using any fragment of $\mathsf{1SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$ as their assertion logic (see e.g. [15]).

To prove the PSPACE upper-bound we relied on the widely used proof technique of the test formulæ, here extended to capture asymmetric spatial connectives. The work presented in [10] show a possible relation between test formulæ and games, paving a way toward better understanding this technique and generalising it even further. In particular, by also using the recurrence systems introduced here in Section 4.2, it seems possible to use this approach to tackle in a new way extensions of separation logic with user-defined inductive predicates.

References

1 Timos Antonopoulos, Nikos Gorogiannis, Christoph Haase, Max I. Kanovich, and Joël Ouaknine. Foundations for Decision Problems in Separation Logic with General Inductive Predicates. In *FoSSaCS*, volume 8412 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2014.

2 Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. Smallfoot: Modular Automatic Assertion Checking with Separation Logic. In *FMCO*, volume 4111 of *Lecture Notes in Computer Science*, pages 115–137. Springer, 2005.

3 Josh Berdine, Byron Cook, and Samin Ishtiaq. SLAyer: Memory Safety for Systems-Level Code. In *CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 178–183. Springer, 2011.

4 Rémi Brochenin, Stéphane Demri, and Étienne Lozes. On the almighty wand. *Inf. Comput.*, 211:106–137, 2012.

5 Luís Caires and Luca Cardelli. A Spatial Logic for Concurrency. In *TACS*, volume 2215 of *Lecture Notes in Computer Science*, pages 1–37. Springer, 2001.

6 Cristiano Calcagno and Dino Distefano. Infer: An Automatic Program Verifier for Memory Safety of C Programs. In *NASA Formal Methods*, volume 6617 of *Lecture Notes in Computer Science*, pages 459–465. Springer, 2011.

7 Cristiano Calcagno, Dino Distefano, Peter W. O'Hearn, and Hongseok Yang. Compositional Shape Analysis by Means of Bi-Abduction. *J. ACM*, 58(6):26:1–26:66, 2011.

8 Cristiano Calcagno, Hongseok Yang, and Peter W. O'Hearn. Computability and Complexity Results for a Spatial Assertion Language for Data Structures. In *FSTTCS*, volume 2245 of *Lecture Notes in Computer Science*, pages 108–119. Springer, 2001.

9 Byron Cook, Christoph Haase, Joël Ouaknine, Matthew J. Parkinson, and James Worrell. Tractable Reasoning in a Fragment of Separation Logic. In *CONCUR*, volume 6901 of *Lecture Notes in Computer Science*, pages 235–249. Springer, 2011.

10 Anuj Dawar, Philippa Gardner, and Giorgio Ghelli. Adjunct Elimination Through Games in Static Ambient Logic. In *FSTTCS*, volume 3328 of *Lecture Notes in Computer Science*, pages 211–223. Springer, 2004.

11 Stéphane Demri and Morgan Deters. Two-Variable Separation Logic and Its Inner Circle. *ACM Trans. Comput. Log.*, 16(2):15:1–15:36, 2015.

12 Stéphane Demri, Didier Galmiche, Dominique Larchey-Wendling, and Daniel Méry. Separation Logic with One Quantified Variable. *Theory Comput. Syst.*, 61(2):371–461, 2017.

13 Stéphane Demri, Étienne Lozes, and Alessio Mansutti. The Effects of Adding Reachability Predicates in Propositional Separation Logic. In *FoSSaCS*, volume 10803 of *Lecture Notes in Computer Science*, pages 476–493. Springer, 2018.

14 Kamil Dudka, Petr Peringer, and Tomás Vojnar. Predator: A Practical Tool for Checking Manipulation of Dynamic Data Structures Using Separation Logic. In *CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 372–378. Springer, 2011.

15 Christoph Haase, Samin Ishtiaq, Joël Ouaknine, and Matthew J. Parkinson. SeLoger: A Tool for Graph-Based Reasoning in Separation Logic. In *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 790–795. Springer, 2013.

16 C. A. R. Hoare. An Axiomatic Basis for Computer Programming. *Commun. ACM*, 12(10):576–580, 1969.

17 Radu Iosif, Adam Rogalewicz, and Jirí Simácek. The Tree Width of Separation Logic with Recursive Definitions. In *CADE*, volume 7898 of *Lecture Notes in Computer Science*, pages 21–38. Springer, 2013.

18 Bart Jacobs, Jan Smans, Pieter Philippaerts, Frédéric Vogels, Willem Penninckx, and Frank Piessens. VeriFast: A Powerful, Sound, Predictable, Fast Verifier for C and Java. In *NASA Formal Methods*, volume 6617 of *Lecture Notes in Computer Science*, pages 41–55. Springer, 2011.

19 Christina Jansen, Jens Katelaan, Christoph Matheja, Thomas Noll, and Florian Zuleger. Unified Reasoning About Robustness Properties of Symbolic-Heap Separation Logic. In *ESOP*, volume 10201 of *Lecture Notes in Computer Science*, pages 611–638. Springer, 2017.

20 Jens Katelaan, Dejan Jovanovic, and Georg Weissenbacher. A Separation Logic with Data: Small Models and Automation. In *IJCAR*, volume 10900 of *Lecture Notes in Computer Science*, pages 455–471. Springer, 2018.

21 Quang Loc Le, Makoto Tatsuta, Jun Sun, and Wei-Ngan Chin. A Decidable Fragment in Separation Logic with Inductive Predicates and Arithmetic. In *CAV (2)*, volume 10427 of *Lecture Notes in Computer Science*, pages 495–517. Springer, 2017.

22 Étienne Lozes. Adjuncts elimination in the static ambient logic. *Electr. Notes Theor. Comput. Sci.*, 96:51–72, 2004.

23 Stephen Magill, Ming-Hsien Tsai, Peter Lee, and Yih-Kuen Tsay. THOR: A tool for reasoning about shape and arithmetic. In *CAV*, volume 5123 of *Lecture Notes in Computer Science*, pages 428–432. Springer, 2008.

24 Ben C. Moszkowski. A Hierarchical Completeness Proof for Propositional Interval Temporal Logic with Finite Time. *Journal of Applied Non-Classical Logics*, 14(1-2):55–104, 2004.

25 Benjamin Charles Moszkowski. *Reasoning About Digital Circuits*. PhD thesis, Stanford University, Stanford, CA, USA, 1983.

26 John C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *LICS*, pages 55–74. IEEE Computer Society, 2002.

27 Walter J. Savitch. Relationships Between Nondeterministic and Deterministic Tape Complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.

28 Sylvain Schmitz. Complexity Hierarchies beyond Elementary. *TOCT*, 8(1):3:1–3:36, 2016.

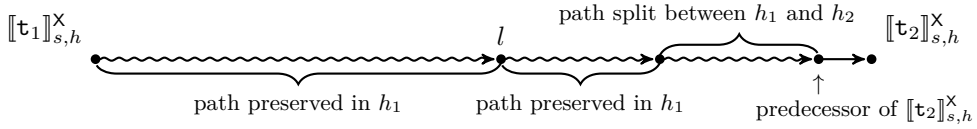
29 Hongseok Yang, Oukseh Lee, Josh Berdine, Cristiano Calcagno, Byron Cook, Dino Distefano, and Peter W. O'Hearn. Scalable Shape Analysis for Systems Code. In *CAV*, volume 5123 of *Lecture Notes in Computer Science*, pages 385–398. Springer, 2008.

A Details on the upper-bounds of $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha)$ formulæ

In this section we provide the inequalities and recurrence systems that have been used to compute the upper-bounds for the test formulæ $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha)$ introduced in Section 4.2. Note that the recurrence systems considered are generally more constrained that the inequalities that we want to satisfy. This is however not a problem, as we just need to find a possible solution to the inequalities, and our recurrence systems provide exactly that. For the cases of $\#\mathtt{loop}_{\mathsf{X}}(\beta_1) \geq \beta_2$ and $\#\mathtt{loop}^{\Uparrow}_{\mathsf{X}} \geq \beta_2$ we refer the reader to the body of the paper. Let (s,h,l) be a memory state, $h_1 + h_2 = h$ and $\alpha, \alpha_1, \alpha_2 \in \mathbb{N}^+$ s.t. $\alpha = \alpha_1 + \alpha_2$ (as in Lemma 8).

- Bound for $\mathtt{size}^{\mathcal{A}}_{\mathsf{X}} \geq \beta$ formulæ. For $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha)$, let us call this upper-bound $\mathcal{S}(\alpha)$, as done in the body of the paper.
  - Inequality: $\mathcal{S}(\alpha) \geq \max_{\alpha=\alpha_1+\alpha_2}(\mathcal{S}(\alpha_1) + \mathcal{S}(\alpha_2))$.
  - Recurrence system: $\{\mathcal{S}(1) = 1,\ \mathcal{S}(\alpha+1) = \mathcal{S}(\alpha) + 1\}$.
  - Solution: $\mathcal{S}(\alpha) = \alpha$.
  - Informal explanation: every location in $\mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h}(\alpha)$ can only appear in $\mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h_1}(\alpha_1)$ or $\mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h_2}(\alpha_2)$.
- Bound for $\#\mathtt{pred}^{\mathcal{A}}_{\mathsf{X}} \geq \beta$ formulæ. For $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha)$, let us call this upper-bound $\mathsf{Pred}(\alpha)$.
  - Inequality: $\mathsf{Pred}(\alpha) \geq \max_{\alpha=\alpha_1+\alpha_2}(\mathsf{Pred}(\alpha_1) + \mathsf{Pred}(\alpha_2))$.
  - Recurrence system: $\{\mathsf{Pred}(1) = 1,\ \mathsf{Pred}(\alpha+1) = \mathsf{Pred}(\alpha) + 1\}$.
  - Solution: $\mathsf{Pred}(\alpha) = \alpha$.
  - Informal explanation: similarly to the previous case, every location in $\mathcal{A}\mathsf{Pred}^{\mathsf{X}}_{s,h}(\alpha)$ can only appear in $\mathcal{A}\mathsf{Pred}^{\mathsf{X}}_{s,h_1}(\alpha_1)$ or $\mathcal{A}\mathsf{Pred}^{\mathsf{X}}_{s,h_2}(\alpha_2)$.
- Bound for $\overrightarrow{\beta}$ in $\mathtt{u} \in \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1, \mathtt{t}_2) \geq (\overleftarrow{\beta}, \overrightarrow{\beta})$ formulæ. For $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha)$, let us call this upper-bound $\mathsf{Right}(\alpha)$.
  - Inequality: $\mathsf{Right}(\alpha) \geq \max_{\alpha=\alpha_1+\alpha_2}(\mathsf{Right}(\max(\alpha_1,\alpha_2)) + \mathcal{S}(\alpha_1) + \mathcal{S}(\alpha_2) + 1)$.
  - Recurrence system: $\{\mathsf{Right}(1) = 2,\ \mathsf{Right}(\alpha+1) = \mathsf{Right}(\alpha) + (\alpha+1) + 1\}$.
  - Solution: $\mathsf{Right}(\alpha) = \frac{1}{2}\alpha(\alpha+3)$.

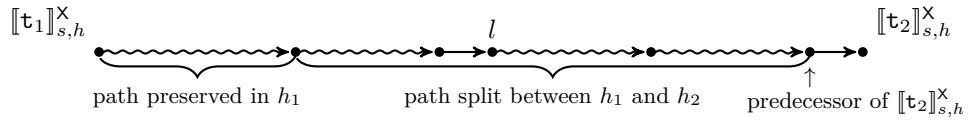
- Informal explanation: the semantics of the formula $\mathtt{u} \in \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1, \mathtt{t}_2) \geq (\overleftarrow{\beta}, \overrightarrow{\beta})$ implies that there is a path from $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$ to $[\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h}$ that goes through l and is such that between l and $[\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h}$ there are at least $\overrightarrow{\beta}$ locations. Suppose $\alpha_1 \geq \alpha_2$ (therefore, we focus our attention on h_1). Then, when the heap h is split into h_1 and h_2 it can hold that:
  - $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$ is a labelled location of (s, h_1, l);
  - in h_1, the location l is still reachable from $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$, as shown in the picture below.



  In this case, the set of locations that are reachable in h_1 from l still counts for the satisfaction of a $\mathtt{u} \in \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1, \mathtt{t}') \geq (\overleftarrow{\beta}, \overrightarrow{\beta})$ predicate, again with respect to $\overrightarrow{\beta}$. This justifies $\mathsf{Right}(\max(\alpha_1, \alpha_2))$ addend of the inequality above. In the figure, notice how there there is a part of the path whose locations are split between h_1 and h_2. These locations can only appear in $\mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h_1}(\alpha_1)$ or $\mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h_2}(\alpha_2)$ and justify the addition of $\mathcal{S}(\alpha_1) + \mathcal{S}(\alpha_2)$ to the inequality. Lastly, if $[\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h}$ corresponds to a program variable in X then its predecessor appears in $\mathcal{A}\mathsf{Pred}^{\mathsf{X}}_{s,h_1}([\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h})$ or in $\mathcal{A}\mathsf{Pred}^{\mathsf{X}}_{s,h_2}([\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h})$. For this reason, the inequality above contains a $+1$ addend. Then, the inequality is maximal for $\alpha_1 = \alpha - 1$ and $\alpha_2 = 1$ or vice-versa (recall that we do not admit α_1 or α_2 to be equal to 0). From $\mathcal{S}(\alpha) = \alpha$ we obtain the recurrence system above with solution $\mathsf{Right}(\alpha) = \frac{1}{2}\alpha(\alpha + 3)$. Notice that the base case for the recurrence system is $\mathsf{Right}(1) = 2$. Under the condition that $[\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h} = s(\mathtt{x})$ for some $\mathtt{x} \in \mathsf{X}$, the test formulæ can always check if the path from l to $s(\mathtt{x})$ has at least length 2 with the conjunction $\mathtt{u} \in \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1, \mathtt{x}) \geq (1, 1) \wedge \neg \mathtt{u} \in \#\mathtt{pred}^{\mathcal{A}}_{\mathsf{X}}(\mathtt{x})$. We therefore require $\mathsf{Right}(1)$ to be at least 2 so that the formula above can be simply expressed in $\mathcal{A}\mathsf{TEST}(\mathsf{X}, 1)$ as $\mathtt{u} \in \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1, \mathtt{x}) \geq (1, 2)$. This makes the proof of Lemma 8 easier as, with respect to $\overrightarrow{\beta}$, we can handle the test formulæ $\mathtt{u} \in \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1, \mathtt{t}_2) \geq (\overleftarrow{\beta}, \overrightarrow{\beta})$ without looking at the satisfaction of other test formulæ.
- Bounds for $\mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1, \mathtt{t}_2) \geq \beta$ formulæ and $\overleftarrow{\beta}$ in $\mathtt{u} \in \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1, \mathtt{t}_2) \geq (\overleftarrow{\beta}, \overrightarrow{\beta})$ formulæ. For $\mathcal{A}\mathsf{TEST}(\mathsf{X}, \alpha)$, we denote the first upper-bound with $\mathsf{Sees}(\alpha)$ whereas the second upper-bound is called $\mathsf{Left}(\alpha)$.
  - Inequalities: $\mathsf{Sees}(\alpha) \geq \mathsf{Left}(\alpha) + \mathsf{Right}(\alpha)$;
  $$\mathsf{Left}(\alpha) + \mathsf{Right}(\alpha) \geq \max_{\alpha = \alpha_1 + \alpha_2}(\mathsf{Sees}(\max(\alpha_1 + \alpha_2)) + 1 + \mathcal{S}(\alpha_1) + \mathcal{S}(\alpha_2) + 1)$$
  - Recurrence system:
  $$\left\{\begin{array}{l} \mathsf{Left}(1) = 2,\ \mathsf{Left}(\alpha + 1) = \mathsf{Sees}(\alpha) + 1, \\ \mathsf{Sees}(\alpha) = \mathsf{Left}(\alpha) + \mathsf{Right}(\alpha) \end{array}\right\}$$
  - Solution: $\mathsf{Left}(\alpha) = \frac{1}{6}\alpha(\alpha + 1)(\alpha + 2) + 1$ and $\mathsf{Sees}(\alpha) = \frac{1}{6}(\alpha + 1)(\alpha + 2)(\alpha + 3)$.
  - Informal explanation: first of, it is easy to see that $\mathsf{Sees}(\alpha) \geq \mathsf{Left}(\alpha) + \mathsf{Right}(\alpha)$ must hold. Indeed if a memory state satisfies the test formula $\mathtt{u} \in \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1, \mathtt{t}_2) \geq (\overleftarrow{\beta}, \overrightarrow{\beta})$ then by definition it also satisfies $\mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1, \mathtt{t}_2) \geq \overleftarrow{\beta} + \overrightarrow{\beta}$. Then, the latter formula needs to be in $\mathcal{A}\mathsf{TEST}(\mathsf{X}, \alpha)$ (otherwise, for instance, Lemma 9 does not hold). The first inequality takes care of this. In the following, we strength this inequality to $\mathsf{Sees}(\alpha) = \mathsf{Left}(\alpha) + \mathsf{Right}(\alpha)$, as done for the recurrence system.
  For the second inequality, the semantics of the formula $\mathtt{u} \in \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1, \mathtt{t}_2) \geq (\overleftarrow{\beta}, \overrightarrow{\beta})$ implies that there is a path from $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$ to $[\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h}$ that goes through l and is such that

between $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$ and l there are at least $\overleftarrow{\beta}$ locations. Suppose $\alpha_1 \geq \alpha_2$ (therefore, we focus our attention on h_1). When the heap h is split into h_1 and h_2 it can hold that:

* $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$ is a labelled location of (s, h_1, l) and is such that $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h_1} \in \mathrm{dom}(h_1)$;
* in h_1, l is no longer reachable from $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$, as shown in the picture below.



In this case, the set of locations that are reachable in h_1 from $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$ counts for the satisfaction of a $\mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1, \mathtt{t}_2) \geq \beta$ predicate. This justifies $\mathsf{Sees}(\max(\alpha_1, \alpha_2))$ addend of the second inequality. In the figure, notice how there there is a part of the path whose locations are split between h_1 and h_2. These locations can only appear in $\mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h_1}(\alpha_1)$ or $\mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h_2}(\alpha_2)$ and justify the addition of $\mathcal{S}(\alpha_1) + \mathcal{S}(\alpha_2)$ to the inequality. If $[\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h}$ corresponds to a program variable in X then its predecessor appears in $\mathcal{A}\mathsf{Pred}^{\mathsf{X}}_{s,h_1}([\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h})$ or in $\mathcal{A}\mathsf{Pred}^{\mathsf{X}}_{s,h_2}([\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h})$. For this reason, the inequality above contains a $+1$ addend. A last $+1$ is instead added to handle a corner case, regarding l, in the proof of Lemma 8. Then, the second inequality is maximal for $\alpha_1 = \alpha - 1$ and $\alpha_2 = 1$ or vice versa (recall that we do not admit α_1 or α_2 to be equal to 0). We then obtain

$$\mathsf{Left}(\alpha) + \mathsf{Right}(\alpha) \geq \mathsf{Sees}(\alpha - 1) + 1 + \mathcal{S}(\alpha) + 1$$

As $\mathsf{Right}(\alpha) \geq \mathcal{S}(\alpha) + 1$, to find a solution is then sufficient consider the recurrence system above. Notice that the base case for the recurrence system is $\mathsf{Left}(1) = 2$. As for the previous case, this is done to ease the proof of Lemma 8, as we can then handle the $\mathtt{u} \in \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1, \mathtt{t}_2) \geq (\overleftarrow{\beta}, \overrightarrow{\beta})$ formulæ without looking at the satisfaction of other test formulæ.

B Lemma 8: the case of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha)$

In this section we complete the construction started in Section 4.2. Let (s, h, l) and (s', h', l') be two memory states. Moreover let $\alpha, \alpha_1, \alpha_2 \in \mathbb{N}^+$ so that $\alpha = \alpha_1 + \alpha_2$ and $h = h_1 + h_2$. In the following, the index k stands for 1 or 2. For h_1 and h_2, we define the three following sets

$$\mathsf{L}_1 \stackrel{\text{def}}{=} \{\mathsf{P} \in \mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha) \mid \mathsf{P} \subseteq \mathrm{dom}(h_1)\} \qquad \mathsf{L}_2 \stackrel{\text{def}}{=} \{\mathsf{P} \in \mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha) \mid \mathsf{P} \subseteq \mathrm{dom}(h_2)\}$$

$$\mathsf{S} \stackrel{\text{def}}{=} \{(\mathsf{P}_1, \mathsf{P}_2) \mid \mathsf{P}_1 \cup \mathsf{P}_2 \in \mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha),\ \mathsf{P}_1 \neq \emptyset \neq \mathsf{P}_2 \text{ and for each } k \in \{1,2\}\ \mathsf{P}_k \subseteq \mathrm{dom}(h_k)\}$$

Then, L_k is the set of loops in $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha)$ that are completely assigned to h_k whereas S contains the loops of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha)$ that are split between h_1 and h_2. Lastly, we denote with $\mathcal{L}(\alpha) \stackrel{\text{def}}{=} \frac{1}{2}\alpha(\alpha+3) - 1$ the upper-bound for $\#\mathtt{loop}^{\Updownarrow}_{\mathsf{X}} \geq \beta$ formulæ and with $\mathcal{S}(\alpha) \stackrel{\text{def}}{=} \alpha$ the one for $\mathtt{size}^{\mathcal{A}}_{\mathsf{X}} \geq \beta'$ formulæ, as introduced in Section 4.2.

The result that we want to prove is the following:

Hypothesis:

H1. $\min(\mathcal{L}(\alpha), \mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha))) = \min(\mathcal{L}(\alpha), \mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)))$

H2. $l \in \mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha) \Leftrightarrow l' \in \mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$.

Thesis:

there are h'_1, h'_2 such that $h' = h'_1 + h'_2$ such that it is possible to define three sets $\mathsf{L}'_1 \subseteq \mathcal{P}(\mathrm{dom}(h'_1))$, $\mathsf{L}'_2 \subseteq \mathcal{P}(\mathrm{dom}(h'_2))$ and $\mathsf{S}' \subseteq \mathcal{P}(\mathrm{dom}(h'_1)) \times \mathcal{P}(\mathrm{dom}(h'_2))$ satisfying the following properties:

- L'_k is the set of loops in $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ that are completely assigned to h'_k
- S' contains the loops of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ that are split between h'_1 and h'_2.
- there is $\mathsf{P} \in \mathsf{L}_k$ with $l \in \mathsf{P}$ if and only if there is $\mathsf{P}' \in \mathsf{L}'_k$ with $l' \in \mathsf{P}'$;
- there is $\mathsf{P} \in \mathsf{S}$ with $l \in \pi_k(\mathsf{P})$ if and only if there is $\mathsf{P}' \in \mathsf{S}'$ with $l' \in \pi_k(\mathsf{P}')$;
- $\min(\mathcal{L}(\alpha_k), \mathrm{card}(\mathsf{L}_k)) = \min(\mathcal{L}(\alpha_k), \mathrm{card}(\mathsf{L}'_k))$;
- $\min(\mathcal{S}(\alpha_k), \mathrm{card}(\bigcup_{\mathsf{P}\in\mathsf{S}} \pi_k(\mathsf{P}))) = \min(\mathcal{S}(\alpha_k), \mathrm{card}(\bigcup_{\mathsf{P}'\in\mathsf{S}'} \pi_k(\mathsf{P}')))$.

Recall that, by definition of the upper-bounds, $\mathcal{L}(\alpha) \geq \mathcal{L}(\alpha_1) + \mathcal{L}(\alpha_2) + \mathcal{S}(\max(\alpha_1, \alpha_2)) + 1$. Moreover, note that from $\alpha = \alpha_1 + \alpha_2$ with $\alpha_1, \alpha_2 \in \mathbb{N}^+$ we obtain that $\alpha \geq 2$ and every set in $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha)$ and $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ contains at least three locations. The construction goes by cases depending on whether or not the cardinalities of L_1, L_2 and S are less than $\mathcal{L}(\alpha_1)$, $\mathcal{L}(\alpha_2)$ and $\mathcal{S}(\max(\alpha_1, \alpha_2))$ respectively. For each case we also need to deal with whether or not l belongs to a set in $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha)$. When we define a set among L'_1, L'_2 and S' we implicitly assume that their locations are assigned to h'_1 and h'_2 accordingly to the definitions of these sets (i.e. L'_1 and $\pi_1(\mathsf{S}')$ only contain locations of h'_1 whereas L'_2 and $\pi_2(\mathsf{S}')$ only contain locations of h'_2).

1. **Case:** $\mathrm{card}(\mathsf{L}_1) < \mathcal{L}(\alpha_1)$, $\mathrm{card}(\mathsf{L}_2) < \mathcal{L}(\alpha_2)$ and $\mathrm{card}(\mathsf{S}) < \mathcal{S}(\max(\alpha_1, \alpha_2))$.
   By hypothesis **H1** and the definition of $\mathcal{L}(\alpha)$, it holds that
   $$\mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha)) = \mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)).$$
   - Select $\mathrm{card}(\mathsf{L}_1)$ sets from $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$. If l appears in one of the sets of L_1, then by hypothesis **H2** l' belongs to one the sets of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ and we require that set to be among the selected ones. These sets constitute L'_1.
   - Select $\mathrm{card}(\mathsf{L}_2)$ sets from $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus \mathsf{L}'_1$. If l appears in one of the sets of L_2, then by hypothesis **H2** l' belongs to one the sets of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ and we require that set to be among the selected ones. These sets constitute L'_2.
   - By $\mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha)) = \mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha))$ it follows that
     $$\mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus (\mathsf{L}'_1 \cup \mathsf{L}'_2)) = \mathrm{card}(\mathsf{S}).$$
     Let $\mathfrak{f}$ be an injection between S and $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus (\mathsf{L}'_1 \cup \mathsf{L}'_2)$ such that if there is $\mathsf{P} \in \mathsf{S}$ with $l \in \mathsf{P}$ then $l' \in \mathfrak{f}(\mathsf{P})$. This constraint on $\mathfrak{f}$ can be always satisfied thanks to **H2**. For each $\mathsf{P} \in \mathsf{S}$, divide $\mathfrak{f}(\mathsf{P})$ into two sets P'_1 and P'_2, as follows:
     - if $\mathrm{card}(\pi_1(\mathsf{P})) < \mathcal{S}(\alpha_1)$ then select $\mathrm{card}(\pi_1(\mathsf{P}))$ locations from $\mathfrak{f}(\mathsf{P})$. If $l \in \pi_1(\mathsf{P})$ then l' is one of the selected locations. These locations form P'_1. Then, $\mathsf{P}'_2 \stackrel{\text{def}}{=} \mathfrak{f}(\mathsf{P}) \setminus \mathsf{P}'_1$;
     - else if $\mathrm{card}(\pi_2(\mathsf{P})) < \mathcal{S}(\alpha_2)$ then select $\mathrm{card}(\pi_2(\mathsf{P}))$ locations from $\mathfrak{f}(\mathsf{P})$. If $l \in \pi_2(\mathsf{P})$ then l' is one of the selected locations. These locations form P'_2. Then, $\mathsf{P}'_1 \stackrel{\text{def}}{=} \mathfrak{f}(\mathsf{P}) \setminus \mathsf{P}'_2$;
     - otherwise $\mathrm{card}(\pi_1(\mathsf{P})) \geq \mathcal{S}(\alpha_1)$ and $\mathrm{card}(\pi_2(\mathsf{P})) \geq \mathcal{S}(\alpha_2)$. Select $\mathcal{S}(\alpha_1)$ locations from $\mathfrak{f}(\mathsf{P})$. If $l \in \pi_1(\mathsf{P})$ then l' is one of the selected locations. These locations form P'_1. Then, $\mathsf{P}'_2 \stackrel{\text{def}}{=} \mathfrak{f}(\mathsf{P}) \setminus \mathsf{P}'_1$.

     Iteratively add $(\mathsf{P}'_1, \mathsf{P}'_2)$ to S'.
2. **Case:** $\mathrm{card}(\mathsf{L}_1) < \mathcal{L}(\alpha_1)$, $\mathrm{card}(\mathsf{L}_2) < \mathcal{L}(\alpha_2)$ and $\mathrm{card}(\mathsf{S}) \geq \mathcal{S}(\max(\alpha_1, \alpha_2))$.
   - Select $\mathrm{card}(\mathsf{L}_1)$ sets from $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$. If l appears in one of the sets of L_1, then by hypothesis **H2** l' belongs to one the sets of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ and we require that set to be among the selected ones. These sets constitute L'_1.
   - Select $\mathrm{card}(\mathsf{L}_2)$ sets from $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus \mathsf{L}'_1$. If l appears in one of the sets of L_2, then by hypothesis **H2** l' belongs to one the sets of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ and we require that set to be among the selected ones. These sets constitute L'_2.
   - By **H1**, $\mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus (\mathsf{L}'_1 \cup \mathsf{L}'_2)) \geq \mathcal{S}(\max(\alpha_1, \alpha_2))$. For each set $\{\ell_1, \ell_2, \cdots, \ell_n\}$ of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus (\mathsf{L}'_1 \cup \mathsf{L}'_2)$, iteratively add $(\{\ell_1\}, \{\ell_2, \cdots, \ell_n\})$ to S' while being careful

that if l' is in the set then $l' = \ell_1$ if and only if there is $\mathsf{P} \in \mathsf{S}$ such that $l \in \pi_1(\mathsf{P})$. This constraint is always satisfiable thanks to **H2**. Notice how, after this step, S' has at least $\mathcal{S}(\max(\alpha_1, \alpha_2))$ elements.

3. **Case:** $\mathrm{card}(\mathsf{L}_1) < \mathcal{L}(\alpha_1)$, $\mathrm{card}(\mathsf{L}_2) \geq \mathcal{L}(\alpha_2)$ and $\mathrm{card}(\mathsf{S}) < \mathcal{S}(\max(\alpha_1, \alpha_2))$.
   - Select $\mathrm{card}(\mathsf{L}_1)$ sets from $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$. If l appears in one of the sets of L_1, then by hypothesis **H2** l' belongs to one the sets of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ and we require that set to be among the selected ones. These sets constitute L'_1.
   - Let $\mathfrak{f}$ be an injection between S and $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus \mathsf{L}'_1$ such that if there is $\mathsf{P} \in \mathsf{S}$ with $l \in \mathsf{P}$ then $l' \in \mathfrak{f}(\mathsf{P})$. This constraint on $\mathfrak{f}$ can be always satisfied thanks to **H2**. For each $\mathsf{P} \in \mathsf{S}$, divide $\mathfrak{f}(\mathsf{P})$ into two sets P'_1 and P'_2 as shown in the third step of the 1st case of the construction. Iteratively add $(\mathsf{P}'_1, \mathsf{P}'_2)$ to S'.
   - The set $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus (\mathsf{L}'_1 \cup \mathsf{S}')$ constitutes L'_2.
4. **Case:** $\mathrm{card}(\mathsf{L}_1) < \mathcal{L}(\alpha_1)$, $\mathrm{card}(\mathsf{L}_2) \geq \mathcal{L}(\alpha_2)$ and $\mathrm{card}(\mathsf{S}) \geq \mathcal{S}(\max(\alpha_1, \alpha_2))$.
   - Select $\mathrm{card}(\mathsf{L}_1)$ sets from $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$. If l appears in one of the sets of L_1, then by hypothesis **H2** l' belongs to one the sets of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ and we require that set to be among the selected ones. These sets constitute L'_1.
   - Select $\mathcal{L}(\alpha_2)$ sets from $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus \mathsf{L}'_1$. If l appears in one of the sets of L_2, then by hypothesis **H2** l' belongs to one the sets of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ and we require that set to be among the selected ones. These sets constitute L'_2.
   - By **H1**, $\mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus (\mathsf{L}'_1 \cup \mathsf{L}'_2)) \geq \mathcal{S}(\max(\alpha_1, \alpha_2))$. For each set $\{\ell_1, \ell_2, \cdots, \ell_n\}$ of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus (\mathsf{L}'_1 \cup \mathsf{L}'_2)$, iteratively add $(\{\ell_1\}, \{\ell_2, \cdots, \ell_n\})$ to S' while being careful that if l' is in the set then $l' = \ell_1$ if and only if there is $\mathsf{P} \in \mathsf{S}$ such that $l \in \pi_1(\mathsf{P})$. This constraint is always satisfiable thanks to **H2**. Notice how, after this step, S' has at least $\mathcal{S}(\max(\alpha_1, \alpha_2))$ elements.
5. **Case:** $\mathrm{card}(\mathsf{L}_1) \geq \mathcal{L}(\alpha_1)$, $\mathrm{card}(\mathsf{L}_2) < \mathcal{L}(\alpha_2)$ and $\mathrm{card}(\mathsf{S}) < \mathcal{S}(\max(\alpha_1, \alpha_2))$. Symmetrical to the 3rd case with respect to the indexes 1 and 2.
6. **Case:** $\mathrm{card}(\mathsf{L}_1) \geq \mathcal{L}(\alpha_1)$, $\mathrm{card}(\mathsf{L}_2) < \mathcal{L}(\alpha_2)$ and $\mathrm{card}(\mathsf{S}) \geq \mathcal{S}(\max(\alpha_1, \alpha_2))$. Symmetrical to the 4rd case with respect to the indexes 1 and 2.
7. **Case:** $\mathrm{card}(\mathsf{L}_1) \geq \mathcal{L}(\alpha_1)$, $\mathrm{card}(\mathsf{L}_2) \geq \mathcal{L}(\alpha_2)$ and $\mathrm{card}(\mathsf{S}) < \mathcal{S}(\max(\alpha_1, \alpha_2))$.
   - Select $\mathcal{L}(\alpha_1)$ sets from $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$. If l appears in one of the sets of L_1, then by hypothesis **H2** l' belongs to one the sets of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ and we require that set to be among the selected ones. These sets constitute L'_1.
   - Let $\mathfrak{f}$ be an injection between S and $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus \mathsf{L}'_1$ such that if there is $\mathsf{P} \in \mathsf{S}$ with $l \in \mathsf{P}$ then $l' \in \mathfrak{f}(\mathsf{P})$. This constraint on $\mathfrak{f}$ can be always satisfied thanks to **H2**. For each $\mathsf{P} \in \mathsf{S}$, divide $\mathfrak{f}(\mathsf{P})$ into two sets P'_1 and P'_2 as shown in the third step of the 1st case of the construction. Iteratively add $(\mathsf{P}'_1, \mathsf{P}'_2)$ to S'.
   - The set $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus (\mathsf{L}'_1 \cup \mathsf{S}')$ constitutes L'_2.
8. **Case:** $\mathrm{card}(\mathsf{L}_1) \geq \mathcal{L}(\alpha_1)$, $\mathrm{card}(\mathsf{L}_2) \geq \mathcal{L}(\alpha_2)$ and $\mathrm{card}(\mathsf{S}) \geq \mathcal{S}(\max(\alpha_1, \alpha_2))$.
   - Select $\mathcal{L}(\alpha_1)$ sets from $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$. If l appears in one of the sets of L_1, then by hypothesis **H2** l' belongs to one the sets of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ and we require that set to be among the selected ones. These sets constitute L'_1.
   - Select $\mathcal{L}(\alpha_2)$ sets from $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus \mathsf{L}'_1$. If l appears in one of the sets of L_2, then by hypothesis **H2** l' belongs to one the sets of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ and we require that set to be among the selected ones. These sets constitute L'_2.
   - By **H1**, $\mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus (\mathsf{L}'_1 \cup \mathsf{L}'_2)) \geq \mathcal{S}(\max(\alpha_1, \alpha_2))$. For each set $\{\ell_1, \ell_2, \cdots, \ell_n\}$ of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus (\mathsf{L}'_1 \cup \mathsf{L}'_2)$, iteratively add $(\{\ell_1\}, \{\ell_2, \cdots, \ell_n\})$ to S' while being careful that if l' is in the set then $l' = \ell_1$ if and only if there is $\mathsf{P} \in \mathsf{S}$ such that $l \in \pi_1(\mathsf{P})$.

This constraint is always satisfiable thanks to **H2**. Notice how, after this step, S' has at least $\mathcal{S}(\max(\alpha_1, \alpha_2))$ elements.

It is easy to see that all the properties are always satisfied for any of the cases above. In particular, for the cases where $\mathrm{card}(\mathsf{S}) \geq \mathcal{S}(\max(\alpha_1, \alpha_2))$, i.e. cases 2, 4, 6 and 8, the property

$$\min(\mathcal{S}(\alpha_k), \mathrm{card}(\textstyle\bigcup_{\mathsf{P}\in\mathsf{S}} \pi_k(\mathsf{P}))) = \min(\mathcal{S}(\alpha_k), \mathrm{card}(\textstyle\bigcup_{\mathsf{P}'\in\mathsf{S}'} \pi_k(\mathsf{P}')))$$

is implied by $\mathrm{card}(\mathsf{S}) \geq \mathcal{S}(\max(\alpha_1, \alpha_2))$ and $\mathrm{card}(\mathsf{S}') \geq \mathcal{S}(\max(\alpha_1, \alpha_2))$. To prove that this property holds also for the cases where $\mathrm{card}(\mathsf{S}) < \mathcal{S}(\max(\alpha_1, \alpha_2))$, i.e. cases 1, 3, 5 and 7, we use the fact that, for every $\mathsf{P} \in \mathsf{S}$ and $k \in \{1, 2\}$, $\pi_k(\mathsf{P}) < \mathcal{S}(\alpha_k)$ implies $\pi_{3-k}(\mathsf{P}) \geq \mathcal{S}(\alpha_{3-k})$. Indeed, this holds as all the loops in $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha)$ have size greater than $\alpha = \mathcal{S}(\alpha) = \mathcal{S}(\alpha_1) + \mathcal{S}(\alpha_2)$.

Bundled Fragments of First-Order Modal Logic: (Un)Decidability

Anantha Padmanabha
Institute of Mathematical Sciences, HBNI, Chennai, India
ananthap@imsc.res.in
https://orcid.org/0000-0002-4265-5772

R Ramanujam
Institute of Mathematical Sciences, HBNI, Chennai, India
jam@imsc.res.in

Yanjing Wang
Department of Philosophy, Peking University, Beijing, China
y.wang@pku.edu.cn

Abstract

Quantified modal logic is notorious for being undecidable, with very few known decidable fragments such as the monodic ones. For instance, even the two-variable fragment over unary predicates is undecidable. In this paper, we study a particular fragment, namely the *bundled* fragment, where a first-order quantifier is always followed by a modality when occurring in the formula, inspired by the proposal of [15] in the context of non-standard epistemic logics of know-what, know-how, know-why, and so on.

As always with quantified modal logics, it makes a significant difference whether the domain stays the same across possible worlds. In particular, we show that the predicate logic with the bundle $\forall\Box$ alone is undecidable over constant domain interpretations, even with only monadic predicates, whereas having the $\exists\Box$ bundle instead gives us a decidable logic. On the other hand, over increasing domain interpretations, we get decidability with both $\forall\Box$ and $\exists\Box$ bundles with unrestricted predicates, where we obtain tableau based procedures that run in PSPACE. We further show that the $\exists\Box$ bundle cannot distinguish between constant domain and variable domain interpretations.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics

Keywords and phrases First-order modal logic, decidability, bundled fragments

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.43

Acknowledgements We thank the reviewers for the insightful and helpful comments.

1 Introduction

Propositional modal logics have been extensively used to reason about labelled transition systems in computer science. These have led to the advent of temporal logics which have been very successful in the formal specification and verification of a wide range of systems. While these have been best used in the context of finite state reactive systems (over infinite behaviours), in the last couple of decades, such logics have been developed for infinite state systems as well ([1]). Finding decidable logics with reasonable complexity over infinite state systems continues to be a challenge.

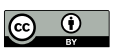


Table 1 x, y refers to the two-variable fragment, P^1 refers to unary predicates. GF is the guarded fragment. $\Box_i$ is multi-modal logic. $\Box_i(x)$ refers to having only 1 free variable inside the scope of modalities (monodic fragment). p refers to propositions and / indicates having one of them.

Language	Decidability	Ref
P^1	undecidable	[10]
x, y, p, P^1	undecidable	[9, 6]
x, y, $\Box_i$, single P^1	undecidable	[11]
single x	decidable	[12, 4]
$x, y/P^1$/GF, $\Box_i(x)$	decidable	[17]

A natural candidate to describe systems with unbounded data is First Order Logic (FO), and it has been extensively used not only in reasoning about mathematical structures, but also about databases and knowledge representation systems. When we wish to describe data updates in such systems, we have labelled transition systems where each state carries information on data, thus making them infinite state systems. It is then easy to specify transitional properties of such systems in First Order Modal Logic (FOML).

However FOML is infamously hard to handle technically: usually you lose good properties of first-order logic and modal propositional logic when putting them together. (FOML is also the theatre in which numerous philosophical controversies have been played out.) On the one hand, the decidable fragments of first-order logic have been well mapped out during the last few decades ([3]). On the other hand, we have a thorough understanding of the robust decidability of propositional modal logics [13]. However, when it comes to finding decidable fragments of FOML, the situation seems quite hopeless: even the *two-variable fragment with one single monadic predicate* is (robustly) undecidable over almost all useful model classes [11].

On the positive side, certain guarded fragments of FOML that are decidable [17]. One promising approach has come from the study of the so-called *monodic fragment*, which requires that there be at most one free variable in the scope of any modal subformula. Combining the monodic restriction with a decidable fragment of FO we often obtain decidable fragments of FOML[17, 2], as Table 1 shows.

The reason behind this sad tale is not far to seek: the addition of $\Box$ gives implicitly an extra quantifier, over a fresh variable. Thus if we consider the two-variable fragment of FOML, with only unary predicates in the syntax, we can use $\Box$ to code up binary relations and we ride out of the two-variable fragment as well as the monadic fragment of FO. The monodic fragment restricts the use of free variables inside the scope of $\Box$ significantly to get decidability.

It is then natural to ask: apart from variable restrictions, is there some other way to obtain syntactic fragments of FOML that are yet decidable?

One answer came, perhaps surprisingly, from epistemic logic. In recent years, interest has grown in studying epistemic logics of knowing-how, knowing-why, knowing-what, and so on (see [16] for a survey). As observed there most of the new epistemic operators essentially share a unified *de re* semantic schema of $\exists x\Box$ where $\Box$ is an epistemic modality ($\mathsf{B}^{\exists\Box}$-FOML). For instance, $\exists x\Box\varphi$ may mean that there exists a mechanism which you know such that executing it will make sure that you end in a φ state [14]. Such reasoning leads to the proposal in [15] of a new fragment of FOML by packing $\exists$ and $\Box$ into a *bundle* modality, but without any restriction on predicates or the occurrences of variables. Formally, $\mathsf{B}^{\exists\Box}$-FOML fragment is given by the syntax:

$$\varphi ::= P\overline{x} \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \exists x\Box\varphi$$

Note that in this language, quantifiers have to always come with modalities. Such a language may seem weak but it already suffices to say many interesting things.

The following examples describe a database model that comes with a binary relation $R(x, y)$ to mean that x "is dominated by" y, and the states describe the possible updates of the database before/after updates.

- $\exists x\Box\ \neg\exists y\Box\ (R(x, y))$: There is a king element such that after any update, no element is sure to dominate it later.
- $\forall x\Diamond\ \exists y\Box\ (R(x, y))$: Every element can be updated in such a way that another can necessarily dominate it ($\forall x\Diamond$ is the dual of $\exists x\Box$).
- $\exists x\Box\ \big(\exists y\Box\ (R(x, y) \wedge \exists z\Box\ (R(x, z) \wedge R(y, z)))\big)$: There is an element x that is dominated by some element y after any possible update and further, there exists z which will always dominate both x and y in any possible update from there on.

It may be noted that the domain does not need to be fixed uniformly at all states to interpret these formulas. For instance, when we consider the first formula above, we refer to some "active" element x present at the current state. It is necessary that x continues to be active at successor states where we compare it against other elements, but there could be new elements in the successor states. When we define the formal semantics in the next section, it will be clear that these (and other similar) formulas may be interpreted over constant domain or increasing domain models uniformly.

It is shown [15] that the $\mathsf{B}^{\exists\Box}$-FOML fragment with arbitrary predicates is in fact PSPACE-complete over FOML. Essentially, the idea is based on the 'secret of success' of modal logic: *guard the quantifiers*, now with a modality. On the other hand, the same fragment is undecidable over equivalence models, and this can be shown by coding first-order sentences in this language using the symmetry property of the accessibility relation.

There are curious features to observe in this tale of (partial) success. The fragment in [15] includes the $\exists\Box$ bundle but not its companion $\forall\Box$ bundle, and considers *only* increasing domain models. The latter observation is particularly interesting when we notice that equivalence models, where the fragment becomes undecidable, force constant domain semantics.

The last distinction is familiar to first-order modal logicians, but might come across as a big fuss to others. Since FOML extends FO, the models contain a first order structure at each state. Then it makes a significant difference whether we work with a single data domain fixed for the entire model, or whether this can vary across transitions (updates). In the latter case, each possible world has its own domain, and quantification extends only over objects that exist in the current world.

Given such subtlety, it is instructive to consider more general bundled fragments of FOML, including both $\exists\Box$ and $\forall\Box$ as the natural first step, and study them over constant domain as well as varying domain models. This is precisely the project undertaken in this paper, and the results are summarized in Table 2. In this paper, the only varying domains we consider are *increasing* ones, whereby the data domain may change across a transition but can only increase monotonically.

As we can see, the $\exists\Box$ bundle behaves better computationally than the $\forall\Box$ bundle. For $\forall\Box$, even the monadic fragment is undecidable over constant domain models: we can encode in this language, qua satisfiability, any first-order logic sentence with binary predicates by exploiting the power of $\forall\Box$.

On the other hand, we can actually give a tableau method for the $\exists\Box$ and $\forall\Box$ fragment together, similar to the tableau in [15], for increasing domain models. The crucial observation

Table 2 Satisfiability problem classification for Bundled FOML fragment, P refers to predicates of arbitrary arity and P^1 refers to monadic predicates. Models are either constant domain or increasing domain.

Language	Model	Decidability	Remark
$\forall\Box$, P^1	Constant	undecidable	
$\exists\Box$, P	Constant	decidable	PSPACE-complete
$\exists\Box, \forall\Box$, P	Increasing	decidable	PSPACE-complete

is that such models allow us to manufacture new witnesses for $\exists x\Box$ and $\exists x\Diamond$ formulas on the fly, giving considerable freedom in model construction, which is not available in constant domain models.

Indeed, the well-behavedness of the $\exists\Box$ bundle is further attested to by the fact that it is decidable over constant domain models as well. So constant domain is not the culprit for undecidability of this fragment over equivalence models. In fact, we show that the $\exists\Box$ bundle cannot distinguish increasing domain models and constant domain models.

The paper is structured as follows. After formal definitions of bundled fragments, we present undecidability results for unary predicate $\forall\Box$ fragment over FOML with constant domain semantics and then move on to tableaux procedures for the decidable fragments. We then show that the validities of $\exists\Box$ over increasing domain are exactly the same as its validities over constant domain models, and end the paper with another look at mapping the terrain of these fragments.

2 Bundled fragment of First order modal logic

Let Var be a countable set of variables, and $\mathbf{P}$ be a countable set of predicate symbols, with $\mathbf{P}_n \subseteq \mathbf{P}$ denoting the set of all predicate symbols of arity n. We use $\overline{x}$ to denote a finite sequence of variables in Var. We only consider the "pure" first order unimodal logic: that is, the vocabulary is restricted to Var (no equality and no constants and no function symbols).

▶ **Definition 1.** Given Var and $\mathbf{P}$, the bundled fragment of FOML denoted by B-FOML is defined as follows:

$$\varphi ::= P\overline{x} \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \exists x\Box\varphi \mid \forall x\Box\varphi$$

where $x \in$ Var, $P \in \mathbf{P}$. We denote the fragment $\mathsf{B}^{\exists\Box}$-FOML to be the formulas which contains only $\exists\Box$(and its dual $\forall\Diamond$) formulas and $\mathsf{B}^{\forall\Box}$-FOML which contains only $\forall\Box$ (and its dual $\exists\Diamond$) formulas.

$\top, \bot, \vee, \rightarrow$ are defined in the standard way. $\forall x\Diamond\varphi = \neg\exists x\Box\neg\varphi$ is the dual of $\exists x\Box\varphi$, and $\exists x\Diamond\varphi = \neg\forall x\Box\neg\varphi$ is the dual of $\forall x\Box\varphi$. With both bundles we can say, that every element is guaranteed an update such that some element is updatable to dominate it: $\forall x\Box\ \exists y\Box\ (R(x, y))$.

The *free* and *bound* occurrences of variables are defined as in first-order logic, by viewing $\exists x\Box$ and $\exists x\Diamond$ as quantifiers. We denote $\mathsf{Fv}(\varphi)$ as the set of free variables of φ. We write $\varphi(\overline{x})$ if all the free variables in φ are included in $\overline{x}$. Given a B-FOML formula φ and $x, y \in$ Var, we write $\varphi[y/x]$ for the formula obtained by replacing every free occurrence of x by y. A formula is said to be a *sentence* if it contains no free variables. As we will see later, $\exists x\Box\varphi$ is equivalent to $\Box\varphi$ if x is not free in φ. Therefore B-FOML is indeed an extension of modal logic.

The semantics presented below is the standard *increasing domain* semantics of FOML.

▶ **Definition 2.** An (increasing domain) model $\mathcal{M}$ for B-FOML is a tuple (W, D, δ, R, ρ) where, W is a non-empty set of worlds, D is a non-empty domain, $R \subseteq (W \times W)$, $\delta : W \to 2^D$ assigns to each $w \in W$ a *non-empty* local domain s.t. wRv implies $\delta(w) \subseteq \delta(v)$ for any $w, v \in W$, and [1] $\rho : (W \times \mathbf{P}) \to \bigcup_{n\in\omega} 2^{D^n}$ such that ρ assigns to each n-ary predicate on each world an n-ary relation on D.

Given a model $\mathcal{M}$, we use $W^{\mathcal{M}}, D^{\mathcal{M}}, \delta^{\mathcal{M}}, \rho^{\mathcal{M}}$ to denote its corresponding components. We often write D_w for $\delta^{\mathcal{M}}(w)$. A *constant domain* model is one where $D_w = D^{\mathcal{M}}$ for any $w \in W^{\mathcal{M}}$. Note that constant domain models are special cases of increasing domain models. A *finite model* is one with both $W^{\mathcal{M}}$ finite and $D^{\mathcal{M}}$ finite.

▶ **Definition 3.** Consider a model $\mathcal{M} = (W, D, \delta, R, \rho)$, $w \in W$. To interpret free variables, we also need a variable assignment $\sigma : \mathsf{Var} \to D$. Given $\mathcal{M} = (W, D, \delta, R, \rho)$, $w \in W$, and an assignment σ, define $\mathcal{M}, w, \sigma \vDash \varphi$ inductively as follows:

$\mathcal{M}, w, \sigma \vDash P(x_1 \cdots x_n)$	$\Leftrightarrow$	$(\sigma(x_1), \cdots, \sigma(x_n)) \in \rho(P, w)$
$\mathcal{M}, w, \sigma \vDash \neg\varphi$	$\Leftrightarrow$	$\mathcal{M}, w, \sigma \nvDash \varphi$
$\mathcal{M}, w, \sigma \vDash (\varphi \wedge \psi)$	$\Leftrightarrow$	$\mathcal{M}, w, \sigma \vDash \varphi$ and $\mathcal{M}, w, \sigma \vDash \psi$
$\mathcal{M}, w, \sigma \vDash \exists x \Box\varphi$	$\Leftrightarrow$	there is some $d \in \delta(w)$ such that $\mathcal{M}, v, \sigma[x \mapsto d] \vDash \varphi$ for all v s.t. wRv
$\mathcal{M}, w, \sigma \vDash \exists x \Diamond\varphi$	$\Leftrightarrow$	there is some $d \in \delta(w)$ and some $v \in W$ such that wRv and $\mathcal{M}, v, \sigma[x \mapsto d] \vDash \varphi$

where $\sigma[x \mapsto d]$ denotes an assignment that is the same as σ except for mapping x to d.

Note that the standard $\Box\alpha(\Diamond\alpha)$ of FOML can be expressed in this logic as $\exists x \Box\alpha(\exists x \Diamond\alpha)$ where x does not occur in α.

In general, when considering the truth of φ in a model, it suffices to consider $\sigma : \mathsf{Fv}(\varphi) \to D$, assignment restricted to the free variables occurring free in φ. When $\mathsf{Fv}(\varphi) = \{x_1, \ldots, x_n\}$ and $\{d_1, \ldots, d_n\} \subseteq D$, We write $\mathcal{M}, w \vDash \varphi[\overline{d}]$ to denote $\mathcal{M}, w, \sigma \vDash \varphi(\overline{x})$ for any σ such that for all $i \leq n$ we have $\sigma(x_i) = d_i$. Finally, when φ is a sentence, we can simply write $\mathcal{M}, w \models \varphi$.

Call σ *relevant* at $w \in W$ if $\sigma(x) \in \delta^{\mathcal{M}}(w)$ for all $x \in \mathsf{Var}$. The increasing domain condition ensures that whenever σ is relevant at w and we have wRv, then σ is relevant at v as well. (In a constant domain model, every assignment σ is relevant at all the worlds.) We say φ is *valid*, if φ is true on any $\mathcal{M}, w$ w.r.t. any σ relevant at w. φ is *satisfiable* if $\neg\varphi$ is not valid. [2]

3 Undecidability results

In this section we prove that the satisfiability problem for the $\mathsf{B}^{\forall\Box}$-FOML fragment over the class of constant domain models is undecidable even when the atomic predicates are restricted to be unary.

Kripke[10] showed that full FOML with constant domain semantics is undecidable even when the atomic predicates are only unary. Gabbay and Shehtman [6] showed that 2-variable Monadic FOML with propositions is undecidable. Kontchakov et al [9] showed that

[1] Note that we do not impose the restriction $\rho(w, P) \subseteq [\delta(w)]^n$ where arity of P is n, since it is not needed for our technical development. For more details about this relaxation, refer Hughes and Creswell [8].

[2] Note that the classical first-order principle *dictum de omne*: $\forall x\psi \to \psi[y/x]$ is not expressible in our language, but validity over relevant assignments gives us classical expressible analogues.

propositions can be eliminated. We take another step in this journey. We show that over constant domain models, the satisfiability problem for $\mathsf{B}^{\forall\Box}$-FOML fragment is undecidable over unary predicates.

Consider $\mathsf{FO}(R)$, the first order logic with only variables as terms and no equality, and the single binary predicate R. We know that $\mathsf{FO}(R)$ satisfiability problem is undecidable [7]. To translate $\mathsf{FO}(R)$ sentences to $\mathsf{B}^{\forall\Box}$-FOML formulas, we use two unary predicate symbols P, Q in the latter. The main idea is that the atomic formula $R(x, y)$ is coded up as the $\mathsf{B}^{\forall\Box}$-FOML formula $\exists z\Diamond\big(P(x) \wedge Q(y)\big)$, where z is a new variable, distinct from x and y.[3]

For any *quantifier-free* $\mathsf{FO}(R)$ formula β, we define the translation of β to $\mathsf{B}^{\forall\Box}$-FOML formula φ_β inductively as follows.

- $Tr(R(x, y)) := \exists z\Diamond\big(P(x) \wedge Q(y)\big)$, where z is distinct from x and y.
- $Tr(\neg\beta) := \neg Tr(\beta)$ and $Tr(\beta_1 \wedge \beta_2) := Tr(\beta_1) \wedge Tr(\beta_2)$.

Note that a quantifier-free FO formula is translated to a $\mathsf{B}^{\forall\Box}$-FOMLformula with modal (quantifier) depth 1. Now consider an $\mathsf{FO}(R)$ sentence α (having no free variables) presented in prenex form: $Q_1x_1\ Q_2x_2 \cdots Q_nx_n(\beta)$ where β is quantifier-free. We define

$$\psi_\alpha := Q_1x_1\Delta_1\ Q_2x_2\Delta_2\ \cdots Q_nx_n\Delta_n\ (Tr(\beta))$$

where $Q_ix_i\Delta_i := \exists x_i\Diamond$ if $Q_i = \exists$ and $Q_ix_i\Delta_i := \forall x_i\Box$ if $Q_i = \forall$.

We claim that satisfiability is preserved over this translation with a few additional formulas. Ideally, we want α to be satisfiable iff ψ_α is satisfiable. However, the translated formula might be satisfiable simply because some $Q_i := \forall$ and there are no successors for the worlds at depth i and thus the corresponding subformula translation $\forall x_i\Box\psi'$ trivially holds. To avoid this, we use formula λ_n which ensures that for all $i \leq n$ and every world at depth i, there is at least one successor: $\lambda_n := \bigwedge_{j=0}^{n} (\forall z\Box)^j(\exists z\Diamond\top)$

Finally to ensure that $\exists z\Diamond(P(x) \wedge Q(y))$ is evaluated uniformly at the "tail" worlds, we have: $\gamma_n := \forall z_1\Box\ \forall z_2\Box\ \big((\exists z\Diamond)^n\ (\exists z\Diamond\ (P(z_1) \wedge Q(z_2)) \rightarrow (\forall z\Box)^n(\exists z\Diamond(P(z_1) \wedge Q(z_2)))\big)$ where z_1, z_2 and z do not appear in α.

Suppose $\mathcal{M}, u \models \gamma_n$ then notice that for any world w at a path length 2 from u, if there is one world at distance n starting from w where $\Diamond(P(z_1) \wedge Q(z_2))$ holds, then $\Diamond(P(z_1) \wedge Q(z_2))$ holds at all worlds at a distance n starting from w. Notice that we use two dummy variables z_1 and z_2 in γ_n. Hence to match the modal depths of the translated formulas, we need to append two $\Box's$ to ψ_α and we need to use λ_{n+2} instead of λ_n. Thus, the complete translation is given by:

▶ **Definition 4.** Given a $\mathsf{FO}(R)$ sentence $\alpha := Q_1x_1Q_2x_2 \cdots Q_nx_n\beta$ in prenex normal form, the translated $\mathsf{B}^{\forall\Box}$-FOML formula φ_α is given by: $\varphi_\alpha := (\forall z\Box)^2(\psi_\alpha) \wedge \lambda_{n+2} \wedge \gamma_n$ where z does not occur in α.

Note that for any $\mathsf{FO}(R)$ sentence α of quantifier depth n, we get a translated formula φ_α of modal (quantifier) depth $n + 3$.

Before stating the theorem, we define some useful notation.

▶ **Definition 5.** For any $\mathsf{FO}(R)$ sentence $\alpha := Q_1x_1Q_2x_2 \cdots Q_nx_n\beta$ in the prenex normal form with β being quantifier-free, we define the following:

[3] This is similar to the approach used by Kripke [10], specialized to the $\mathsf{B}^{\forall\Box}$-FOML fragment.

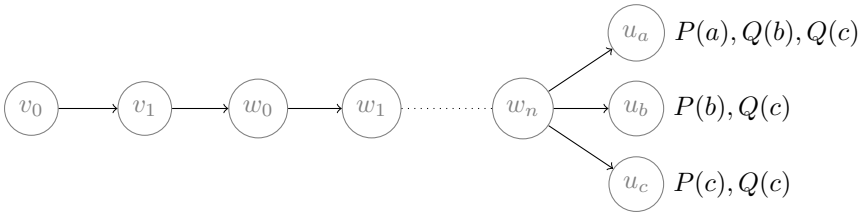


Figure 1 The translated model for (D, I) where $D = \{a, b, c\}$ and $I = \{(a, b),\ (a, c),\ (b, c),\ (c, c)\}$. For any sentence $\alpha \in FO(R)$ of quantifier depth n, $(D, I) \models \alpha$ iff $M, v_0 \models \varphi_\alpha$.

- For all $1 \leq i \leq n$ let $x_1 \cdots x_i$ be denoted by $\overline{x^i}$ and the vector $[d_1, d_2 \cdots d_i]$ be denoted by $\overline{d^i}$ where every $d_j \in D$.
- Let $[\overline{x^i} \mapsto \overline{d^i}]$ denote the interpretation where $\sigma(x_j) = d_j$.
- For $0 \leq i < n$, let $\alpha[i] = Q_{i+1}x_{i+1} \cdots Q_n x_n \beta$ and $\psi_\alpha[i] = Q_{i+1}x_{i+1}\Delta_{i+1} \cdots Q_n x_n \Delta_n(\varphi_\beta)$ be the corresponding translated formula. Also, let $\alpha[n] = \beta$ and $\psi_\alpha[n] = Tr(\beta)$.

▶ **Theorem 6.** *For any* FO(R) *sentence* $\alpha := Q_1x_1Q_2x_2 \cdots Q_nx_n\beta$ *in prenex normal form,* α *is satisfiable iff* φ_α *is constant domain satisfiable.*

Proof. Let $\alpha := Q_1x_1 \cdots Q_nx_n\beta$, where β is quantifier-free. To prove ($\Rightarrow$), assume that α is satisfiable. Let D be some domain such that $(D, I) \models \alpha$ where $I \subseteq (D \times D)$ is the interpretation for R. We use the same D as the domain and construct a FOML model. Define $\mathcal{M} = (W, R, D, \delta, \rho)$ where:

$W = \{v_0, v_1\} \cup \{w_i \mid 0 \leq i \leq n\} \cup \{u_d \mid d \in D\}$.
$R = \{(v_0, v_1), (v_1, w_0)\} \cup \{(w_i, w_{i+1}) \mid 0 \leq i < n\} \cup \{(w_n, u_d) \mid u_d \in W\}$.
$\delta(u) = D$ for all $u \in W$.
For all $i \in \{0, 1\}$ and $0 \leq j \leq n$ and $v_i, w_j \in W$ define $\rho(v_i, P) = \rho(v_i, Q) = \rho(w_j, P) = \rho(w_j, Q) = \emptyset$ and for all $u_d \in W$, $\rho(u_d, P) = \{d\}$ and $\rho(u_d, Q) = \{c \mid (d, c) \in I\}$.

Note that $\mathcal{M}$ is a constant domain model. $\mathcal{M}$ is illustrated in Figure 1 for one such translation. Note that $\mathcal{M}$ has exactly one path of length $n + 2$ starting from v_0 which ends at w_n. Hence, $\mathcal{M}, v_1 \models \lambda_{n+2} \wedge \gamma_n$.

Finally, we claim that $\mathcal{M}, v_0 \models (\forall z \Box)^2 \psi_\alpha$ which completes the proof of the forward direction. Again, since $v_0 \to v_1 \to w_0$ is the only path of length 2 starting from v_0, it is enough to verify that $\mathcal{M}, w_0 \models \psi_\alpha$. We set up an induction to prove this.

Claim. For all $0 \leq i \leq n$, $w_i \in W$, for all vectors $\overline{d^i} \in D^i$ of length i, we have $D, I, [\overline{x^i} \mapsto \overline{d^i}] \models \alpha[i]$ iff $\mathcal{M}, w_i, [\overline{x^i} \mapsto \overline{d^i}] \models \psi_\alpha[i]$.

The proof is by reverse induction on i. The base case, when $i = n$, we have $\alpha[n] = \beta$. Now we induct on the structure of β, to prove the claim. In the base case we have $R(x_i, x_j)$. By definition of ρ, if $(a, b) \in I$ then $\mathcal{M}, u_a, [x_i \to a, x_j \to b] \models (P(x) \wedge Q(y))$ and hence $\mathcal{M}, w_n, [x_i \to a, x_j \to b] \models \exists z \Diamond (P(x_1) \wedge Q(x_2))$. On the other hand if $\mathcal{M}, w_n, [x_i \to a, x_j \to b] \models \exists z \Diamond (P(x_1) \wedge Q(x_2))$ then since $\mathcal{M}, u_a \not\models P(b)$ for all $b \neq a$, it has to be the case that $\mathcal{M}, u_a, [x_i \to a, x_j \to b] \models (P(x) \wedge Q(y))$ and thus $(a, b) \in I$. The $\neg$ and $\wedge$ cases are standard.

For the induction step, we need to consider formulas $\alpha[i-1]$ and $\psi_\alpha[i-1]$ and the world w_{i-1}. Now $\alpha[i-1]$ is either $\exists x_i \alpha[i]$ or $\forall x_i \alpha[i]$.

For the case when $\alpha[i-1]$ is $\exists x_i \alpha[i]$ the corresponding $\psi_\alpha[i-1]$ is $\exists x_i \Diamond (\psi_\alpha[i])$. We have $D, I, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}] \models \exists x_i \alpha[i]$ iff there is some $c \in D$ such that

$D, I, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}, x_i \to c] \models \alpha[i]$ iff (by induction hypothesis)
$\mathcal{M}, w_i, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}, x_i \to c] \models \psi_\alpha[i]$ iff
$\mathcal{M}, w_{i-1}, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}] \models \exists x_i \Diamond \psi_\alpha[i]$, as required.

For the case when $\alpha[i-1]$ is $\forall x_i \alpha[i]$, we have $\psi_\alpha[i-1] = \forall x_i \Box \psi_\alpha[i]$. Now,
$D, I, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}] \models \forall x_i \alpha[i]$ iff
for all $c \in D$ we have $D, I, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}, x_i \to c] \models \alpha[i]$
iff (by induction hypothesis) for all $c \in D$ we have
$\mathcal{M}, w_i, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}, x_i \to c] \models \psi_\alpha[i]$ iff
$\mathcal{M}, w_{i-1}, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}] \models \forall x_i \Box \psi_\alpha[i]$ (since w_i is the only successor of w_{i-1}).
This completes ($\Rightarrow$) since $(D, I) \models \alpha[0]$ and we have $\alpha[0] = \alpha$. Thus $\mathcal{M}, w_0 \models \psi_\alpha$.

To prove ($\Leftarrow$), suppose that φ_α is satisfiable, and let $\mathcal{M} = (W, D, R, \gamma, V)$ be a constant domain model such that $\mathcal{M}, r \models \psi_\alpha$. Note that since $\mathcal{M}, r \models \lambda_{n+2}$, every path starting from r has length at least $n+2$ and there is at least one such path.

Let w be any world at height 2. Since $\mathcal{M}, r \models \lambda_{n+2} \wedge (\forall z \Box)^2 \psi_\alpha$, there is at least one path of length n starting from w and also we have $\mathcal{M}, w \models \psi_\alpha$. Further since, $\mathcal{M}, r \models \gamma_n$, for any $c, d \in D$ we have $\mathcal{M}, w, [z_1 \to c, z_2 \to d] \models (\exists z \Diamond)^n (\exists z \Diamond (P(z_1) \wedge Q(z_2)) \to (\forall z \Box)^n (\exists z \Diamond (P(z_1) \wedge Q(z_2))$.

Define $I = \{(c, d) \mid c, d \in D$ and $\mathcal{M}, w, [x \to c, y \to d] \models (\forall z \Box)^n \exists z \Diamond (P(x) \wedge Q(y))\}$.

For $0 \leq i \leq n$ let W_i denote the set of all worlds at distance i from w with $W_0 = \{w\}$. The $\mathsf{FO}(R)$ model for α is given by $\mathcal{M}' = (D, I)$. We now claim that the formula α is satisfied in this model, which is proved by induction on $n - i$. Again, the relevant claim is as follows:

Claim. For all $0 \leq i \leq n$ and for all $d_1 \cdots d_i \in D$, we have:
(a) if there is some $v_i \in W_i$ such that $\mathcal{M}, v_i, [\overline{x^i} \mapsto \overline{d^i}] \models \psi_\alpha[i]$ then for all $u_i \in W_i$ we have $\mathcal{M}, u_i, [\overline{x^i} \mapsto \overline{d^i}] \models \psi_\alpha[i]$
(b) $D, I, [\overline{x^i} \mapsto \overline{d^i}] \models \alpha[i]$ iff for all $v_i \in W_i$, $\mathcal{M}, v_i, [\overline{x^i} \mapsto \overline{d^i}] \models \psi_\alpha[i]$.

The proof is by induction on $n - i$. In the base case, $i = n$. Now we induct on the structure of β (assume that β is in negation normal form).

In the base case we have $R(x_i, x_j)$. To prove (a), if for some $v_n \in W_n$ suppose $\mathcal{M}, v_n, [x_i \to c, x_j \to d] \models (\exists z \Diamond)(P(x_i) \wedge Q(x_j))$. Recall that $\mathcal{M}, w, [z_1 \to c, z_2 \to d] \models (\exists z \Diamond)^n (\exists z \Diamond (P(z_1) \wedge Q(z_2)) \to (\forall z \Box)^n (\exists z \Diamond (P(z_1) \wedge Q(z_2))$. Hence we have $\mathcal{M}, w, [z_1 \to c, z_2 \to d] \models (\forall z \Box)^n (\exists z \Diamond (P(z_1) \wedge Q(z_2))$. Thus, for all $u_n \in W_n$, we have $\mathcal{M}, u_n, [x_i \to c, x_j \to d] \models (\exists z \Diamond)(P(x_i) \wedge Q(x_j))$.

For (b), $(D, I) \models R(c, d)$ iff $\mathcal{M}, w, [x_i \to c, x_j \to d] \models (\exists z \Box)^n (\exists z \Diamond z (P(x_i) \wedge Q(x_j))$ iff (by definition of R) for all $v_n \in W_n$ we have $\mathcal{M}, v_n, [x_i \to c, x_j \to d] \models (\exists z \Diamond)(P(x_i) \wedge Q(x_j))$.

For the case $\neg R(x, y)$ let $M, v_n, [x_i \to c, x_j \to d] \models \neg(\exists z \Diamond (P(x_i) \wedge Q(x_j))$ this implies $M, w, [x_i \to c, x_j \to d] \not\models (\exists z \Box)^n [\exists z \Box (P(x_i) \wedge Q(x_j))$. Now suppose (a) does not hold, then there is some v_n' such that $M, v_n', [x_i \to c, x_j \to d] \models (\exists z \Diamond (P(x) \wedge Q(y))$ but this implies $M, w, [x_i \to c, x_j \to d] \models (\exists z \Diamond)^n (\exists z \Diamond (P(x) \wedge Q(y))$ and hence $M, w, [x_i \to c, x_j \to d] \models (\exists z \Box)^n (\exists z \Box (P(x) \wedge Q(y))$ which contradicts the assumption. Further (b) follows but routine induction.

The cases of $\vee$ and $\wedge$ are standard.

For the induction step, consider the case when $\alpha[i-1]$ is of the form $\exists x_i \alpha[i]$; the corresponding translated formula is $\exists x_i \Diamond \psi_\alpha[i]$.

To prove (a), suppose for some $v_{i-1} \in W_{i-1}$ we have $\mathcal{M}, v_{i-1}, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}] \models \exists x \Diamond \psi_\alpha[i]$ then there is some $c \in D$ and some successor of v_{i-1}, $v_i' \in W_i$ such that $\mathcal{M}, v_i', [\overline{x^{i-1}} \mapsto$

$\overline{d^{i-1}}, x^i \to c] \models \psi_\alpha[i]$. Now by induction, for all $u_i \in W_i$, we have $\mathcal{M}, u_i, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}, x^i \to c] \models \psi_\alpha[i]$. Further since $\mathcal{M}, r \models \lambda_{n+2}$, every $u_{i-1} \in W_{i-1}$ has at least one successor $u_i' \in W_i$. Thus, $\mathcal{M}, u_{i-1}, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}] \models \exists x_i \Diamond \psi_\alpha[i]$.

For (b) suppose, $D, I, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}] \models \exists x_i \alpha[i]$ then there is some $c \in D$ such that $D, I, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}, x_i \to c] \models \alpha[i]$ iff (by induction hypothesis) $\mathcal{M}, v_i, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}, x_i \to c] \models \psi_\alpha[i]$ for every $v_i \in W_i$ at height i. Now any $w_{i-1} \in W_{i-1}$ is at height $< n$ and since $\mathcal{M}, r \models \lambda_{n+2}$, there is some $v_i' \in W_i$ which is a successor of w_{i-1}. Hence, for all $w_{i-1} \in W_{i-1}$ we have $\mathcal{M}, w_{i-1}, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}] \models \exists x_i \Diamond \psi_\alpha[i]$.

On the other hand, suppose for all $v_{i-1} \in W_{i-1}$ we have $\mathcal{M}, v_{i-1}, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}] \models \exists x_i \Diamond \psi_\alpha[i]$. Choose arbitrary $w_{i-1} \in W_{i-1}$. By semantics, there is some $c \in D$ and $u_i \in W_i$ which is a successor of w_{i-1} such that $\mathcal{M}, u_i, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}, x_i \to c] \models \psi_\alpha[i]$. Now by induction (a) at step i, for all $u' \in W_i$ we have $\mathcal{M}, u', [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}, x_i \to c] \models \psi_\alpha[i]$ and hence $D, I, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}, x_i \to c] \models \alpha[i]$. Hence $D, I, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}] \models \exists x_i \alpha[i]$.

For the case when $\alpha[i-1]$ is of the form $\forall x_i \alpha[i]$, to prove (a), suppose for some $v_{i-1} \in W_{i-1}$ we have $\mathcal{M}, v_{i-1}, [\overline{x^i} \mapsto \overline{d^i}] \models \forall x_i \Box \psi_\alpha[i]$. Choose arbitrary $c \in D$. Then for all $v_i' \in W_i$ which are successors of v_{i-1}, we have $\mathcal{M}, v_i', [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}, x^i \to c] \models \psi_\alpha[i]$. Since there is at least once such successor of v_{i-1}, by induction (a) for all $u_i \in W_i$, we have $\mathcal{M}, u_i, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}, x^i \to c] \models \psi_\alpha[i]$. Now, note that for all $w_{i-1} \in W_i$ we have successors of $w_{i-1} \subseteq W_i$ and c was chosen arbitrarily. Hence for all $w_{i-1} \in W_{i-1}$ we have $\mathcal{M}, w_{i-1}, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}] \models \forall x_i \Box \alpha[i]$.

To prove (b), suppose $D, I, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}] \models \forall x_i \alpha[i]$. Choose arbitrary $c \in D$. Then $D, I, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}, x^i \to c] \models \alpha[i]$ and by induction hypothesis, for all $v_i \in W_i$ we have $\mathcal{M}, v_i, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}, x^i \to c] \models \psi_\alpha[i]$. Again for any $w_{i-1} \in W_{i-1}$, since successors of w_i are in W_{i-1} and c was chosen arbitrarily we have $\mathcal{M}, w_{i-1}, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}] \models \forall x_i \Box \psi_\alpha[i]$.

Finally, suppose for all $w_{i-1} \in W_{i-1}$ we have $\mathcal{M}, w_{i-1}, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}] \models \forall x_i \Box \psi_\alpha[i]$. Choose arbitrary $c \in D$. Since every $u_i \in W_i$ is a successor of some $w_{i-1} \in W_{i-1}$, for all $u_i \in W_i$ we have $\mathcal{M}, u_i, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}, x_i \to c] \models \psi_\alpha[i]$. Now by induction hypothesis, $D, I, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}, x_i \to c] \models \alpha[i]$. Since c was chosen arbitrarily, $D, I, [\overline{x^{i-1}} \mapsto \overline{d^{i-1}}] \models \forall x_i \alpha[i]$. ◀

4 Decidability results

Having seen that the $\mathsf{B}^{\forall\Box}$-FOML (and hence full B-FOML) fragment is undecidable over constant domain models, and noted that the $\exists\Box$ bundle is decidable over increasing domain models ([15]), it is natural to wonder whether the problem is undecidable because of $\exists\Diamond(\forall\Box)$ bundle or constant domain semantics, or both. In this section, we show that it is indeed the combination that is the culprit, by proving that relaxing either of the conditions leads to decidability. First, we show that the full B-FOML fragment is decidable over increasing domain models, and then show that the $\exists\Box$ bundle is decidable over constant domain models. For technical reasons, we consider formulas given in negation normal form (NNF):

$$\varphi ::= P\overline{x} \mid \neg P\overline{x} \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid \exists x \Box \varphi \mid \exists x \Diamond \varphi \mid \forall x \Box \varphi \mid \forall x \Diamond \varphi$$

Formulas of the form $P\overline{x}$ and $\neg P\overline{x}$ are *literals*. Clearly, every B-FOML-formula φ can be rewritten into an equivalent formula in NNF. We call a formula *clean* if no variable occurs both bound and free in it and every use of a quantifier quantifies a distinct variable. A

finite set of formulas is *clean* if their conjunction is clean. Note that every B-FOML-formula can be rewritten into an equivalent clean formula. (For instance, $\exists x \Box P(x) \vee \exists x \Box Q(x)$ and $P(x) \wedge \exists x \Box Q(x)$ are unclean formulas, whereas $\exists x \Box P(x) \vee \exists y \Box Q(y)$ and $P(x) \wedge \exists y \Box Q(y)$ are their clean equivalents.)

A tableau is a tree structure $T = (W, V, E, \lambda)$ where W is a finite set, (V, E) is a rooted tree and $\lambda : V \to L$ is a labelling map. Each element in L is of the form (w, Γ, F), where $w \in W$, Γ is a finite set of formulas and $F \subseteq \mathsf{Var}$ is a finite set. The intended meaning of the label is that the node constitutes a world w that satisfies the formulas in Γ with the "assignment" F, with each variable in F denoting one that occurs free in Γ and as we will see, the assignment will be the identity.

Tableau procedures offer an intuitive way of constructing a canonical model for the given formula. See Fitting and Mendelson [5] for details on tableau procedures for first order modal logics.

4.1 Increasing domain models

Tableau procedures for first order logics typically add witnesses for existential quantifiers using "new" elements (either variables or constants) while simultaneously instantiate universally quantified formulas by the newly added ones. Tableau procedures for modal logics add successor worlds for $\Diamond$ modalities that inherit formulas α when $\Box\alpha$ is in the parent node. Clearly, we need a combination of both. Increasing domain semantics enables us to easily add new witnesses "as we need", so we consider this first.

One complication with bundled quantifiers and modalities is that we need to ass witnesses for existential quantifiers and successor worlds "simultaneously", in the sense that any decision for one affects the choice of the other. To be specific, suppose that we are in an intermediate step of tableau construction when we have formulas $\{\exists x \Diamond \alpha, \exists y \Box \beta, \forall z \Diamond \varphi, \forall z' \Box \psi\}$ at a node w. We need new witnesses for x and y. Further, we need to add a new successor node wv_x; this new node inherits not only α but also β and ψ. But there is plenty more to consider. We already have "active" variables F, which has been updated to F' now. For each $y' \in F'$ we need a φ-successor (which inherits β and ψ as well).

The (BR) rule in the tableau formalizes this intuition when there are multiple occurrences of the bundled formulas. In general if we have formulas $\{\exists x_1 \Diamond \alpha_1 .. \exists x_{n_1} \Diamond \alpha_{n_1}\} \cup \{\exists y_1 \Box \beta_1 .. \exists y_{n_2} \Box \beta_{n_2}\} \cup \{\forall z_1 \Diamond \varphi_1 .. \forall z_{m_1} \varphi_{m_1}\} \cup \{\forall z'_1 \Box \psi_1 .. \forall z'_{m_2} \Box \psi'_{m_2}\}$ at a world w, we need two kinds of successors. The first kind is where a new successor wv_{x_i} is created for every α_i (where x_i is the witness). These successors should satisfy all $\Box$ formulas and hence we add β_j, ψ_l appropriately. The second kind are the ones that take care of $\forall\Diamond$ formulas and hence we have one successor $wv^{y'}_{z_k}$ for every φ_k and every $y' \in F'$. Again β_j, ψ_l are added appropriately to handle $\Box$ constraints.

The ($\vee$) and ($\wedge$) rules are standard and The rule (END) says that in the absence of any $Qx\Diamond$ formulas, with $Q \in \{\exists, \forall\}$, the branch does not need to be explored further, as only the literals remain.

The corresponding tableau rules are given as follows:

▶ **Definition 7.** Tableau rules for increasing domain models for the B-FOML fragment are given by:

$$\frac{w : \varphi_1 \vee \varphi_2, \Gamma, F}{w : \varphi_1, \Gamma, F \mid w : \varphi_2, \Gamma, F}\ (\vee) \qquad \frac{w : \varphi_1 \wedge \varphi_2, \Gamma, F}{w : \varphi_1, \varphi_2, \Gamma, F}\ (\wedge)$$

Given $n_1 \geq 1$ or $m_1 \geq 1$; $n_2, m_2, s \geq 0$:

$$\frac{\begin{array}{c} w : \exists x_1 \Diamond \alpha_1 \cdots, \exists x_{n_1} \Diamond \alpha_{n_1}, \exists y_1 \Box \beta_1, \cdots, \exists y_{n_2} \Box \beta_{n_2}, \\ \forall z_1 \Diamond \varphi_1, \cdots, \forall z_{m_1} \Diamond \varphi_{m_1}, \forall z'_1 \Box \psi_1, \cdots, \forall z'_{m_2} \Box \psi_{m_2}, \end{array} \quad r_1 \ldots r_s, F}{\begin{array}{c} \langle w v_{x_i} : \alpha_i, \{\beta_j \mid 1 \leq j \leq n_2\}, \{\psi_l[z/z'_l] \mid z \in F', l \in [1, m_2]\}, F' \rangle \text{ where } i \in [1, n_1] \\ \cup \langle w v^{y'}_{z_k} : \varphi_k[y'/z_k], \{\beta_j \mid 1 \leq j \leq n_2\}, \{\psi_l[z/z'_l] \mid z \in F', l \in [1, m_2]\}, F' \rangle \\ \text{where } k \in [1, m_1], y' \in F' \end{array}}\ (\texttt{BR})$$

Given $n_2 \geq 1$ or $m_2 \geq 1$; $s \geq 0$:

$$\frac{w : \exists y_1 \Box \beta_1, \cdots, \exists y_{n_2} \Box \beta_{n_2}, \forall z'_1 \Box \psi_1, \cdots, \forall z'_{m_2} \Box \psi_{m_2}, r_1 \ldots r_s, F}{w : r_1 \ldots r_s, F}\ (\texttt{END})$$

where $F' = F \cup \{x_i \mid i \in [1, n_1]\} \cup \{y_j \mid j \in [1, n_2]\}$ and $r_1 \cdots r_s \in \mathit{lit}$ (the literals).

Note that we use variables themselves as witnesses and F' extends F with one witness for each α_i (x_i) and one for each β_j (y_j). Further, there is an implicit ordering on how rules are applied: (BR) insists on the label containing no top level conjuncts or disjuncts, and hence may be applied only after the $\wedge$ and $\vee$ rules have been applied as many times as necessary.

For a given formula φ, we start building the tableau with the root node $(\{w\}, \{r\}, \emptyset, L)$ where $L(r) = (w, \{\varphi\}, \mathsf{Fv}(\varphi))$. A rule specifies that if a node labelled by the premise of the rule exists at a node, it can cause one or more new nodes to be created as children with the labels as given by the completion of the rule. A tableau is saturated when no more rules can be applied. For any formula φ, we refer to the saturated tableau of φ simply as tableau of φ.[4]

The rule (BR) looks complicated but actually asserts standard modal validities with multiplicity. To see how it works, consider a model $\mathcal{M}$, a world u and assignment σ such that $(\mathcal{M}, u, \sigma) \models \exists x \Diamond \alpha \wedge \exists y \Box \beta \wedge \forall z \Box \psi$. Then there are some domain elements $c, d \in \delta(u)$, and a successor world v_c such that $(\mathcal{M}, v_c, \sigma') \models \alpha \wedge \beta \wedge \psi$, where $\sigma'(x) = \sigma'(z) = c$ and $\sigma'(y) = d$. Further if $(\mathcal{M}, u, \sigma) \models \forall z \Diamond \varphi \wedge \forall z' \Box \psi$ then for all $d \in \delta(u)$, we have a successor world v^d such that for all $c \in \delta(u)$, $(\mathcal{M}, v^d, \sigma') \models \varphi \wedge \psi$, where $\sigma'(z) = d$ and $\sigma'(z') = c$. When the domain elements is a finite set (C) which are themselves variables, then we could as well write $(\mathcal{M}, v^d, \sigma') \models \varphi[z] \wedge \bigwedge_{z' \in C} \psi[z']$. The rule (BR) achieves just this, but has to do all this simultaneously for all the quantified formulas at the node "in one shot", and has to keep the formulas clean too.

We need to check that the rule (BR) is well-defined. Specifically, if the label in the premise contains only clean formulas, we need to check that the label in the conclusion does the same. To see this, observe the following, with Γ being the set of clean formulas in the premise. Let Δ, Δ' stand for any modality.

- Note that if $\exists x \Delta \varphi$ and $Qy\Delta'\psi$ are both in Γ, with Q any quantifier, then $x \neq y$ and neither x occurs free in ψ nor y occurs free in φ; also φ or ψ do not contain any subformula that quantifies over x or y.

[4] Refer Wang[15] for an illustration of a similar tableau construction.

- Hence, in the conclusion of (BR), every substitution of the form $\varphi[z/y]$ results in a clean formula, since z occurs free y does not occur at all. Similar argument holds for ψ. Hence the resulting set of formulas in the successors are always clean.

Thus, maintaining "cleanliness" allows us to treat existential quantifiers as giving their own witnesses. The "increase" in the domain is given by the added elements in F' in the conclusion. Note that with each node creation either the number of boolean connectives or the maximum quantifier rank of formulas in the label goes down, and hence repeated applications of the tableau rules must terminate, thus guaranteeing that the tableau generated is always finite.

A tableau is said to be *open* if it does not contain any node u such that its label contains a literal r as well as its negation. Given a tableau T, we say a node $(w : \Gamma, F)$ is a *branching node* if it is branching due to the application of BR. We call $(w : \Gamma, F)$ the *last node of* w, if it is a leaf node or a branching node. Clearly, given any label w appearing in any node of a tableau T, the last node of w uniquely exists. If it is a non-leaf node, every child of w is labelled wu for some u.

Let t_w denote the last node of w in tableau T and let $\lambda(t_w) = (w : \Gamma, F)$. If it is a non-leaf node, then it is a branching node with rule (BR) applying to it with F' as its conclusion. We let $Dom(t_w)$ denote the set F' in this case and $Dom(t_w) = F$ otherwise.

▶ **Theorem 8.** *For any clean* B-FOML*-formula θ in NNF let $F_r = \{x \mid x \text{ is free in } \theta\} \cup \{z\}$, where $z \in \mathsf{Var}$, z does not appear in θ. Then:*
There is an open tableau from $(r : \{\theta\}, F_r)$ iff θ is satisfiable in an increasing domain model.

Proof. Note that we include a new variable $z \in F_r$ to ensure that the domain is always non-empty.

Let T be any (saturated) tableau T starting from $(r : \{\theta\}, F_r)$ where θ is clean. We observe that for any node t with label $(w : \Gamma, F)$ in T, we have the following. If $x \in F$ and occurs in a formula in Γ then every occurrence of x is free. Further, every variable x occurring free in a formula in Γ is in F. These are proved by induction on the height of t using the fact that the rule (BR), when applied to clean formulas, results in clean formulas.

To prove the theorem, given an open tableau T starting from $(r : \{\theta\}, F_r)$, we define $\mathcal{M} = (W, D, \delta, R, \rho)$ where: $W = \{w \mid (w : \Gamma, F)$ occurs in some label of T for some $\Gamma, F\}$; $D = \mathsf{Var}$; wRv iff $v = wv'$ for some v'; $\delta(w) = Dom(t_w)$; $\overline{x} \in \rho(w, P)$ iff $P\overline{x} \in \Gamma$, where $\lambda(t_w) = (w, \Gamma, F)$. Clearly, if wRv then $Dom(t_w) \subseteq Dom(t_v)$, and hence $\mathcal{M}$ is indeed an increasing domain model.

Moreover ρ is well-defined due to openness of T. We now show that $\mathcal{M}, r$ is indeed a model of θ, and this is proved by the following claim.

Claim. For any $w \in W$ if $\lambda(t_w) = (w : \Gamma, F)$ and if $\alpha \in \Gamma$ then $(\mathcal{M}, w, id_F) \models \alpha$. (Below, we abuse notation and write $(\mathcal{M}, w, F) \models \alpha$ for $(\mathcal{M}, w, id_F) \models \alpha$ where $id_F = \{(x, x) \mid x \in F\}$.)

The proof proceeds by reverse induction on the height of the node at which w occurs as label. The base case is when the node considered is a leaf node and hence it is also the last node with that label. The definition of ρ ensures that the literals are evaluated correctly in the model and hence the base case follows.

For the induction step, the conjunction and disjunction cases, the current node is not the last node. Thus the induction applies to its successor which will also have the same label w and the claim follows.

Now consider the application of rule (BR) at a branching node t_w with label $(w : \Gamma, F)$. Let [5]

$$\Gamma = \{\exists x_i \Diamond \alpha_i \mid i \in [1, n_1]\} \cup \{\exists y_j \Box \beta_j \mid j \in [1, n_2]\} \cup \{\forall z_k \Diamond \varphi_k \mid k \in [1, m_1]\} \cup \{\forall z'_l \Box \psi_l \mid l \in [1, m_2]\} \cup \{r_1 \dots r_s\}.$$

By induction hypothesis, we have that for every $i \leq n_1$, $\mathcal{M}, wv_{x_i}, F' \vDash \alpha_i \wedge \bigwedge_{j \leq n_2} \beta_j \wedge \psi'$ and for every $y \in F'$ and $k \in [1, m_1]$, $\mathcal{M}, wv^y_{z_k}, F' \vDash \varphi_k[y/z_k] \wedge \psi'$, where $\psi' = \bigwedge_{l \leq m_2}^{z \in F'} \psi_l[z/z'_l]$.

Note that $D_w = Dom(t_w) = F'$. We need to show that $\mathcal{M}, w, F \vDash \alpha$ for each $\alpha \in \Gamma$. Every such α is either a literal or a bundle formula. The assertion for literals follows from the definition of ρ. For $\exists x_i \Diamond \alpha_i \in \Gamma$ we have the successor wv_{x_i} where $\mathcal{M}, wv_{x_i}, F' \models \alpha_i$ and (by observation at the beginning of the proof) $\mathsf{Fv}(\alpha_i) \subseteq F$ and hence we have $\mathcal{M}, w, F \models \exists x_i \Diamond \alpha_i$. Similarly for every $\forall z_k \Diamond \varphi_k \in \Gamma$ and $y \in D_w$ we have the successor $wv^y_{z_k}$ where $\mathcal{M}, wv^y_{z_k}, F \models \varphi_k[y/z_k]$ and thus $\mathcal{M}, w, F \models \forall z_k \Diamond \varphi_k$.

Now for the case $\exists y_j \Box \beta_j$: by induction hypothesis, for all successors $wv^\#_z$ of w where $\#$ is either empty or $\# \in F'$ we have $\mathcal{M}, wv^\#_z, F' \vDash \beta_j$. Since $\mathsf{Fv}(\beta_j) \subseteq F \cup \{y_j\}$, we have $\mathcal{M}, wv^\#_z, id_F[y_j \mapsto y_j] \vDash \beta_j$ for each $wv^\#_z$. Finally note that $y_j \in F' = D_w$ and hence we have $\mathcal{M}, w, id_F \vDash \exists y_j \Box \beta_j$.

The case $\forall z'_l \Box \psi_l$ is similar. By induction hypothesis, we have $\mathcal{M}, wv^\#_z, F' \vDash \psi_l[z/z'_l]$ for every $z \in F'$ and again by cleanliness preservation, $\mathcal{M}, wv^\#_z, F'[z_l \mapsto z] \vDash \psi_l$ for all $z \in F' = D_w$.

Hence $\mathcal{M}, w, id_F \vDash \forall z'_l \Box \psi_l$.

Finally note that for the root r, if $t_r = (r : \Gamma, F)$ then $F = F_r$ since domain changes only for a (BR) rule which will not be the t_r. Hence it follows that $\mathcal{M}, r, F_r \vDash \theta$.

Completeness of tableau construction. For the other direction, we show that all rule applications preserve the satisfiability of the formula sets in the labels. This would ensure that there is an open tableau since satisfiability of formula sets ensures lack of contradiction among literals. It is easy to see that the rules ($\wedge$) preserves satisfiability and so does the (END) rule, since F is non-empty at every step. If one of the conclusions of the ($\vee$) rule is satisfiable then so is the premise. It remains only to show that (BR) preserves satisfiability. Consider a label set Γ of clean formulas at a branching node. Let

$$\Gamma = \{\exists x_i \Diamond \alpha_i \mid i \in [1, n_1]\} \cup \{\exists y_j \Box \beta_j \mid j \in [1, n_2]\} \cup \{\forall z_k \Diamond \varphi_k \mid k \in [1, m_1]\} \cup \{\forall z'_l \Box \psi_l \mid l \in [1, m_2]\} \cup \{r_1 \dots r_s\}.$$

be satisfiable at a model $\mathcal{M} = \{W, D, \delta, R, \rho\}$, $w \in W$ and a relevant assignment η such that $\eta(x) \in D_w$ for all $x \in \mathsf{Fv}(\Gamma)$ and $\mathcal{M}, w, \eta \vDash \bigwedge_{\chi \in \Gamma} \chi$.

By the semantics, we have the following:

(A) There exist $a_1, \dots, a_{n_1} \in D_w$ and $v_1 \dots v_{n_1} \in W$ where wRv_i such that $\mathcal{M}, v_i, \eta[x_i \mapsto a_i] \vDash \alpha_i$ for all $i \leq n_1$.

(B) For all $c \in D_w$ there exist $v^c_1 \dots v^c_{m_1} \in W$, where $wRv^c_{m_i}$ such that $\mathcal{M}, v^c_k, \eta[z_k \mapsto c] \models \varphi_k$ for all for all $k \leq m_1$.

(C) There exist $b_1, \dots b_{n_2} \in D_w$ such that for all $v \in W$ if wRv then $\mathcal{M}, v, \eta[y_j \mapsto b_j] \models \beta_j$ for all $j \leq n_2$.

(D) For all $d \in D_w$ and for all $v \in W$ if wRv then $\mathcal{M}, v, \eta[z'_l \mapsto d] \models \psi_l$ for all $l \leq m_2$.

Moreover, due to the fact that Γ is clean, we observe that:

(O) $\overline{x}, \overline{y}, \overline{z}$ and $\overline{z'}$ only occur in $\alpha_i, \beta_j, \varphi_k$ and ψ_l respectively.

[5] Note that the argument holds even if either of n_1 or m_1 is 0.

We now need to show:
(1) $\{\alpha_i\} \cup \{\beta_j \mid 1 \leq j \leq n_2\} \cup \{\psi_l[z/z'_l] \mid z \in F', 1 \leq l \leq m_2\}$ is satisfiable for all $i \leq n_1$.
(2) $\{\varphi_k[y'/z_k]\} \cup \{\beta_j \mid 1 \leq j \leq n_2\}, \{\psi_l[z/z'_l] \mid z \in F', 1 \leq l \leq m_2\}$ is satisfiable for all $k \leq m_1$, $y' \in F'$.

For (1): given $i \leq n_1$, due to (A), (C) and (O), we can pick an $a_i \in D_w$ and a successor v_i of w, and some $\overline{b} \in D_w$ such that

$$\mathcal{M}, v_i, \eta[x_i \mapsto a_i;\ \overline{y} \mapsto \overline{b}] \vDash \alpha_i \wedge \bigwedge_j \beta_j$$

By (D), (O) and the fact that η only assigns variables the elements in D_w, we can also show that

$$\mathcal{M}, v_i, \eta[x_i \mapsto a_i;\ \overline{y} \mapsto \overline{b}] \vDash \bigwedge\{\psi_l[z/z'_l] \mid z \in F', 1 \leq l \leq m_2\}.$$

Note that $\eta[x_i \mapsto a_i;\ \overline{y} \mapsto \overline{b}]$ is relevant for v_i since $\mathcal{M}$ is an increasing domain model and wRv_i. This completes the proof for (1).

For (2): Given $k \leq m_1$ and $y' \in F'$. Suppose $\eta(y') = c \in D_w$, then due to (B) we have a successor v_k^c of w such that $\mathcal{M}, v_k^c, \eta \vDash \varphi_k[y'/z_k]$. Now again, due to (C), (D), (O) and the fact that η is a relevant assignment for w, we have:

$$\mathcal{M}, v_k^c, \eta[\overline{y} \mapsto \overline{b}] \vDash \varphi_k[y'/z_k] \wedge \bigwedge_j \beta_j \wedge \bigwedge\{\psi_l[z/z'_l] \mid z \in F', 1 \leq l \leq m_2\}.$$

Again, $\eta[\overline{y} \mapsto \overline{b}]$ is also a relevant assignment for v_k^c, and this completes the proof for (2). ◀

The theorem offers us a decision procedure for checking satisfiability. Note that not only is the depth of the tableau linear in the size of the formula, but also that labels are never repeated across siblings. Hence an algorithm can explore the tableau depth wise and reuse the same space when exploring other branches. The techniques are standard as in tableau procedures for modal logics. The extra space overhead for keeping track of domain elements is again only linear in the size of the formula. Further, observe that every B-FOML formula has an equivalent formula in negation normal form with linear blow-up. This way, we can get a PSPACE-algorithm for checking satisfiability. The PSPACE lower bound follows from propositional modal logic, of which our language is an extension.

▶ **Corollary 9.** *Satisfiability of* B-FOML *fragment over increasing domain models is* PSPACE*-complete.*

4.2 Constant domain models

We now take up the second task, to show that over constant domain models, the culprit is the $\forall\Box$ bundle, by proving that the satisfiability problem for the $\mathsf{B}^{\exists\Box}$-FOML is decidable over constant domain models.

In these models, we need to fix the domain right at the start of the tableau construction and use only these elements as witnesses. We do this by calculating a precise bound on how many new elements need to be added for each subformula of the form $\exists x\Box\varphi$ and include as many as needed at the beginning of the tableau construction.

Let $\mathsf{Sub}(\theta)$ stand for the finite set of subformulas of θ. Given a clean formula $\theta \in \mathsf{B}^{\exists\Box}$-FOML in NNF, for every $\exists x_j\Box\varphi \in \mathsf{Sub}(\theta)$ let $\mathsf{Var}^{\exists}(\theta) = \{x \mid \exists x\Box\varphi \in \mathsf{Sub}(\theta)\}$. Now, cleanliness has its advantages: every subformula of a clean formula is clean as well. Hence,

when θ_1 and θ_2 are both in $\mathsf{Sub}(\theta)$, $\mathsf{Var}^{\exists}(\theta_1) \cap \mathsf{Var}^{\exists}(\theta_2) = \emptyset$. Similarly, when $\theta_1 \in \mathsf{Sub}(\theta)$ and $\theta_2 \in \mathsf{Sub}(\theta_1)$, again $\mathsf{Var}^{\exists}(\theta_1) \cap \mathsf{Var}^{\exists}(\theta_2) = \emptyset$.

Fix a clean formula θ in NNF with modal depth h. For every $x \in \mathsf{Var}^{\exists}(\theta)$ define Var_x to be the set of h *fresh* variables $\{x^k \mid 1 \leq k \leq h\}$, and let $\mathsf{Var}^{+}(\theta) = \bigcup\{\mathsf{Var}_x \mid x \in \mathsf{Var}^{\exists}(\theta)\}$ be the set of new variables to be added. Note that $\mathsf{Var}_x \cap \mathsf{Var}_y = \emptyset$ when $x \neq y$. Fix a variable z not occurring in $\mathsf{Var}^{+}(\theta)$. Define $D_\theta = \mathsf{Fv}(\theta) \cup \mathsf{Var}^{+}(\theta) \cup \{z\}$.

The tableau rules for constant domain models for $\mathsf{B}^{\exists\Box}$-FOML fragment are given by:

$$\frac{w : \varphi_1 \vee \varphi_2, \Gamma, C}{w : \varphi_1, \Gamma, C \mid w : \varphi_2, \Gamma, C}\ (\vee) \qquad \frac{w : \varphi_1 \wedge \varphi_2, \Gamma, C}{w : \varphi_1, \varphi_2, \Gamma, C}\ (\wedge)$$

Given $n, s \geq 0$; $m \geq 1$:

$$\frac{w : \exists x_1 \Box\varphi_1, \ldots, \exists x_n \Box\varphi_n, \forall y_1 \Diamond\psi_1, \ldots, \forall y_m \Diamond\psi_m, r_1 \ldots r_s, C}{\langle (w v^{y}_{y_i} : \{\varphi_j[x_j^{k_j}/x_j] \mid 1 \leq j \leq n\}, \psi_i[y/y_i], C')\rangle where\ y \in D_\theta, i \in [1,m]}\ (\mathtt{BR_C})$$

Given $n \geq 1, s \geq 0$:

$$\frac{w : \exists x_1 \Box\varphi_1, \cdots, \exists x_n \Box\varphi_n, r_1, \cdots r_s, C}{w : r_1 \cdots r_s, C}\ (\mathtt{END_C})$$

where $C \subseteq D_\theta$ and $C' = C \cup \{x_j^{k_j} \mid 1 \leq j \leq n\}$ where k_j is the smallest number such that $x_j^{k_j} \in \mathsf{Var}_{x_j} \setminus C$ and $r_1 \ldots r_s \in lit$.

Note that the rule $\mathtt{BR_C}$ starts off one branch for each $y \in D_\theta$, since the $\forall\Diamond$ connective requires this over the fixed constant domain D_θ. C keeps track of the variables used already along the path from the root till the current node. These are now fixed, so the witness for $\exists x\Box\varphi$ is picked from the remaining variables in $\mathsf{Var}_x(\theta)$. The variables in Var_{x_j} are introduced only by applying $\mathtt{BR}$. Since $|\mathsf{Var}_{x_j}|$ is the modal depth, we always have a fresh x_j^k to choose.

▶ **Theorem 10.** *For any clean* $\mathsf{B}^{\exists\Box}$-FOML*-formula* θ *in NNF, there is an open constant tableau from* $(r, \{\theta\}, \mathsf{Fv}(\theta))$ *iff* θ *is satisfiable in a constant domain model.*

The structure of the proof is very similar to that of Theorem 8. But we need to be careful to check that sufficient witnesses exist as needed, since the domain is fixed at the beginning of tableau construction. The proof details are presented in the appendix.

▶ **Corollary 11.** *The satisfiability problem for* $\mathsf{B}^{\exists\Box}$-FOML*-formulas over constant domain models is* PSPACE*-complete.*

5 Between Constant Domain and Increasing Domain

We now show that the $\mathsf{B}^{\exists\Box}$-FOML fragment cannot distinguish increasing domain models and constant domain models. Note that in FOML this distinction is captured by the Barcan formula $\forall x\Box\varphi \to \Box\forall x\varphi$; but this is not expressible in $\mathsf{B}^{\exists\Box}$-FOML.[6]

[6] However, with equality added in the language, we can distinguish the two by:
$\exists x_1\Box(\forall x_2\Diamond(\forall z\Diamond(x_1 = x_2)) \wedge \forall y_1\Diamond(\exists z\Box(\exists y_2\Box(y_1 \neq y_2))$.
We can also accomplish this in the $\forall\Box$ fragment: $\forall x\Box\forall y\Box\neg P(x) \wedge \forall z\Box\exists x\Diamond\neg P(x)$.

The tableau construction of $\mathsf{B}^{\exists\Box}$-FOML fragment over increasing domain models is a restriction of the BR rule in the last section presented in [15].

Given $n, s \geq 0$; $m \geq 1$:

$$\frac{w : \exists x_1 \Box\varphi_1, \ldots, \exists x_n \Box\varphi_n, \forall y_1 \Diamond\psi_1, \ldots, \forall y_m \Diamond\psi_m, r_1 \ldots r_s, F}{\langle (wv^{y}_{y_i} : \{\varphi_j \mid 1 \leq j \leq n\}, \psi_i[y/y_i], F')\rangle where\ y \in F', i \in [1, m]} \ (\mathtt{BR_W})$$

where $F' = F \cup \{x_j \mid j \in [1, n]\}$.

Note that $\mathtt{BR_C}$ produces a constant domain tableau whereas $\mathtt{BR_W}$ produces an increasing domain tableau. Now, to prove that the $\mathsf{B}^{\exists\Box}$-FOML fragment cannot distinguish increasing domain models and constant domain models, it is sufficient to show that any formula $\varphi \in \mathsf{B}^{\exists\Box}$-FOML is satisfiable over increasing domain model is also satisfiable in a constant domain model. We prove this by showing that any $\varphi \in \mathsf{B}^{\exists\Box}$-FOML fragment that has an open tableau also has a constant domain tableau. From this tableau, we can extract the constant domain model where φ is satisfiable.

▶ **Theorem 12.** *For any* $\mathsf{B}^{\exists\Box}$-FOML *formula* φ *satisfiable on some increasing domain model, the constant domain tableau of* φ *is open.*

The proof is sketched in the appendix.

6 Discussion

We have considered a decidable fragment of FOML by bundling quantifiers together with modalities, retaining the same complexity as propositional modal logic, while yet admitting arbitrary k-ary predicates.

We note that choice of how this bundling is done is crucial. The $\exists\Box$ bundle is shown to be robustly decidable, for both constant domain and increasing domain semantics, whereas the $\forall\Box$ bundle is undecidable over constant domain models. Indeed, other ways of "bundling" quantifiers and modalities is possible. For instance, the $\Box\forall$ bundle seems to be similar to the $\forall\Box$ that we have considered (over constant domain models) but $\Box\exists$ seems to be interestingly different. Indeed, we could proceed further and consider bundles determined by a shape of quantifier prefix: $\exists x_1 \ldots \exists x_n \Box$ or $\exists x_1 \ldots \exists x_n \forall z_1 \ldots \forall z_n \Box$ might be worthy of study as a bundle as well. In this sense, this paper is envisaged as a study of "bundling" quantifiers and modalities and its impact on decidability rather than proposing the definitive bundled fragment.

An obvious extension is to consider the language with constants, function symbols and equality. This would be of importance in the study of systems with unbounded data. A crucial direction for further development is to consider the *transitive closure* modality so that reachability properties are specified. The tableau procedures presented already give us a basis for exploring model checking algorithms, but working with finite presentations of data domains needs some care.

References

1 Parosh Aziz Abdulla, Karlis Cerans, Bengt Jonsson, and Yih-Kuen Tsay. General decidability theorems for infinite-state systems. In *Logic in Computer Science, 1996. LICS'96. Proceedings., Eleventh Annual IEEE Symposium on*, pages 313–321. IEEE, 1996.

2 Francesco Belardinelli and Alessio Lomuscio. Interactions between knowledge and time in a first-order logic for multi-agent systems: completeness results. *Journal of Artificial Intelligence Research*, 45:1–45, 2012.

3 Egon Börger, Erich Grädel, and Yuri Gurevich. *The classical decision problem*. Springer Science & Business Media, 2001.

4 Gisèle Fischer-Servi et al. The finite model property for MIPQ and some consequences. *Notre Dame Journal of Formal Logic*, 19(4):687–692, 1978.

5 Melvin. Fitting and Richard L Mendelsohn. *First-order modal logic*. Synthese Library. Springer, 1998.

6 D. M. Gabbay and V. B. Shehtman. Undecidability of Modal and Intermediate First-Order Logics with Two Individual Variables. *J. Symbolic Logic*, 58(3):800–823, September 1993. URL: `https://projecteuclid.org:443/euclid.jsl/1183744299`.

7 Kurt Gödel. Zum entscheidungsproblem des logischen funktionenkalküls. *Monatshefte für Mathematik und Physik*, 40(1):433–443, 1933.

8 G. E. Hughes and M. J. Cresswell. *A New Introduction to Modal Logic*. Routledge, 1996.

9 Roman Kontchakov, Agi Kurucz, and Michael Zakharyaschev. Undecidability of first-order intuitionistic and modal logics with two variables. *Bull. Symbolic Logic*, 11(3):428–438, September 2005. `doi:10.2178/bsl/1122038996`.

10 Saul A. Kripke. The Undecidability of Monadic Modal Quantification Theory. *Mathematical Logic Quarterly*, 8(2):113–116, 1962. `doi:10.1002/malq.19620080204`.

11 Mikhail N. Rybakov and Dmitry Shkatov. Undecidability of first-order modal and intuitionistic logics with two variables and one monadic predicate letter. *CoRR*, abs/1706.05060, 2017. `arXiv:1706.05060`.

12 Krister Segerberg. Two-Dimensional Modal Logic. *Journal of Philosophical Logic*, 2(1):77–96, 1973. URL: `http://www.jstor.org/stable/30226970`.

13 Moshe Y Vardi. Why is modal logic so robustly decidable? Technical report, Rice University, 1997.

14 Yanjing Wang. A logic of goal-directed knowing how. *Synthese*, 2017. forthcoming.

15 Yanjing Wang. A New Modal Framework for Epistemic Logic. In *Proceedings Sixteenth Conference on Theoretical Aspects of Rationality and Knowledge, TARK 2017, Liverpool, UK, 24-26 July 2017.*, pages 515–534, 2017. `doi:10.4204/EPTCS.251.38`.

16 Yanjing Wang. Beyond knowing that: a new generation of epistemic logics. In *Jaakko Hintikka on knowledge and game theoretical semantics*, pages 499–533. Springer, 2018.

17 Frank Wolter and Michael Zakharyaschev. Decidable Fragments of First-Order Modal Logics. *The Journal of Symbolic Logic*, 66(3):1415–1438, 2001. URL: `http://www.jstor.org/stable/2695115`.

Appendix

Proof of Theorem 10

▶ **Theorem 10.** *For any clean* $\mathsf{B}^{\exists\Box}$-FOML*-formula* θ *in NNF, there is an open constant tableau from* $(r, \{\theta\}, Fv(\theta))$ *iff* θ *is satisfiable in a constant domain model.*

We show that existence of a constant open tableau is equivalent to satisfiability over constant domain models. First, the following observation on the rule (BR) is useful.

▶ **Proposition 13.** *The rule* ($\mathtt{BR_C}$) *preserves cleanliness of formulas: if a tableau node is labelled by* $(w : \Gamma, C)$*,* Γ *is clean, and a child node labelled* $(wv : \Gamma', C')$ *is created by* (BR) *then* Γ' *is clean as well.*

An important corollary of this proposition is that for all $x \in D_\theta$, at any tableau node all occurrences of x in Γ are free. Therefore, for any formula of the form $\psi_i[z/y_i]$ in the conclusion of the rule, z is free and y_i does not occur at all.

The following fact, familiar from first order logic, will be handy in the proof.

▶ **Proposition 14.** *For any* FOML *formula φ and any model $\mathcal{M}, w$ and any variable z:*

$$\mathcal{M}, w, \sigma \vDash \varphi[z/x] \iff \mathcal{M}, w, \sigma'[x \mapsto \sigma(z)] \vDash \varphi$$

if $\sigma(y) = \sigma'(y)$ for all $y \neq x$ with y occurring free in φ.

Now we shall prove Theorem 10.

Proof. To prove the soundness of tableau construction, given an open tableau T from the root node labelled $(r : \{\theta\}, \mathsf{Fv}(\theta))$, we define $\mathcal{M} = \{W, D_\theta, R, \rho\}$ where $W = \{w \mid (w : \Gamma, C)$ is a label at some node in $T\}$. and wRv iff $v = wv'$ for some v'. For the valuation, we have $\overline{x} \in \rho(w, P)$ iff $P\overline{x} \in \Gamma$, where $\lambda(t_w) = (w, \Gamma)$.

By definition, D_θ is not empty. Further, ρ is well-defined due to the openness of T. As before, we prove that $\mathcal{M}, r$ is indeed a model of θ, and this is proved by the following claim.

Claim. For any tree node w in T if $\lambda(t_w) = (w : \Gamma, C)$ and if $\alpha \in \Gamma$ then $(\mathcal{M}, w, id_C) \models \alpha$. (Again, we abuse notation and write $(\mathcal{M}, w, C) \models \alpha$ for $(\mathcal{M}, w, id_C) \models \alpha$ and denote $C(w)$ to be the C associated with the node labelled w.)

The proof proceeds exactly as before for all the rules except for a slight modification for the ($\mathtt{BR_C}$) rule. We shall consider only this rule in the proof here.

Suppose $(w : \Gamma, C)$ is a branching node where

$$\Gamma = \{\exists x_1 \Box\varphi_1 \dots \exists x_n \Box\varphi_n, \forall y_1 \Diamond\psi_1 \dots \forall y_m \Diamond\psi_m, r_1, \dots r_s\}.$$

By induction hypothesis,

$$\mathcal{M}, wv^y_{y_i}, C'(wv^y_{y_i}) \vDash \psi_i[y/y_i] \wedge \bigwedge_1^n \varphi_j[x_j^{k_j}/x_j]$$

for every $y \in D_\theta$ and $i \in [1, m]$. We need to show that $\mathcal{M}, w, C \vDash \chi$ for each $\chi \in \Gamma$.

The assertion for literals in Γ follows from the definition of ρ. For each $\exists x_j \Box\varphi_j \in \Gamma$ and each $wv^y_{y_i}$, with $y \in D_\theta$, we have $\mathcal{M}, wv^y_{y_i}, C' \vDash \varphi_j[x_j^{k_j}/x_j]$ by induction hypothesis. It is clear that $\{x_j^{k_j} \mid 1 \leq j \leq n\}$ are not free in φ_j since they are chosen to be new. Further, since $x_j^{k_j}$ are not free in φ_j, by Proposition 14, $\mathcal{M}, wv^y_{y_i}, id_C[x_j \mapsto x_j^{k_j}] \vDash \varphi_j$ for all $wv^y_{y_i}$. Therefore $\mathcal{M}, w, C \vDash \exists x_j \Box\varphi_j$.

For $\forall y_i \Diamond\psi_i \in \Gamma$, and $y \in D_\theta$, by induction hypothesis, we have $\mathcal{M}, wv^y_{y_i}, C' \models \psi_i[y/y_i]$. By Proposition 13 and its corollary, y_i is not free in $\psi_i[y/y_i]$ and hence by Proposition 14, $\mathcal{M}, wv^y_{y_i}, id_C[y_i \mapsto y] \vDash \psi_i$. Since this holds for each $y \in D_\theta$, we get $\mathcal{M}, w, id_C \vDash \forall y_i \Diamond\psi_i$ for each i.

Thus, it follows that $\mathcal{M}, r, \sigma(r) \vDash \theta$.

To prove the completeness of the tableau construction, we show that rule applications preserve the satisfiability of the formula set. Again, we only discuss the $\mathtt{BR_C}$ case.

Consider a label set Γ of clean formulas at a branching node. Let

$$\Gamma = \{\exists x_j \Box\varphi_j \mid j \in [1, n]\} \cup \{\forall y_i \Diamond\psi_i \mid j \in [1, m]\} \cup \{r_1 \dots r_s\}$$

be satisfiable in a model $\mathcal{M} = \{W, D, R, \rho\}$, $w \in W$ and an assignment η such that $\mathcal{M}, w, \eta \vDash \varphi$ for all $\varphi \in \Gamma$.

By the semantics:

(A) for all $c \in D^{\mathcal{M}}$ there exist $v_1^c \dots v_m^c \in W$, successors of w such that $\mathcal{M}, v_i^c, \eta[y_i \mapsto c] \models \psi_i$ for each $i \in [1, m]$.

(B) there exist $c_1, \dots c_n \in D$ such that for all $v \in W$, if wRv then $\mathcal{M}, v, \eta[x_j \mapsto c_j] \models \varphi_j$.

By cleanliness of formulas in Γ, each x_j is free only in φ_j, and each y_i is free only in ψ_i. Thus we can merge the assignments without changing the truth values of φ_j and ψ_i, and obtain:

(A') for all $c \in D$ there exist $v_1^c \dots v_m^c \in W$, successors of w, such that

$$\mathcal{M}, v_i^c, \eta[\overline{x_j} \mapsto \overline{c_j}, y_i \mapsto c] \models \varphi_1 \wedge \dots \wedge \varphi_n \wedge \psi_i$$

where $i \in [1, m]$.

Fixing a $y \in D_\theta$ and an $i \in [1, m]$, in the following we show that $\{\varphi_j[x_j^{k_j}/x_j] \mid 1 \leq j \leq n\} \cup \{\psi_i[y/y_i]\}$ is satisfiable. There are two cases to be considered:

1. y is not one of $x_j^{k_j}$. First since η is an assignment for all the variables in Var, we can suppose $\eta(y) = b \in D$. By (A') above, there exists a successor v_i^b of w such that $\mathcal{M}, v_i^b, \eta[\overline{x_j} \mapsto \overline{c_j}, y_i \mapsto b] \models \varphi_1 \wedge \dots \wedge \varphi_n \wedge \psi_i$.
   Note that $\overline{x_j}$ and y_i are not in D_θ thus they are different from y. On the other hand, by cleanliness of Γ, y_i does not occur in φ_j and $\eta(y) = b$, hence $\mathcal{M}, v_i^b, \eta[\overline{x_j} \mapsto \overline{c_j}] \models \varphi_1 \wedge \dots \wedge \varphi_n \wedge \psi_i[y/y_i]$.
   Finally, since each x_j only occurs in φ_j and each $x_j^{k_j}$ does not occur in $\varphi_1 \dots \varphi_j$ and $\psi_i[y/y_i]$, we have: $\mathcal{M}, v_i^b, \eta[\overline{x_j^{k_j}} \mapsto \overline{c_j}] \models \varphi_1[x_1^{k_1}/x_1] \wedge \dots \wedge \varphi_n[x_n^{k_n}/x_n] \wedge \psi_i[y/y_i]$.
2. y is $x_j^{k_j}$ for some j. Then we pick c_j, the witness for x_j, and by (A'), $\mathcal{M}, v_i^{c_j}, \eta[\overline{x_j} \mapsto \overline{c_j}, y_i \mapsto c_j] \models \varphi_1 \wedge \dots \wedge \varphi_n \wedge \psi_i$.
   Since y is $x_j^{k_j}$, we have $\mathcal{M}, v_i^{c_j}, \eta[\overline{x_j} \mapsto \overline{c_j}, x_j^{k_j} \mapsto c_j] \models \varphi_1 \wedge \dots \wedge \varphi_n \wedge \psi_i[y/y_i]$.
   Now proceeding similarly as in the case above we can show that: $\mathcal{M}, v_i^{c_j}, \eta[\overline{x_j^{k_j}} \mapsto \overline{c_j}] \models \varphi_1[x_1^{k_1}/x_1] \wedge \dots \wedge \varphi_n[x_n^{k_n}/x_j] \wedge \psi_i[y/y_i]$.

Finally note that all formulas resulting after applying ($\mathtt{BR_C}$) rule will be of the form A' and is satisfiable as argued above. This completes the proof of the theorem. ◀

Proof of Theorem 12

▶ **Theorem 12.** *For any* $\mathsf{B}^{\exists\Box}$-FOML *formula* φ *satisfiable on some increasing domain model, the constant domain tableau of* φ *is open.*

Proof. (Sketch) We give a proof sketch. Consider a clean $\mathsf{B}^{\exists\Box}$-FOML formula φ, and let $\varphi' = \varphi \wedge \bigwedge\{\exists x' \Box\top \mid x' \in \mathsf{Var}^+(\varphi)\}$ (recall that $\mathsf{Var}^+(\varphi) = \bigcup_{x \in \mathsf{Var}^\exists(\varphi)} \mathsf{Var}_x$). Clearly φ is satisfiable in an increasing domain model iff φ' is as well. Let T be an open tableau for φ'. We show that T can be transformed into a constant open tableau T' for φ.

Suppose T has no applications of ($\mathtt{BR}$), it is also a constant tableau and we are done, so suppose that T has at least one application of the rule ($\mathtt{BR}$). By construction, all the $x' \in \mathsf{Var}^+(\varphi)$ are added to the domain of the root, thus they are also at all the local domains in T. Note that we may have more elements in the local domains, such as x that get added when we apply $\mathtt{BR}$ to $\exists x \Box\varphi$, and therefore there are more branches than needed for a constant domain tableau of φ (such as those for x).

We can get rid of them by the following process:

- Fix $\psi = \exists x \Box \theta \in \mathsf{Sub}(\varphi)$:
  - Fix a node s where BR rule is applied and ψ is in s. Since φ is clean, there is no other node in any path of T from the root passing through s such that $\exists x \Box \theta' \in \mathsf{Sub}(\varphi)$ occurs for some θ'. Let m be the modal depth of φ. The path from the root to the predecessor of s can use at most $m-1$ different variables in $\mathsf{Var}_x(\varphi)$ when generating successors by applying the BR rule to some $\forall y \Diamond \theta$ formula. Pick the first $x^h \in \mathsf{Var}_x$ which is not used in the path up to this node.
  - Delete all the descendent nodes of s that are named using x^h when applying BR to some $\forall y \Diamond$ formula, i.e., the nodes named in the form of $stv_y^{x^h}$ where t can be empty. It is not hard to see that the resulting sub-tableau rooted at s has no occurrence of x^h at all since x^h could only be introduced among the children of s using BR.
  - Rename all the occurrences of x by x^h (in formulas and node names) in all the descendent nodes of s. Then the branching structure from the sub-tableau rooted at s will comply with the BR rule for constant-domain tableau.
  - Repeat the above for all the application nodes of the BR rule w.r.t. ψ
- Repeat the above procedure for all ψ of the form $\exists x \Box \theta \in \mathsf{Sub}(\varphi)$.

The core idea is to simply use the newly introduced variable x as if it were x^h in a constant-domain tableau. Note that each branch-cutting operation and renaming operation (by new variables) above will preserve openness, since openness is merely about contradictions among literals. We then obtain a constant domain tableau by setting the domain as D_φ. ◀

Note that the constant domain tableau T of φ constructed can be viewed as a sub-tree embedded inside the increasing domain tableau T' of φ'. However, showing that it is generated precisely by the tableau rules in Section 4.2 involves some tedious detail.

On the Boundedness Problem for Higher-Order Pushdown Vector Addition Systems

Vincent Penelle
LaBRI, Univ. Bordeaux, CNRS, Bordeaux-INP, Talence, France
vincent.penelle@labri.fr

Sylvain Salvati
CRIStAL, Univ. Lille, INRIA, Lille, France
sylvain.salvati@univ-lille.fr

Grégoire Sutre
LaBRI, Univ. Bordeaux, CNRS, Bordeaux-INP, Talence, France
gregoire.sutre@labri.fr

Abstract

Karp and Miller's algorithm is a well-known decision procedure that solves the termination and boundedness problems for vector addition systems with states (VASS), or equivalently Petri nets. This procedure was later extended to a general class of models, well-structured transition systems, and, more recently, to pushdown VASS. In this paper, we extend pushdown VASS to higher-order pushdown VASS (called HOPVASS), and we investigate whether an approach à la Karp and Miller can still be used to solve termination and boundedness. We provide a decidable characterisation of runs that can be iterated arbitrarily many times, which is the main ingredient of Karp and Miller's approach. However, the resulting Karp and Miller procedure only gives a semi-algorithm for HOPVASS. In fact, we show that coverability, termination and boundedness are all undecidable for HOPVASS, even in the restricted subcase of one counter and an order 2 stack. On the bright side, we prove that this semi-algorithm is in fact an algorithm for higher-order pushdown automata.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory, Theory of computation → Logic and verification

Keywords and phrases Higher-order pushdown automata, Vector addition systems, Boundedness problem, Termination problem, Coverability problem

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.44

Funding This work was supported by the grant ANR-17-CE40-0028 of the French National Research Agency ANR (project BRAVAS).

1 Introduction

Termination of a program is a desirable feature in computer science. As it is undecidable on Turing machines, an important challenge is to find models as expressive as possible while retaining decidability of termination. A prominent model having this property is vector addition systems with states (or VASS for short), introduced by Karp and Miller (without states) to model and analyse concurrent systems [10]. They also provide an algorithm, known as the Karp and Miller tree, which can decide termination as well as boundedness (i.e., finiteness of the set of reachable configurations). This algorithm is not optimal complexity-wise, as it has an Ackermannian worst-case running time [19, 20], whereas termination and boundedness for VASS are ExpSpace-complete [17, 22]. But it is conceptually simple, and

so amenable to other models. Karp and Miller's algorithm has been extended, into a so-called *reduced reachability tree*, to the general class of well-structured transition systems (WSTS), to which VASS belong [6, 7]. It has also recently been applied to VASS equipped with one pushdown stack [14]. These pushdown VASS are not WSTS, thus showing that Karp and Miller's algorithm can apply outside the realm of WSTS.

A well-known extension of pushdown automata is higher-order pushdown automata (or HOPDA for short), introduced in [18, 8, 1], in which stacks are replaced by higher-order stacks – an order n stack being a stack of order $(n-1)$ stacks, with an order 1 stack being a classical stack, and the operations on an order n stack being the copying of the topmost order $(n-1)$ stack on top of it, and its inverse operation. This model is interesting for modelling because of its equivalence to safe higher-order recursion schemes [11]. Furthermore, their transition graphs are exactly the graphs of the so-called prefix-recognisable hierarchy [5, 4], which are known to enjoy decidable MSO model-checking. As all the graphs of the hierarchy enjoy the same decidable properties, a tempting conjecture to make is that what holds for models with an order 1 auxiliary stack also holds for the same models with an order n auxiliary stack, for any order n. The starting point of the present work was thus the conjecture that Karp and Miller's algorithm would be a decision tool for termination and boundedness for higher-order pushdown VASS (or HOPVASS for short).

Contribution. Our contribution in this paper is twofold. We first show that termination, and therefore boundedness, are undecidable for HOPVASS by reducing from termination of Minsky counter machines through a stepwise simulation. We also show that the coverability problem (also known as the control-state reachability problem) is undecidable as well, through the same simulation. Our undecidability results hold even in the restricted subcase of one counter and an order 2 stack. This is in sharp contrast with the same model at order 1, for which boundedness and coverability are decidable [14, 16].

We then give a decidable criterion over sequences of higher-order stack operations which characterises which ones can be applicable arbitrarily many times. The detection of such sequences is crucial for the implementation of Karp and Miller's algorithm. Our criterion, which is decidable in quadratic time, makes Karp and Miller's approach implementable for HOPVASS, but the resulting procedure is only a semi-algorithm. It can find witnesses of non-termination or unboundedness, but it does not terminate in general because, contrary to WSTS and order 1 pushdown VASS, there might be infinite runs that contain no iterable factor. We provide an example that illustrates this fact. More interestingly, we prove, thanks to the same iterability criterion, that our semi-algorithm always terminates on HOPDA. This means that Karp and Miller's algorithm also applies to HOPDA, and thus provides a decision procedure that solves termination and boundedness for HOPDA.

Related work and discussion. The coverability and reachability problems for order 1 pushdown VASS are inter-reducible (in logspace) and Tower-hard [12, 13]. Their decidability status is still open. The boundedness problem for the same model is decidable, and its complexity is between Tower and Hyper-Ackermann [14]. For the subcase of only one counter, coverability is decidable [16] and boundedness is solvable in exponential time [15].

The main framework for our presentation comes from the description of regular sets of higher-order stacks from Carayol presented in [2, 3]. We borrow from it the notion of *reduced* sequence of operations as a short description of the effect of a sequence. Our criterion for iterability is a modification of that reduction notion, in which we aim to keep the domain of definition of the sequence (which is not stable through reduction). To solve this issue,

Carayol introduced *test operations*. Instead, we simply weaken the reduction by forbidding it to reduce *destructive tests* of the highest level, and considering a so-obtained *weak-reduced sequence* for every order. This iterability criterion is similar to the result of Parys in [21], in the sense that the underlying idea is that a sequence of operations is iterable if, and only if, it does not decrease the number of k-stack in the topmost $(k-1)$ stack, for every k. Otherwise, our presentation and our techniques are very different from those of Parys.

To our knowledge, termination and boundedness have never been directly studied on HOPDA. However, there are several existing works from which decidability of termination and boundedness for HOPDA could be easily derived. For example, in [9], Hague, Kochems and Ong study the downward closure of languages of HOPDA, and compute it by deciding the *simultaneous unboundedness problem* of their languages. It follows that finiteness of the language defined by a HOPDA is decidable. Termination and boundedness are easily reducible[1] to the latter problem.

2 Preliminaries

Higher-order pushdown automata. We consider a finite alphabet Σ. The set of *order 1 stacks* (or 1-stacks) over Σ is the set $\mathrm{Stacks}_1(\Sigma) = \Sigma^*$. We denote a 1-stack s as $s = [s_1 \dots s_{|s|}]_1$, where $|s|$ is the length of s, where $s_{|s|}$ is the *topmost letter* of s. The empty stack is denoted $[]_1$. For every $a \in \Sigma$, we define the operations $\mathtt{push}_a$ which adds an a at the top of a stack, and $\mathtt{pop}_a$ which removes the topmost letter of a stack if it is an a and is not applicable otherwise. Formally, $\mathtt{push}_a$ and $\mathtt{pop}_a$ are partial functions from $\mathrm{Stacks}_1(\Sigma)$ to $\mathrm{Stacks}_1(\Sigma)$, defined by $\mathtt{push}_a([s_1 \cdots s_{|s|}]_1) = [s_1 \cdots s_{|s|}a]_1$, and $\mathtt{pop}_a(s) = s'$ if and only if $\mathtt{push}_a(s') = s$. We define the set of order 1 operations $\mathtt{Op}_1(\Sigma) = \{\mathtt{push}_a, \mathtt{pop}_a \mid a \in \Sigma\}$. When Σ is understood, we omit it (we will do it from now on).

For $n > 1$, we define the set of *order n stacks* (or n-stacks) over Σ as $\mathrm{Stacks}_n = (\mathrm{Stacks}_{n-1})^+$. We denote an n-stack s as $s = [s_1 \dots s_{|s|}]_n$, where $s_{|s|}$ is the *topmost* $(n-1)$-*stack* of s. The stack $[[]_{n-1}]_n$ is denoted $[]_n$ for short, and abusively called the *empty stack*. We define the operations $\mathtt{copy}_n$ which copies the topmost $(n-1)$-stack on the top of the stack it is applied to, and $\overline{\mathtt{copy}}_n$ its inverse, i.e., it removes the topmost $(n-1)$-stack of a stack if it is equal to the one right below it, and is not applicable otherwise. Formally, $\mathtt{copy}_n$ and $\overline{\mathtt{copy}}_n$ are partial functions from Stacks_n to Stacks_n, defined by $\mathtt{copy}_n([s_1 \cdots s_{|s|}]_n) = [s_1 \cdots s_{|s|}s_{|s|}]_n$, and $\overline{\mathtt{copy}}_n(s) = s'$ if and only if $\mathtt{copy}_n(s') = s$. We define the set of order n operations $\mathtt{Op}_n = \{\mathtt{copy}_n, \overline{\mathtt{copy}}_n\} \cup \mathtt{Op}_{n-1}$ and we define the application of an operation θ of $\mathtt{Op}_{n-1}$ to an n-stack s as $\theta(s) = [s_1 \cdots s_{|s|-1}\theta(s_{|s|})]_n$. Given $\theta \in \mathtt{Op}_n$, we define $\bar{\theta}$ its inverse, i.e., $\overline{\mathtt{push}_a} = \mathtt{pop}_a$, $\overline{\mathtt{pop}_a} = \mathtt{push}_a$ and $\overline{\overline{\mathtt{copy}}_i} = \mathtt{copy}_i$. Finally, we inductively define the *topmost k-stack* of an n-stack $s = [s_1 \cdots s_{|s|}]_n$ as $\mathrm{top}_n(s) = s$, and $\mathrm{top}_k(s) = \mathrm{top}_k(s_{|s|})$ for $k < n$.

▶ **Example 1.** Assuming that $\Sigma = \{a, b\}$, we have $\mathtt{push}_a([[ab]_1[b]_1]_2) = [[ab]_1[ba]_1]_2$, $\mathtt{copy}_2([[[ab]_1]_2[[a]_1[b]_1]_2]_3) = [[[ab]_1]_2[[a]_1[b]_1[b]_1]_2]_3$, and $\overline{\mathtt{copy}}_2([[b]_1[a]_1]_2)$ is not defined.

An *order n pushdown automaton*, or *n-PDA* for short, or *HOPDA* if the order is left implicit, is a tuple $\mathcal{A} = (Q, q_{\mathrm{init}}, \Sigma, \Delta)$, where Q is a finite set of *states*, q_{init} is an *initial state*, Σ is a *stack alphabet* and $\Delta \subseteq Q \times \mathtt{Op}_n \times Q$ is a finite set of *transitions*. A transition

[1] For termination, simply make all transitions output a letter and make all states accepting, then observe that the resulting HOPDA terminates if, and only if, its language is finite. This observation follows from König's lemma together with the fact that HOPDA are finitely branching. For boundedness, add a new accepting state that the HOPDA may non-deterministically jump to, and from which it "dumps" the contents of the stack on the output tape. All original states are non-accepting and all original transitions are silent. It is readily seen that the resulting HOPDA is bounded if, and only if, its language is finite.

$(p, \theta, q) \in \Delta$ is also written as $p \xrightarrow{\theta} q$. A *configuration* of $\mathcal{A}$ is a pair (q, s), where $q \in Q$ and $s \in \mathrm{Stacks}_n$. The *initial configuration* is $(q_{\mathrm{init}}, []_n)$. A *step* of $\mathcal{A}$ is a triple $((p, s), \theta, (q, t))$, where (p, s) and (q, t) are configurations and θ is an operation, such that $p \xrightarrow{\theta} q$ and $t = \theta(s)$. Such a step is also written as $(p, s) \xrightarrow{\theta} (q, t)$. A *run* of $\mathcal{A}$ is an alternating sequence $(q_0, s_0), \theta_1, (q_1, s_1), \ldots, \theta_k, (q_k, s_k)$ of configurations (q_i, s_i) and operations θ_i, such that $(q_{i-1}, s_{i-1}) \xrightarrow{\theta_i} (q_i, s_i)$ for every $0 < i \leq k$. Such a run is also written as $(q_0, s_0) \xrightarrow{\theta_1} (q_1, s_1) \cdots \xrightarrow{\theta_k} (q_k, s_k)$, and it is called *initialised* when (q_0, s_0) is the initial configuration. The *reachability set* of $\mathcal{A}$ is the set of configurations (q, s) such that there is an initialised run in $\mathcal{A}$ that ends with (q, s).

▶ Remark. Instead of the $\overline{\mathtt{copy}}_n$ operation, the literature usually considers a $\mathtt{pop}_n$ operation that destroys the topmost $(n-1)$-stack (provided that there is one below it). Formally, $\mathtt{pop}_n([s_1 \cdots s_{|s|-1} s_{|s|}]_n) = [s_1 \cdots s_{|s|-1}]_n$ if $|s| > 1$ and is undefined otherwise. Following Carayol [2], we prefer the more symmetric operation $\overline{\mathtt{copy}}_n$ that destroys the topmost $(n-1)$-stack only if it is equal to the previous one.

Higher-order pushdown vector addition systems with states. We let $\mathbb{N}$ denote the set of natural numbers $\mathbb{N} = \{0, 1, \ldots\}$ and we let $\mathbb{Z}$ denote the set of integers $\mathbb{Z} = \{\ldots, -1, 0, 1, \ldots\}$. Consider a *dimension* $d \in N$ with $d > 0$. Given a set S and a vector $\mathbf{x}$ in S^d, we let $\mathbf{x}(c)$ denote the cth component of $\mathbf{x}$, i.e., $\mathbf{x} = (\mathbf{x}(1), \ldots, \mathbf{x}(d))$.

An *order n pushdown vector addition system with states of dimension d*, or *d-dim n-PVASS* for short, or *HOPVASS* if the order and the dimension are left implicit, is a tuple $\mathcal{S} = (Q, q_{\mathrm{init}}, \Sigma, \Delta)$, where Q is a finite set of *states*, q_{init} is an *initial state*, Σ is a *stack alphabet* and $\Delta \subseteq Q \times \mathbb{Z}^d \times \mathtt{Op}_n \times Q$ is a finite set of *transitions*. Vectors $\mathbf{a} \in \mathbb{Z}^d$ are called *actions*. A *configuration* of $\mathcal{S}$ is a triple $(q, \mathbf{x}, s)$, where $q \in Q$, $\mathbf{x} \in \mathbb{N}^d$ and $s \in \mathrm{Stacks}_n$. Intuitively, the dimension d is the number of counters, and $\mathbf{x}(1), \ldots, \mathbf{x}(d)$ are the values of these counters. The *initial configuration* is $(q_{\mathrm{init}}, \mathbf{0}, []_n)$. A *step* of $\mathcal{S}$ is a triple $(p, \mathbf{x}, s) \xrightarrow{\mathbf{a}, \theta} (q, \mathbf{y}, t)$, where $(p, \mathbf{x}, s)$ and $(q, \mathbf{y}, t)$ are configurations, $\mathbf{a}$ is an action and θ is an operation, such that $p \xrightarrow{\mathbf{a}, \theta} q$, $\mathbf{y} = \mathbf{x} + \mathbf{a}$ and[2] $t = \theta(s)$. The notions of *run*, *initialised run* and *reachability set* are defined in the same way as for n-PDA.

Coverability, termination and boundedness. We investigate in this paper three basic verification problems on HOPVASS, namely coverability, termination and boundedness. The *coverability problem* asks, given a HOPVASS $\mathcal{S}$ and a state q of $\mathcal{S}$, whether the reachability set of $\mathcal{S}$ contains a configuration whose state is q. The *termination problem* asks, given a HOPVASS $\mathcal{S}$, whether all initialised runs of a $\mathcal{S}$ are finite. The *boundedness problem* asks, given a HOPVASS $\mathcal{S}$, whether the reachability set of $\mathcal{S}$ is finite. Observe that HOPVASS are *finitely branching*, i.e., each configuration is the source of only finitely many steps. This entails that termination is Turing-reducible to boundedness for HOPVASS. Indeed, if the reachability set of a HOPVASS is infinite, then it necessarily has an infinite initialised run, by König's lemma (applied to its reachability tree). Otherwise, we may decide whether it has an infinite initialised run by exploring its reachability graph, which is finite and computable.

[2] The definition of configurations requires counters to be nonnegative. So the equality $\mathbf{y} = \mathbf{x} + \mathbf{a}$ carries the implicit condition that $\mathbf{x} + \mathbf{a} \geq \mathbf{0}$.

3 Undecidability of Coverability and Termination for 1-dim 2-PVASS

It is known that coverability is decidable for 1-dim 1-PVASS [16] and that termination and boundedness are decidable for 1-PVASS of arbitrary dimension [14]. We show in this section that all three problems are undecidable for HOPVASS, even in the restricted subcase of 1-dim 2-PVASS. Our proof proceeds by reduction from coverability and termination in Minsky counter machines, through a stepwise simulation of these machines by 1-dim 2-PVASS.

We use a non-standard presentation of Minsky counter machines (simply called counter machines in the sequel) that is close to VASS and more convenient for our purpose than the standard one. A *d-counter machine* is a triple $\mathcal{M} = (Q, q_{\text{init}}, \Delta)$, where Q is a finite set of *states*, q_{init} is an *initial state* and $\Delta \subseteq Q \times (\mathbb{Z} \cup \{\mathtt{T}\})^d \times Q$ is a finite set of *transitions*. Vectors $\mathbf{a} \in (\mathbb{Z} \cup \{\mathtt{T}\})^d$ are called *actions*. A *configuration* of $\mathcal{M}$ is a pair $(q, \mathbf{x})$, where $q \in Q$ and $\mathbf{x} \in \mathbb{N}^d$. The *initial configuration* is $(q_{\text{init}}, \mathbf{0})$. A *step* of $\mathcal{M}$ is a triple $(p, \mathbf{x}) \xrightarrow{\mathbf{a}} (q, \mathbf{y})$, where $(p, \mathbf{x})$ and $(q, \mathbf{y})$ are configurations and $\mathbf{a}$ is an action, such that $p \xrightarrow{\mathbf{a}} q$ and

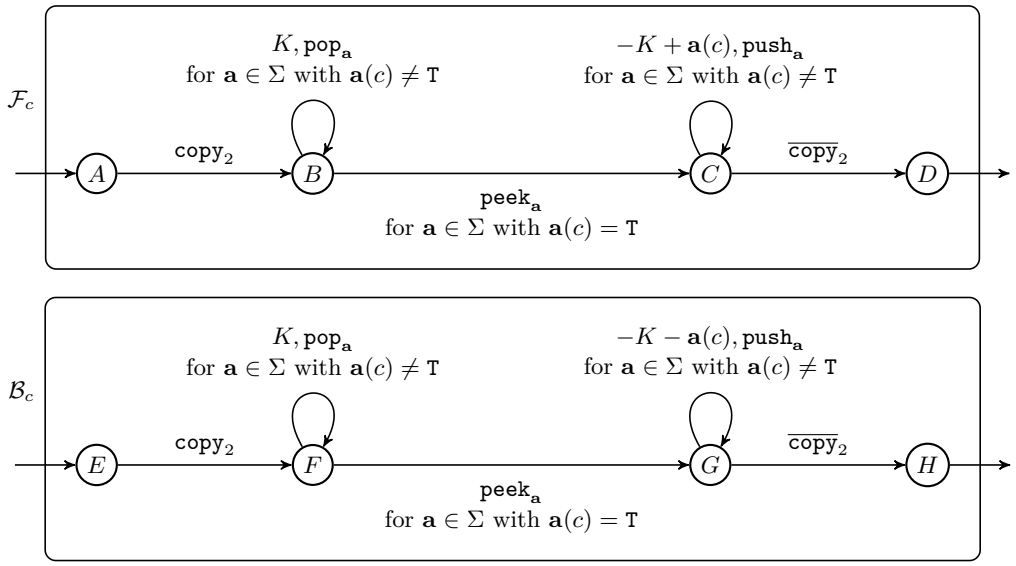
$$\begin{cases} \mathbf{y}(c) = \mathbf{x}(c) + \mathbf{a}(c) & \text{if } \mathbf{a}(c) \in \mathbb{Z} \\ \mathbf{y}(c) = \mathbf{x}(c) = 0 & \text{if } \mathbf{a}(c) = \mathtt{T} \end{cases} \tag{1}$$

for every counter $1 \leq c \leq d$. The notions of *run*, *initialised run* and *reachability set* are defined in the same way as for n-PDA. It is well-known that, for every $d \geq 2$, coverability, termination and boundedness are undecidable for d-counter machines.

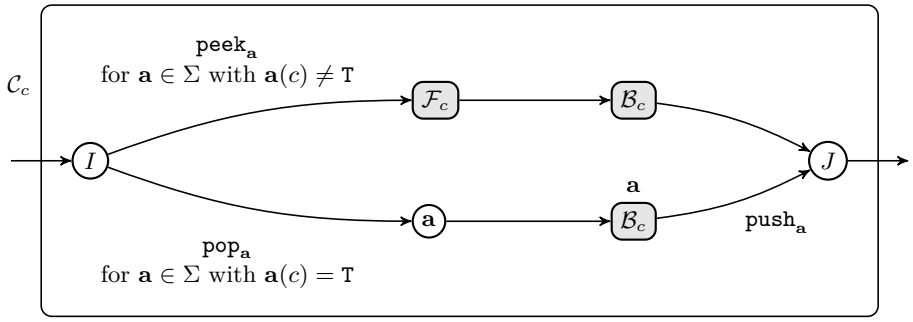
Our simulation of a d-counter machine $\mathcal{M}$ by a 1-dim 2-PVASS $\mathcal{S}$ roughly proceeds as follows. To prevent confusion between the counters of $\mathcal{M}$ and the counter of $\mathcal{S}$, we will denote the latter by κ. When $\mathcal{S}$ is idle, meaning that it is not simulating a step of $\mathcal{M}$, its counter κ is zero and its stack is of the form $[[(\mathtt{T}, \ldots, \mathtt{T})\mathbf{a}_1 \cdots \mathbf{a}_k]_1]_2$, where each $\mathbf{a}_i \in (\mathbb{Z} \cup \{\mathtt{T}\})^d$ is an action of $\mathcal{M}$. Intuitively, the word $w = \mathbf{a}_1 \cdots \mathbf{a}_k$, which we call the *history*, is the sequence of actions that $\mathcal{M}$ has taken to reach its current configuration. The vector $(\mathtt{T}, \ldots, \mathtt{T})$ acts as a bottom symbol. To simulate a step $(p, \mathbf{x}) \xrightarrow{\mathbf{a}} (q, \mathbf{y})$ of $\mathcal{M}$, $\mathcal{S}$ pushes $\mathbf{a}$ onto its stack, which becomes $[[(\mathtt{T}, \ldots, \mathtt{T})w\mathbf{a}]_1]_2$, and then it checks that its new history $w\mathbf{a}$ corresponds to a legal sequence of actions (starting from $\mathbf{0}$) with respect to Equation 1. To perform this check, $\mathcal{S}$ uses the history w and its counter κ to verify, for each counter $1 \leq c \leq d$, that $w\mathbf{a}$ is legal with respect to c. It can do so without destroying the history thanks to $\mathtt{copy}_2$ and $\overline{\mathtt{copy}}_2$ operations. When all checks are complete, $\mathcal{S}$ is again idle, its counter κ is zero and its stack is $[[(\mathtt{T}, \ldots, \mathtt{T})w\mathbf{a}]_1]_2$.

We now present our simulation of d-counter machines by 1-dim 2-PVASS in detail. We start with some additional notations. Given an action $\mathbf{a} \in (\mathbb{Z} \cup \{\mathtt{T}\})^d$, we let $\xrightarrow{\mathbf{a}}$ denote the binary relation on $\mathbb{N}^d$ defined by $\mathbf{x} \xrightarrow{\mathbf{a}} \mathbf{y}$ if Equation 1 holds. Given a word $w = \mathbf{a}_1 \cdots \mathbf{a}_k$ of actions $\mathbf{a}_i \in (\mathbb{Z} \cup \{\mathtt{T}\})^d$, we let $\xrightarrow{w}$ denote the binary relation on $\mathbb{N}^d$ defined by $\mathbf{x} \xrightarrow{w} \mathbf{y}$ if there exists $\mathbf{x}_0, \ldots, \mathbf{x}_k$ such that $\mathbf{x} = \mathbf{x}_0 \xrightarrow{\mathbf{a}_1} \mathbf{x}_1 \cdots \xrightarrow{\mathbf{a}_k} \mathbf{x}_k = \mathbf{y}$, with the convention that $\xrightarrow{\varepsilon}$ is the identity relation on $\mathbb{N}^d$. The notation $\mathbf{x} \xrightarrow{w}$ means that $\mathbf{x} \xrightarrow{w} \mathbf{y}$ for some $\mathbf{y}$. We define the *displacement* $\delta(w)$ of a word $w = \mathbf{a}_1 \cdots \mathbf{a}_k$ in $(\mathbb{Z}^d)^*$ by $\delta(w) = \mathbf{a}_1 + \cdots + \mathbf{a}_k$. Observe that, for such a word $w \in (\mathbb{Z}^d)^*$, it holds that $\mathbf{x} \xrightarrow{w} \mathbf{y}$ if, and only if, $\mathbf{x} + \delta(w) = \mathbf{y}$ and $\mathbf{x} + \delta(v) \geq \mathbf{0}$ for every prefix v of w.

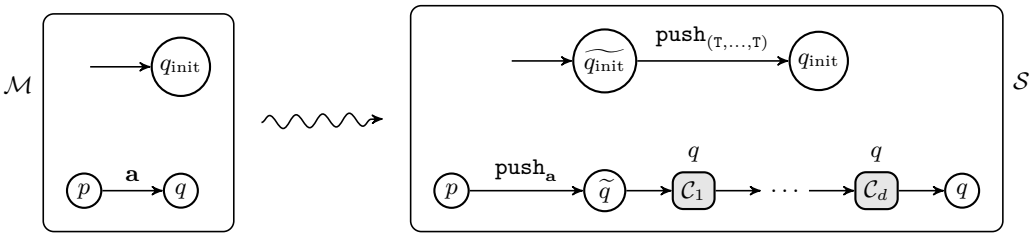
We extend the vector notation $\mathbf{a}(c)$ to sequences of actions $\mathbf{a}_1 \cdots \mathbf{a}_k \in ((\mathbb{Z} \cup \{\mathtt{T}\})^d)^*$ by letting $(\mathbf{a}_1 \cdots \mathbf{a}_k)(c)$ denote the word in $\mathbb{Z}^*$ defined by $(\mathbf{a}_1 \cdots \mathbf{a}_k)(c) = \mathbf{a}_1(c) \cdots \mathbf{a}_k(c)$. Note that for every $\mathbf{x}, \mathbf{y} \in \mathbb{N}^d$ and $w \in ((\mathbb{Z} \cup \{\mathtt{T}\})^d)^*$, it holds that $\mathbf{x} \xrightarrow{w} \mathbf{y}$ if, and only if, $\mathbf{x}(c) \xrightarrow{w(c)} \mathbf{y}(c)$ for every $1 \leq c \leq d$. Observe that the relation $\xrightarrow{w}$ is *forward-deterministic*,



(a) The gadgets $\mathcal{F}_c$ and $\mathcal{B}_c$ that apply, forward for $\mathcal{F}_c$ and backward for $\mathcal{B}_c$, the current history of the cth counter. The constant $K \in \mathbb{N}$ satisfies $|\mathbf{a}(c)| < K$ for every $\mathbf{a} \in \Sigma$ with $\mathbf{a}(c) \neq \mathtt{T}$.



(b) The gadget $\mathcal{C}_c$ that checks that the most recent action of the history is applicable for the cth counter.



(c) Translation of a d-counter machine $\mathcal{M}$ into a 1-dim 2-PVASS $\mathcal{S}$.

Figure 1 Simulation of a d-counter machine $\mathcal{M}$ by a 1-dim 2-PVASS $\mathcal{S}$. The stack alphabet $\Sigma \subseteq (\mathbb{Z} \cup \{\mathtt{T}\})^d$ is the finite set of actions of $\mathcal{M}$. Edges containing "for $\mathbf{a} \in \Sigma$ with φ" denote multiple transitions, one for each $\mathbf{a} \in \Sigma$ satisfying the condition φ.

i.e., $\mathbf{x} \xrightarrow{w} \mathbf{y} \wedge \mathbf{x} \xrightarrow{w} \mathbf{y}' \implies \mathbf{y} = \mathbf{y}'$. In a d-counter machine or 1-dim 2-PVASS, given two configurations α and β, we let $\alpha \xrightarrow{*} \beta$ denote the existence of a run from α to β.

We fix, for the remainder of this section, a d-counter machine $\mathcal{M} = (Q, q_{\text{init}}, \Delta)$. Let $\Sigma \subseteq (\mathbb{Z} \cup \{\mathtt{T}\})^d$ denote the set of actions of $\mathcal{M}$, formally, $\Sigma = \{\mathbf{a} \mid \exists p, q : p \xrightarrow{\mathbf{a}} q\}$. We build from $\mathcal{M}$ a 1-dim 2-PVASS $\mathcal{S}$ with stack alphabet Σ. To simplify the presentation,

we introduce, for every $\mathbf{a} \in \Sigma$, a new order 1 operation $\mathtt{peek}_{\mathbf{a}} = \mathtt{push}_{\mathbf{a}} \circ \mathtt{pop}_{\mathbf{a}}$ that tests, without changing the stack, that the topmost letter of the stack is an $\mathbf{a}$, and is not applicable otherwise. The addition of these $\mathtt{peek}_{\mathbf{a}}$ operations has no impact on the decidability status of coverability, termination and boundedness for 1-dim 2-PVASS.

We present the 1-dim 2-PVASS $\mathcal{S}$ that simulates the d-counter machine $\mathcal{M}$ in a "bottom up" fashion. Recall that κ denotes the counter of $\mathcal{S}$. We start with two gadgets $\mathcal{F}_c$ and $\mathcal{B}_c$, where $1 \leq c \leq d$, that apply on κ, forward for $\mathcal{F}_c$ and backward for $\mathcal{B}_c$, the current history of the cth counter of $\mathcal{M}$. More precisely, they apply the displacement of the suffix v of the history that starts after the most recent zero-test on c. These gadgets are depicted in Figure 1a. Let us explain the behaviour of $\mathcal{F}_c$. We ignore K for the moment. Firstly, the $\mathtt{copy}_2$ from A to B copies the history so that it can be restored before leaving the gadget. The loop on B together with the transition from B to C locates the most recent zero-test on c in the history. The loop on C guesses the suffix v of the history and replays $v(c) \in \mathbb{Z}^*$ on the counter κ. Lastly, the $\overline{\mathtt{copy}}_2$ from C to D ensures that the guesses made in state C are correct and restores the stack to its original contents before entering the gadget. The increments by K in the loop on B are matched by decrements by K in the loop on C. So they do not change the global displacement realised by $\mathcal{F}_c$, which is $\delta(v(c))$. Their purpose is to ensure that the loop on C can be taken only finitely many times. This is crucial for termination. The behaviour of $\mathcal{B}_c$ is identical to that of $\mathcal{F}_c$ except that when $v(c) \in \mathbb{Z}^*$ is replayed on the counter κ, the opposite of each action is applied instead of the action itself. So the global displacement realised by $\mathcal{B}_c$ is $-\delta(v(c))$. The following lemma shows that $\mathcal{F}_c$ and $\mathcal{B}_c$ behave as expected. All proofs of this section can be found in Appendix A.

▶ **Lemma 2.** *Let $x, y \in \mathbb{N}$ and $s, t \in \mathrm{Stacks}_2$. Assume that $s = [[u\mathbf{b}v]_1]_2$ where $\mathbf{b} \in \Sigma$ and $u, v \in \Sigma^*$ are such that $\mathbf{b}(c) = \mathtt{T}$ and $v(c) \in \mathbb{Z}^*$. Then the following assertions hold:*

- *$(A, x, s) \xrightarrow{*} (D, y, t)$ in $\mathcal{F}_c$ if, and only if, $s = t$ and $x + \delta(v(c)) = y$,*
- *$(E, x, s) \xrightarrow{*} (H, y, t)$ in $\mathcal{B}_c$ if, and only if, $s = t$ and $x - \delta(v(c)) = y$.*

Our next gadget, the 1-dim 2-PVASS $\mathcal{C}_c$, where $1 \leq c \leq d$, is depicted in Figure 1b. It uses the gadgets $\mathcal{F}_c$ and $\mathcal{B}_c$ as subsystems. It is understood that each action $\mathbf{a} \in \Sigma$ with $\mathbf{a}(c) = \mathtt{T}$ induces a distinct copy of $\mathcal{B}_c$ in $\mathcal{C}_c$. Provided that $\kappa = 0$ and that w is a legal sequence of actions $\mathbf{0} \xrightarrow{w} \mathbf{x}$, the gadget $\mathcal{C}_c$ checks that the most recent action $\mathbf{a}$ of the history $w\mathbf{a}$ is applicable for the cth counter of $\mathcal{M}$, i.e., that $\mathbf{x}(c) \xrightarrow{\mathbf{a}(c)}$. If $\mathbf{a}(c) \in \mathbb{Z}$ then $\mathcal{C}_c$ goes through $\mathcal{F}_c$, which checks that $\mathbf{x}(c) + \mathbf{a}(c) \geq 0$ and exits with $\kappa = \mathbf{x}(c) + \mathbf{a}(c)$, and then it goes through $\mathcal{B}_c$, which reverts the changes that $\mathcal{F}_c$ did on κ. If $\mathbf{a}(c) = \mathtt{T}$ then $\mathcal{C}_c$ pops $\mathbf{a}$ and then goes through $\mathcal{B}_c$, which checks that $\mathbf{x}(c) \leq 0$ (hence, $\mathbf{x}(c) = 0$) and exits with $\kappa = 0$, and then pushes $\mathbf{a}$ back. In both cases, κ and the stack are restored to their original contents before entering the gadget. The following lemma shows that $\mathcal{C}_c$ behaves as expected.

▶ **Lemma 3.** *Let $y \in \mathbb{N}$ and $s, t \in \mathrm{Stacks}_2$. Assume that $s = [[(\mathtt{T}, \ldots, \mathtt{T})w\mathbf{a}]_1]_2$ where $\mathbf{a} \in \Sigma$ and $w \in \Sigma^*$ are such that $\mathbf{0} \xrightarrow{w}$. Then $(I, 0, s) \xrightarrow{*} (J, y, t)$ in $\mathcal{C}_c$ if, and only if, $y = 0$, $s = t$ and $0 \xrightarrow{w(c)\mathbf{a}(c)}$.*

We are now ready to present our translation of the d-counter machine $\mathcal{M}$ into an "equivalent" 1-dim 2-PVASS $\mathcal{S}$. The translation is depicted in Figure 1c, and corresponds to the informal description of $\mathcal{S}$ that followed the definition of d-counter machines. It is understood that each state q of $\mathcal{M}$ induces distinct copies of $\mathcal{C}_1, \ldots, \mathcal{C}_d$ in $\mathcal{S}$. The initial state of $\mathcal{S}$ is $\widetilde{q_{\mathrm{init}}}$. We need a few additional notations to prove that this translation is correct in the sense that it preserves coverability and termination. Given $\mathbf{x} \in \mathbb{N}^d$ and $s \in \mathrm{Stacks}_2$,

we let $\mathbf{x} \bowtie s$ denote the existence of $w \in \Sigma^*$ such that $\mathbf{0} \xrightarrow{w} \mathbf{x}$ and $s = [[(\mathtt{T}, \dots, \mathtt{T})w]_1]_2$. Given two configurations $(\widetilde{q}, x, s)$ and (q, y, t) of $\mathcal{S}$, where $q \in Q$ is a state of $\mathcal{M}$, we let $(\widetilde{q}, x, s) \rightsquigarrow (q, y, t)$ denote a run in $\mathcal{S}$ from $(\widetilde{q}, x, s)$ to (q, y, t) with no intermediate state in Q. Put differently, such a run is the concatenation of runs of $\mathcal{C}_1, \dots, \mathcal{C}_d$ except for its first and last steps (see Figure 1c). The correctness of the translation of $\mathcal{M}$ into $\mathcal{S}$ is shown by the following two lemmas. Lemma 4 shows that $\bowtie$ induces a "weak simulation" relation from $\mathcal{M}$ to $\mathcal{S}$. This lemma is used to show that every (possibly infinite) initialised run of $\mathcal{M}$ can be translated into an initialised run of $\mathcal{S}$. Lemma 6 shows that the subsystem $\widetilde{q} \to \mathcal{C}_1 \to \cdots \to \mathcal{C}_d \to q$ of $\mathcal{S}$ works as expected, in that it correctly checks that the most recent action of the history is applicable provided that the previous actions of the history are applicable. This lemma is used to show that every (possibly infinite) initialised run of $\mathcal{S}$ can be translated back into an initialised run of $\mathcal{M}$.

▶ **Lemma 4.** *Let $\mathbf{x} \in \mathbb{N}^d$ and $s \in \mathrm{Stacks}_2$ such that $\mathbf{x} \bowtie s$. For every step $(p, \mathbf{x}) \xrightarrow{\mathbf{a}} (q, \mathbf{y})$ in $\mathcal{M}$, there exists a run $(p, 0, s) \xrightarrow{\mathtt{push}_{\mathbf{a}}} (\widetilde{q}, 0, t) \rightsquigarrow (q, 0, t)$ in $\mathcal{S}$ with $\mathbf{y} \bowtie t$.*

▶ **Corollary 5.** *For every initialised run $(q_0, \mathbf{x}_0) \xrightarrow{\mathbf{a}_1} (q_1, \mathbf{x}_1) \cdots \xrightarrow{\mathbf{a}_k} (q_k, \mathbf{x}_k) \cdots$ in $\mathcal{M}$, there is an initialised run $(\widetilde{q_{\mathrm{init}}}, 0, []_2) \xrightarrow{\mathtt{push}_{(\mathtt{T},\dots,\mathtt{T})}} (q_0, 0, s_0) \xrightarrow{\mathtt{push}_{\mathbf{a}_1}} (\widetilde{q_1}, 0, s_1) \rightsquigarrow (q_1, 0, s_1) \cdots \xrightarrow{\mathtt{push}_{\mathbf{a}_k}} (\widetilde{q_k}, 0, s_k) \rightsquigarrow (q_k, 0, s_k) \cdots$ in $\mathcal{S}$.*

▶ **Lemma 6.** *Assume that $t = [[(\mathtt{T}, \dots, \mathtt{T})w\mathbf{a}]_1]_2$ where $\mathbf{a} \in \Sigma$ and $w \in \Sigma^*$ are such that $\mathbf{0} \xrightarrow{w}$. For every run $(\widetilde{q}, 0, t) \rightsquigarrow (q, x, s)$, it holds that $x = 0$, $s = t$ and $\mathbf{0} \xrightarrow{w\mathbf{a}}$.*

▶ **Corollary 7.** *Every initialised run of $\mathcal{S}$ that is infinite or ends with a configuration whose state is in Q, is of the form $(\widetilde{q_{\mathrm{init}}}, 0, []_2) \xrightarrow{\mathtt{push}_{(\mathtt{T},\dots,\mathtt{T})}} (q_0, 0, s_0) \xrightarrow{\mathtt{push}_{\mathbf{a}_1}} (\widetilde{q_1}, 0, s_1) \rightsquigarrow (q_1, 0, s_1) \cdots \xrightarrow{\mathtt{push}_{\mathbf{a}_k}} (\widetilde{q_k}, 0, s_k) \rightsquigarrow (q_k, 0, s_k) \cdots$ with $q_i \in Q$. Moreover, for every such run in $\mathcal{S}$, there is an initialised run $(q_0, \mathbf{x}_0) \xrightarrow{\mathbf{a}_1} (q_1, \mathbf{x}_1) \cdots \xrightarrow{\mathbf{a}_k} (q_k, \mathbf{x}_k) \cdots$ in $\mathcal{M}$.*

An immediate consequence of Corollaries 5 and 7 is that the coverability and termination problems for d-counter machines are many-one reducible to the coverability and termination problems for 1-dim 2-PVASS, respectively. Since coverability and termination are undecidable for 2-counter machines, they are also undecidable for 1-dim 2-PVASS. Moreover, as mentioned in Section 2, termination is Turing-reducible to boundedness for 1-dim 2-PVASS, since they are finitely branching. We have shown the following theorem.

▶ **Theorem 8.** *The coverability problem, the termination problem and the boundedness problem are undecidable for 1-dim 2-PVASS.*

▶ Remark. Theorem 8 also holds for 1-dim 2-PVASS defined with $\mathtt{pop}_2$ operations instead of $\overline{\mathtt{copy}}_2$ operations. Indeed, we may replace the gadgets $\mathcal{F}_c$ and $\mathcal{B}_c$ by "equivalent" ones using $\mathtt{pop}_2$ and no $\overline{\mathtt{copy}}_2$. Intuitively, instead of guessing and replaying in state C the suffix of the history, $\mathcal{F}'_c$ copies the history twice. Each loop uses a fresh copy of the history and then destroys this copy with a $\mathtt{pop}_2$. Both loops use $\mathtt{pop}_{\mathbf{a}}$ operations to browse through the history (backwards). The construction of $\mathcal{B}'_c$ is similar. The new gadgets $\mathcal{F}'_c$ and $\mathcal{B}'_c$ also satisfy Lemma 2, and it follows that the resulting 1-dim 2-PVASS $\mathcal{S}'$ also simulates the d-counter machine $\mathcal{M}$.

4 Iterability of Operation Sequences

In this section, we show that we can characterise exactly the sequences of operations which can be applied arbitrarily many times to a given stack. This result is used in the next section to provide a semi-decision procedure for the non-boundedness and non-termination problems for HOPVASS, using the Karp and Miller reduced tree (as used in [14]).

We consider sequences of operations in $\mathtt{Op}_n$, called n-blocks for short. Given $\rho = \theta_1 \cdots \theta_m \in \mathtt{Op}_n^*$, we identify it with the partial function $\rho = \theta_m \circ \theta_{m-1} \circ \cdots \circ \theta_1$. We denote by $\mathrm{dom}(\rho)$ the set of n-stacks s such that $\rho(s)$ is defined. We define $\bar{\rho} = \overline{\theta_m} \cdots \overline{\theta_1}$, and observe that $\bar{\rho}$ is the partial inverse of ρ. We want to characterise n-blocks which are *iterable*, i.e., which can be applied arbitrarily many times to a given stack.

▶ **Definition 9.** An n-block ρ is *iterable* on a stack s if for all i, $s \in \mathrm{dom}(\rho^i)$.

To investigate iterability, we are interested in the global effect of an n-block while keeping track of its condition of application. We thus need a normal form of sequences which keeps track of these two things, and a criterion on this normal form to determine if it is iterable or not. Following Carayol [2, 3], we say that an n-block is *reduced* if it does not contain any factor of the form $\theta\bar{\theta}$ with $\theta \in \mathtt{Op}_n$ (in [2], they are called minimal, most details come from [3]). Given an n-block ρ, we let $\mathrm{red}(\rho)$ denote the unique reduced operation sequence obtained from ρ by recursively removing $\theta\bar{\theta}$ factors. It is immediate to observe that $\mathrm{red}(\rho)(s) = \rho(s)$ for every stack $s \in \mathrm{dom}(\rho)$. In particular, $\mathrm{dom}(\rho) \subseteq \mathrm{dom}(\mathrm{red}(\rho))$. Intuitively, $\mathrm{red}(\rho)$ is the minimal n-block performing the transformation performed by ρ, in the sense that it does not contain a factor which does not modify the stack. The reduced n-block is thus a good normal form for determining the global effect of an n-block. However, reducing an n-block may yield an n-block with a wider domain, e.g., $\mathtt{pop}_a\mathtt{push}_a$ is only applicable to stacks whose topmost symbol is an a, while its reduced n-block is ε which is applicable to all stacks.

Thus, red is not a good tool to investigate iterability. We need to preserve the domain of applicability of an n-block, while getting rid of factors that are always applicable and do not modify the stack. To do this, Carayol adds test operations in the normal form of regular sets of n-blocks, at the expense of augmenting the number of operations and having no real normal form for n-blocks themselves as some tests may be redundant and not easy to eliminate. We propose here a different approach which consists in associating to each n-block, n normal blocks, one for every order. Each keeps track of the destructive operations of its order the original n-block has to perform to be applicable, while getting rid of factors which do not modify the stack and do not restrict the domain of application, and reducing the block in the classical sense for lower orders. To this end, we define a weaker variant of reduction, $\overline{\mathrm{red}}_n$, which does not remove factors of the form $\overline{\mathtt{copy}}_n\mathtt{copy}_n$ but is otherwise identical to red. The idea will thus be to consider for an n-block ρ, all its weak reduced blocks at every order: ρ will be iterable if, and only if, for every k, $\overline{\mathrm{red}}_k(\rho)$ is iterable. Furthermore, it will be possible to check syntactically whether $\overline{\mathrm{red}}_k(\rho)$ is iterable or not.

The *restriction* of an n-block ρ to an order k, written $\rho|_k$, is the k-block obtained by removing every operation of order strictly higher than k in ρ, e.g., $(\mathtt{push}_a\mathtt{copy}_2\mathtt{push}_b)|_1 = \mathtt{push}_a\mathtt{push}_b$.

▶ **Definition 10.** Given an n-block ρ and an order $k \leq n$, we call $\overline{\mathrm{red}}_k(\rho)$ the only k-block obtained from $\rho|_k$ by applying the following rewriting system:

$$\theta\bar{\theta} \to \varepsilon, \text{ for } \theta \in \mathtt{Op}_k \backslash \{\overline{\mathtt{copy}}_k\} \text{ if } k > 1, \text{ and } \theta \in \{\mathtt{push}_a \mid a \in \Sigma\} \text{ if } k = 1.$$

Observe that this rewriting system is confluent, as is the classical reduction rewriting system. Therefore, $\overline{\mathrm{red}}_k(\rho)$ can be computed in linear time, by always reducing the leftmost factor first.

The following theorem shows that the domain of an n-block ρ is equal to the intersection of the domains of the weak reductions of ρ of every order. This implies that $\overline{\mathrm{red}}_1(\rho), \cdots, \overline{\mathrm{red}}_n(\rho)$ is indeed a good normal form of ρ in the sense it describes entirely the effect of ρ in a canonical way, and it preserves its domain of definition (contrary to $\mathrm{red}(\rho)$).

Intuitively, this result comes from the fact than whenever a reduction step (in red) enlarges the domain of applicability of ρ, it is due to the removal of a factor of the form $\overline{\mathtt{copy}}_k\mathtt{copy}_k$ (or $\mathtt{pop}_a\mathtt{push}_a$). As such factors are left intact in $\overline{\mathrm{red}}_k(\rho)$, a stack added to the domain of $\mathrm{red}(\rho)$ in this way is not added to the domain of $\overline{\mathrm{red}}_k(\rho)$.

▶ **Theorem 11.** *For every n-stack s and n-block ρ, $s \in \mathrm{dom}(\rho)$ if, and only if, for every $k \leq n$, $s \in \mathrm{dom}(\overline{\mathrm{red}}_k(\rho))$.*

Proof. ($\Rightarrow$) Suppose $s \in \mathrm{dom}(\rho)$. By definition of application, $s \in \mathrm{dom}(\rho|_k)$ for every k. We observe that $\mathrm{dom}(\rho|_k) \subseteq \mathrm{dom}(\overline{\mathrm{red}}_k(\rho))$ as every reduction step cannot restrict the domain of application. It follows that $s \in \mathrm{dom}(\overline{\mathrm{red}}_k(\rho))$.

($\Leftarrow$) For the sake of simplicity, in the following we suppose that when we reduce a $\overline{\mathtt{copy}}_k\mathtt{copy}_k$, $\overline{\mathtt{copy}}_k$ could not be matched with some $\mathtt{copy}_k$ at its left and similarly for $\mathtt{copy}_k$ at its right (w.l.o.g., as the system is confluent).

We show that for every weak reduction step, either $\mathrm{dom}(\rho)$ is not modified, either it is increased, but every stack added to it is not in one of the $\mathrm{dom}(\overline{\mathrm{red}}_k(\rho))$:

- If $\rho = \rho_1\mathtt{copy}_k\overline{\mathtt{copy}}_k\rho_2$ for $k \leq n$ (resp. $\rho_1\mathtt{push}_a\mathtt{pop}_a\rho_2$), then $\mathrm{dom}(\rho) = \mathrm{dom}(\rho_1\rho_2)$.
- If $\rho = \rho_1\overline{\mathtt{copy}}_k\mathtt{copy}_k\rho_2$ for $k < n$ (resp. $\rho_1\mathtt{pop}_a\mathtt{push}_a\rho_2$), then for every stack s in $\mathrm{dom}(\rho_1\rho_2)\backslash\mathrm{dom}(\rho)$, we get that $\rho_1(s) \notin \mathrm{dom}(\overline{\mathtt{copy}}_k)$ (resp. $\rho_1(s) \notin \mathrm{dom}(\mathtt{pop}_a)$). As $\overline{\mathrm{red}}_k(\rho) = \overline{\mathrm{red}}_k(\rho_1)\overline{\mathtt{copy}}_k\mathtt{copy}_k\overline{\mathrm{red}}_k(\rho_2)$ (resp. $\overline{\mathrm{red}}_1(\rho_1)\mathtt{pop}_a\mathtt{push}_a\overline{\mathrm{red}}_1(\rho_2)$) and $\overline{\mathrm{red}}_k(\rho_1)(s) = \rho_1|_k(s)$, we get that $s \notin \mathrm{dom}(\overline{\mathrm{red}}_k(\rho))$.

Therefore, by induction on the weak reduction steps of ρ, if $s \in \mathrm{dom}(\overline{\mathrm{red}}_n(\rho))\backslash\mathrm{dom}(\rho)$ then $s \notin \mathrm{dom}(\overline{\mathrm{red}}_k(\rho))$ for some $k < n$. As furthermore for every $k \leq n$, $s \in \mathrm{dom}(\rho)$ implies that $s \in \mathrm{dom}(\rho|_k)$, and therefore that $s \in \mathrm{dom}(\overline{\mathrm{red}}_k(\rho))$, we get the result. ◀

The rest of this subsection is devoted to proving that it is decidable whether an n-block is iterable on some stack or not. The decision algorithm is based on the observation that when ρ is iterable then for every k, $\overline{\mathrm{red}}_k(\rho)$ can be written as $\overline{\rho_{E_k}}\rho_{I_k}\rho_{E_k}$ and ρ_{I_k} does not contain $\overline{\mathtt{copy}}_k$ (see Theorem 15). Thus, if we iterate ρ, at each level, the accumulated effect will only be the accumulated effect of ρ_{I_k}, as ρ_{E_k} and $\overline{\rho_{E_k}}$ cancel each other. We show that ρ is iterable if, and only if, this accumulated effect does not decrease the "size of the stack" at any level, i.e., ρ_{I_k} does not contain $\overline{\mathtt{copy}}_k$ for any k. The proof, if rather technical in its formulation, only relies on the definition of the weak reduction and the two following auxiliary lemmas, the second being proven in Appendix B.

▶ **Lemma 12** ([3], Lemme 4.1.7)**.** *For every n-block ρ, $\mathrm{red}(\rho) = \varepsilon$ if, and only if, there is a stack s such that $s = \rho(s)$.*

▶ **Lemma 13.** *For every n-block ρ and order k, if $\overline{\mathrm{red}}_k(\rho)$ contains a factor of the form $\mathtt{copy}_k\rho'\overline{\mathtt{copy}}_k$ (or $\mathtt{push}_a\rho'\mathtt{pop}_b$ if $k = 1$), then $\mathrm{dom}(\rho) = \emptyset$.*

▶ **Corollary 14.** *For every n-block ρ and order k, if $\overline{\mathrm{red}}_k(\rho)$ is of the form $\overline{\rho_1}\rho_2\rho_1$ with ρ_1 containing a $\overline{\mathtt{copy}}_k$ (or a $\mathtt{pop}_a$ if $k = 1$), then $\mathrm{dom}(\rho) = \emptyset$.*

Proof. If ρ_1 contains a $\overline{\texttt{copy}}_k$ (resp. $\texttt{pop}_a$), then $\overline{\rho_1}$ contains a $\texttt{copy}_k$ (resp. $\texttt{push}_a$), therefore, $\overline{\text{red}}_k(\rho)$ contains a factor of the form $\texttt{copy}_k\rho'\overline{\texttt{copy}}_k$ (resp. $\texttt{push}_a\rho'\texttt{pop}_a$). ◀

▶ **Theorem 15.** *Given a stack s and an n-block ρ, ρ is iterable on s if, and only if, $s \in \text{dom}(\rho)$ and for every $k \leq n$, $\overline{\text{red}}_k(\rho)$ is of the form $\overline{\rho_{E_k}}\rho_{I_k}\rho_{E_k}$ with ρ_{I_k} containing no $\overline{\texttt{copy}}_k$ (or no $\texttt{pop}_a$ if $k = 1$).*

Proof. ($\Rightarrow$) We only do the proof for $k \geq 2$. The case for $k = 1$ is similar. It suffices to replace $\texttt{copy}_k$ and $\overline{\texttt{copy}}_k$ with $\texttt{push}_a$ and $\texttt{pop}_a$. Suppose that ρ is iterable on a stack s, i.e., for every i, $s \in \text{dom}(\rho^i)$. Fix k and let ρ_{E_k} be the largest suffix of $\overline{\text{red}}_k(\rho)$ such that $\overline{\text{red}}_k(\rho) = \overline{\rho_{E_k}}\rho_{I_k}\rho_{E_k}$. Notice that this choice of ρ_{E_k} implies that $\overline{\text{red}}_k(\rho^2_{I_k}) = \rho^2_{I_k}$. We prove by way of contradiction that so does ρ_{I_k}. Suppose now that ρ_{I_k} contains an occurrence of $\overline{\texttt{copy}}_k$. From Lemma 13, we get that $\rho_{I_k} = \rho_1\overline{\texttt{copy}}_k\rho_2$ with ρ_1 containing no order k operation. Suppose that ρ_2 contains a $\texttt{copy}_k$. Lemma 13 entails that $\rho_2 = \rho_3\texttt{copy}_k\rho_4$ and ρ_4 contains no order k operation. By maximality of ρ_{E_k}, we get $\overline{\text{red}}_k(\rho^2) = \overline{\rho_{E_k}}\rho_1\overline{\texttt{copy}}_k\rho_3\texttt{copy}_k\rho_4\rho_1\overline{\texttt{copy}}_k\rho_3\texttt{copy}_k\rho_4\rho_{E_k}$. As $s \in \text{dom}(\rho^2)$, by the definition of application, we get that $(\rho_4\rho_1)(s') = s'$, for $s' = (\overline{\rho_{E_k}}\rho_1\overline{\texttt{copy}}_k\rho_3\texttt{copy}_k)(s)$. From Lemma 12, we get $\text{red}(\rho_4\rho_1) = \varepsilon$. As it contains no order k operations, we also have that $\overline{\text{red}}_k(\rho_4\rho_1) = \varepsilon$. But then, we obtain that $\overline{\text{red}}_k(\rho^2_{I_k}) = \overline{\text{red}}_k(\rho_1\overline{\texttt{copy}}_k\rho_3\texttt{copy}_k\rho_4\rho_1\overline{\texttt{copy}}_k\rho_3\texttt{copy}_k\rho_4) = \overline{\text{red}}_k(\rho_1\overline{\texttt{copy}}_k\rho_3\texttt{copy}_k\rho_4\rho_1\overline{\texttt{copy}}_k\rho_3\texttt{copy}_k\rho_4) = \overline{\text{red}}_k(\rho_1\overline{\texttt{copy}}_k\rho_3\texttt{copy}_k\overline{\texttt{copy}}_k\rho_3\texttt{copy}_k\rho_4)$ and $\overline{\text{red}}_k(\rho^2_{I_k}) = \overline{\text{red}}_k(\rho_1\overline{\texttt{copy}}_k\rho_3\rho_3\texttt{copy}_k\rho_4)$ so that $\overline{\text{red}}_k(\rho^2_{I_k}) \neq \rho^2_{I_k}$. This is in contradiction with the remark we made above. Therefore, ρ_2 does not contain any $\texttt{copy}_k$.

Thus, ρ_{I_k} contains some $\overline{\texttt{copy}}_k$ and does not contain any $\texttt{copy}_k$. Therefore for every s', $\text{top}_k(\rho_{I_k}(s'))$ has strictly less $(k-1)$-stacks than s' and ρ_{I_k} is only applicable a finite number of times to all stacks. From Theorem 11, as $s \in \text{dom}(\rho^i)$ for all i, we get that $s \in \text{dom}(\overline{\text{red}}_k(\rho)^i)$ for all i, and therefore $\overline{\rho_{E_k}}(s) \in \text{dom}(\rho^i_{I_k})$, for all i. We thus get a contradiction. Therefore, for every k, ρ_{I_k} contains no $\overline{\texttt{copy}}_k$ operation.

($\Leftarrow$) We proceed by induction on the order n. Let us consider a 1-block ρ and a stack $s \in \text{dom}(\rho)$ such that $\overline{\text{red}}_1(\rho) = \overline{\rho_{E_1}}\rho_{I_1}\rho_{E_1}$, with ρ_{I_1} containing no $\texttt{pop}_a$ with $a \in \Sigma$. As $s \in \text{dom}(\rho)$, Corollary 14 shows that ρ_{E_1} contains no $\texttt{pop}_a$ with $a \in \Sigma$. As a consequence, we have that $\text{dom}(\rho_{I_1}) = \text{dom}(\rho_{E_1}) = \text{Stacks}_1$. Therefore $\overline{\rho_{E_1}}(s) \in \text{dom}(\rho^i_{I_1}\rho_{E_1})$ for all i, and $\overline{\rho_{E_1}}(s)$ is defined as $s \in \text{dom}(\rho)$. As $\overline{\text{red}}_1(\rho^i) = \overline{\rho_{E_1}}\rho^i_{I_1}\rho_{E_1}$, we get $s \in \text{dom}(\overline{\text{red}}_1(\rho^i))$ for all i. Using Theorem 11, we get that ρ is iterable on s.

Suppose now that the property holds for $(n-1)$-blocks, and consider an n-block ρ and a stack $s \in \text{dom}(\rho)$ such that for all $k \leq n$, $\overline{\text{red}}_k(\rho) = \overline{\rho_{E_k}}\rho_{I_k}\rho_{E_k}$ with ρ_{I_k} containing no $\overline{\texttt{copy}}_k$. From Corollary 14, we get that ρ_{E_k} contains no $\overline{\texttt{copy}}_k$ as well. Let us show that $s \in \text{dom}(\overline{\text{red}}_n(\rho^i))$ for all i. By hypothesis of induction, $\rho|_{n-1}$ is iterable on s, thus $s \in \text{dom}((\rho|_{n-1})^i)$ for all i. Observe $(\rho|_{n-1})^i = \rho^i|_{n-1}$. Therefore, $s \in \text{dom}((\overline{\text{red}}_n(\rho^i)|_{n-1}))$ for all i, as the latter can be obtained from $\rho^i|_{n-1}$ by applying reduction steps. As $s \in \text{dom}(\rho)$, $\rho_{E_n}(s)$ is defined, we deduce that $\overline{\rho_{E_n}}(s) \in \text{dom}((\rho^i_{I_n}\rho_{E_n})|_{n-1})$ for all i. Given an n-block ρ' containing no $\overline{\texttt{copy}}_n$, as $\texttt{copy}_n$ is applicable to all stacks, we get that for all s', if $s' \in \text{dom}(\rho'|_{n-1})$, then $s' \in \text{dom}(\rho')$. Thus $\overline{\rho_{E_n}}(s) \in \text{dom}(\rho^i_{I_n}\rho_{E_n})$ for all i, which entails $s \in \text{dom}(\overline{\text{red}}_n(\rho^i))$ for all i. As by hypothesis of induction, $s \in \text{dom}(\overline{\text{red}}_k(\rho^i))$ for all i and $k < n$, by Theorem 11, we deduce that ρ is iterable on s. ◀

An important remark which stems from the characterisation of Theorem 15 is that a block is either applicable only finitely many times to every stack, or it is applicable arbitrarily many times to every stack on which it can be applied once.

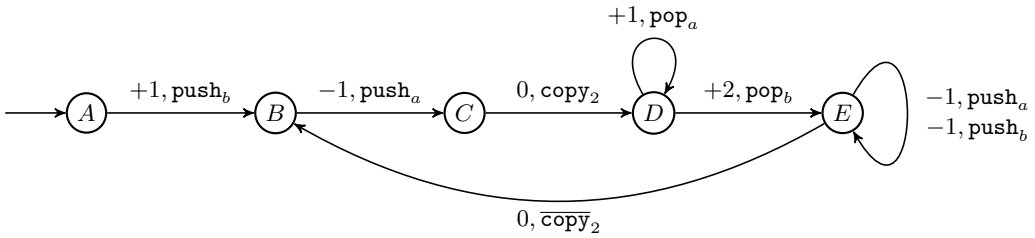


Figure 2 A 1-dim 2-PVASS whose reduced reachability tree is infinite.

5 Testing Non-Termination and Unboundedness

In this section, we use the characterisation of iterable blocks of Theorem 15 to obtain a semi-algorithm à la Karp and Miller for the termination and boundedness problems for HOPVASS. As seen in Section 3, these problems are undecidable. It is however possible to search for witnesses of non-termination, and, with a slight modification, unboundedness. We first present the semi-algorithm, called reduced reachability tree, and prove its correctness. Then, we present an example of HOPVASS on which it does not terminate. Finally, we prove that the semi-algorithm always terminates on HOPDA, and so is a decision procedure for the termination and boundedness problems for HOPDA. We also recall that it is a decision procedure for 1-PVASS as well [14]. We borrow its presentation from that paper.

We define the *reachability tree* of a d-dim n-PVASS $\mathcal{S}$ as follows. Nodes of the tree are labelled by configuration of $\mathcal{S}$. The root r is labelled by the initial configuration $(q_{\text{init}}, \mathbf{0}, []_n)$, written $r : (q_{\text{init}}, \mathbf{0}, []_n)$. Each node $u : (p, \mathbf{x}, s)$ has one child $v : (q, \mathbf{y}, t)$ for each step $(p, \mathbf{x}, s) \xrightarrow{\mathbf{a},\theta} (q, \mathbf{y}, t)$ in $\mathcal{S}$, and the edge from u to v is labelled by the pair $(\mathbf{a}, \theta)$. Notice that the reachability tree of $\mathcal{S}$ is finitely branching.

We say that a node $u : (p, \mathbf{x}, s)$ *subsumes* a node $v : (q, \mathbf{y}, t)$ if u is a proper ancestor of v, $p = q$, $\mathbf{x} \leq \mathbf{y}$, and the block ρ from u to v is iterable on s. Furthermore, we say that u *strictly subsumes* v if $\mathbf{x} < \mathbf{y}$ or $\text{red}(\rho)$ is not ε.

▶ **Theorem 16.** *If the reachability tree of a d-dim n-PVASS $\mathcal{S}$ contains two nodes u and v such that u* subsumes *v (resp., u* strictly subsumes *v), then $\mathcal{S}$ has an infinite initialised run (resp., an infinite reachability set).*

For VASS [10], WSTS [6, 7] and 1-PVASS [14], it can be shown that every infinite branch of the reachability tree contains two nodes such that one subsumes the other. Therefore, termination and boundedness can be solved by constructing the so-called *reduced reachability tree*, or *RRT* for short, which is constructed like the reachability tree, but on which every branch is stopped at the first node subsumed by one of its ancestors. When the RRT of an HOPVASS is finite, it can be computed and it contains enough information to decide termination and boundedness.

As seen in Section 3, boundedness is undecidable for HOPVASS. Figure 2 depicts an example of a 1-dim 2-PVASS whose reduced reachability tree is infinite. There is only one infinite run in this HOPVASS, and for any two configurations with the same state in this run, either the latter one has a smaller counter value, or the sequence of operations between them is not an iterable n-block. That can be proven by an easy case study on the configurations (see Appendix C).

We now turn to show that the RRT is finite in natural subcases of HOPVASS. In [14], it is shown that the RRT is finite for 1-PVASS, by replacing, in the definition of subsumption, our iterability condition with the condition that s is a prefix of every stack appearing on the path from u to v. Actually, the technique presented here yields, at order 1, a (slightly) smaller RRT than in [14], as contrary to it, we can detect that a block in the tree is iterable even if it destructs the stack and then reconstructs it. The rest of the section is devoted to the proof that the RRT is also finite in the case of HOPDA. We first have to introduce some notations and recall some facts.

▶ **Lemma 17.** *If ρ is a block applicable to $[]_n$, then for every order k, $\overline{\mathrm{red}}_k(\rho) = \mathrm{red}(\rho|_k)$ and $\overline{\mathrm{red}}_k(\rho)$ contains no $\overline{\mathtt{copy}}_k$ (no $\mathtt{pop}_a$ if $k = 1$).*

From [2, 3], for every n-stack s, there exists a *unique reduced block* ρ_s such that $\rho_s([]_n) = s$. We define a norm on n-stacks, such that $||s||$ is the length $|\rho_s|$ of the reduced block ρ_s. For every $k \leq n$, we define $||s||_k = ||\mathrm{top}_k(s)||$. We make the following observations.

▶ **Lemma 18.** *For every n-stack s and orders $k < k' \leq n$, it holds that $||s||_k \leq ||s||_{k'}$.*

▶ **Lemma 19.** *Given m, there are at most $(2(|\Gamma|+n-1)-1)^m$ n-stacks s such that $||s||_n = m$.*

We are now ready to prove the main result of this section, namely that the RRT of an HOPDA is finite. To do so, we investigate all possible forms of infinite branch that can appear in the reachability tree of an HOPDA and show that, in all cases, it is possible to extract an iterable block between two nodes with the same state. The easy case is when the branch visits only finitely many stacks, hence, finitely many configurations. In that case, there are two identical configurations on the branch, and the block between them is obviously iterable.

The other case is more involved. When the RRT has an infinite branch, this branch represents an infinite run $(q_0, s_0) \xrightarrow{\theta_1} (q_1, s_1) \cdots \xrightarrow{\theta_k} (q_k, s_k) \cdots$, with $q_0 = q_{\mathrm{init}}$ and $s_0 = []_n$. We then consider the smallest order k for which the sequence $(||s_i||_k)_{i\in\mathbb{N}}$, is unbounded. For every m, we show that we can extract a particular subsequence of positions $j_1, \cdots, j_m$ such that $||s_{j_i}||_k = i$, and that for all stacks s between s_{j_i+1} and s_{j_m}, $||s||_k > i$. We then show that the reduced sequence $\overline{\mathrm{red}}(\theta_{j_i+1} \cdots \theta_{j_{i'}})$ does not contain any $\overline{\mathtt{copy}}_{k'}$ with $k' \geq k$. As k is the smallest order such that $(||s_i||_k)_{i\in\mathbb{N}}$ is unbounded, there are finitely many $(k-1)$-stacks that can appear at the top of the stacks s_i. Consequently we can find a subsequence of $j_1, \cdots, j_m$ such that the topmost $k-1$ stack is the same for all the stacks s_{j_i} with $1 \leq i \leq m$. When m is chosen to be large enough, there must be i and i' so that $q_{j_i} = q_{j'_i}$. Then the conditions are met for us to use Theorem 15. This gives us an iterable block between two nodes with the same state on the infinite branch we considered.

▶ **Theorem 20.** *The reduced reachability tree of an HOPDA is finite.*

Proof. We consider an n-PDA and suppose its RRT is infinite. By Koenig's Lemma, it contains an infinite branch $(q_0, s_0 = []_n), (q_1, s_1), (q_2, s_2), \cdots$, and for every $i \geq 1$, we call θ_i the operation such that $s_i = \theta_i(s_{i-1})$. We thus get an infinite n-block $\theta_1\theta_2 \cdots$. Observe that for every i, we get $\rho_{s_i} = \mathrm{red}(\theta_1 \cdots \theta_i)$.

Suppose that the sequence of $||s_i||_n$ is bounded, i.e., there exists $m \in \mathbb{N}$ such that for every i, $||s_i||_n \leq m$. From Lemma 19 there are finitely many n-stacks of norm at most m. Therefore, there is a stack s such that there are infinitely many i such that $s = s_i$. As Q is finite, there are two positions $i < j$ such that $(q_i, s_i) = (q_j, s_j)$. Thus, $s_i = (\theta_{i+1} \cdots \theta_j)(s_i)$, and therefore $\theta_{i+1} \cdots \theta_j$ is iterable on s_i. Therefore i subsumes j, which contradicts the fact that the branch considered is infinite in the RRT.

Suppose now that the sequence of $||s_i||_n$ is unbounded, i.e., for every $m \in \mathbb{N}$, there exists i such that $||s_i||_n > m$. As for every $k < k' \leq n$ and $s \in \text{Stacks}_n$, $||s||_k \leq ||s||_{k'}$ (Lemma 18), we can fix k such that for every $k' \geq k$, the sequence of $||s_i||_{k'}$ is unbounded, but for every $k' < k$ the sequence of $||s_i||_{k'}$ is bounded. For every $m \in \mathbb{N}$, we define j_m the *first position* at which $||s_i||_k = m$, i.e., $j_m = \min(i \mid ||s_i||_k = m)$. Given $p < m \in \mathbb{N}$, we define $i(p,m)$ the *last position before* j_m at which $||s_i||_k = p$, i.e., $i(p,m) = \max(i < j_m \mid ||s_i||_k = p)$. As for every stack s, operation θ and order k, $||s||_k - 1 \leq ||\theta(s)||_k \leq ||s||_k + 1$, the j_m and $i(p,m)$ are defined for every $p \leq m$. Furthermore, observe that $i(p,m)$ is strictly increasing with respect to p.

Let us show that for every $k' \geq k$, for every $p < p' < m$, there is no $\overline{\texttt{copy}}_{k'}$ in $\overline{\text{red}}_{k'}(\theta_{i(p,m)+1} \cdots \theta_{i(p',m)})$. Observe first that, as by definition $||s_{i(p,m)}||_k < ||s_{i(p,m)+1}||_k$, $\theta_{i(p,m)} \in \texttt{Op}_k$. Suppose there is a position i with $i(p,m) < i < i(p',m)$ such that $\theta_i = \overline{\texttt{copy}}_{k'}$. As $\theta_1 \cdots \theta_i$ is applicable to $[]_n$, $\overline{\text{red}}_{k'}(\theta_0 \cdots \theta_i)$ does not contain any $\overline{\texttt{copy}}_{k'}$ (Lemma 17), and therefore there is a position $i' < i$ such that $\theta_{i'} = \texttt{copy}_{k'}$, $\theta_{i'+1} \cdots \theta_{i-1}$ does not contain any $\texttt{copy}_k$ nor $\overline{\texttt{copy}}_k$ and $\overline{\text{red}}_{k'}(\theta_{i'+1} \cdots \theta_{i-1}) = \varepsilon$. As $\theta_{i'+1} \cdots \theta_{i-1}$ contains no $\texttt{copy}_k$ nor $\overline{\texttt{copy}}_k$, by definition of reduction, for every $i' < \ell < i$, $||s_\ell||_k \geq ||s_{i'}||_k = ||s_i||_k$. Suppose $i' < i(p,m) + 1$, we thus have $||s_{i(p,m)}|| \geq ||s_i||$, which contradicts the definition of $i(p,m)$. Therefore $i' \geq i(p,m) + 1$, and $\overline{\text{red}}_{k'}(\theta_{i(p,m)+1} \cdots \theta_i)$ does not contain any $\overline{\texttt{copy}}_{k'}$. Thus, in any case, $\overline{\text{red}}_{k'}(\theta_{i(p,m)+1} \cdots \theta_{i(p',m)})$ does not contain any $\overline{\texttt{copy}}_{k'}$.

From Lemma 19, there are at most $(2(|\Gamma| + n - 1) - 1)^{h+1}$ $(k-1)$-stacks of norm at most h. We take $m > |Q| * (2(|\Gamma| + n - 1) - 1)^{h+1}$, where h is the highest value for $||s_i||_{k-1}$. Therefore, we can find $|Q|+1$ positions $p_1 < p_2 < \cdots < p_{|Q|+1}$ such that $\overline{\text{red}}_{k-1}(\theta_1 \cdots \theta_{i(p_i,m)})$ is the same for every p_i, and therefore, for every $i < j$, $\overline{\text{red}}_{k-1}(\theta_{i(p_i,m)+1} \cdots \theta_{i(p_j,m)}) = \varepsilon$. As from what precedes, for every $k' \geq k$ and $i < j$, $\overline{\text{red}}_{k'}(\theta_{i(p_i,m)+1} \cdots \theta_{i(p_j,m)})$ does not contain any $\overline{\texttt{copy}}_{k'}$, from Theorem 15 we get that $\theta_{i(p_i,m)+1} \cdots \theta_{i(p_j,m)}$ is iterable on $s_{i(p_i,m)}$. We can furthermore find $i < j$ such that $q_{i(p_i,m)} = q_{i(p_j,m)}$, and therefore, we get that $(q_{i(p_i,m)}, s_{i(p_i,m)})$ subsumes $(q_{i(p_j,m)}, s_{i(p_j,m)})$, which contradicts the fact that the RRT is infinite. ◀

We derive from Theorem 20 that we can solve termination and boundedness for HOPDA by computing the RRT and checking whether it contains a (strictly) subsumed node.

6 Conclusion

In this paper, we have investigated whether an approach à la Karp and Miller can be used to solve termination and boundedness for HOPVASS.

On the negative side, we have shown that coverability, termination, and boundedness are all undecidable for HOPVASS, even in the restricted subcase of one counter and an order 2 stack. This is in sharp contrast with the same model at order 1, for which all three problems are decidable [14, 16].

On the positive side, we have identified a simple and decidable criterion characterising which sequences of higher-order stack operations can be iterated. Such a criterion is crucial for the implementation of Karp and Miller's approach. While the resulting Karp and Miller procedure is only a semi-algorithm for HOPVASS, we have shown that it always terminates for HOPDA. Moreover, when dealing with 1-PVASS, this algorithm is a variant of the algorithm proposed in [14].

We have considered symmetric higher-order operations (as in [2]), namely $\texttt{copy}_n$ operations and their inverse $\overline{\texttt{copy}}_n$. Our undecidability results still hold for HOPVASS defined with $\texttt{pop}_n$ operations instead of $\overline{\texttt{copy}}_n$. We conjecture that Karp and Miller's approach can still be applied to HOPVASS with $\texttt{pop}_n$ and yields an algorithm for HOPDA with $\texttt{pop}_n$.

References

1. Alfred V. Aho. Nested Stack Automata. *J. ACM*, 16(3):383–406, 1969. doi:10.1145/321526.321529.
2. Arnaud Carayol. Regular Sets of Higher-Order Pushdown Stacks. In Joanna Jedrzejowicz and Andrzej Szepietowski, editors, *Mathematical Foundations of Computer Science 2005, 30th International Symposium, MFCS 2005, Gdansk, Poland, August 29 - September 2, 2005, Proceedings*, volume 3618 of *Lecture Notes in Computer Science*, pages 168–179. Springer, 2005. doi:10.1007/11549345_16.
3. Arnaud Carayol. *Automates infinis, logiques et langages*. PhD thesis, University of Rennes 1, France, 2006. URL: https://tel.archives-ouvertes.fr/tel-00628513.
4. Arnaud Carayol and Stefan Wöhrle. The Caucal Hierarchy of Infinite Graphs in Terms of Logic and Higher-Order Pushdown Automata. In Paritosh K. Pandya and Jaikumar Radhakrishnan, editors, *FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science, 23rd Conference, Mumbai, India, December 15-17, 2003, Proceedings*, volume 2914 of *Lecture Notes in Computer Science*, pages 112–123. Springer, 2003. doi:10.1007/978-3-540-24597-1_10.
5. Didier Caucal. On Infinite Terms Having a Decidable Monadic Theory. In Krzysztof Diks and Wojciech Rytter, editors, *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Warsaw, Poland, August 26-30, 2002, Proceedings*, volume 2420 of *Lecture Notes in Computer Science*, pages 165–176. Springer, 2002. doi:10.1007/3-540-45687-2_13.
6. Alain Finkel. A Generalization of the Procedure of Karp and Miller to Well Structured Transition Systems. In Thomas Ottmann, editor, *Automata, Languages and Programming, 14th International Colloquium, ICALP87, Karlsruhe, Germany, July 13-17, 1987, Proceedings*, volume 267 of *Lecture Notes in Computer Science*, pages 499–508. Springer, 1987. doi:10.1007/3-540-18088-5_43.
7. Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001. doi:10.1016/S0304-3975(00)00102-X.
8. Sheila A. Greibach. Full AFLs and Nested Iterated Substitution. *Information and Control*, 16(1):7–35, 1970. doi:10.1016/S0019-9958(70)80039-0.
9. Matthew Hague, Jonathan Kochems, and C.-H. Luke Ong. Unboundedness and downward closures of higher-order pushdown automata. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 151–163. ACM, 2016. doi:10.1145/2837614.2837627.
10. Richard M. Karp and Raymond E. Miller. Parallel Program Schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969. doi:10.1016/S0022-0000(69)80011-5.
11. Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Higher-Order Pushdown Trees Are Easy. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002. doi:10.1007/3-540-45931-6_15.
12. Ranko Lazic. The reachability problem for vector addition systems with a stack is not elementary. *CoRR*, abs/1310.1767, 2013. Presented at RP'12. arXiv:1310.1767.
13. Ranko Lazic and Patrick Totzke. What Makes Petri Nets Harder to Verify: Stack or Data? In Thomas Gibson-Robinson, Philippa J. Hopcroft, and Ranko Lazic, editors, *Concurrency, Security, and Puzzles - Essays Dedicated to Andrew William Roscoe on the Occasion of His 60th Birthday*, volume 10160 of *Lecture Notes in Computer Science*, pages 144–161. Springer, 2017. doi:10.1007/978-3-319-51046-0_8.

14 Jérôme Leroux, M. Praveen, and Grégoire Sutre. Hyper-Ackermannian bounds for pushdown vector addition systems. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 63:1–63:10. ACM, 2014. doi:10.1145/2603088.2603146.

15 Jérôme Leroux, Grégoire Sutre, and Patrick Totzke. On Boundedness Problems for Pushdown Vector Addition Systems. In Mikolaj Bojanczyk, Slawomir Lasota, and Igor Potapov, editors, *Reachability Problems - 9th International Workshop, RP 2015, Warsaw, Poland, September 21-23, 2015, Proceedings*, volume 9328 of *Lecture Notes in Computer Science*, pages 101–113. Springer, 2015. doi:10.1007/978-3-319-24537-9_10.

16 Jérôme Leroux, Grégoire Sutre, and Patrick Totzke. On the Coverability Problem for Pushdown Vector Addition Systems in One Dimension. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 324–336. Springer, 2015. doi:10.1007/978-3-662-47666-6_26.

17 Richard J. Lipton. The reachability problem requires exponential space. Technical Report 63, Yale University, January 1976.

18 A.N. Maslov. Multilevel Stack Automata. *Probl. Inf. Transm.*, 12(1):38–43, 1976.

19 Ernst W. Mayr and Albert R. Meyer. The Complexity of the Finite Containment Problem for Petri Nets. *J. ACM*, 28(3):561–576, 1981. doi:10.1145/322261.322271.

20 Ken McAloon. Petri nets and large finite sets. *Theor. Comput. Sci.*, 32:173–183, 1984. doi:10.1016/0304-3975(84)90029-X.

21 Pawel Parys. A Pumping Lemma for Pushdown Graphs of Any Level. In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France*, volume 14 of *LIPIcs*, pages 54–65. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012. doi:10.4230/LIPIcs.STACS.2012.54.

22 Charles Rackoff. The Covering and Boundedness Problems for Vector Addition Systems. *Theor. Comput. Sci.*, 6:223–231, 1978. doi:10.1016/0304-3975(78)90036-1.

A Proofs of Section 3

▶ **Lemma 2.** *Let $x, y \in \mathbb{N}$ and $s, t \in \mathrm{Stacks}_2$. Assume that $s = [[u\mathbf{b}v]_1]_2$ where $\mathbf{b} \in \Sigma$ and $u, v \in \Sigma^*$ are such that $\mathbf{b}(c) = \mathtt{T}$ and $v(c) \in \mathbb{Z}^*$. Then the following assertions hold:*

- *$(A, x, s) \xrightarrow{*}_{\mathcal{F}_c} (D, y, t)$ in $\mathcal{F}_c$ if, and only if, $s = t$ and $x + \delta(v(c)) = y$,*
- *$(E, x, s) \xrightarrow{*}_{\mathcal{B}_c} (H, y, t)$ in $\mathcal{B}_c$ if, and only if, $s = t$ and $x - \delta(v(c)) = y$.*

Proof. We start with the proof of the first assertion. Define $w = u\mathbf{b}v$. Suppose that there are runs from (A, x, s) to (D, y, t) in $\mathcal{F}_c$, and pick one of them. Since $\mathbf{b}(c) = \mathtt{T}$ and $\mathbf{a}(c) \neq \mathtt{T}$ for every action $\mathbf{a}$ occurring in v, the run necessarily begins with the following steps:

$$(A, x, s) \xrightarrow{\mathtt{copy}_2} (B, x, [[w]_1[u\mathbf{b}v]_1]_2) \xrightarrow{*} (B, x', [[w]_1[u\mathbf{b}]_1]_2) \xrightarrow{\mathtt{peek}_{\mathbf{b}}} (C, x', [[w]_1[u\mathbf{b}]_1]_2) \quad (2)$$

where $x' = x + |v|K$. Then, the run necessarily continues with the following steps:

$$(C, x', [[w]_1[u\mathbf{b}]_1]_2) \xrightarrow{-K+\mathbf{a}_1(c),\mathtt{push}_{\mathbf{a}_1}} \cdots \xrightarrow{-K+\mathbf{a}_k(c),\mathtt{push}_{\mathbf{a}_k}} (C, z, [[w]_1[u\mathbf{b}\mathbf{a}_1 \cdots \mathbf{a}_k]_1]_2) \quad (3)$$

for some $z \in \mathbb{N}$ and some actions $\mathbf{a}_1, \ldots, \mathbf{a}_k$ in Σ such that $\mathbf{a}_i(c) \neq \mathtt{T}$ for every $1 \leq i \leq k$. It follows from the definition of steps in 1-dim 2-PVASS that $x' \xrightarrow{(-K+\mathbf{a}_1(c))\cdots(-K+\mathbf{a}_k(c))} z$.

This entails that $z = x' - kK + \delta(\mathbf{a}_1(c)\cdots\mathbf{a}_k(c))$. Finally, the run necessarily ends with the following step:

$$(C, z, [[w]_1[u\mathbf{b}\mathbf{a}_1\cdots\mathbf{a}_k]_1]_2) \xrightarrow{\overline{\mathtt{copy}}_2} (D, y, t) \tag{4}$$

It follows that $y = z$ and that $t = \overline{\mathtt{copy}}_2([[w]_1[u\mathbf{b}\mathbf{a}_1\cdots\mathbf{a}_k]_1]_2)$. The last equality entails that $t = [[w]_1]_2 = s$ and $w = u\mathbf{b}\mathbf{a}_1\cdots\mathbf{a}_k$. Since $w = u\mathbf{b}v$, we get that $v = \mathbf{a}_1\cdots\mathbf{a}_k$, hence, $v(c) = \mathbf{a}_1(c)\cdots\mathbf{a}_k(c)$. We conclude that $y = z = x' - |v|K + \delta(v(c)) = x + \delta(v(c))$.

Conversely, suppose that $s = t$ and $x + \delta(v(c)) = y$. Let us write v as $v = \mathbf{a}_1\cdots\mathbf{a}_k$ with $\mathbf{a}_i \in \Sigma$. Note that $\mathbf{a}_i(c) \neq \mathtt{T}$ for every $1 \leq i \leq k$, by assumption. Therefore, Equation 2 is a run in $\mathcal{F}_c$, where $x' = x + kK$. Observe that, for every $1 \leq i \leq k$,

$$\begin{aligned} x' + (-K + \mathbf{a}_1(c)) + \cdots + (-K + \mathbf{a}_i(c)) &= x + (k-i)K + \mathbf{a}_1(c) + \cdots + \mathbf{a}_i(c) \\ &= y + (K - \mathbf{a}_{i+1}(c)) + \cdots + (K - \mathbf{a}_k(c)) \\ &\geq 0 \end{aligned}$$

It follows that $x' \xrightarrow{(-K+\mathbf{a}_1(c))\cdots(-K+\mathbf{a}_k(c))} y$. We deduce that Equation 3 is also a run in $\mathcal{F}_c$, by letting $z = y$. Moreover, Equation 4 is also a run in $\mathcal{F}_c$ since $z = y$ and $w = u\mathbf{b}v = u\mathbf{b}\mathbf{a}_1\cdots\mathbf{a}_k$. By concatenating these three runs, we obtain that $(A, x, s) \xrightarrow{*} (D, y, t)$ in $\mathcal{F}_c$.

The second assertion follows from the first assertion by replacing $v = \mathbf{a}_1\cdots\mathbf{a}_k$ with $v' = \mathbf{a}'_1\cdots\mathbf{a}'_k$, where $\mathbf{a}'_i$ differs from $\mathbf{a}_i$ only in c, with $\mathbf{a}'_i(c) = -\mathbf{a}_i(c)$. ◀

▶ **Lemma 3.** *Let $y \in \mathbb{N}$ and $s, t \in \mathrm{Stacks}_2$. Assume that $s = [[(\mathtt{T}, \ldots, \mathtt{T})w\mathbf{a}]_1]_2$ where $\mathbf{a} \in \Sigma$ and $w \in \Sigma^*$ are such that $\mathbf{0} \xrightarrow{w}$. Then $(I, 0, s) \xrightarrow{*} (J, y, t)$ in $\mathcal{C}_c$ if, and only if, $y = 0$, $s = t$ and $0 \xrightarrow{w(c)\mathbf{a}(c)}$.*

Proof. We may write $(\mathtt{T}, \ldots, \mathtt{T})w = u\mathbf{b}v$ for some $\mathbf{b} \in \Sigma$ and $u, v \in \Sigma^*$ such that $\mathbf{b}(c) = \mathtt{T}$ and $v(c) \in \mathbb{Z}^*$. This entails that $\mathtt{T}w(c) = u(c)\mathtt{T}v(c)$. Since $\mathbf{0} \xrightarrow{w}$, we get that $0 \xrightarrow{u(c)\mathtt{T}v(c)}$, hence, $0 \xrightarrow{u(c)\mathtt{T}} 0 \xrightarrow{v(c)} x$ for $x = \delta(v(c))$. Observe that $x \xrightarrow{\mathbf{a}(c)}$ if, and only if, $0 \xrightarrow{w(c)\mathbf{a}(c)}$. The "only if" direction follows from $0 \xrightarrow{w(c)} x$ and the "if" direction follows from forward determinism of $\xrightarrow{w(c)}$. We now proceed with the proof of the lemma.

Suppose that $(I, 0, s) \xrightarrow{*} (J, y, t)$ in $\mathcal{C}_c$. We consider two cases, depending on $\mathbf{a}(c)$. If $\mathbf{a}(c) \in \mathbb{Z}$ then, by definition of $\mathcal{C}_c$ (see Figure 1b), we have $(A, 0, s) \xrightarrow{*} (D, z, t')$ in $\mathcal{F}_c$ and $(E, z, t') \xrightarrow{*} (H, y, t)$ in $\mathcal{B}_c$, for some $z \in \mathbb{N}$ and $t' \in \mathrm{Stacks}_2$. Recall that $s = [[u\mathbf{b}v\mathbf{a}]_1]_2$. We get from Lemma 2 that $s = t'$ and $\delta(v(c)\mathbf{a}(c)) = z$, and we get from Lemma 2 that $t' = t$ and $z - \delta(v(c)\mathbf{a}(c)) = y$. It follows that $y = 0$ and $x + \mathbf{a}(c)) = \delta(v(c)) + \mathbf{a}(c)) = z \geq 0$. We derive that $x \xrightarrow{\mathbf{a}(c)}$, hence, $0 \xrightarrow{w(c)\mathbf{a}(c)}$.

The other case is when $\mathbf{a}(c) = \mathtt{T}$. In that case, by definition of $\mathcal{C}_c$ (see Figure 1b), we have $(E, 0, s') \xrightarrow{*} (H, y, t')$ in $\mathcal{B}_c$, for some $s', t' \in \mathrm{Stacks}_2$ such that $s' = \mathtt{pop}_{\mathbf{a}}(s)$ and $t = \mathtt{push}_{\mathbf{a}}(t')$. Note that $s' = [[u\mathbf{b}v]_1]_2$. We get from Lemma 2 that $s' = t'$ and $-\delta(v(c)) = y$, hence, $\delta(v(c)) \leq 0$. It follows that $x = \delta(v(c)) = 0$, and therefore $x \xrightarrow{\mathtt{T}}$. Moreover, we deduce from $s' = t'$ that $t = \mathtt{push}_{\mathbf{a}}(\mathtt{pop}_{\mathbf{a}}(s)) = s$. We have shown that $y = 0$, $s = t$ and $x \xrightarrow{\mathbf{a}(c)}$. Hence, $0 \xrightarrow{w(c)\mathbf{a}(c)}$.

Conversely, suppose that $0 \xrightarrow{w(c)\mathbf{a}(c)}$ and let us show that $(I, 0, s) \xrightarrow{*} (J, 0, s)$. Note that $x \xrightarrow{\mathbf{a}(c)}$ and let $z \in \mathbb{N}$ such that $x \xrightarrow{\mathbf{a}(c)} z$. It follows that $z = \delta(v(c)\mathbf{a}(c))$ since $x = \delta(v(c))$.

Recall that $s = [[u\mathbf{b}v\mathbf{a}]_1]_2$. We again consider two cases, depending on $\mathbf{a}(c)$. If $\mathbf{a}(c) \in \mathbb{Z}$ then we get from Lemma 2 that $(A,0,s) \xrightarrow{*} (D,z,s)$ in $\mathcal{F}_c$, and we get from Lemma 2 that $(E,z,s) \xrightarrow{*} (H,0,s)$ in $\mathcal{B}_c$. It follows that $(I,0,s) \xrightarrow{*} (J,0,s)$ in $\mathcal{C}_c$. If $\mathbf{a}(c) = \mathtt{T}$ then $x = 0$ since $x \xrightarrow{\mathbf{a}(c)}$. Hence, $\delta(v(c)) = 0$. Let $s' = [[u\mathbf{b}v]_1]_2$ and note that $s' = \mathtt{pop}_{\mathbf{a}}(s)$. We get that from Lemma 2 that $(E,0,s') \xrightarrow{*} (H,0,s')$ in $\mathcal{B}_c$. It follows that $(I,0,s) \xrightarrow{*} (J,0,s)$ in $\mathcal{C}_c$. ◀

▶ **Lemma 4.** *Let $\mathbf{x} \in \mathbb{N}^d$ and $s \in \mathrm{Stacks}_2$ such that $\mathbf{x} \bowtie s$. For every step $(p,\mathbf{x}) \xrightarrow{\mathbf{a}} (q,\mathbf{y})$ in $\mathcal{M}$, there exists a run $(p,0,s) \xrightarrow{\mathtt{push}_{\mathbf{a}}} (\widetilde{q},0,t) \rightsquigarrow (q,0,t)$ in $\mathcal{S}$ with $\mathbf{y} \bowtie t$.*

Proof. Consider a step $(p,\mathbf{x}) \xrightarrow{\mathbf{a}} (q,\mathbf{y})$ in $\mathcal{M}$. Since $\mathbf{x} \bowtie s$, there exists of $w \in \Sigma^*$ such that $\mathbf{0} \xrightarrow{w} \mathbf{x}$ and $s = [[(\mathtt{T},\dots,\mathtt{T})w]_1]_2$. Let $t = \mathtt{push}_{\mathbf{a}}(s) = [[(\mathtt{T},\dots,\mathtt{T})w\mathbf{a}]_1]_2$. Observe that, by construction of $\mathcal{S}$ from $\mathcal{M}$ (see Figure 1c), $(p,0,s) \xrightarrow{\mathtt{push}_{\mathbf{a}}} (\widetilde{q},0,t)$ is a step in $\mathcal{S}$, since $p \xrightarrow{\mathbf{a}} q$ is a transition in $\mathcal{M}$. Notice that $\mathbf{0} \xrightarrow{w\mathbf{a}} \mathbf{y}$ since $\mathbf{0} \xrightarrow{w} \mathbf{x}$ and $\mathbf{x} \xrightarrow{\mathbf{a}} \mathbf{y}$. This entails that $\mathbf{y} \bowtie t$ and that $0 \xrightarrow{w(c)\mathbf{a}(c)}$ for every $1 \le c \le d$. We derive from Lemma 3 that $(I,0,t) \xrightarrow{*} (J,0,t)$ in $\mathcal{C}_c$ for every $1 \le c \le d$. It follows that $(\widetilde{q},0,t) \rightsquigarrow (q,0,t)$ in $\mathcal{S}$. ◀

▶ **Corollary 5.** *For every initialised run $(q_0,\mathbf{x}_0) \xrightarrow{\mathbf{a}_1} (q_1,\mathbf{x}_1) \cdots \xrightarrow{\mathbf{a}_k} (q_k,\mathbf{x}_k) \cdots$ in $\mathcal{M}$, there is an initialised run $(\widetilde{q_{\mathrm{init}}},0,[]_2) \xrightarrow{\mathtt{push}_{(\mathtt{T},\dots,\mathtt{T})}} (q_0,0,s_0) \xrightarrow{\mathtt{push}_{\mathbf{a}_1}} (\widetilde{q_1},0,s_1) \rightsquigarrow (q_1,0,s_1) \cdots \xrightarrow{\mathtt{push}_{\mathbf{a}_k}} (\widetilde{q_k},0,s_k) \rightsquigarrow (q_k,0,s_k) \cdots$ in $\mathcal{S}$.*

Proof. Recall that the initial configuration of $\mathcal{M}$ is $(q_0,\mathbf{x}_0) = (q_{\mathrm{init}},\mathbf{0})$ and that the initial configuration of $\mathcal{S}$ is $(\widetilde{q_{\mathrm{init}}},0,[]_2)$. Observe that $(\widetilde{q_{\mathrm{init}}},0,[]_2) \xrightarrow{\mathtt{push}_{(\mathtt{T},\dots,\mathtt{T})}} (q_{\mathrm{init}},0,s_0)$ in $\mathcal{S}$ for the stack $s_0 = [[(\mathtt{T},\dots,\mathtt{T})]_1]_2$. Note that $\mathbf{0} \bowtie s_0$. It follows from Lemma 4, by induction on i, that there exists $s_i \in \mathrm{Stacks}_2$ and runs $(q_{i-1},0,s_{i-1}) \xrightarrow{\mathtt{push}_{\mathbf{a}_i}} (\widetilde{q_i},0,s_i) \rightsquigarrow (q_i,0,s_i)$ in $\mathcal{S}$ with $\mathbf{x}_i \bowtie s_i$, for every $i \ge 1$. We obtain the desired initialised run of $\mathcal{S}$ by concatenating these runs. ◀

▶ **Lemma 6.** *Assume that $t = [[(\mathtt{T},\dots,\mathtt{T})w\mathbf{a}]_1]_2$ where $\mathbf{a} \in \Sigma$ and $w \in \Sigma^*$ are such that $\mathbf{0} \xrightarrow{w}$. For every run $(\widetilde{q},0,t) \rightsquigarrow (q,x,s)$, it holds that $x = 0$, $s = t$ and $\mathbf{0} \xrightarrow{w\mathbf{a}}$.*

Proof. Consider a run $(\widetilde{q},0,t) \rightsquigarrow (q,x,s)$. Recall that $\rightsquigarrow$ means that the run can be decomposed into a first step (moving from $\widetilde{q}$ to $\mathcal{C}_1$), a last step (moving from $\mathcal{C}_d$ to q), and runs of $\mathcal{C}_1,\dots,\mathcal{C}_d$ in between. So there exists $x_0,\dots,x_d \in \mathbb{N}$ and $s_0,\dots,s_d \in \mathrm{Stacks}_2$, with $x_0 = 0$, $s_0 = t$, $x_d = x$ and $s_d = s$, such that $(I,x_{c-1},s_{c-1}) \xrightarrow{*} (J,x_c,s_c)$ in $\mathcal{C}_c$, for every $1 \le c \le d$. We derive from Lemma 3, by induction on c, that $x_c = 0$, $s_c = t$ and $0 \xrightarrow{w(c)\mathbf{a}(c)}$, for every $1 \le c \le d$. It follows that $x = x_d = 0$, $t = s_d = s$ and $\mathbf{0} \xrightarrow{w\mathbf{a}}$. ◀

▶ **Corollary 7.** *Every initialised run of $\mathcal{S}$ that is infinite or ends with a configuration whose state is in Q, is of the form $(\widetilde{q_{\mathrm{init}}},0,[]_2) \xrightarrow{\mathtt{push}_{(\mathtt{T},\dots,\mathtt{T})}} (q_0,0,s_0) \xrightarrow{\mathtt{push}_{\mathbf{a}_1}} (\widetilde{q_1},0,s_1) \rightsquigarrow (q_1,0,s_1) \cdots \xrightarrow{\mathtt{push}_{\mathbf{a}_k}} (\widetilde{q_k},0,s_k) \rightsquigarrow (q_k,0,s_k) \cdots$ with $q_i \in Q$. Moreover, for every such run in $\mathcal{S}$, there is an initialised run $(q_0,\mathbf{x}_0) \xrightarrow{\mathbf{a}_1} (q_1,\mathbf{x}_1) \cdots \xrightarrow{\mathbf{a}_k} (q_k,\mathbf{x}_k) \cdots$ in $\mathcal{M}$.*

Proof. Consider an initialised run in $\mathcal{S}$ that is infinite or ends with a configuration whose state is in Q. If the run is infinite, then it visits infinitely many configurations whose state are in Q. Because if it were not the case, then an infinite suffix of the run would remain forever in the same $\mathcal{F}_c$ or $\mathcal{B}_c$. This is impossible as each loop in $\mathcal{F}_c$ or $\mathcal{B}_c$ either shrinks the stack or decreases the counter κ, since K satisfies $|\mathbf{a}(c)| < K$ for every $\mathbf{a} \in \Sigma$ with $\mathbf{a}(c) \neq \mathtt{T}$. So the initialised run under consideration starts with the step $(\widetilde{q_{\mathrm{init}}},0,[]_2) \xrightarrow{\mathtt{push}_{(\mathtt{T},\dots,\mathtt{T})}} (q_{\mathrm{init}},0,[[(\mathtt{T},\dots,\mathtt{T})]_1]_2)$ followed by a run of the form:

$$(q_0,x_0,s_0) \xrightarrow{\mathtt{push}_{\mathbf{a}_1}} (\widetilde{q_1},y_1,t_1) \rightsquigarrow (q_1,x_1,s_1) \cdots \xrightarrow{\mathtt{push}_{\mathbf{a}_k}} (\widetilde{q_k},y_k,t_k) \rightsquigarrow (q_k,x_k,s_k) \cdots$$

where $(q_0, x_0, s_0) = (q_{\text{init}}, 0, [[(\mathtt{T}, \ldots, \mathtt{T})]_1]_2)$ and $q_1, \ldots, q_k, \ldots$ are in Q. Observe that $y_i = x_{i-1}$ and $t_i = \mathtt{push}_{\mathbf{a}_i}(s_{i-1})$, for every $i \geq 1$. We derive from Lemma 6, by induction on i, that $x_i = 0$, $s_i = t_i = [(\mathtt{T}, \ldots, \mathtt{T})\mathbf{a}_1 \cdots \mathbf{a}_i]_2$ and $\mathbf{0} \xrightarrow{\mathbf{a}_1 \cdots \mathbf{a}_i}$, for every $i \geq 1$. Let $\mathbf{x}_i \in \mathbb{N}^d$ such that $\mathbf{0} \xrightarrow{\mathbf{a}_1 \cdots \mathbf{a}_i} \mathbf{x}_i$. It is readily seen that $\mathbf{0} \xrightarrow{\mathbf{a}_1} \mathbf{x}_1 \cdots \xrightarrow{\mathbf{a}_k} \mathbf{x}_k \cdots$. This comes from the observation that $\xrightarrow{w}$ is forward deterministic. Moreover, the construction of $\mathcal{S}$ from $\mathcal{M}$ (see Figure 1c) entails that $q_{i-1} \xrightarrow{\mathbf{a}_i} q_i$ is a transition of $\mathcal{M}$, for every $i \geq 1$. It follows that $(q_0, \mathbf{0}) \xrightarrow{\mathbf{a}_1} (q_1, \mathbf{x}_1) \cdots \xrightarrow{\mathbf{a}_k} (q_k, \mathbf{x}_k) \cdots$ is a run in $\mathcal{M}$. ◀

B Proofs of Section 4

▶ **Lemma 13.** *For every n-block ρ and order k, if $\overline{\text{red}}_k(\rho)$ contains a factor of the form $\mathtt{copy}_k\rho'\overline{\mathtt{copy}}_k$ (or $\mathtt{push}_a\rho'\mathtt{pop}_b$ if $k = 1$), then $\text{dom}(\rho) = \emptyset$.*

Proof. By contradiction, suppose $s \in \text{dom}(\rho)$ and $\overline{\text{red}}_k(\rho)$ contains a factor of the form $\mathtt{copy}_k\rho'\overline{\mathtt{copy}}_k$ (or $\mathtt{push}_a\rho'\mathtt{pop}_b$ if $k = 1$). Let ρ_2 be one of the smallest such ρ'.

If $k = 1$ then we get that $\rho_2 = \varepsilon$, hence, $\overline{\text{red}}_k(\rho)$ contains a factor of the form $\mathtt{push}_a\mathtt{pop}_b$. If $a = b$, this contradicts the fact that $\overline{\text{red}}_k(\rho)$ is weakly reduced. If $a \neq b$, this contradicts the assumption that $\text{dom}(\rho) \neq \emptyset$.

If $k > 1$ then we get that $\rho_2 \in \mathtt{Op}^*_{k-1}$. We may write $\overline{\text{red}}_k(\rho) = \rho_1\mathtt{copy}_k\rho_2\overline{\mathtt{copy}}_k\rho_3$ for some ρ_1 and ρ_3. By Theorem 11, $s \in \text{dom}(\overline{\text{red}}_k(\rho))$. Therefore $(\rho_1\mathtt{copy}_k\rho_2)(s) \in \text{dom}(\overline{\mathtt{copy}}_k)$. As $\rho_2 \in \mathtt{Op}^*_{k-1}$, we necessarily have $\rho_2(s) = s$. By Lemma 12, $\text{red}(\rho_2) = \varepsilon$, and as it is in $\mathtt{Op}^*_{k-1}$, we derive that $\overline{\text{red}}_k(\rho_2) = \varepsilon$. Therefore, $\overline{\text{red}}_k(\rho) = \rho_1\mathtt{copy}_k\overline{\mathtt{copy}}_k\rho_3$, which contradicts the fact that $\overline{\text{red}}_k(\rho)$ is weakly reduced. ◀

C Proofs and comments of Section 5

▶ **Lemma 21.** *For every n-block ρ and every $i \geq 1$, $\text{red}(\rho) = \varepsilon$ if, and only if, $\text{red}(\rho^i) = \varepsilon$.*

Proof. We only prove the "if" direction as the "only if" direction is trivial. By contradiction, suppose that $i \geq 2$, $\text{red}(\rho^i) = \varepsilon$ and $\text{red}(\rho) \neq \varepsilon$. Let us decompose $\text{red}(\rho)$ as $\text{red}(\rho) = \overline{\rho_1}\rho_2\rho_1$ where ρ_1 is maximal in length. Note that $\rho_2 \neq \varepsilon$ since we would get $\text{red}(\rho) = \text{red}(\overline{\rho_1}\rho_1) = \varepsilon$ otherwise. By definition of red, it holds that $\text{red}(\rho^i) = \text{red}(\text{red}(\rho)^i) = \text{red}(\overline{\rho_1}\rho_2^i\rho_1) = \varepsilon$. So there exists a $\theta\bar{\theta}$ factor in $\overline{\rho_1}\rho_2^i\rho_1$. Recall that $\overline{\rho_1}\rho_2\rho_1$ is reduced. This means this $\theta\bar{\theta}$ factor is necessarily at the junction between to consecutive ρ_2. Formally, we get that $\rho_2 = \bar{\theta}\rho_3\theta$ for some ρ_3. Hence, $\text{red}(\rho) = \overline{\rho_1}\bar{\theta}\rho_3\theta\rho_1$, which contradicts the maximality of ρ_1. ◀

▶ **Corollary 22.** *For every n-block ρ and stack s such that ρ is iterable on s, if $\rho(s) \neq s$ then the infinite sequence $s, \rho(s), \rho^2(s), \ldots, \rho^i(s), \ldots$ contains no repetition.*

Proof. If the sequence $s, \rho(s), \rho^2(s), \ldots, \rho^i(s), \ldots$ contains a repetition, then there is a stack t satisfying $\rho^j(t) = t$ for some $j \geq 1$. This entails, by Lemmas 12 and 21, that $\text{red}(\rho) = \varepsilon$, hence, $\rho(s) = s$. ◀

▶ **Theorem 16.** *If the reachability tree of a d-dim n-PVASS $\mathcal{S}$ contains two nodes u and v such that u* subsumes *v (resp., u* strictly subsumes *v), then $\mathcal{S}$ has an infinite initialised run (resp., an infinite reachability set).*

Proof. We have $v : (q, \mathbf{x}, s)$ and $v' : (q, \mathbf{x} + \mathbf{y}, \rho(s))$, where $\mathbf{y}$ is a componentwise nonnegative vector, and ρ is the n-block on the run from v to v'. As $\mathbf{x} \leq \mathbf{x} + \mathbf{y}$, we know that vectorwise, the run from v to v' is applicable to v' (by monotony). As ρ is iterable on s, we know that stackwise, the run from v to v' is applicable to v'. Thus we can apply the run from v to v' on v', and obtain a new node $z : (q, \mathbf{x} + 2\mathbf{y}, \rho^2(s))$, and iterate the process to obtain an infinite sequence of nodes $v_i : (q, \mathbf{x} + i\mathbf{y}, \rho^i(s))$. We therefore have an infinite run in $\mathcal{S}$.

If furthermore v strictly subsumes v', then $\mathbf{y} \neq 0$ or $\rho(s) \neq s$, and we get that all these nodes are labelled by distinct configurations (this claim is obvious if $\mathbf{y} \neq 0$ and comes from Corollary 22 if $\rho(s) \neq s$). Thus $\mathcal{S}$ can reach infinitely many configurations and is thus unbounded. ◀

Non-completeness of the test. We detail here why the only run of the HOPVASS of Figure 2 does not contain any iterable subrun.

- One can show that the only possible configurations containing state B are of the form $(B, 1, [[ba^n]_1]_2)$. Moreover, the only run moving from $(B, 1, [[ba^n]_1]_2)$ to $(B, 1, [[ba^{n+1}]_1]_2)$ passes through the states C, D, E and performs the stack operations sequence $\rho = \mathtt{push}_a \mathtt{copy}_2 \mathtt{pop}_a^{n+1} \mathtt{pop}_b \mathtt{push}_b \mathtt{pop}_a^{n+1} \overline{\mathtt{copy}}_2$. $\overline{\mathrm{red}}_1(\rho) = \mathtt{pop}_a^n \mathtt{pop}_b \mathtt{push}_b \mathtt{push}_a^{n+1}$, and one can see it cannot be written as $\overline{\rho_{E_1}} \rho_{I_1} \rho_{E_1}$ with ρ_{I_1} containing no $\mathtt{pop}$ operation. Intuitively, this run adds an a at the top of the stack, copies the stack, and pops it until the b on the bottom before reconstructing it. As it needs to go to the bottommost symbol while adding a new symbol on top, it cannot be applied a second time, as it doesn't go deep enough anymore.
- Similarly, all possible configurations containing state C are of the form $(C, 0, [[ba^n]_1]_2)$, and the same reasoning applies.
- Configurations containing D are of the form $(D, k, [[ba^n]_1[ba^{n-k}]_1]_2)$. The run going from $(D, k, [[ba^n]_1[ba^{n-k}]_1]_2)$ to $(D, k', [[ba^{n+1}]_1[ba^{n+1-k'}]_1]_2)$ passes through E, B, C, and performs the stack operations sequence $\rho = \mathtt{pop}_a^k \mathtt{pop}_b \mathtt{push}_b \mathtt{push}_a^n \overline{\mathtt{copy}}_2 \mathtt{push}_a \mathtt{copy}_2 \mathtt{pop}_a^{k'}$. It is easy to see that either $\overline{\mathrm{red}}_1(\rho)$ is not iterable, or $\overline{\mathrm{red}}_2(\rho)$ is not iterable (depending on k and k'). The run going from $(D, k, [[ba^n]_1[ba^{n-k}]_1]_2)$ to $(D, k', [[ba^n]_1[ba^{n-k'}]_1]_2)$ stays on D and performs the stack operations sequence $\mathtt{pop}_a^{k-k'}$, which is not iterable.
- Configuration containing E are of the form $(E, k, [[ba^n]_1[ba^{n-k}]_1]_2)$. The run going from $(E, k, [[ba^n]_1[ba^{n-k}]_1]_2)$ to $(E, k', [[ba^{n+1}]_1[ba^{n+1-k'}]_1]_2)$ is similar to the previous case. The run going from $(E, k, [[ba^n]_1[ba^{n-k}]_1]_2)$ to $(E, k', [[ba^n]_1[ba^{n-k'}]_1]_2)$ decreases the counter value, and thus cannot be iterated.

▶ **Lemma 17.** *If ρ is a block applicable to $[]_n$, then for every order k, $\overline{\mathrm{red}}_k(\rho) = \mathrm{red}(\rho|_k)$ and $\overline{\mathrm{red}}_k(\rho)$ contains no $\overline{\mathtt{copy}}_k$ (no $\mathtt{pop}_a$ if $k = 1$).*

Proof. Suppose there is a $\overline{\mathtt{copy}}_k$ in $\rho|_k$ at position j in $\rho = \theta_1 \cdots \theta_m$. As ρ is applicable to $[]_n$, there is a $i < j$ such that $\theta_i = \mathtt{copy}_k$, there is no other $\mathtt{copy}_k$ and $\overline{\mathtt{copy}}_k$ between i and j (w.l.o.g) and $\theta_{i+1} \cdots \theta_{j-1}(\theta_1 \cdots \theta_i([]_n)) = \theta_1 \cdots \theta_i([]_n)$. Thus, by Lemma 12, $\mathrm{red}(\theta_{i+1} \cdots \theta_{j-1}|_k) = \varepsilon$. Therefore, $\overline{\mathrm{red}}_k(\rho)$ does not contain any $\overline{\mathtt{copy}}_k$.

Furthermore, for orders lower than k, both red and $\overline{\mathrm{red}}_k$ coincide syntactically, therefore $\overline{\mathrm{red}}_k(\rho)$ is reduced for red, and by unicity of the reduced k-block, we get the result. ◀

▶ **Lemma 18.** *For every n-stack s and orders $k < k' \leq n$, it holds that $||s||_k \leq ||s||_{k'}$.*

Proof. Given a stack s, and two orders $k < k'$, by definition of application, we have $\rho_{\mathrm{top}_{k'}(s)}|_k([]_k) = \mathrm{top}_k(s) = \rho_{\mathrm{top}_k(s)}([]_k)$. By minimality of the reduced k-block and the definition of restriction, we have $||s||_k = |\rho_{\mathrm{top}_k(s)}| \leq |\rho_{\mathrm{top}_{k'}(s)}|_k| \leq |\rho_{\mathrm{top}_{k'}(s)}| = ||s||_{k'}$. ◀

▶ **Lemma 19.** *Given m, there are at most $(2(|\Gamma|+n-1)-1)^m$ n-stacks s such that $||s||_n = m$.*

Proof. An n-stack s has norm m if and only if its reduced n-block has length m. Such an n-block is a word in $\mathtt{Op}_{n-1} \cup \{\mathtt{copy}_n\}^m$, hence there are at most $(2(|\Gamma| + n - 1) - 1)^m$ such words. ◀

Stronger Tradeoffs for Orthogonal Range Querying in the Semigroup Model

Swaroop N. Prabhakar

The Institute of Mathematical Sciences, HBNI, Chennai, 600113, India
npswaroop@imsc.res.in

Vikram Sharma

The Institute of Mathematical Sciences, HBNI, Chennai, 600113, India
vikram@imsc.res.in

Abstract

In this paper, we focus on lower bounds for data structures supporting orthogonal range querying on m points in n-dimensions in the semigroup model. Such a data structure usually maintains a family of "canonical subsets" of the given set of points and on a range query, it outputs a disjoint union of the appropriate subsets. Fredman showed that in order to prove lower bounds in the semigroup model, it suffices to prove a lower bound on a certain combinatorial tradeoff between *two parameters*: (a) the total sizes of the canonical subsets, and (b) the total number of canonical subsets required to cover all query ranges. In particular, he showed that the arithmetic mean of these two parameters is $\Omega(m \log^n m)$. We strengthen this tradeoff by showing that the *geometric mean* of the same two parameters is $\Omega(m \log^n m)$.

Our second result is an alternate proof of Fredman's tradeoff in the one dimensional setting. The problem of answering range queries using canonical subsets can be formulated as factoring a specific boolean matrix as a product of two boolean matrices, one representing the canonical sets and the other capturing the appropriate disjoint unions of the former to output all possible range queries. In this formulation, we can ask what is an optimal data structure, i.e., a data structure that minimizes the sum of the two parameters mentioned above, and how does the balanced binary search tree compare with this optimal data structure in the two parameters? The problem of finding an optimal data structure is a non-linear optimization problem. In one dimension, Fredman's result implies that the minimum value of the objective function is $\Omega(m \log m)$, which means that at least one of the parameters has to be $\Omega(m \log m)$. We show that both the parameters in an optimal solution have to be $\Omega(m \log m)$. This implies that balanced binary search trees are near optimal data structures for range querying in one dimension. We derive intermediate results on factoring matrices, not necessarily boolean, while trying to minimize the norms of the factors, that may be of independent interest.

2012 ACM Subject Classification Information systems → Multidimensional range search

Keywords and phrases range querying, lower bounds, matrix factorization, Lagrange dual function

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.45

Acknowledgements We are thankful to Dr. Prashant Batra and Dr. Arijit Ghosh for pointing out the relevant literature on the eigenvalues of tridiagonal matrices, and range querying, respectively. We would also like to thank the anonymous refrees for their careful comments; in particular, one of the reviewers who spotted an error in our argument in Lemma 4.

38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018).
Editors: Sumit Ganguly and Paritosh Pandya; Article No. 45; pp. 45:1–45:14

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

1 Introduction

Orthogonal range querying is one of the fundamental problems in computational geometry. The **range querying** problem is the following: Given a set X of m points in $\mathbb{R}^n$ and a *range* set $\mathcal{R}$ of subsets of points in $\mathbb{R}^n$, the goal is to pre-process the set X into a data structure so that given a query range $R \in \mathcal{R}$, the set of points in $X \cap R$ can be output efficiently. For *orthogonal range querying*, a range is simply an axis aligned box in $\mathbb{R}^n$. In this paper, we only consider the problem of orthogonal range querying. Sometimes, we are also interested in the number of points in the set $X \cap R$. The case where we output all the points in $X \cap R$ is called range reporting and the case where we only report the number of points in $X \cap R$ is called range counting. Other types of queries include whether or not $X \cap R$ is empty and so on. To capture these different types of queries in the range querying framework, it is typical to associate with every point $X_i \in X$ a weight w_i, where w_i comes from a commutative semigroup $(S, +)$[1]. Then, for every query range R, the output is $\sum_{X_i \in X \cap R} w_i$. For instance, for the orthogonal range reporting problem, we can take the semigroup $(2^X, \cup)$ and set $w_i = \{X_i\}$.; for the range counting problem, we can take the semigroup $(\mathbb{N}, +)$ and set $w_i = 1$.

Data structures for range querying typically store certain **canonical subsets** of the input set X and on a query range R, the query algorithm comes up with a set of disjoint canonical subsets such that their union is exactly $X \cap R$. The performance of a data structure for range querying is measured by the time spent in answering a query, the space requirement of the data structure and also the preprocessing cost involved in building the data structure. Often, the preprocessing is ignored as the data structure is built only once. In the dynamic setting where operations such as delete and insert are permitted, update time is also important. Most data structures for geometric problems are described in the real RAM model [16] and the pointer-machine model [1, 2]. A popular data structure for orthogonal range querying is the *range tree* which was introduced by Bentley [3]; for an exposition, see [4, chap. 5]. For orthogonal range reporting on m points in n dimensions, the range tree can be built in time $O(m \log^{n-1} m)$ and every query can be answered in time $O(\log^n m + k)$, where k is the number of points in the output. The query time though can be improved to $O(\log^{n-1} m + k)$ through a technique called fractional cascading [7, 14]. These upper bounds have been subsequently improved for range querying in various computation models [1, 2].

Fredman gave some of the first lower bounds on orthogonal range querying in the semigroup model [10, 11]. These lower bounds are in the dynamic setting where insertions and deletions are allowed. More specifically, in [11], he showed that for any m, there is a sequence of m operations consisting of insert, delete and querying such that the time required for this sequence is $\Omega(m \log^n m)$ in the semigroup model. The crux of Fredman's lower bound argument lies in exploiting a certain combinatorial tradeoff between the sizes of the canonical sets and the number of canonical sets needed to answer all the query ranges [15, p. 69, Lemma 9]. To state this more precisely, we set up some definitions and notations.

From here on, we take the set X to be the n-dimensional grid of m points, i.e.,

$$X := \left\{1, \ldots, \left\lfloor m^{1/n} \right\rfloor\right\}^n.$$

The n coordinates of the point X_i are represented as X_{ij}, $j = 1, \ldots, n$. A **one-sided range**

[1] Another algebraic structure from which weights are assigned are groups [12, 9], but in this paper we restrict ourselves to the case where weights come from a semigroup.

query on the set X takes as input a $Y \in \mathbb{R}^n$ and outputs

$$R_Y := \{X_i \in X : X_i \leq Y\},$$

where $X_i \leq Y$ iff $X_{ij} \leq Y_j$, for all $j \in [n]$. In this paper, range queries will always be one-sided. Corresponding to m points in X, we have m range queries whose outputs are $R_j := R_{X_j}$, for $j \in [m]$. A set $\mathcal{D} := \{W_1, \ldots, W_r\}$ of subsets of X is a **data structure for answering range queries on** X if every output to a range query on X is represented as a *disjoint union* over the canonical sets $W_1, \ldots, W_r$. Let $\langle R_j \rangle_{\mathcal{D}}$ denote the set of indices of W_k's used in the representation of R_j. Fredman showed the following result:

▶ **Proposition 1.** *If $\mathcal{D}$ is a data structure that answers range queries on X then*

$$\sum_{k=1}^{r} |W_k| + \sum_{j=1}^{m} |\langle R_j \rangle_{\mathcal{D}}| = \Omega(m \log^n m).$$

This tradeoff between the sizes of canonical sets and the number of canonical sets needed for covering all the query ranges is the central theme of this paper. Many more lower bounds on orthogonal range querying in different computation models are also known (see [1, p. 7] and [2, p. 11]).

The tradeoff in Proposition 1 gives us a lower bound on the arithmetic mean of the total size of the canonical sets and the number of canonical sets needed for covering query ranges. But in practice, data structures such as range trees need $\Theta(m \log^n m)$ many canonical sets for orthogonal range querying on m points and the total size of these sets is $\Theta(m \log^n m)$ as well. In view of this fact, we prove the following stronger result:

▶ **Theorem 2.** *If $\mathcal{D}$ is a data structure that answers range queries on X then*

$$\left(\sum_{k=1}^{r} |W_k|\right)\left(\sum_{j=1}^{m} |\langle R_j \rangle_{\mathcal{D}}|\right) = \Omega(m^2 \log^{2n} m).$$

From the AM-GM inequality it is clear that Theorem 2 implies Proposition 1. Theorem 2 also implies that any data structure $\mathcal{D}$ that is *tight* with respect to Proposition 1, i.e.,

$$\sum_{k=1}^{r} |W_k| + \sum_{j=1}^{m} |\langle R_j \rangle_{\mathcal{D}}| = \Theta(m \log^n m), \tag{1}$$

must satisfy:

$$\sum_{k=1}^{r} |W_k| = \Theta(m \log^n m) \text{ and } \sum_{j=1}^{m} |\langle R_j \rangle_{\mathcal{D}}| = \Theta(m \log^n m). \tag{2}$$

The proof of Theorem 2 will be given in Section 2.

From (2), we see that the balanced binary search tree is an optimal data structure in the *boolean* setting where the outputs are represented as disjoint unions over canonical sets. This leaves open the possibility of existence of a more efficient data structure that does not take disjoint unions of its canonical subsets but takes their weighted sum in order to represent an output. In such a relaxed setting, Proposition 1 and Theorem 2 are not applicable. Can balanced binary search tree be an optimal data structure even in this setting? In Section 3, we give a positive answer to this question.

In order to account for data structures that take weighted sums of their canonical sets, we will reinterpret range querying differently from Proposition 1. In the proof of Proposition 1 [15, p. 69, Lemma 9 and 10], the problem of range querying is interpreted in a graph theoretic setting, namely expressing a bipartite graph as a "product" of two bipartite graphs. This can also be interpreted in terms of matrices [6, Sec. 2.2]. Let $U_{m\times r}$ be the incidence matrix of the set X with the canonical sets W_k's, i.e, $U_{ik} = 1$ iff $X_i \in W_k$. Similarly, define $V_{r\times m}$ to be the incidence matrix of the canonical sets W_k's and the outputs R_j's. Let $R_{m\times m}$ be the matrix whose columns are the characteristic vectors of the sets R_j's. To give a proof of Proposition 1, it suffices to derive a lower bound on the optimal value of the following optimization problem:

$$\min \left(||U||_F^2 + ||V||_F^2 \right) \text{ subject to } UV = R, \tag{3}$$

where $R \in \{0,1\}^{m\times m}$, $U \in \{0,1\}^{m\times r}$ and $V \in \{0,1\}^{r\times m}$ and $||.||_F$ refers to the Frobenius norms of the respective matrices.

In this optimization based formulation of the problem, the objective function aims to minimize the sum of the two parameters we are interested in: The total size of the canonical sets, $||U||_F^2$ and the total number of canonical sets needed to cover all the query ranges, $||V||_F^2$. Every data structure that supports range querying in one dimension is a feasible solution to the problem above. When the entries of the matrices are restricted to be boolean, Proposition 1 implies that the optimal value of the objective function is $\Omega(m \log m)$. Hence, from (2), we see that for an optimal solution $(U_{\text{bool}}, V_{\text{bool}})$ of (3) we must have,

$$||U_{\text{bool}}||_F^2 = \Theta(m \log m) \text{ and } ||V_{\text{bool}}||_F^2 = \Theta(m \log m).$$

To extend these bounds for data structures that take weighted sums of their canonical sets, we consider the relaxation of the problem in (3) where the matrix entries are allowed to be arbitrary reals. For an optimal solution (U^*, V^*) of this relaxation, we show that

$$||U^*||_F^2 = \Omega(m \log m) \text{ and } ||V^*||_F^2 = \Omega(m \log m).$$

The lower bounds above imply that the balanced binary search tree is near optimal **not only** in the boolean framework but also in a more relaxed setting.

The main idea in proving the lower bounds above is to use the Lagrangian dual of the relaxation and show that

$$||U^*||_F^2 = \text{Trace}((R^t R)^{1/2}) \text{ and } ||V^*||_F^2 = \text{Trace}((R^t R)^{1/2}), \tag{4}$$

where we take the principal square-root of a matrix [13]. The result in (4) holds for an **arbitrary** matrix R (see Theorem 3) and is the key technical ingredient in our proof. Then, by taking R to be the lower triangular all ones matrix, which corresponds to range querying in one dimension, and by using some well established results on explicit forms for the eigenvalues of tri-diagonal matrix (in this case $(R^t R)^{-1}$), we show that

$$\text{Trace}((R^t R)^{1/2}) = \Omega(m \log m).$$

We believe that our proof gives more understanding on the optimality of Fredman's lower bound by relating it to some intrinsic parameters of the matrix R, which is a natural representation of the range query problem in one-dimension. To the best of our knowledge, our proof is more general than the existing proofs in the literature; e.g., [17] works only in the boolean setting. Whether our proof technique can be generalized to obtain an alternative proof of Proposition 1 in all dimensions remains an open and interesting question.

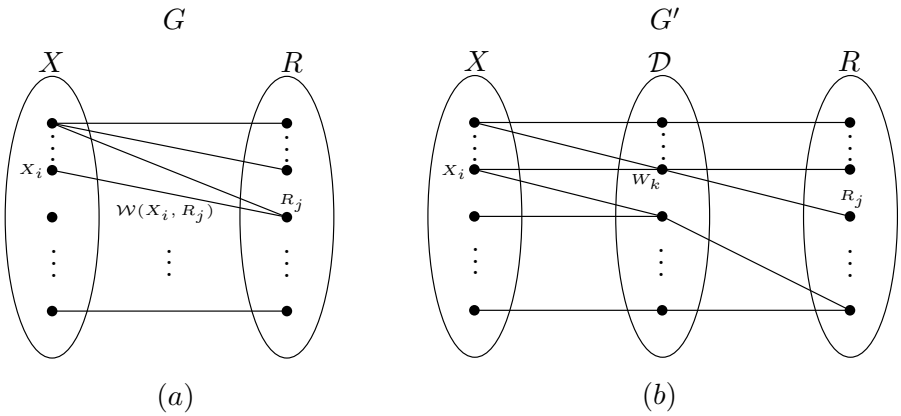


Figure 1 (a): Bipartite graph G with the vertex sets X, R and the edge set E. (b): Tripartite graph G' with vertex sets X, $\mathcal{D}$ and R.

2 Tradeoff between Sizes of Canonical Sets and Outputs to Query Ranges

In this section, we will prove Theorem 2. The proof of the theorem follows by altering the argument in the proof of Proposition 1. Before we proceed, we first present some high level details of the proof of Proposition 1. For the complete details, we refer to [15, p. 69].

The argument relies on interpreting range querying in a graph theoretic setting: Consider the weighted bipartite graph $G(X \cup R, E)$, where $R := \{R_1, R_2, \ldots, R_m\}$ and the edge set $E := \{(X_i, R_j) : X_i \in R_j\}$; see Figure 1(a) for illustration. The edge $(X_i, R_j) \in E$ is assigned the weight

$$\mathcal{W}(X_i, R_j) := \frac{1}{\prod_{\kappa=1}^{n} (X_{j\kappa} - X_{i\kappa} + 1)}. \tag{5}$$

The graph G can be "factored" into a tripartite graph G' whose vertex set is $\{X \cup \mathcal{D} \cup R\}$. There is an edge (X_i, W_k) iff $X_i \in W_k$ and there is an edge (W_k, R_j) iff $k \in \langle R_j \rangle_{\mathcal{D}}$; see Figure 1(b) for an illustration. Note that the edges of G are a disjoint union over the sets $\{(X_i, R_j) : X_i \in W_k, k \in \langle R_j \rangle_{\mathcal{D}}\}$, for all $k \in [r]$, as every R_j is a disjoint union of W_k's. For every set W_k, define

$$I_k := \{X_i \in X : X_i \in W_k\}$$

i.e., the edges of G' incident on W_k from the left and

$$O_k := \left\{R_j \in R : k \in \langle R_j \rangle_{\mathcal{D}}\right\},$$

i.e., the edges of G' incident on W_k from the right. Therefore,

$$|I_k| = |W_k| \text{ and } \sum_{j=1}^{m} |\langle R_j \rangle_{\mathcal{D}}| = \sum_{k=1}^{r} |O_k|.$$

At a high level, the proof of Proposition 1 can be broken down into two steps [15, p. 69, Lemma 9 and p. 71, Lemma 10]:

Step 1. For every $k \in [r]$,

$$\sum_{\substack{X_i \in I_k \\ R_j \in O_k}} \mathcal{W}(X_i, R_j) = \sum_{\substack{X_i \in I_k \\ R_j \in O_k}} \frac{1}{\prod_{\kappa=1}^{n} (X_{j\kappa} - X_{i\kappa} + 1)} \leq (2\pi)^n (|I_k| + |O_k|), \tag{6}$$

where n is the dimension of the points in X. The outline of the proof is as follows. Let $M_j = \max\{X_{ij} : X_i \in I_k\}$. Define the **associate sets** of W_k as

$$B := \{(M_1 - X_{i1}, M_2 - X_{i2}, \ldots, M_n - X_{in}) : X_i \in I_k\}$$

and

$$C := \{(X_{j1} - M_1, X_{j2} - M_2, \ldots, X_{jn} - M_n) : R_j \in O_k\}.$$

Since every term of the form

$$(X_{j\kappa} - X_{i\kappa} + 1) = (M_\kappa - X_{i\kappa} + X_{j\kappa} - M_\kappa + 1),$$

the summation in (6) is equal to

$$\sum_{\substack{X_i \in I_k \\ R_j \in O_k}} \frac{1}{\prod_{\kappa=1}^{n} (X_{j\kappa} - X_{i\kappa} + 1)} = \sum_{\substack{u \in B \\ v \in C}} \frac{1}{\prod_{\kappa=1}^{n} (u_\kappa + v_\kappa + 1)},$$

where u_κ and v_κ are non-negative integers. Then, by applying a generalized version of Hilbert's inequality for points with natural numbers as their coordinates, one obtains

$$\sum_{\substack{u \in B \\ v \in C}} \frac{1}{\prod_{\kappa=1}^{n} (u_\kappa + v_\kappa + 1)} \leq (2\pi)^n (|B| + |C|) = (2\pi)^n (|I_k| + |O_k|). \tag{7}$$

Step 2. The second step is to show that the total sum of weights over all edges in E satisfies

$$\sum_{(X_i, R_j) \in E} \mathcal{W}(X_i, R_j) = \sum_{k=1}^{r} \sum_{\substack{X_i \in I_k \\ R_j \in O_k}} \mathcal{W}(X_i, R_j) = \Omega(m \log^n m). \tag{8}$$

By summing the inequality in (6) over all W_k and applying the lower bound from (8), we get the claim in Proposition 1.

We now give the proof of Theorem 2.

Proof. Since $\sum_{k=1}^{r} |W_k| = \sum_{k=1}^{r} |I_k|$ and $\sum_{j=1}^{m} |\langle R_j \rangle_{\mathcal{D}}| = \sum_{k=1}^{r} |O_k|$, we will show that

$$\sum_{k=1}^{r} |I_k| \cdot \sum_{k=1}^{r} |O_k| = \Omega(m^2 \log^{2n} m).$$

To prove this, consider a pair of canonical sets, W_k and W_ℓ. Using the same weight function as in (5) on the edges of G, we have

$$\sum_{\substack{X_i \in I_k \\ R_j \in O_k}} \sum_{\substack{X_c \in I_\ell \\ R_d \in O_\ell}} \mathcal{W}(X_i, R_j) \cdot \mathcal{W}(X_c, R_d) = \sum_{\substack{X_i \in I_k \\ R_j \in O_k}} \sum_{\substack{X_c \in I_\ell \\ R_d \in O_\ell}} \frac{1}{\prod_{\kappa=1}^{n} (X_{j\kappa} - X_{i\kappa} + 1)(X_{d\kappa} - X_{c\kappa} + 1)}. \tag{9}$$

Define the **associate sets** B, C of W_k as in Step 1; sets B' and C' are defined analogously for W_ℓ. Notice that $|B| = |I_k|$, $|C| = |O_k|$, $|B'| = |I_\ell|$ and $|C'| = |O_\ell|$. Using the associate sets, equation (9) can be expressed as

$$\sum_{\substack{X_i \in I_k \\ R_j \in O_k}} \sum_{\substack{X_c \in I_\ell \\ R_d \in O_\ell}} \mathcal{W}(X_i, R_j) \cdot \mathcal{W}(X_c, R_d) = \sum_{\substack{u \in B \\ v \in C}} \sum_{\substack{u' \in B' \\ v' \in C'}} \frac{1}{\prod_{\kappa=1}^{n} (u_\kappa + v_\kappa + 1)(u'_\kappa + v'_\kappa + 1)}. \quad (10)$$

The importance of the associate sets of W_k and W_ℓ is that they have non-negative coordinates and they are in some sense independent of the actual coordinates of the points in X, since difference choices of the point set X give the same associate sets. We now use the upper bound from (7) to upper bound the RHS of (10). Since the RHS of (10) can be interpreted as a function over $2n$ dimensional points, we define the following sets in $\mathbb{R}^{2n}$

$$\mathcal{B} := \{(u, u') : u \in B, u' \in B'\}, \mathcal{C} := \{(v, v') : v \in C, v' \in C'\}$$

and

$$\mathcal{B}' := \{(u, v') : u \in B, v' \in C'\}, \mathcal{C}' := \{(v, u') : v \in C, u' \in B'\}.$$

The pair of sets $(\mathcal{B}, \mathcal{C})$ and $(\mathcal{B}', \mathcal{C}')$ allow us to express the RHS of (10) in two different ways as:

$$\begin{aligned}\sum_{\substack{u \in B \\ v \in C}} \sum_{\substack{u' \in B' \\ v' \in C'}} \frac{1}{\prod_{\kappa=1}^{n} (u_\kappa + v_\kappa + 1)(u'_\kappa + v'_\kappa + 1)} &= \sum_{\substack{\mathcal{U} \in \mathcal{B} \\ \mathcal{V} \in \mathcal{C}}} \frac{1}{\prod_{\kappa=1}^{2n} (\mathcal{U}_\kappa + \mathcal{V}_\kappa + 1)} \\ &= \sum_{\substack{\mathcal{U}' \in \mathcal{B}' \\ \mathcal{V}' \in \mathcal{C}'}} \frac{1}{\prod_{\kappa=1}^{2n} (\mathcal{U}'_\kappa + \mathcal{V}'_\kappa + 1)}.\end{aligned}$$

From (7), the RHS of the equalities above can be upper bounded as

$$\sum_{\substack{\mathcal{U} \in \mathcal{B} \\ \mathcal{V} \in \mathcal{C}}} \frac{1}{\prod_{\kappa=1}^{2n} (\mathcal{U}_\kappa + \mathcal{V}_\kappa + 1)} \leq (2\pi)^{2n}(|\mathcal{B}| + |\mathcal{C}|) \text{ and } \sum_{\substack{\mathcal{U}' \in \mathcal{B}' \\ \mathcal{V}' \in \mathcal{C}'}} \frac{1}{\prod_{\kappa=1}^{2n} (\mathcal{U}'_\kappa + \mathcal{V}'_\kappa + 1)} \leq (2\pi)^{2n}(|\mathcal{B}'| + |\mathcal{C}'|).$$

So, from the two inequalities above and (10) we obtain

$$\begin{aligned}\sum_{\substack{X_i \in I_k \\ R_j \in O_k}} \sum_{\substack{X_c \in I_\ell \\ R_d \in O_\ell}} \mathcal{W}(X_i, R_j) \cdot \mathcal{W}(X_c, R_d) &\leq (2\pi)^{2n} \min\{|\mathcal{B}| + |\mathcal{C}|, |\mathcal{B}'| + |\mathcal{C}'|\}. \\ &= (2\pi)^{2n} \min\{|B||B'| + |C||C'|, |B||C'| + |B'||C|\}, \\ &= (2\pi)^{2n} \min\{|I_k||I_\ell| + |O_k||O_\ell|, |I_k||O_\ell| + |I_\ell||O_k|\}\end{aligned}$$

Therefore, for an arbitrary pair W_k, W_ℓ, we have

$$\sum_{\substack{X_i \in I_k \\ R_j \in O_k}} \sum_{\substack{X_c \in I_\ell \\ R_d \in O_\ell}} \mathcal{W}(X_i, R_j) \cdot \mathcal{W}(X_c, R_d) \leq (2\pi)^{2n}(|I_k||O_\ell| + |I_\ell||O_k|). \quad (11)$$

Note that every edge (X_i, R_j) in E maps to a unique path (X_i, W_k, R_j) in the graph G'. Hence the sum of the LHS of (11) over all possible pairs of W_k and W_ℓ gives us the sum of the product of weights of all possible pairs of edges (X_i, R_j) and (X_c, R_d) in E. Hence from (8) we obtain that

$$\sum_{\substack{W_k \\ W_\ell}} \sum_{\substack{X_i \in I_k \\ R_j \in O_k}} \sum_{\substack{X_c \in I_\ell \\ R_d \in O_\ell}} \mathcal{W}(X_i, R_j) \cdot \mathcal{W}(X_c, R_d) = \sum_{\substack{(X_i, R_j) \in E \\ (X_c, R_d) \in E}} \mathcal{W}(X_i, R_j) \cdot \mathcal{W}(X_c, R_d) = \Omega(m^2 \log^{2n} m).$$

(12)

Similarly, summing the RHS of (11) over all pairs of W_k and W_ℓ and using the fact that $|I_k| = |W_k|$ and $\sum_{k=1}^{r} |O_k| = \sum_{j=1}^{m} |\langle R_j \rangle_{\mathcal{D}}|$, we get

$$\sum_{W_k} \sum_{W_\ell} (2\pi)^{2n} (|I_k||O_\ell| + |I_\ell||O_k|) = 2 \cdot (2\pi)^{2n} \left(\sum_{j=1}^{m} |\langle R_j \rangle_{\mathcal{D}}| \right) \left(\sum_{k=1}^{r} |W_k| \right).$$

Therefore, from (11), (12) and the equality above, we conclude that

$$\left(\sum_{j=1}^{m} |\langle R_j \rangle_{\mathcal{D}}| \right) \left(\sum_{k=1}^{r} |W_k| \right) = \Omega(m^2 \log^{2n} m). \quad \blacktriangleleft$$

3 Optimality of the Balanced Binary Search Tree

In this section, we will show the optimality of the balanced binary search tree in a relaxed framework where the data structures are allowed to take a weighted sum of their canonical subsets. Let $X := \{1, 2, \ldots, m\}$. We again consider the set of one sided range queries: For $j \in X$, output $R_j := \{i \in X : i \leq j\}$. Let $\mathcal{D} := \{W_1, W_2, \ldots, W_r\}$ be an arbitrary data structure that answers range queries on X. Proposition 1 in one dimension reduces to:

$$\left(\sum_{k=1}^{r} |W_k| \right) + \left(\sum_{j=1}^{m} |\langle R_j \rangle_{\mathcal{D}}| \right) = \Omega(m \log m). \tag{13}$$

To extend the lower bound above for data structures that are allowed to take weighted sums of their canonical subsets, we will reinterpret range querying in a different setting. The problem of range querying can be interpreted in a linear algebraic setting as follows: Consider the 0/1 matrix U whose rows are indexed by the m numbers and columns are indexed by the r sets, W_k's. The (i, j)th entry is one iff the number i is a member of W_j. Consider the range query that asks for all the numbers less than or equal to the jth number. The output is a union of at most, say ℓ sets. Then, R_j, which is an m-dimensional vector with ones from the jth position onwards can be expressed as a linear combination of at most ℓ columns of U. Let $\mathbf{v}_j$ be this linear combination, i.e.

$$R_j = U\mathbf{v}_j$$

where $\mathbf{v}_j$ is a 0/1 vector. Since there are m distinct range queries, we have $\mathbf{v}_1, \ldots, \mathbf{v}_m$ such vectors. If V is the matrix with these vectors as its columns, then our observation regarding these m range queries can be succinctly represented by the following matrix equation

$$UV = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 1 & 1 & 1 & \cdots & 1 \end{bmatrix} =: R, \tag{14}$$

where R is the lower triangular matrix with all ones on and below the diagonal and the rows of R are indexed by the numbers $m, m-1, \ldots, 1$ and the columns by $R_m, R_{m-1}, \ldots, R_1$.

Also, $U \in \{0,1\}^{m \times r}$ and $V \in \{0,1\}^{r \times m}$. Now,

$$\sum_{i=1}^{r} |W_i| = ||U||_F^2 \quad \text{and} \quad \sum_{j=1}^{m} |\langle R_j \rangle| = ||V||_F^2,$$

where $||A||_F$ denotes the Frobenius norm of the matrix A. So, in terms of factorizations of R as a product of U and V, proving the claim in (13) is equivalent to lower bounding the optimal value of the following optimization problem

$$\begin{aligned} &\min \left(||U||_F^2 + ||V||_F^2 \right) \\ &\text{subject to } UV = R, U \in \{0,1\}^{m \times r}, V \in \{0,1\}^{r \times m}. \end{aligned} \tag{15}$$

In order to consider data structures that may take weighted sum of their canonical subsets instead of disjoint unions, we focus on the following relaxation of the problem in (15): Given an arbitrary matrix $T \in \mathbb{R}^{m \times m}$,

$$\begin{aligned} &\min \left(||U||_F^2 + ||V||_F^2 \right) \\ &\text{subject to } UV = T, U \in \mathbb{R}^{m \times r}, V \in \mathbb{R}^{r \times m}. \end{aligned} \tag{16}$$

It is clear that a lower bound on the optimal value of (16), when T is taken to be R, is also a lower bound on the optimal value of (15). So, we first prove that

▶ **Theorem 3.** *Any optimal solution (U^*, V^*) for the optimization problem in (16) satisfies:*

$$||U^*||_F^2 = \text{Trace}((T^tT)^{1/2}) \text{ and } ||V^*||_F^2 = \text{Trace}((T^tT)^{1/2}),$$

where the matrix $(T^tT)^{1/2}$ is defined to be the principal square root of the matrix T^tT [13, p .20, Theorem 1.29].

Proof. The Lagrangian dual function associated with the problem in (16) is defined as [5, p. 216]

$$\inf_{U,V} L(U,V,\Lambda) = \inf_{U,V} \left(||U||_F^2 + ||V||_F^2 + \sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{r} (T_{ij} - U_{ik} \cdot V_{kj})\lambda_{ij} \right), \tag{17}$$

where $\Lambda \in \mathbb{R}^{m \times m}$. The Lagrange dual problem is now defined as

$$\max_{\Lambda} \left(\inf_{U,V} L(U,V,\Lambda) \right), \tag{18}$$

where $\Lambda \in \mathbb{R}^{m \times m}$. Any optimal solution (U^*, V^*) for the primal problem satisfies the following inequality:

$$||U^*||_F^2 + ||V^*||_F^2 \geq \max_{\Lambda} \left(\inf_{U,V} L(U,V,\Lambda) \right).$$

From the inequality above, we see that it suffices to lower bound the optimal value of the dual problem in order to prove the required claim. Consider the function

$$\inf_{U,V} L(U,V,\Lambda).$$

Applying the optimality conditions, we take the partial derivative of $L(U,V,\Lambda)$ with respect to variables in U to get the following matrix equation:

$$\nabla_U L(U,V,\Lambda) = 2U^t - V\Lambda^t = 0. \tag{19}$$

Similarly, taking derivative with respect to variables in V, we get

$$\nabla_V L(U, V, \Lambda) = 2V - U^t\Lambda = 0. \tag{20}$$

From (19) and (20), we have

$$V\Lambda^t = 2U^t \quad \text{and} \quad U^t\Lambda = 2V. \tag{21}$$

Since the dual problem is convex and aims to maximize the Lagrangian dual function in (17) with respect to Λ, we apply the first order condition to $L(U, V, \Lambda)$ with respect to Λ to get

$$UV = T.$$

By left multiplying the first equation in (21) by U, we get

$$\begin{aligned} UV\Lambda^t &= 2UU^t \\ T\Lambda^t &= 2UU^t \text{ since } UV = T. \end{aligned}$$

Using the equality above and the fact that $||U||_F^2 = \text{Trace}(UU^t)$, we get

$$||U||_F^2 = \frac{1}{2}\text{Trace}(T\Lambda^t).$$

Similarly, we can show that

$$||V||_F^2 = \frac{1}{2}\text{Trace}(\Lambda^t T).$$

Therefore, for an optimal solution Λ of the dual problem, we have

$$||U||_F^2 = \frac{1}{2}\text{Trace}(T\Lambda^t), \quad ||V||_F^2 = \frac{1}{2}\text{Trace}(\Lambda^t T).$$

Since $\text{Trace}(T\Lambda^t) = \text{Trace}(\Lambda^t T)$, it suffices to show that the trace of $\Lambda^t T$ is $2 \cdot \text{Trace}((T^t T)^{1/2})$ to prove the theorem.

We begin by multiplying the transpose of the second equation in (21) with the first equation in (21) to obtain

$$\Lambda^t UV\Lambda^t = 4(UV)^t.$$

Since $UV = T$, we see that any optimal solution for the dual problem must satisfy:

$$\Lambda^t T\Lambda^t = 4T^t.$$

Multiplying the equality above by T from the right, we get

$$(\Lambda^t T)^2 = 4T^t T.$$

Since $T^t T$ is a positive semidefinite matrix [2], it is diagonalizable. Assuming Q to be the $m \times m$ matrix whose columns are the eigenvectors of $T^t T$ and $\gamma_1 \geq \gamma_2 \geq \cdots \geq \gamma_m$ to be the eigenvalues of $T^t T$, we can express the equality above as

$$(\Lambda^t T)^2 = 4Q^{-1}\Gamma Q$$

[2] Here we use the fact that any matrix A that can be written as $A = B^t B$, is positive semidefinite.

where Γ is the diagonal matrix with γ_k's being the kth diagonal entry. Therefore, we have

$$(\Lambda^t T) = 2Q^{-1}\Gamma^{1/2}Q,$$

where $Q^{-1}\Gamma^{1/2}Q$ is defined to be the *principle square root of* T^tT whose eigenvalues are $\sqrt{\gamma_1}, \sqrt{\gamma_2}, \ldots, \sqrt{\gamma_m}$ and for all k, $\sqrt{\gamma_k} \in \mathbb{R}_{\geq 0}$. So,

$$\text{Trace}(\Lambda^t T) = 2\text{Trace}((T^tT)^{1/2}) = 2\sum_{k=1}^{m}\sqrt{\gamma_k}.$$

Hence, we conclude that

$$||U^*||_F^2 = \text{Trace}((T^tT)^{1/2}) \quad ||V^*||_F^2 = \text{Trace}((T^tT)^{1/2}). \qquad \blacktriangleleft$$

We bound the trace of $(R^tR)^{1/2}$ in the following result:

▶ **Lemma 4.** *Let R be the matrix as in* (14)*. The trace of the principal square root of R^tR satisfies*

$$\text{Trace}((R^tR)^{1/2}) = \sum_{k=1}^{m}\sqrt{\gamma_k} = \Omega(m\log m),$$

where γ_k, $k \in [m]$, are the eigenvalues of the matrix R^tR.

Proof. To compute γ_k's, consider the inverse matrix

$$(R^tR)^{-1} = R^{-1}(R^{-1})^t.$$

The inverse of the matrix R is the bidiagonal matrix

$$R^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & \ldots & 0 \\ -1 & 1 & 0 & 0 & \ldots & 0 \\ 0 & -1 & 1 & 0 & \ldots & 0 \\ 0 & 0 & -1 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & 1 \end{bmatrix}$$

So, the matrix $R^{-1}(R^{-1})^t$ is the following tridiagonal matrix

$$R^{-1}(R^{-1})^t = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & \ldots & 0 \\ -1 & 2 & -1 & 0 & 0 & \ldots & 0 \\ 0 & -1 & 2 & -1 & 0 & \ldots & 0 \\ 0 & 0 & -1 & 2 & -1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \ldots & -1 \\ 0 & 0 & 0 & 0 & 0 & \ldots & 2 \end{bmatrix},$$

which obtained as a special case of tridiagonal matrices of the form

$$\begin{bmatrix} a+d & b & 0 & 0 & 0 & \ldots & 0 \\ b & a & b & 0 & 0 & \ldots & 0 \\ 0 & b & a & b & 0 & \ldots & 0 \\ 0 & 0 & b & a & b & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \ldots & b \\ 0 & 0 & 0 & 0 & 0 & \ldots & a+c \end{bmatrix},$$

by substituting $a = 2$, $b = -1$, $d = -1$, and $c = 0$. From [8, p. 27], we know that the roots of the characteristic polynomial of the matrix above is given by

$$a + 2b\cos\theta \tag{22}$$

where θ varies over the m zeros of the following function

$$\frac{\sin(m+1)\theta - \frac{c+d}{b}\sin m\theta + \frac{cd}{b^2}\sin(m-1)\theta}{\sin\theta}.$$

Substituting $a = 2$, $b = d = -1$, and $c = 0$ in the formula above, we obtain the following expression

$$\frac{\sin(m+1)\theta - \sin m\theta}{\sin\theta}.$$

Simplifying the formula above using the sum-to-product identity[3], we get

$$\frac{2\sin\frac{(m+1)\theta - m\theta}{2}\cos\frac{(m+1)\theta + m\theta}{2}}{\sin\theta} = \frac{2\sin(\theta/2)\cos(m\theta+\theta/2)}{2\sin(\theta/2)\cos(\theta/2)} = \frac{\cos(m\theta+\theta/2)}{\cos(\theta/2)}.$$

The expression above vanishes at the values

$$\left(\frac{2k-1}{2m+1}\right)\pi,$$

where $k = 1, 2, \ldots, m$. Substituting in (22), we see that the eigenvalues of $R^{-1}(R^{-1})^t$ are

$$2\left(1 - \cos\left(\frac{2k-1}{2m+1}\right)\pi\right) = 4\sin^2\left(\frac{2k-1}{4m+2}\right)\pi,$$

for $k = 1, 2, \ldots, m$, where we use the identity $(1 - \cos x) = 2\sin^2 x/2$ above. So, the eigenvalues of R^tR are

$$\gamma_k = \frac{1}{4\sin^2\left(\frac{2k-1}{4m+2}\right)\pi},$$

for $k = 1, 2, \ldots, m$, and, therefore, the trace of the principal square root of R^tR is

$$\text{Trace}((R^tR)^{1/2}) = \sum_{k=1}^{m}\sqrt{\gamma_k} = \sum_{k=1}^{m}\frac{1}{2\sin\left(\frac{2k-1}{4m+2}\right)\pi}.$$

Since for $k = 1, \ldots, m$, the reciprocal of the sine functions is a monotonically decreasing concave function, the summation above can be lower bounded as follows:

$$\text{Trace}((R^tR)^{1/2}) = \sum_{k=1}^{m}\frac{1}{2\sin\frac{(2k-1)\pi}{4m+2}} \geq \int_1^m \frac{dx}{2\sin\frac{(2x-1)\pi}{4m+2}} = \int_1^m \frac{\csc\frac{(2x-1)\pi}{4m+2}}{2}dx.$$

Substituting $y = \frac{(2x-1)\pi}{4m+2}$ and using the fact that $\int \csc y \cdot dy = \ln|\tan(y/2)|$, we obtain

$$\text{Trace}((R^tR)^{1/2}) \geq \frac{4m+2}{4\pi}\left(\ln\left(\tan\frac{(2m-1)\pi}{(8m+4)}\right) - \ln\left(\tan\frac{\pi}{(8m+4)}\right)\right).$$

[3] Namely, $\sin A - \sin B = 2\sin\frac{(A-B)}{2}\cos\frac{(A+B)}{2}$

As m tends to infinity, the term $\tan((2m-1)\pi/(8m+4))$ tends to one. Therefore, we have

$$\text{Trace}((R^tR)^{1/2}) = \Omega\left(m \ln\left(\cot\frac{\pi}{(8m+4)}\right)\right).$$

From the Taylor series of the cotangent function, we know that for $m \geq 1$, $\cot\frac{\pi}{(8m+4)} = \Theta(m)$. Therefore,

$$\text{Trace}((R^tR)^{1/2}) = \Omega(m \log m).$$ ◀

We note that from Theorem 3 and Lemma 4, we have a stronger conclusion than in (13). From (13), we can only infer that one of the two parameters is $\Omega(m \log m)$ whereas for data structures such as balanced binary search trees, both the parameters are $\Theta(m \log m)$. Our proof shows that

$$||U_{\text{BST}}||_F = \Theta(||U^*||_F) \text{ and } ||V_{\text{BST}}||_F = \Theta(||V^*||_F),$$

where (U^*, V^*) is an optimal solution for the problem in (15) and $(U_{\text{BST}}, V_{\text{BST}})$ are the matrices U and V corresponding to the balanced binary search tree. Therefore, binary search trees are optimal with respect to both the parameters, The total size of the canonical sets and the total number of canonical sets needed for covering all the query ranges. Also, our proof implies that balanced binary search trees are near optimal in a more relaxed framework where the data structures are allowed to take weighted sums of their canonical subsets.

4 Conclusion

In this paper, we have shown that there is a stronger tradeoff between the sizes of canonical sets and the outputs to query ranges than the one shown by Fredman. In Section 3, we show the optimality of balanced binary search trees in a more general setting where we allow weighted combinations of the canonical sets in the data structure. A natural continuation would be to generalize this proof to higher dimensions. One can start by bounding the integrality gap between an optimal solution in the boolean setting and an optimal solution of the relaxation. In one dimension, our proof shows that this gap is at most a constant.

We also believe that the optimization problem introduced in Section 3 is interesting in its own right. For instance, the lower bound for the average complexity of the partial sums problem [12] in the one dimensional setting can be obtained from the lower bound on the optimization problem in (15).

References

1. Pankaj Agarwal. Range Searching. Technical report, Duke University, 2016. URL: `https://users.cs.duke.edu/~pankaj/publications/surveys/rs3ed.pdf`.
2. Pankaj Agarwal and Jeff Erickson. Geometric Range Searching and its Relatives. Technical report, University of Illinois, 1997. URL: `http://jeffe.cs.illinois.edu/pubs/pdf/survey.pdf`.
3. Jon Louis Bentley. Multidimensional Divide-and-Conquer. *Commun. ACM*, 23(4):214–229, April 1980. `doi:10.1145/358841.358850`.
4. Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008.
5. Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

6 Richard A. Brualdi and Dragos Cvetkovic. *A Combinatorial Approach to Matrix Theory and Its Applications*. Chapman and Hall/CRC, 1st edition, 2008.

7 Bernard Chazelle and Leonidas J. Guibas. Fractional Cascading: A Data Structuring Technique. *Algorithmica*, 1(1):133–162, November 1986. doi:10.1007/BF01840440.

8 J. F. Elliott. The characteristic roots of certain real symmetric matrices. Master's thesis, University of Tennessee, 1953.

9 M. Fredman and M. Saks. The Cell Probe Complexity of Dynamic Data Structures. In *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*, STOC '89, pages 345–354, New York, NY, USA, 1989. ACM. doi:10.1145/73007.73040.

10 M. L. Fredman. The Inherent Complexity of Dynamic Data Structures which Accommodate Range Queries. In *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)*, pages 191–199, October 1980. doi:10.1109/SFCS.1980.47.

11 Michael L. Fredman. A Lower Bound on the Complexity of Orthogonal Range Queries. *J. ACM*, 28(4):696–705, October 1981.

12 Michael L. Fredman. The Complexity of Maintaining an Array and Computing its Partial Sums. *J. ACM*, 29(1):250–260, January 1982. doi:10.1145/322290.322305.

13 N. Higham. *Functions of Matrices*. Society for Industrial and Applied Mathematics, 2008. URL: https://epubs.siam.org/doi/abs/10.1137/1.9780898717778.

14 G. S. Lueker. A Data Structure for Orthogonal Range Queries. In *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*, pages 28–34, October 1978. doi:10.1109/SFCS.1978.1.

15 Kurt Mehlhorn. *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*. Springer-Verlag New York, Inc., New York, NY, USA, 1984.

16 Franco P. Preparata and Michael I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

17 A. Toni. Lower Bounds on Zero–One Matrices. *Linear Algebra and its Applications*, 376:275–282, 2004. doi:10.1016/j.laa.2003.06.018.

Parameterized Dynamic Cluster Editing

Junjie Luo[1]
Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Berlin, Germany
Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China
School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing, China
junjie.luo@campus.tu-berlin.de, luojunjie@amss.ac.cn

Hendrik Molter[2]
Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Berlin, Germany
h.molter@tu-berlin.de

André Nichterlein
Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Berlin, Germany
andre.nichterlein@tu-berlin.de

Rolf Niedermeier
Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Berlin, Germany
rolf.niedermeier@tu-berlin.de

Abstract

We introduce a dynamic version of the NP-hard Cluster Editing problem. The essential point here is to take into account dynamically evolving input graphs: Having a cluster graph (that is, a disjoint union of cliques) that represents a solution for a first input graph, can we cost-efficiently transform it into a "similar" cluster graph that is a solution for a second ("subsequent") input graph? This model is motivated by several application scenarios, including incremental clustering, the search for compromise clusterings, or also local search in graph-based data clustering. We thoroughly study six problem variants (edge editing, edge deletion, edge insertion; each combined with two distance measures between cluster graphs). We obtain both fixed-parameter tractability as well as parameterized hardness results, thus (except for two open questions) providing a fairly complete picture of the parameterized computational complexity landscape under the perhaps two most natural parameterizations: the distance of the new "similar" cluster graph to (i) the second input graph and to (ii) the input cluster graph.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis, Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases graph-based data clustering, goal-oriented clustering, compromise clustering, NP-hard problems, fixed-parameter tractability, parameterized hardness

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.46

1 Introduction

The NP-hard Cluster Editing problem [6, 31], also known as Correlation Clustering [5], has developed into one of the most popular graph-based data clustering problems in algorithm theory. Given an undirected graph, the task is to transform it into a disjoint union of cliques (also known as cluster graph) by performing a minimum number of edge

[1] Supported by CAS-DAAD Joint Fellowship Program for Doctoral Students of UCAS. Work done while with TU Berlin.
[2] Supported by the DFG, project MATE (NI 369/17).

38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018).
Editors: Sumit Ganguly and Paritosh Pandya; Article No. 46; pp. 46:1–46:15

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

modifications (deletions or insertions). Being NP-hard, Cluster Editing gained high popularity in studies concerning parameterized algorithmics, e.g. [1, 4, 8, 9, 12, 18, 20, 22, 25]. To the best of our knowledge, to date these parameterized studies mostly focus on a "static scenario". Chen et al. [12] are an exception by also looking at temporal and multilayer graphs. In their work, the input is a set of graphs (multilayer) or an ordered list of graphs (temporal), in both cases defined over the same vertex set. The goal is to transform each input graph into a cluster graph such that, in the multilayer case, the number of vertices in which any two cluster graphs may differ is bounded, and in the temporal case, the number of vertices in which any consecutive (with respect to their position in the list) cluster graphs may differ is bounded. In this work, we introduce a dynamic view on Cluster Editing by, roughly speaking, assuming that the input graph changes. Thus we seek to efficiently and effectively adapt an existing solution, namely a corresponding cluster graph. In contrast to the work of Chen et al. [12], we do not assume that all future changes are known. We consider the scenario where given an input graph, we only know changes that lie immediately ahead, that is, we know the "new" graph that the input graph changes to. Motivated by the assumption that the "new" cluster graph should only change moderately but still be a valid representation of the data, we parameterize both on the number of edits necessary to obtain the "new" cluster graph and the difference between the "old" and the "new" cluster graph. We finally remark that there have been previous parameterized studies of dynamic (or incremental) graph problems, but they deal with coloring [23], domination [16, 2], or vertex deletion [3, 26] problems.

Mathematical model. In principle, the input for a dynamic version of a static problem X are two instances I and I' of X, a solution S for I, and an integer d. The task is to find a solution S' for I' such that the distance between S and S' is upper-bounded by d. Often, there is an additional constraint on the size of S'. Moreover, the symmetric difference between I and I' is used as a parameter for the problem many times. We arrive at the following "original dynamic version" of Cluster Editing (phrased as decision version).

Original Dynamic Cluster Editing
Input: Two graphs G_1 and G_2 and a cluster graph G_c over the same vertex set, and integers k and d such that $|E(G_1) \oplus E(G_c)| \leq k$.
Question: Is there a cluster graph G' for G_2 such that $|E(G_2) \oplus E(G')| \leq k$ and $\mathrm{dist}(G', G_c) \leq d$?

Herein, $\oplus$ denotes the symmetric difference between two sets and $\mathrm{dist}(\cdot, \cdot)$ is a generic distance function for cluster graphs, which we discuss later. Moreover, G_c is supposed to be the "solution" given for the input graph G_1. However, since the question in this problem formulation is independent from G_1 we can remove this graph from the input and arrive at the following simplified version of the problem. For the remainder of this paper we focus on this simplified version of Dynamic Cluster Editing.

Dynamic Cluster Editing
Input: A graph G and a cluster graph G_c over the same vertex set, and two integers: a budget k and a distance upper bound d.
Question: Is there a cluster graph G' for G such that $|E(G) \oplus E(G')| \leq k$ and $\mathrm{dist}(G', G_c) \leq d$?

There are many different distance measures for cluster graphs [28, 29]. Indeed, we will study two standard ways of measuring the distance between two cluster graphs. One is called

classification error distance, which measures the number of vertices one needs to move to make two cluster graphs the same – we subsequently refer to it as *matching-based distance*. The other is called disagreement distance, which is the symmetric distance between two edge sets – we subsequently refer to it as *edge-based distance*. Notably, the edge-based distance upper-bounds the matching-based distance. We give formal definitions in Section 2.

Motivation and related work. Beyond parameterized algorithmics and static Cluster Editing, dynamic clustering in general has been subject to many studies, mostly in applied computer science [32, 15, 14, 34, 33, 10]. We mention in passing that there are also close ties to reoptimization (e.g., [7, 30]) and parameterized local search (e.g., [17, 19, 21, 23, 27]).

There are several natural application scenarios that motivate the study of Dynamic Cluster Editing. Next, we list four of them.

Dynamically updating an existing cluster graph. Dynamic Cluster Editing can be interpreted to model a smooth transition between cluster graphs, reflecting that "customers" working with clustered data in a dynamic setting may only tolerate a moderate change of the clustering from "one day to another" since "revolutionary" transformations would require too dramatic changes in their work. In this spirit, when employing small parameter values, Dynamic Cluster Editing has kind of an evolutionary flavor with respect to the history of the various cluster graphs in a dynamic setting.

Editing a graph into a target cluster graph. For a given graph G, there may be many cluster graphs which are at most k edge modifications away. The goal then is to find one of these which is close to the given target cluster graph G_c since in a corresponding application one is already "used to" work with G_c. Alternatively, the editing into the target cluster graph G_c might be too expensive (that is, $|E(G) \oplus E(G')|$ is too big), and one has to find one with small enough modification costs but being still close to the target G_c.

Local search for an improved cluster graph. Here the scenario is that one may have found an initial clustering expressed by G_c, and one searches for another solution G' for G within a certain local region around G_c (captured by our parameter d).

Editing into a compromise clustering. When focusing on the edge-based distance, one may generalize the definition of Dynamic Cluster Editing by allowing G_c to be any graph (not necessarily a cluster graph). This may be used as a model for "compromise cluster editing" in the sense that the goal cluster graph then is a compromise for a cluster graph suitable for both input graphs since it is close to both of them.

Our results. We investigate the (parameterized) computational complexity of Dynamic Cluster Editing. We study Dynamic Cluster Editing as well as two restricted versions where only edge deletions ("Deletion") or edge insertions ("Completion") are allowed. We show that all problem variants (notably also the completion variants, whose static counterpart is trivially polynomial-time solvable) are NP-complete even if the input graph G is already a cluster graph. Table 1 surveys our main parameterized complexity results.

The general versions of Dynamic Cluster Editing all turn out to be parameterized intractable (W[1]-hard) by the single natural parameters "budget k" and "distance d"; however, when both parameters are combined, one achieves a polynomial kernel. We also derive a generic approach towards fixed-parameter tractability for several deletion and completion variants with respect to the budget k as well as with respect to the distance d. Proofs of results marked with ($\star$) are deferred to a full version of the paper.

Table 1 Result overview for Dynamic Cluster Editing. We primarily categorize the problem variants by the distance measure (Matching, Edge) they use and secondarily by the allowed modification operation. NP-completeness for all problem variants (last column) even holds if the input graph G is a cluster graph. PK stands for polynomial kernel.

		Parameter					
	Problem Variant	$k+d$		k	d		
Matching	Editing	FPT (PK)	Thm. 3	W[1]-h Thm. 2	W[1]-h	Thm. 2	NP-c Thm. 1
	Deletion	FPT (PK)		*open*	W[1]-h		NP-c
	Completion	FPT (PK)		*open*	FPT Thm. 4		NP-c
Edge	Editing	FPT (PK)	Thm. 3	W[1]-h Thm. 2	W[1]-h	Thm. 2	NP-c Thm. 1
	Deletion	FPT (PK)		FPT	Thm. 4		NP-c
	Completion	FPT (PK)		FPT	Thm. 4		NP-c

2 Preliminaries and Problems Variants

In this section we give a brief overview on concepts and notation of graph theory and parameterized complexity theory that are used in this paper. We also give formal definitions of the distance measures for cluster graphs we use and of our problem variants.

Graph-theoretic concepts and notations. Given a graph $G = (V, E)$, we say that a vertex set $C \subseteq V$ is a *clique in* G if $G[C]$ is a complete graph. We say that a vertex set $C \subseteq V$ is *isolated* in G if there is no edge $\{u, v\} \in E$ with $u \in C$ and $v \in V \setminus C$. A P_3 is a path with three vertices. We say that vertices $u, v, w \in V$ form an induced P_3 in G if $G[\{u, v, w\}]$ is a P_3. We say that an edge $\{u, v\} \in E$ is part of a P_3 in G if there is a vertex $w \in V$ such that $G[\{u, v, w\}]$ is a P_3. Analogously, we say that a non-edge $\{u, v\} \notin E$ is part of a P_3 in G if there is a vertex $w \in V$ such that $G[\{u, v, w\}]$ is a P_3. A graph $G = (V, E)$ is a *cluster graph* if for all $u, v, w \in V$ we have that $G[\{u, v, w\}]$ is not a P_3, or in other words, P_3 is a forbidden induced subgraph for cluster graphs.

Distance measures for cluster graphs. A cluster graph is simply a disjoint union of cliques. We use two basic distance measures for cluster graphs [28, 29]. The first one is called "matching-based distance" and counts how many vertices have to be moved from one cluster to another to make two cluster graphs the same. It is formally defined as follows.

▶ **Definition 1** (Matching-based distance)**.** Let $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ be two cluster graphs defined over the same vertex set. Let $B(G_1, G_2) = (V_1 \uplus V_2, E, w)$ be a weighted complete bipartite graph, where each vertex $u \in V_1$ corresponds to a cluster in G_1, denoted by $C_u \subseteq V$, and each vertex $v \in V_2$ corresponds to a cluster of G_2, denoted by $C_v \subseteq V$. The weight of the edge between $u \in V_1$ and $v \in V_2$ is $w(\{u, v\}) = |C_u \cap C_v|$. Let W be the weight of a maximum-weight matching in $B(G_1, G_2)$. The *matching-based distance* d_M between G_1 and G_2 is $d_M(G_1, G_2) := |V| - W$.

The second distance measure is called "edge-based distance" and simply measures the symmetric distance between the edge sets of two cluster graphs.

▶ **Definition 2** (Edge-based distance)**.** Let $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ be two cluster graphs defined over the same vertex set. The *edge-based distance* d_E between G_1 and G_2 is $d_E(G_1, G_2) := |E_1 \oplus E_2|$.

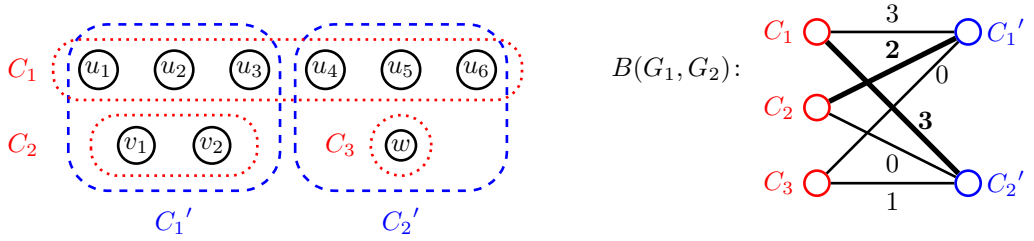


Figure 1 An illustration of the two distance measures. On the left side, red dotted boundaries represent cliques in cluster graph G_1, and blue dashed boundaries represent cliques in cluster graph G_2. The bipartite graph on the right side is the edge-weighted bipartite graph $B(G_1, G_2)$. The maximum-weight matching for $B(G_1, G_2)$ is formed by the two edges represented by the two bold lines.

See Figure 1 for an example illustration of two cluster graphs G_1 and G_2 defined over the same vertex set $V = \{u_1, u_2, u_3, u_4, u_5, u_6, v_1, v_2, w\}$. In G_1 there are three cliques (clusters) $C_1 = \{u_1, u_2, u_3, u_4, u_5, u_6\}$, $C_2 = \{v_1, v_2\}$ and $C_3 = \{w\}$. In G_2 there are two cliques $C_1' = \{u_1, u_2, u_3, v_1, v_2\}$ and $C_2' = \{u_4, u_5, u_6, w\}$. Then in $B(G_1, G_2)$ we have three vertices on the left side for the cliques in G_1 and two vertices on the right side for the cliques in G_2. A maximum-weight matching for $B(G_1, G_2)$ matches C_1 with C_2' and C_2 with C_1', and has weight $W = 5$. Thus we have $d_M(G_1, G_2) = |V| - W = 9 - 5 = 4$, while $d_E(G_1, G_2) = 3^2 + 2 \cdot 3 + 1 \cdot 3 = 18$.

Problem names and definitions. In the following we present the six problem variants we are considering. We use Dynamic Cluster Editing as a basis for our problem variants. In Dynamic Cluster Deletion we add the restriction that $E(G') \subseteq E(G)$ and in Dynamic Cluster Completion we add the restriction that $E(G) \subseteq E(G')$. For each of these three variants we distinguish a matching-based version and an edge-based version, where the generic "dist" in the problem definition of Dynamic Cluster Editing is replaced by d_M and d_E, respectively. This gives us a total of six problem variants. We use the following abbreviations for our problem names. The letters "DC" stand for "Dynamic Cluster", and "Matching Dist" is short for "Matching-Based Distance". Analogously, "Edge Dist" is short for "Edge-Based Distance". As an example, we abbreviate Dynamic Cluster Editing with Matching-Based Distance as DCEditing (Matching Dist). All other problem variants are abbreviated in an analogous way.

Parameterized complexity. A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet. We call the second component the *parameter* of the problem. A parameterized problem is *fixed-parameter tractable* (in the complexity class FPT) if there is an algorithm that solves each instance (I, r) in $f(r) \cdot |I|^{O(1)}$ time, for some computable function f. A parameterized problem L admits a *polynomial kernel* if there is a polynomial-time algorithm that transforms each instance (I, r) into an instance (I', r') such that $(I, r) \in L$ if and only if $(I', r') \in L$ and $|(I', r')| \leq f(r)$, for some computable function f. If a parameterized problem is hard for the parameterized complexity class W[1], then it is (presumably) not in FPT. The complexity class W[1] is closed under parameterized reductions, which may run in FPT-time and additionally set the new parameter to a value that exclusively depends on the old parameter.

3 Intractability Results

In this section we first show that all problem variants of Dynamic Cluster Editing are NP-complete even if the input graph G is already a cluster graph. Intuitively, this means that on top of the NP-hard task of transforming a graph into a cluster graph, it is computationally hard to improve an already found clustering (with respect to being closer to the target cluster graph). In particular, while the dynamic versions of Cluster Completion are NP-complete, it is simple to see that classical Cluster Completion is solvable in polynomial time. In a second part we show W[1]-hardness results both for budget parameter k and for distance parameter d for several variants of Dynamic Cluster Editing.

▶ **Theorem 1.** *All considered problem variants of* Dynamic Cluster Editing *are NP-complete, even if the input graph G is a cluster graph.*

Next, we present several parameterized hardness results showing that for certain problem variants we cannot hope for fixed-parameter tractability. Formally, we show the following.

▶ **Theorem 2.** DCEditing (Matching Dist) *and* DCEditing (Edge Dist) *are W[1]-hard with respect to the budget k. The following problems are W[1]-hard with respect to the distance d:* DCEditing (Matching Dist), DCDeletion (Matching Dist), DCEditing (Edge Dist), *and* DCDeletion (Edge Dist).

As a representative for the results of Theorem 2, we present a parameterized reduction showing that DCEditing (Matching Dist) is W[1]-hard when parameterized by the budget k. The remaining results are deferred to a full version of the paper.

▶ **Lemma 1.** DCEditing (Matching Dist) *is NP-complete and W[1]-hard with respect to the budget k, even if the input graph G is a cluster graph.*

Proof. We present a parameterized reduction from Clique, where given a graph G_0 and an integer ℓ, we are asked to decide whether G_0 contains a complete subgraph of order ℓ. Clique is W[1]-hard when parameterized by ℓ [13]. Given an instance (G_0, ℓ) of Clique, we construct an instance (G, G_c, k, d) of DCEditing (Matching Dist) as follows.

The construction is illustrated in Figure 2. Let $n = |V(G_0)|$. We first construct G. For every vertex v of G_0, create a clique C_v of size $\ell^7 + \ell^4 + \ell^2$. For every edge e of G_0, create a clique C_e of size $\ell^4 + 2$. Lastly, create a big clique C_B of size ℓ^8. Note that G is already a cluster graph. Next we construct G_c. We first create ℓ cliques D_i of size $n\ell^3$ for each $1 \leq i \leq \ell$. Every D_i contains ℓ^3 vertices in every C_v in G. In other words, every C_v in G contains ℓ^3 vertices in every D_i in G_c. Then create a big clique D_B which contains all vertices in C_B and ℓ^7 vertices in every C_v. For every vertex v of G_0, create clique D_v which contains ℓ^2 vertices in C_v and one vertex in every C_e for $v \in e$. Lastly, for every edge e create D_e which contains ℓ^4 vertices in C_e. Set $k = \binom{\ell}{2}(2\ell^4 + 1) + \ell\binom{\ell-1}{2}$ and set $d = d_0 - \ell(\ell - 1)$, where $d_0 = d_M(G, G_c)$ is the matching-based distance between G and G_c, which is computed as follows.

To compute $d_M(G, G_c)$, we need to find an optimal matching in $B(G, G_c)$, the weighted bipartite graph between G and G_c. First, in an optimal matching D_B must be matched with C_B since $|C_B \cap D_B| = \ell^8 > |C_v \cap D_B| = \ell^7$ for any $v \in V(G_0)$ and $C_B \subseteq D_B$. Similarly, D_e must be matched with C_e for every $e \in E(G_0)$. Then the remaining n cliques C_v in G need to be matched to ℓ cliques D_i and n cliques D_v in G_c. Since $|C_v \cap D_i| = \ell^3 > |C_v \cap D_v| = \ell^2$ for any $v \in V(G_0)$ and $1 \leq i \leq \ell$, it is always better to match C_v with some D_i. Since there are only ℓ cliques D_i, we can choose any ℓ cliques from $\{C_v \mid v \in V(G_0)\}$ to be matched

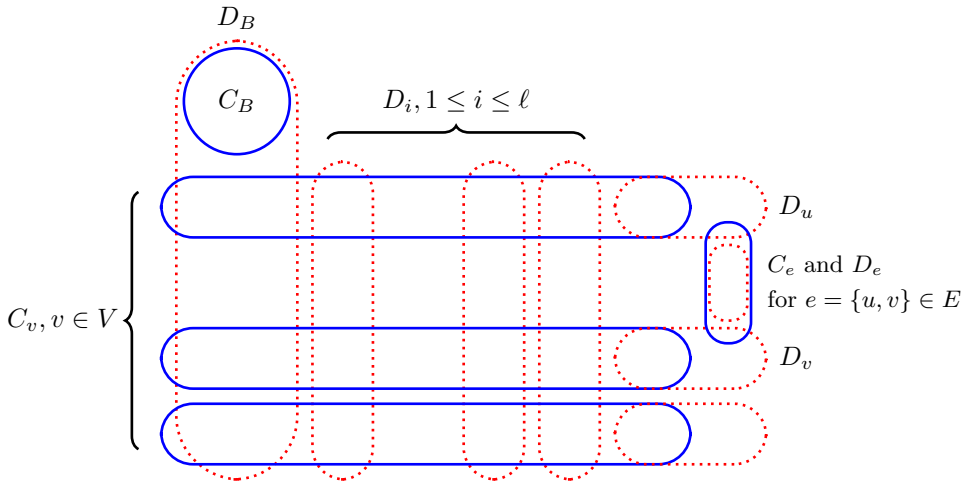


Figure 2 Illustration of the constructed instance for the proof of Lemma 1. Blue solid borders represent cliques in G and red dotted borders represent cliques in G_c. One horizontal long blue border represents a clique C_v in G. It has $\ell + 2$ parts and each part is contained in one clique of G_c. The first part contains ℓ^7 vertices which are contained in the big clique D_B of G_c. The following ℓ parts each contain ℓ^3 vertices which are contained in the ℓ cliques D_i of G_c, and the last part contains ℓ^2 vertices which are contained in D_v of G_c.

with D_i for $1 \leq i \leq \ell$ and the remaining $n - \ell$ cliques to be matched with D_v. Thus we have many different matchings in $B(G, G_c)$ which have the same maximum weight, and each of them corresponds to choosing ℓ different cliques from $\{C_v \mid v \in V(G_0)\}$ to be matched with D_i for $1 \leq i \leq \ell$. For each optimal matching, there are ℓ *free* cliques D_v in G_c which are not matched.

This reduction works in polynomial time. We show that there is a clique of size ℓ in G_0 if and only if there is a cluster graph $G' = (V, E')$ such that $|E(G') \oplus E(G)| \leq k$ and $d_M(G', G_c) \leq d$.

($\Rightarrow$): Assume that there is a clique C^* of size ℓ in G_0. We modify the graph G as follows. First, for every edge e in the clique C^* partition the corresponding clique C_e in G into three parts; one part contains all vertices in D_e and the other two parts each have one vertex. After this we get $\ell(\ell - 1)$ single vertices. Since C^* is a clique, all these single vertices can be partitioned into ℓ groups such that each group has $\ell - 1$ vertices and all these $\ell - 1$ vertices are contained in the same D_v for some $v \in C^*$. Then for each $v \in C^*$, we combine the corresponding $\ell - 1$ vertices into one clique $C_v^{\ell-1}$. Denote the resulting graph as G'. For an illustration see Figure 3. Along the way to get G', we delete $\binom{\ell}{2}(2\ell^4 + 1)$ edges and add $\ell\binom{\ell-1}{2}$ edges, thus $|E(G) \oplus E(G')| = \binom{\ell}{2}(2\ell^4 + 1) + \ell\binom{\ell-1}{2} = k$. Next we show that $d_M(G', G_c) \leq d_0 - \ell(\ell - 1)$. Recall that an optimal matching in $B(G, G_c)$ can choose ℓ cliques from $\{C_v \mid v \in V(G_0)\}$ to be matched with D_i for $1 \leq i \leq \ell$. Now in $B(G, G_c)$ we can choose all cliques in $\{C_v \mid v \in C^*\}$ to be matched with D_i for $1 \leq i \leq \ell$, and then match $C_v^{\ell-1}$ with D_v for all $v \in C^*$. Then in the new matching we have ℓ additional edges between $C_v^{\ell-1}$ and D_v for $v \in C^*$, each with weight $\ell - 1$. Hence $d_M(G', G_c) \leq d_0 - \ell(\ell - 1)$.

($\Leftarrow$): Assume that there is a cluster graph $G' = (V, E')$ such that $|E' \oplus E(G)| \leq k$ and $d_M(G', G_c) \leq d$. Note that $k < \ell^7$, thus $k < |C_v|$ and $k < |C_B|$. Consequently, we can only modify edges between vertices in C_e. It is easy to see that in any optimal matching in $B(G', G_c)$, we still have that clique C_B must be matched with D_B and clique C_e must be matched with D_e for every $e \in E(G_0)$. And we should choose ℓ cliques from $\{C_v \mid v \in V(G_0)\}$

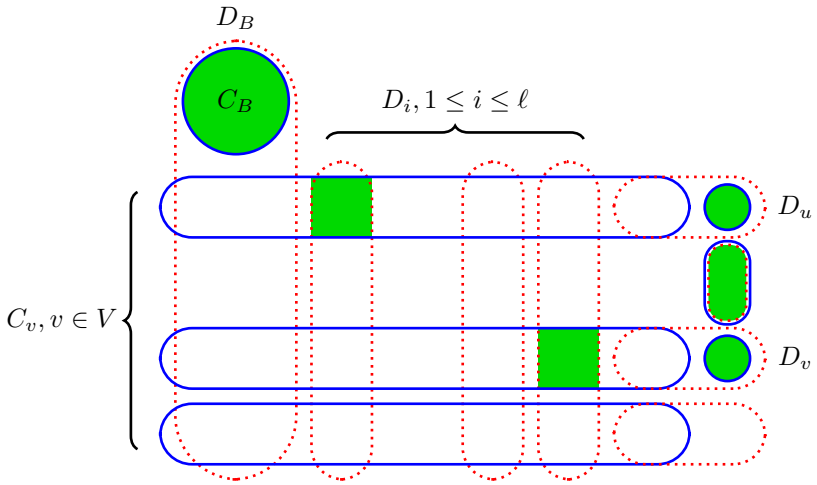


Figure 3 Illustration of a possible solution for the constructed instance (see Figure 2) in the proof of Lemma 1. Blue solid borders represent cliques in G' and red dotted borders represent cliques in G_c. Green shaded areas indicate how cliques of G' and G_c are matched. If two horizontal cliques of G' (blue) are matched with two of the ℓ vertical cliques of G_c, then the corresponding vertices are part of the clique and hence are adjacent. This means the cliques corresponding to the edge can be matched in the indicated way.

to be matched with D_i for $1 \leq i \leq \ell$, which creates ℓ free cliques D_v. Hence, to decrease the distance between G and G_c, or to increase the matching, we have to create new cliques to be matched with these ℓ free cliques D_v. Since for every D_v, except for vertices contained in C_v, it only contains single vertices from C_e with $v \in e$, to create new cliques we need to first separate D_e to get single vertices and then combine them. To decrease the distance by $\ell(\ell-1)$, we need to separate at least $\ell(\ell-1)$ single vertices from C_e. This will cost at least $\ell(\ell-1)(\ell^4+1) - \binom{\ell}{2} = \binom{\ell}{2}(2\ell^4+1)$ edge deletions if we always separate one C_e into three parts and get two single vertices. Then we need to combine these single vertices into at most ℓ cliques since there are at most ℓ free cliques D_v. This will cost at least $\ell\binom{\ell-1}{2}$ edge insertions if all these $\ell(\ell-1)$ single vertices can be partitioned into ℓ groups and each group has $\ell-1$ vertices. Since $k = \binom{\ell}{2}(2\ell^4+1) + \ell\binom{\ell-1}{2}$, we have that in the first step we have to choose $\binom{\ell}{2}$ cliques C_e and separate them into three parts and all these $\ell(\ell-1)$ single vertices are evenly distributed in ℓ free cliques D_v. This means that in G_0 we can select $\binom{\ell}{2}$ edges between ℓ vertices and each vertex has $\ell-1$ incident edges. Thus there is a clique of size ℓ in G_0. ◀

4 Fixed-Parameter Tractability Results

In this section we identify tractable cases for the considered variants of Dynamic Cluster Editing. We first show that all problem variants admit a polynomial kernel for the combination of the budget k and the distance d. Then we present further FPT-results with respect to single parameters.

4.1 Polynomial Kernels for the Combined Parameter $(k+d)$

In this section we present polynomial kernels with respect to the parameter combination $(k+d)$ for all considered variants of Dynamic Cluster Editing:

▶ **Theorem 3.** *The following problems admit an $O(k^2 + d^2)$-vertex kernel:* DCEditing (Matching Dist), DCDeletion (Matching Dist), *and* DCCompletion (Matching Dist). *The following problems admit an $O(k^2 + k \cdot d)$-vertex kernel:* DCEditing (Edge Dist), DCDeletion (Edge Dist), *and* DCCompletion (Edge Dist). *All kernels can be computed in $O(|V|^3)$ time.*

We describe data reduction rules that each take an instance $(G = (V, E), G_c = (V, E_c), k, d)$ as input and output a reduced instance that is a yes-instance if and only if the original instance is a yes-instance (of the corresponding problem variant). In the correctness proof of each reduction rule, we assume that all previous rules are not applicable.

We first use some well-known reduction rules for classical Cluster Editing [20] to get a graph which consists of isolated cliques plus one vertex set of size $k^2 + 2k$ that does not contain any isolated cliques. These rules remove edges that are part of $k + 1$ induced P_3s and add edges between non-adjacent vertex pairs that are part of $k + 1$ induced P_3s. We defer a formal description and correctness proofs of these rules to a full version of the paper. The reason we use these data reduction rules even though there are linear-vertex kernels for classical Cluster Editing [9, 11] is that they do not eliminate any possible solutions.

Now we introduce new reduction rules that are specific to our problem setting, allowing us to use $k + d$ to upper-bound the size of all remaining isolated cliques and their number to get a polynomial kernel. First, we observe that if there is a vertex set that forms an isolated clique both in G and G_c, then we can remove it since it has no influence on k or d in any problem variant. This is formalized in the next rule. We omit a formal correctness proof.

▶ **Reduction Rule 1.** If there is a vertex set $C \subseteq V$ that is an isolated clique in G and G_c, then remove all vertices in C from G and G_c.

The next rules deal with large cliques and allow us to either remove them or conclude that we face a no-instance.

▶ **Reduction Rule 2a** (Matching-based distance)**.** If there is a vertex set $C \subseteq V$ with $|C| > k + 2d + 1$ that is an isolated clique in G, then

- if for each vertex set $C' \subseteq V$ that is an isolated clique in G_c we have that $|C \cap C'| \leq d$, then answer NO,
- otherwise, if there is a vertex set $C' \subseteq V$ that is an isolated clique in G_c and $|C \cap C'| > d$, then we remove vertices in C from G and G_c and decrease d by $|C \setminus C'|$. Furthermore, if $d \geq 0$, then add a set C_d of $k + d + 1$ fresh vertices to V. Add all edges between vertices in C_d to E and add all edges between vertices in $C_d \cup (C' \setminus C)$ to G_c (if not already present).

▶ **Reduction Rule 2b** (Edge-based distance)**.** If there is a vertex set $C \subseteq V$ with $|C| > k$ that is an isolated clique in G, then decrease d by $|E_c| + \binom{|C|}{2} - 2|E(G_c[C])| - |E(G_c[V \setminus C])|$ and remove vertices in C from G and G_c.

If none of the previous rules are applicable, then we know that there are no large cliques left in the graph. The next rule allows us to conclude that we face a no-instance if there are too many small cliques left.

▶ **Reduction Rule 3.** If there are more than $2(k + d)$ isolated cliques in G, then output NO.

In the following we show that the rules we presented decrease the number of vertices of the instance to a number polynomial in $k + d$.

▶ **Lemma 2.** *Let $(G = (V, E), G_c = (V, E_c), k, d)$ be an instance of any one of the considered problem variants of* Dynamic Cluster Editing *that uses the matching-based distance. If none of the appropriate reduction rules applies, then $|V| \in O(k^2 + d^2)$.*

▶ **Lemma 3.** *Let $(G = (V, E), G_c = (V, E_c), k, d)$ be an instance of any one of the considered problem variants of* Dynamic Cluster Editing *that uses the edge-based distance. If none of the appropriate reduction rules applies, then $|V| \in O(k^2 + k \cdot d)$.*

Finally, we can apply all data reduction rules exhaustively in $O(|V|^3)$ time.

▶ **Lemma 4.** *Let $(G = (V, E), G_c = (V, E_c), k, d)$ be an instance of any one of the considered problem variants of* Dynamic Cluster Editing*. Then the respective reduction rules can be exhaustively applied in $O(|V|^3)$ time.*

It is easy to see that Theorem 3 directly follows from Lemma 2, Lemma 3, and Lemma 4. We remark that the number of edges that are not part of an isolated clique can be bounded by $O(k^3)$ [20].

4.2 Fixed-Parameter Tractable Cases for Single Parameters

In this section we show that several variants of Dynamic Cluster Editing are fixed-parameter tractable with respect to either the budget k or the distance d.

▶ **Theorem 4.** DCDeletion (Edge Dist) *and* DCCompletion (Edge Dist) *are in* FPT *with respect to the budget k.* DCCompletion (Matching Dist) *and* DCCompletion (Edge Dist) *are in* FPT *with respect to the distance d.*

All our FPT results are using the same approach: We reduce (in FPT time) the input to an instance of Multi-Choice Knapsack (MCK), formally defined as follows.

> Multi-Choice Knapsack (MCK)
> **Input:** A family of ℓ mutually disjoint sets $S_1, \ldots, S_\ell$ of items, a weight $w_{i,j}$ and a profit $p_{i,j}$ for each item $j \in S_i$, and two integers W and P.
> **Question:** Is it possible to select one item from each set S_i such that the profit sum is at least P and the weight sum is at most W?

MCK is solvable in pseudo-polynomial time by dynamic programming [24]:

▶ **Lemma 5** ([24, Section 11.5])**.** *MCK can be solved in $O(W \cdot \sum_{i=1}^{\ell} |S_i|)$ time.*

As our approach is easier to explain with the edge-based distance, we start with this case and afterwards show how to extend it to the matching-based distance. As already exploited in our reductions showing NP-hardness (see Theorem 1), all variants of Dynamic Cluster Editing carry some number-problem flavor. Our generic approach will underline this flavor: We will focus on cases where we can partition the vertex set of the input graph into parts such that we will neither add nor delete an edge between two parts. Moreover, we require that the parts are "easy" enough to list all Pareto-optimal (with respect to k and d) solutions in FPT-time (this is usually achieved by some kernelization arguments). However, even with these strict requirements we cannot solve the parts independently from each other: The challenge is that we have to select for each part an appropriate Pareto-optimal solution. Finding a feasible combination of these part-individual solutions leads to a knapsack-type problem (in this case MCK). Indeed, this is common to all studied variants of Dynamic Cluster Editing.

The details for our generic approach (for edge-based distance) are as follows:

1. When necessary, apply data reduction rules from Section 4.1. Partition the input graph $G = (V, E)$ into different parts $G_1, G_2, \ldots, G_{\ell+1}$ such that there exists a solution (if there is a solution) where no edge between two parts will be inserted or deleted. (In particular, this implies that in G there is no edge between the parts.)
2. Compute for each part $G_i = (V_i, E_i)$, $1 \leq i \leq \ell$, a set $S_i \subseteq \mathbb{N}^2$ encoding "cost" and "gain" of all "representative" solutions for G_i. The size of the set S_i has to be upper-bounded in a function of the parameter p. (Here, p will be either k or d.)
   More precisely, select a family $\mathcal{E}_i$ of $f(p)$ edge sets such that for each edge set $E_i' \subseteq \binom{V_i}{2}$ in $\mathcal{E}_i$ the graph $G_i' = (V_i, E_i' \oplus E_i)$ is a cluster graph achievable with the allowed number of modification operations (edge deletions or edge insertions). For each such edge set E_i', add to S_i a tuple containing the cost ($= |E_i'|$) and "decrease" of the distance from G_i to the target cluster graph G_c. More formally, add $(|E_i'|, |E_i' \cap E_c| - |E_i' \setminus E_c|)$ to S_i, where E_c is the edge set of G_c. Note that we allow $E_i' = \emptyset$, that is, if G_i is a cluster graph, then S_i contains the tuple $(0, 0)$.
   The set S_i has to fulfill the following property: If there is a solution, then there is a solution G' such that restricting G' to V_i yields a tuple in S_i. More precisely, we require that $(|E(G'[V_i]) \oplus E_i|, |(E(G'[V_i]) \oplus E_i) \cap E_c| - |(E(G'[V_i]) \oplus E_i) \setminus E_c|) \in S_i$.
3. Create an MCK instance I with $W = k$, $P = |E \oplus E_c| - d$, and the sets $S_1, S_2, \ldots, S_\ell$ where the tuples in the sets correspond to the items with the first number in the tuple being its weight and the second number being its profit.
4. Return true if and only if I is a yes-instance.

Note that the requirement in Step 1 implies that a part is a collection of connected components in G. Furthermore, note that the part $G_{\ell+1}$ will be ignored in the subsequent steps. Thus $G_{\ell+1}$ contains all vertices which are not contained in an edge of the edge modification set. Observe that $\ell \leq n$. Hence, we have $\sum_{i=1}^{\ell} |S_i| \in O(f(p) \cdot n)$. (The parameter p will be either k or d.) Moreover, as k and d are smaller than n^2, it follows that $W < n^2$ and thus, by Lemma 5, the MCK instance I created in Step 3 can be solved in $O(f(p) \cdot n^3)$ time in Step 4. This yields the following.

▶ **Observation 1.** *If the partition in Step 1 and the sets S_i in Step 2 can be computed in FPT-time with respect to p, then the above four-step-approach runs in FPT-time with respect to p.*

Note that Steps 1 and 2 are different for every problem variant we consider. There are, however, some similarities between the variants where only edge insertions are allowed. Note that the requirements of Steps 1 and 2 seem impossible to achieve in FPT-time when allowing edge insertions and deletions. Indeed, as shown in Theorem 2, the corresponding edge-edit variants are W[1]-hard with respect to the studied (single) parameters k and d respectively.

Next, we use the above approach to show that DCDeletion (Edge Dist) is fixed-parameter tractable with respect to k. The fixed-parameter tractability of DCCompletion (Edge Dist) with respect to k and with respect to d is deferred to a full version of the paper.

▶ **Lemma 6.** DCDeletion (Edge Dist) *is FPT with respect to k.*

Proof (Sketch). We first apply the known reduction rules for Cluster Editing (see discussion after Theorem 3). As a result, we end up with a graph where at most $k^2 + 2k$ vertices are contained in an induced P_3; all other vertices form a cluster graph with cliques containing at most k vertices each. We define the parts $G_1, G_2, \ldots, G_\ell, G_{\ell+1}$ of Step 1 as follows: The first part $G_1 = (V_1, E_1)$ contains the graph induced by all vertices contained in

a P_3. Each of the cliques in the cluster graph $G[V \setminus V_1]$ forms another part G_i, $2 \leq i \leq \ell$. Finally, set $G_{\ell+1} = (\emptyset, \emptyset)$, that is, we include all vertices in the subsequent steps of our generic approach. Clearly, each part contains less than $2k^2$ vertices. Moreover, observe that there are no edges between the parts.

As to Step 2, we add, for every edge set $E'_i \subseteq E_i$ such that $G'_i = (V_i, E'_i \setminus E_i)$ is a cluster graph, a tuple $(|E'_i|, |E'_i \cap E_c| - |E'_i \setminus E_c|)$ to S_i. As this enumerates all possible solutions for G_i, the requirement in Step 2 is fulfilled. Together with Observation 1 we get the statement of the lemma. ◀

We next discuss how to adjust our generic four-step approach for DCCompletion (Matching Dist). The main difference to the edge-based distance variants is an additional search tree of size $O(d^{d+2})$ in the beginning. Each leaf of the search tree then corresponds to a simplified instance where we have additional knowledge on the matching defining the distance of a solution to G_c. With this additional knowledge, we can apply our generic four-step approach in each leaf, yielding the following.

▶ **Lemma 7.** DCCompletion (Matching Dist) *is FPT with respect to d.*

Proof. We apply our generic four-step approach and thus need to provide the details how to implement Steps 1 and 2.

We can assume that our input graph is a cluster graph. Let $\mathcal{C}$ be the set of all cliques in G and $\mathcal{D} = \{D_1, D_2, \ldots, D_q\}$ the set of all cliques in G_c. Then we classify all cliques in $\mathcal{C}$ into two classes $\mathcal{C}_1$ and $\mathcal{C}_2$, where every clique in $\mathcal{C}_1$ has the property that all its vertices are contained in one clique in $\mathcal{D}$ and every clique in $\mathcal{C}_2$ contains vertices from at least two different cliques in $\mathcal{D}$. Observe that $|\mathcal{C}_2| \leq d$ as otherwise the input is a no-instance. Similarly, every clique in $\mathcal{C}_2$ contains vertices from at most $d+1$ different cliques in $\mathcal{D}$ as otherwise the input is a no-instance.

This allows us to do the following branching step. For each clique in $\mathcal{C}_2$ we try out all "meaningful" possibilities to match it to a clique in $\mathcal{D}$, where "meaningful" means that the cliques in $\mathcal{C}_2$ and $\mathcal{D}$ should share some vertices or we decide to not match the clique of $\mathcal{C}_2$ to any clique in $\mathcal{D}$. For each clique this gives us $d+2$ possibilities and hence we have at most d^{d+2} different cases each of which defines a mapping $M : \mathcal{C}_2 \to \mathcal{D} \cup \{\emptyset\}$ that maps a clique in $\mathcal{C}_2$ to the clique in $\mathcal{D}$ it is matched to.

Given the mapping M from cliques in $\mathcal{C}_2$ to cliques $\mathcal{D}$ or $\emptyset$, we partition G into $q+1$ groups $G_1, G_2, \ldots, G_q, G_{q+1}$ with $G_i = G[V_i]$, where $V_i = \{C \in \mathcal{C}_1 \mid C \subseteq D_i\} \cup \{C \in \mathcal{C}_2 \mid M(C) = D_i\}$ and $V_{q+1} = \{C \in \mathcal{C}_2 \mid M(C) = \emptyset\}$.

If there is a solution with a matching that uses the matches given by M, then there is a solution only combining cliques within every group G_i, $1 \leq i \leq q$, since all cliques in G_i that are not matched by M are completely contained in D_i and hence would not be merged with cliques in G_j for some $i \neq j$. This shows that with $\ell = q$ the requirements of Step 1 of our generic approach are met.

Next we describe Step 2, that is, for every part G_i, we show how to compute a set S_i corresponding to all "representative" solutions. Note that all except at most d cliques from G_i need to be merged into one clique that is then matched with D_i, otherwise the matching distance would be too large. For each clique in G_i that is not completely contained in D_i we already know that it is matched to D_i, hence we need to merge all cliques of this kind to one clique $C_i^\star$. Each clique in G_i that is completely contained in D_i and has size at least $d+1$ also needs to be merged to $C_i^\star$, otherwise the matching distance would be too large. For all cliques of G_i that are completely contained in D_i with size x for some $1 \leq x \leq d$ we merge all but d cliques to $C_i^\star$. This leaves us with one big clique $C_i^\star$ and d^2 cliques of size at most d

each. Now we can brute-force all possibilities to merge some of the remaining cliques to $C_i^\star$. There are less than d^d possibilities to do so and for each possibility we add to S_i a tuple representing the cost and gain of merging the cliques according to the partition. ◀

5 Conclusion

Our work provides a first thorough (parameterized) analysis of Dynamic Cluster Editing, addressing a natural dynamic setting for graph-based data clustering. We deliver both (parameterized) tractability and intractability results. Our positive algorithmic results (fixed-parameter tractability and kernelization) are mainly of classification nature. To get practically useful algorithms, one needs to further improve our running times.

The main difference to static Cluster Editing seems to come from the fact that all six variants of Dynamic Cluster Editing remain NP-hard when the input graph is a cluster graph (see Theorem 1). Moreover, Dynamic Cluster Editing (both matching- and edge-based distance) is W[1]-hard with respect to the budget k (see Theorem 2) whereas Cluster Editing is FPT with respect to k. The obvious approach to solve Dynamic Cluster Editing is to compute (almost) all cluster graphs achievable with at most k edge modifications, then from this set of cluster graphs pick one at distance at most d to the target cluster graph. However, listing these cluster graphs is computationally hard. Indeed, our W[1]-hardness results indicate that we might not do much better than using this simple approach.

We mention in passing that our results can also be used to show fixed-parameter tractability for the case when both input graphs are arbitrary graphs and one wants to find a "compromise" cluster graph being close enough (in terms edge-based distance) to both input graphs. The parameter herein is the symmetric distance of the edge sets.

We conclude with few open questions. First, we left open the parameterized complexity of Dynamic Cluster Editing (deletion variant and completion variant) with matching-based distance when parameterized by the budget k, see Table 1 in Section 1. Moreover, the existence of polynomial-size problem kernels for our fixed-parameter tractable cases in case of single parameters (budget k or distance d) is open.

References

1 Faisal N. Abu-Khzam. On the complexity of multi-parameterized cluster editing. *Journal of Discrete Algorithms*, 45:26–34, 2017.

2 Faisal N. Abu-Khzam, Shaowei Cai, Judith Egan, Peter Shaw, and Kai Wang. Turbo-Charging Dominating Set with an FPT Subroutine: Further Improvements and Experimental Analysis. In *Proceedings of the 14th Annual Conference on Theory and Applications of Models of Computation, TAMC 2017*, volume 10185 of *LNCS*, pages 59–70. Springer, 2017.

3 Faisal N Abu-Khzam, Judith Egan, Michael R Fellows, Frances A Rosamond, and Peter Shaw. On the parameterized complexity of dynamic problems. *Theoretical Computer Science*, 607:426–434, 2015.

4 Faisal N. Abu-Khzam, Judith Egan, Serge Gaspers, Alexis Shaw, and Peter Shaw. Cluster Editing with Vertex Splitting. In *Proceeding of the 5th International Symposium on Combinatorial Optimization, ISCO 2018*, volume 10856 of *LNCS*, pages 1–13. Springer, 2018.

5 Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation Clustering. *Machine Learning*, 56:89–113, 2004.

6 Amir Ben-Dor, Ron Shamir, and Zohar Yakhini. Clustering Gene Expression Patterns. *Journal of Computational Biology*, 6(3-4):281–297, 1999.

7 Hans-Joachim Böckenhauer, Juraj Hromkovič, Tobias Mömke, and Peter Widmayer. On the Hardness of Reoptimization. In *Proceedings of the 34th Conference on Theory and Practice of Computer Science, SOFSEM 2008*, volume 4910 of *LNCS*, pages 50–65. Springer, 2008.

8 Sebastian Böcker and Jan Baumbach. Cluster Editing. In *Proceedings of the 9th Conference on Computability in Europe, CiE 2013*, volume 7921 of *LNCS*, pages 33–44. Springer, 2013.

9 Yixin Cao and Jianer Chen. Cluster Editing: Kernelization Based on Edge Cuts. *Algorithmica*, 64(1):152–169, 2012.

10 Moses Charikar, Chandra Chekuri, Tomás Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. *SIAM Journal on Computing*, 33(6):1417–1440, 2004.

11 Jianer Chen and Jie Meng. A $2k$ kernel for the cluster editing problem. *Journal of Computer and System Sciences*, 78(1):211–220, 2012.

12 Jiehua Chen, Hendrik Molter, Manuel Sorge, and Ondrej Suchý. Cluster Editing in Multi-Layer and Temporal Graphs. In *Proceedings of the 29th International Symposium on Algorithms and Computation (ISAAC '18)*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. To appear.

13 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

14 Tamal K. Dey, Alfred Rossi, and Anastasios Sidiropoulos. Temporal Clustering. In *Proceedings of the 25th Annual European Symposium on Algorithms, ESA 2017*, volume 87 of *LIPIcs*, pages 34:1–34:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.

15 X. Dong, P. Frossard, P. Vandergheynst, and N. Nefedov. Clustering With Multi-Layer Graphs: A Spectral Perspective. *IEEE Transactions on Signal Processing*, 60(11):5820–5831, 2012.

16 Rodney G Downey, Judith Egan, Michael R Fellows, Frances A Rosamond, and Peter Shaw. Dynamic dominating set and turbo-charging greedy heuristics. *Tsinghua Science and Technology*, 19(4):329–337, 2014.

17 Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Frances A. Rosamond, Saket Saurabh, and Yngve Villanger. Local search: Is brute-force avoidable? *Journal of Computer and System Sciences*, 78(3):707–719, 2012.

18 Fedor V. Fomin, Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Yngve Villanger. Tight Bounds for Parameterized Complexity of Cluster Editing with a Small Number of Clusters. *Journal of Computer and System Sciences*, 80(7):1430–1447, 2014.

19 Serge Gaspers, Eun Jung Kim, Sebastian Ordyniak, Saket Saurabh, and Stefan Szeider. Don't Be Strict in Local Search! In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, pages 486–492. AAAI Press, 2012.

20 Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Graph-Modeled Data Clustering: Exact Algorithms for Clique Generation. *Theory of Computing Systems*, 38(4):373–392, 2005.

21 Jiong Guo, Sepp Hartung, Rolf Niedermeier, and Ondrej Suchý. The Parameterized Complexity of Local Search for TSP, More Refined. *Algorithmica*, 67(1):89–110, 2013.

22 Sepp Hartung and Holger H. Hoos. Programming by Optimisation Meets Parameterised Algorithmics: A Case Study for Cluster Editing. In *Proceedings of the 9th International Conference on Learning and Intelligent Optimization, LION 2015*, volume 8994 of *LNCS*, pages 43–58. Springer, 2015.

23 Sepp Hartung and Rolf Niedermeier. Incremental list coloring of graphs, parameterized by conservation. *Theoretical Computer Science*, 494:86–98, 2013.

24 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Springer, 2004.

25 Christian Komusiewicz and Johannes Uhlmann. Cluster editing with locally bounded modifications. *Discrete Applied Mathematics*, 160(15):2259–2270, 2012.

26 R Krithika, Abhishek Sahu, and Prafullkumar Tale. Dynamic parameterized problems. *Algorithmica*, 80(9):2637–2655, 2018.

27 Dániel Marx and Ildikó Schlotter. Parameterized Complexity and Local Search Approaches for the Stable Marriage Problem with Ties. *Algorithmica*, 58(1):170–187, 2010.

28 Marina Meilă. Comparing clusterings: an axiomatic view. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 577–584. ACM, 2005.

29 Marina Meilă. Local equivalences of distances between clusterings—a geometric perspective. *Machine Learning*, 86(3):369–389, 2012.

30 Baruch Schieber, Hadas Shachnai, Gal Tamir, and Tami Tamir. A theory and algorithms for combinatorial reoptimization. *Algorithmica*, 80(2):576–607, 2018.

31 Ron Shamir, Roded Sharan, and Dekel Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004.

32 W. Tang, Z. Lu, and I. S. Dhillon. Clustering with Multiple Graphs. In *Proceedings of the Ninth IEEE International Conference on Data Mining, ICDM 2009*, pages 1016–1021, 2009.

33 C. Tantipathananandh and T. Y. Berger-Wolf. Finding Communities in Dynamic Social Networks. In *Proceedings of the IEEE 11th International Conference on Data Mining, ICDM 2011*, pages 1236–1241, 2011.

34 Chayant Tantipathananandh, T. Y. Berger-Wolf, and David Kempe. A Framework for Community Identification in Dynamic Social Networks. In *Proceedings of the 13th SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2007*, pages 717–726. ACM, 2007.

The Complexity of Separation for Levels in Concatenation Hierarchies

Thomas Place
LaBRI Bordeaux University and IUF, France

Marc Zeitoun
LaBRI Bordeaux University, France

Abstract

We investigate the complexity of the separation problem associated to classes of regular languages. For a class $\mathcal{C}$, $\mathcal{C}$-separation takes two regular languages as input and asks whether there exists a third language in $\mathcal{C}$ which includes the first and is disjoint from the second. First, in contrast with the situation for the classical membership problem, we prove that for most classes $\mathcal{C}$, the complexity of $\mathcal{C}$-separation does not depend on how the input languages are represented: it is the same for nondeterministic finite automata and monoid morphisms. Then, we investigate specific classes belonging to finitely based concatenation hierarchies. It was recently proved that the problem is always decidable for levels 1/2 and 1 of any such hierarchy (with inefficient algorithms). Here, we build on these results to show that when the alphabet is fixed, there are polynomial time algorithms for both levels. Finally, we investigate levels 3/2 and 2 of the famous Straubing-Thérien hierarchy. We show that separation is PSpace-complete for level 3/2 and between PSpace-hard and EXPTime for level 2.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Regular languages, separation, concatenation hierarchies, complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.47

Related Version A full version of the paper is available at `https://arxiv.org/abs/1810.09287`.

Funding Both authors acknowledge support from the DeLTA project (ANR-16-CE40-0007).

1 Introduction

For more than 50 years, a significant research effort in theoretical computer science was made to solve the membership problem for regular languages. This problem consists in determining whether a class of regular languages is decidable, that is, whether there is an algorithm inputing a regular language and outputing 'yes' if the language belongs to the investigated class, and 'no' otherwise.

Many results were obtained in a long and fruitful line of research. The most prominent one is certainly Schützenberger's theorem [19], which gives such an algorithm for the class of star-free languages. For most interesting classes also, we know precisely the computational cost of the membership problem. As can be expected, this cost depends on the way the input language is given. Indeed, there are several ways to input a regular language. For instance, it can be given by a nondeterministic finite automaton (NFA), or, alternately, by a morphism into a finite monoid. While obtaining an NFA representation from a morphism into a monoid has only a linear cost, the converse direction is much more expensive: from an NFA

with n states, the smallest monoid recognizing the same language may have an exponential number of elements (the standard construction yields 2^{n^2} elements). This explains why the complexity of the membership problem depends on the representation of the input. For instance, for the class of star-free languages, it is PSpace-complete if one starts from NFAs (and actually, even from DFAs [2]) while it is NL when starting from monoid morphisms.

Recently, another problem, called separation, has replaced membership as the cornerstone in the investigation of regular languages. It takes as input *two* regular langages instead of one, and asks whether there exists a third language from the class under investigation including the first input language and having empty intersection with the second one. This problem has served recently as a major ingredient in the resolution of difficult membership problems, such as the so-called dot-depth two problem [16] which remained open for 40 years (see [13, 18, 6] for recent surveys on the topic). Dot-depth two is a class belonging to a famous *concatenation hierarchy* which stratifies the star-free languages: the dot-depth [1]. A specific concatenation hierarchy is built in a generic way. One starts from a base class (level 0 of the hierarchy) and builds increasingly growing classes (called levels and denoted by 1/2, 1, 3/2, 2, ...) by alternating two standard closure operations: polynomial and Boolean closure. Concatenation hierarchies account for a significant part of the open questions in this research area. The state of the art regarding separation is captured by only three results [17, 9]: in finitely based concatenation hierarchies (i.e. those whose basis is a finite class) levels 1/2, 1 and 3/2 have decidable separation. Moreover, using specific transfer results [15], this can be pushed to the levels 3/2 and 2 for the two most famous finitely based hierarchies: the dot-depth [1] and the Straubing-Thérien hierarchy [21, 22].

Unlike the situation for membership and despite these recent decidability results for separability in concatenation hierarchies, the complexity of the problems and of the corresponding algorithms has not been investigated so far (except for the class of piecewise testable languages [3, 11, 5], which is level 1 in the Straubing-Thérien hierarchy). The aim of this paper is to establish such complexity results. Our contributions are the following:

- We present a **generic** reduction, which shows that for many natural classes, the way the input is given (by NFAs or finite monoids) has **no impact** on the complexity of the separation problem. This is proved using two LogSpace reductions from one problem to the other. This situation is surprising and opposite to that of the membership problem, where an exponential blow-up is unavoidable when going from NFAs to monoids.
- Building on the results of [17], we show that when the alphabet is fixed, there are polynomial time algorithms for levels 1/2 and 1 in any finitely based hierarchy.
- We investigate levels 3/2 and 2 of the famous Straubing-Thérien hierarchy, and we show that separation is PSpace-complete for level 3/2 and between PSpace-hard and EXPTime for level 2. The upper bounds are based on the results of [17] while the lower bounds are based on independent reductions.

Organization. In Section 2, we give preliminary terminology on the objects investigated in the paper. Sections 3, 4 and 5 are then devoted to the three above points. Due to space limitations, many proofs are postponed to the full version of the paper.

2 Preliminaries

In this section, we present the key objects of this paper. We define words and regular languages, classes of languages, the separation problem and finally, concatenation hierarchies.

2.1 Words and regular languages

An alphabet is a *finite* set A of symbols, called *letters*. Given some alphabet A, we denote by A^+ the set of all nonempty finite words and by A^* the set of all finite words over A (*i.e.*, $A^* = A^+ \cup \{\varepsilon\}$). If $u \in A^*$ and $v \in A^*$ we write $u \cdot v \in A^*$ or $uv \in A^*$ for the concatenation of u and v. A *language* over an alphabet A is a subset of A^*. Abusing terminology, if $u \in A^*$ is some word, we denote by u the singleton language $\{u\}$. It is standard to extend concatenation to languages: given $K, L \subseteq A^*$, we write $KL = \{uv \mid u \in K \text{ and } v \in L\}$. Moreover, we also consider marked concatenation, which is less standard. Given $K, L \subseteq A^*$, *a marked concatenation* of K with L is a language of the form KaL, for some $a \in A$.

We consider *regular languages*, which can be equivalently defined by *regular expressions*, *nondeterministic finite automata* (NFAs), *finite monoids* or *monadic second-order logic* (MSO). In the paper, we investigate the separation problem which takes regular languages as input. Since we are focused on complexity, how we represent these languages in our inputs matters. We shall consider two kinds of representations: NFAs and monoids. Let us briefly recall these objects and fix the terminology (we refer the reader to [7] for details).

NFAs. An NFA is a tuple $\mathcal{A} = (A, Q, \delta, I, F)$ where A is an alphabet, Q a finite set of states, $\delta \subseteq Q \times A \times Q$ a set of transitions, $I \subseteq Q$ a set of initial states and $F \subseteq Q$ a set of final states. The language $L(\mathcal{A}) \subseteq A^*$ consists of all words labeling a run from an initial state to a final state. The regular languages are exactly those which are recognized by an NFA. Finally, we write "DFA" for *deterministic* finite automata, which are defined in the standard way.

Monoids. We turn to the algebraic definition of regular languages. A *monoid* is a set M endowed with an associative multiplication $(s, t) \mapsto s \cdot t$ (also denoted by st) having a neutral element 1_M, *i.e.*, such that $1_M \cdot s = s \cdot 1_M = s$ for every $s \in M$. An *idempotent* of a monoid M is an element $e \in M$ such that $ee = e$.

Observe that A^* is a monoid whose multiplication is concatenation (the neutral element is ε). Thus, we may consider monoid morphisms $\alpha : A^* \to M$ where M is an arbitrary monoid. Given such a morphism, we say that a language $L \subseteq A^*$ is *recognized* by α when there exists a set $F \subseteq M$ such that $L = \alpha^{-1}(F)$. It is well-known that the regular languages are also those which are recognized by a morphism into a *finite* monoid. When representing a regular language L by a morphism into a finite monoid, one needs to give both the morphism $\alpha : A^* \to M$ (*i.e.*, the image of each letter) and the set $F \subseteq M$ such that $L = \alpha^{-1}(F)$.

2.2 Classes of languages and separation

A class of languages $\mathcal{C}$ is a correspondence $A \mapsto \mathcal{C}(A)$ which, to an alphabet A, associates a set of languages $\mathcal{C}(A)$ over A.

▶ Remark. When two alphabets A, B satisfy $A \subseteq B$, the definition of classes does not require $\mathcal{C}(A)$ and $\mathcal{C}(B)$ to be comparable. In fact, it may happen that a particular language $L \subseteq A^* \subseteq B^*$ belongs to $\mathcal{C}(A)$ but not to $\mathcal{C}(B)$ (or the opposite). For example, we may consider the class $\mathcal{C}$ defined by $\mathcal{C}(A) = \{\emptyset, A^*\}$ for every alphabet A. When $A \subsetneq B$, we have $A^* \in \mathcal{C}(A)$ while $A^* \notin \mathcal{C}(B)$.

We say that $\mathcal{C}$ is a *lattice* when for every alphabet A, we have $\emptyset, A^* \in \mathcal{C}(A)$ and $\mathcal{C}(A)$ is closed under finite union and finite intersection: for any $K, L \in \mathcal{C}(A)$, we have $K \cup L \in \mathcal{C}(A)$ and $K \cap L \in \mathcal{C}(A)$. Moreover, a *Boolean algebra* is a lattice $\mathcal{C}$ which is additionally closed under complement: for any $L \in \mathcal{C}(A)$, we have $A^* \setminus L \in \mathcal{C}(A)$. Finally, a class $\mathcal{C}$ is *quotienting*

if it is closed under quotients. That is, for every alphabet A, $L \in \mathcal{C}(A)$ and word $u \in A^*$, the following properties hold:

$$u^{-1}L \stackrel{\text{def}}{=} \{w \in A^* \mid uw \in L\} \quad \text{and} \quad Lu^{-1} \stackrel{\text{def}}{=} \{w \in A^* \mid wu \in L\} \quad \text{both belong to } \mathcal{C}(A).$$

All classes that we consider in the paper are (at least) quotienting lattices consisting of *regular languages*. Moreover, some of them satisfy an additional property called *closure under inverse image*.

Recall that A^* is a monoid for any alphabet A. We say that a class $\mathcal{C}$ is *closed under inverse image* if for every two alphabets A, B, every monoid morphism $\alpha : A^* \to B^*$ and every language $L \in \mathcal{C}(B)$, we have $\alpha^{-1}(L) \in \mathcal{C}(A)$. A quotienting lattice (resp. quotienting Boolean algebra) closed under inverse image is called a *positive variety* (resp. *variety*).

Separation. Consider a class of languages $\mathcal{C}$. Given an alphabet A and two languages $L_1, L_2 \subseteq A^*$, we say that L_1 is $\mathcal{C}$-separable from L_2 when there exists a third language $K \in \mathcal{C}(A)$ such that $L_1 \subseteq K$ and $L_2 \cap K = \emptyset$. In particular, K is called a *separator* in $\mathcal{C}$. The $\mathcal{C}$-separation problem is now defined as follows:

Input: An alphabet A and two regular languages $L_1, L_2 \subseteq A^*$.
Output: Is L_1 $\mathcal{C}$-separable from L_2 ?

▶ Remark. Separation generalizes the simpler *membership problem*, which asks whether a single regular language belongs to $\mathcal{C}$. Indeed $L \in \mathcal{C}$ if and only if L is $\mathcal{C}$-separable from $A^* \setminus L$ (which is also regular and computable from L).

Most papers on separation are mainly concerned about decidability. Hence, they do not go beyond the above presentation of the problem (see [3, 16, 12, 17] for example). However, this paper specifically investigates complexity. Consequently, we shall need to be more precise and take additional parameters into account. First, it will be important to specify whether the alphabet over which the input languages is part of the input (as above) or a constant. When considering separation for some fixed alphabet A, we shall speak of "$\mathcal{C}(A)$-separation". When the alphabet is part of the input, we simply speak of "$\mathcal{C}$-separation".

Another important parameter is how the two input languages are represented. We shall consider NFAs and monoids. We speak of *separation for* NFAs *and separation for monoids*. Note that one may efficiently reduce the latter to the former. Indeed, given a language $L \subseteq A^*$ recognized by some morphism $\alpha : A^* \to M$, it is simple to efficiently compute a NFA with $|M|$ states recognizing L (see [7] for example). Hence, we have the following lemma.

▶ **Lemma 1.** *For any class $\mathcal{C}$, there is a* LogSpace *reduction from $\mathcal{C}$-separation for monoids to $\mathcal{C}$-separation for* NFAs.

Getting an efficient reduction for the converse direction is much more difficult since going from NFAs (or even DFAs) to monoids usually involves an exponential blow-up. However, we shall see in Section 3 that for many natural classes $\mathcal{C}$, this is actually possible.

2.3 Concatenation hierarchies

We now briefly recall the definition of concatenation hierarchies. We refer the reader to [18] for a more detailed presentation. A particular concatenation hierarchy is built from a starting class of languages $\mathcal{C}$, which is called its *basis*. In order to get robust properties, we restrict $\mathcal{C}$ to be a quotienting Boolean algebra of regular languages. The basis is the only parameter in the construction. Once fixed, the construction is generic: each new level is built from the previous one by applying generic operators: either Boolean closure, or polynomial closure. Let us first define these two operators.

Definition. Consider a class $\mathcal{C}$. We denote by $Bool(\mathcal{C})$ the *Boolean closure* of $\mathcal{C}$: for every alphabet A, $Bool(\mathcal{C})(A)$ is the least set containing $\mathcal{C}(A)$ and closed under Boolean operations. Moreover, we denote by $Pol(\mathcal{C})$ the *polynomial closure* of $\mathcal{C}$: for every alphabet A, $Pol(\mathcal{C})(A)$ is the least set containing $\mathcal{C}(A)$ and closed under union and marked concatenation (if $K, L \in Pol(\mathcal{C})(A)$ and $a \in A$, then $K \cup L, KaL \in Pol(\mathcal{C})(A)$).

Consider a quotienting Boolean algebra of regular languages $\mathcal{C}$. The concatenation hierarchy of basis $\mathcal{C}$ is defined as follows. Languages are classified into levels of two kinds: full levels (denoted by 0, 1, 2,...) and half levels (denoted by 1/2, 3/2, 5/2,...). Level 0 is the basis (*i.e.*, $\mathcal{C}$) and for every $n \in \mathbb{N}$,

- The *half level* $n + 1/2$ is the *polynomial closure* of the previous full level, *i.e.*, of level n.
- The *full level* $n + 1$ is the *Boolean closure* of the previous half level, *i.e.*, of level $n + 1/2$.

$$0 \xrightarrow{Pol} 1/2 \xrightarrow[Bool]{} 1 \xrightarrow{Pol} 3/2 \xrightarrow[Bool]{} 2 \xrightarrow{Pol} 5/2 \cdots\cdots$$

We write $\frac{1}{2}\mathbb{N} = \{0, 1/2, 1, 2, 3/2, 3, \dots\}$ for the set of all possible levels in a concatenation hierarchy. Moreover, for any basis $\mathcal{C}$ and $n \in \frac{1}{2}\mathbb{N}$, we write $\mathcal{C}[n]$ for level n in the concatenation hierarchy of basis $\mathcal{C}$. It is known that every half-level is a quotienting lattice and every full level is a quotienting Boolean algebra (see [18] for a recent proof).

We are interested in finitely based concatenation hierarchies: if $\mathcal{C}$ is the basis, then $\mathcal{C}(A)$ is finite for every alphabet A. Indeed, it was shown in [17] that for such hierarchies separation is always decidable for the levels 1/2 and 1 (in fact, while we do not discuss this in the paper, this is also true for level 3/2, see [9] for a preliminary version). In Section 4, we build on the results of [17] and show that when the alphabet is fixed, this can be achieved in polynomial time for both levels 1/2 and 1. Moreover, we shall also investigate the famous *Straubing-Thérien* hierarchy in Section 5. Our motivation for investigating this hierarchy in particular is that the results of [17] can be pushed to levels 3/2 and 2 in this special case.

3 Handling NFAs

In this section, we investigate how the representation of input languages impact the complexity of separation. We prove that for many natural classes $\mathcal{C}$ (including most of those considered in the paper), $\mathcal{C}$-separation has the same complexity for NFAs as for monoids. Because of these results, we shall be able to restrict ourselves to monoids in later sections.

▶ Remark. This result highlights a striking difference between separation and the simpler membership problem. For most classes $\mathcal{C}$, $\mathcal{C}$-membership is strictly harder for NFAs than for monoids. This is because when starting from a NFA, typical membership algorithms require to either determinize $\mathcal{A}$ or compute a monoid morphism recognizing $L(\mathcal{A})$ which involves an exponential blow-up in both cases. Our results show that the situation differs for separation.

We already have a generic efficient reduction from $\mathcal{C}$-separation for monoids to $\mathcal{C}$-separation for NFAs (see Lemma 1). Here, we investigate the opposite direction: given some class $\mathcal{C}$, is it possible to *efficiently* reduce $\mathcal{C}$-separation for NFAs to $\mathcal{C}$-separation for monoids ? As far as we know, there exists no such reduction which is generic to all classes $\mathcal{C}$.

▶ Remark. There exists an *inefficient* generic reduction from separation for NFAs to the separation for monoids. Given as input two NFAs $\mathcal{A}_1, \mathcal{A}_2$, one may compute monoid morphisms recognizing $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$. This approach is not satisfying as it involves an exponential blow-up: we end-up with monoids M_i of size $2^{|Q_i|^2}$ where Q_i is the set of states of $\mathcal{A}_i$.

Here, we present a set of conditions applying to a pair of classes $(\mathcal{C}, \mathcal{D})$. When they are satisfied, there exists an efficient reduction from $\mathcal{C}$-separation for NFAs to $\mathcal{D}$-separation for monoids. By themselves, these conditions are abstract. However, we highlight two concrete applications. First, for every positive variety $\mathcal{C}$, the pair $(\mathcal{C}, \mathcal{C})$ satisfies the conditions. Second, for every finitely based concatenation hierarchies of basis $\mathcal{C}$, there exists another finite basis $\mathcal{D}$ such that for every $n \in \frac{1}{2}\mathbb{N}$, the pair $(\mathcal{C}[n], \mathcal{D}[n])$ satisfies the conditions

We first introduce the notions we need to present the reduction and the conditions required to apply it. Then, we state the reduction itself and its applications.

3.1 Generic theorem

We fix a special two letter alphabet $\mathbb{E} = \{0, 1\}$. For the sake of improved readability, we abuse terminology and assume that when considering an arbitrary alphabet A, it always has empty intersection with $\mathbb{E}$. This is harmless as we may work up to bijective renaming.

We exhibit conditions applying to a pair of classes $(\mathcal{C}, \mathcal{D})$. Then, we prove that they imply the existence of an efficient reduction from $\mathcal{C}$-separation for NFAs to $\mathcal{D}$-separation for monoids. This reduction is based on a construction which takes as input a NFA $\mathcal{A}$ (over some arbitrary alphabet A) and builds a modified version of the language $L(\mathcal{A})$ (over $A \cup \mathbb{E}$) which is recognized by a "small" monoid. Our conditions involve two kinds of hypotheses:

1. First, we need properties related to inverse image: "$\mathcal{D}$ must be an an extension of $\mathcal{C}$".
2. The construction is parametrized by an object called "tagging". We need an algorithm which builds special taggings (with respect to $\mathcal{D}$) efficiently.

We now make these two notions more precise. Let us start with extension.

Extensions. Consider two classes $\mathcal{C}$ and $\mathcal{D}$. We say that $\mathcal{D}$ is an extension of $\mathcal{C}$ when for every alphabet A, the two following conditions hold:

- If $\gamma : (A \cup \mathbb{E})^* \to A^*$ is the morphism defined by $\gamma(a) = a$ for $a \in A$ and $\gamma(b) = \varepsilon$ for $b \in \mathbb{E}$, then for every $K \in \mathcal{C}(A)$, we have $\gamma^{-1}(K) \in \mathcal{D}(A \cup \mathbb{E})$.
- For every $u \in \mathbb{E}^*$, if $\lambda_u : A^* \to (A \cup \mathbb{E})^*$ is the morphism defined by $\lambda_u(a) = au$ for $a \in A$, then for every $K \in \mathcal{D}(A \cup \mathbb{E})$, we have $\lambda_u^{-1}(K) \in \mathcal{C}(A)$.

Positive varieties give an important example of extension. Since they are closed under inverse image, it is immediate that for every positive variety $\mathcal{C}$, $\mathcal{C}$ is an extension of itself.

Taggings. A *tagging* is a pair $P = (\tau : \mathbb{E}^* \to T, G)$ where τ is a morphism into a finite monoid and $G \subseteq T$. We call $|G|$ the *rank* of P and $|T|$ its size. Moreover, given some NFA $\mathcal{A} = (A, Q, \delta, I, F)$, P is *compatible with* $\mathcal{A}$ when the rank $|G|$ is larger than $|\delta|$.

For our reduction, we shall require special taggings. Consider a class $\mathcal{D}$ and a tagging $P = (\tau : \mathbb{E}^* \to T, G)$. We say that P *fools* $\mathcal{D}$ when, for every alphabet A and every morphism $\alpha : (A \cup \mathbb{E})^* \to M$ into a finite monoid M, if all languages recognized by α belong to $Bool(\mathcal{D})(A \cup \mathbb{E})$, then, there exists $s \in M$, such that for every $t \in G$, we have $w_t \in \mathbb{E}^*$ which satisfies $\alpha(w_t) = s$ and $\tau(w_t) = t$.

Our reduction requires an efficient algorithm for computing taggings which fool the output class $\mathcal{D}$. Specifically, we say that a class $\mathcal{D}$ is *smooth* when, given as input $k \in \mathbb{N}$, one may compute in LogSpace (with respect to k) a tagging of rank at least k which fools $\mathcal{D}$.

Main theorem. We may now state our generic reduction theorem. The statement has two variants depending on whether the alphabet is fixed or not.

▶ **Theorem 2.** *Let $\mathcal{C}, \mathcal{D}$ be quotienting lattices such that $\mathcal{D}$ is smooth and extends $\mathcal{C}$. Then the two following properties hold:*

- *There is a* LogSpace *reduction from $\mathcal{C}$-separation for* NFAs *to $\mathcal{D}$-separation for monoids.*
- *For every fixed alphabet A, there is a* LogSpace *reduction from $\mathcal{C}(A)$-separation for* NFAs *to $\mathcal{D}(A \cup \mathbb{E})$-separation for monoids.*

We have two main applications of Theorem 2 which we present at the end of the section. Let us first describe the reduction. As we explained, we use a construction building a language recognized by a "small" monoid out of an input NFA and a compatible tagging.

Consider a NFA $\mathcal{A} = (A, Q, \delta, I, F)$ and let $P = (\tau : \mathbb{E}^* \to T, G)$ be a compatible tagging (i.e. $|\delta| \leq |G|$). We associate a new language $L[\mathcal{A}, P]$ over the alphabet $A \cup \mathbb{E}$ and show that one may efficiently compute a recognizing monoid whose size is polynomial with respect to $|Q|$ and the rank of P (i.e $|G|$). The construction involves two steps. We first define an intermediary language $K[\mathcal{A}, P]$ over the alphabet $A \times T$ and then define $L[\mathcal{A}, P]$ from it.

We define $K[\mathcal{A}, P] \subseteq (A \times T)^*$ as the language recognized by a new NFA $\mathcal{A}[P]$ which is built by relabeling the transitions of $\mathcal{A}$. Note that the definition of $\mathcal{A}[P]$ depends on arbitrary linear orders on G and δ. We let $\mathcal{A}[P] = (A \times T, Q, \delta[P], I, F)$ where $\delta[P]$ is obtained by relabeling the transitions of $\mathcal{A}$ as follows. Given $i \leq |\delta|$, if $(q_i, a_i, r_i) \in \delta$ is the i-th transition of $\mathcal{A}$, we replace it with the transition $(q_i, (a_i, t_i), r_i) \in \delta[P]$ where $t_i \in G$ is the i-th element of G (recall that $|\delta| \leq |G|$ by hypothesis).

▶ Remark. A key property of $\mathcal{A}[P]$ is that, by definition, all transitions are labeled by distinct letters in $A \times T$. This implies that $K[\mathcal{A}, P] = L(\mathcal{A}[P])$ is recognized by a monoid of size at most $|Q|^2 + 2$.

We may now define the language $L[\mathcal{A}, P] \subseteq (A \cup \mathbb{E})^*$. Observe that we have a natural map $\mu : (A\mathbb{E}^*)^* \to (A \times T)^*$. Indeed, consider $w \in (A\mathbb{E}^*)^*$. Since $A \cap \mathbb{E} = \emptyset$ (recall that this is a global assumption), it is immediate that w admits a *unique* decomposition $w = a_1 w_1 \cdots a_n w_n$ with $a_1, \ldots, a_n \in A$ and $w_1, \ldots, w_n \in \mathbb{E}^*$. Hence, we may define $\mu(w) = (a_1, P(w_1)) \cdots (a_n, P(w_n)) \in (A \times T)^*$. Finally, we define,

$$L[\mathcal{A}, P] = \mathbb{E}^* \cdot \mu^{-1}(K[\mathcal{A}, P]) \subseteq (A \cup \mathbb{E})^*$$

We may now state the two key properties of $L[\mathcal{A}, P]$ upon which Theorem 2 is based. It is recognized by a small monoid and the construction is connected to the separation.

▶ **Proposition 3.** *Given a* NFA $\mathcal{A} = (A, Q, \delta, I, F)$ *and a compatible tagging P of rank n, one may compute in* LogSpace *a monoid morphism $\alpha : (A \cup \mathbb{E})^* \to M$ recognizing $L[\mathcal{A}, P]$ and such that $|M| \leq n + |A| \times n^2 \times (|Q|^2 + 2)$.*

▶ **Proposition 4.** *Let $\mathcal{C}, \mathcal{D}$ be quotienting lattices such that $\mathcal{D}$ extends $\mathcal{C}$. Consider two* NFAs *$\mathcal{A}_1$ and $\mathcal{A}_2$ over some alphabet A and let P be a compatible tagging that fools $\mathcal{D}$. Then, $L(\mathcal{A}_1)$ is $\mathcal{C}(A)$-separable from $L(\mathcal{A}_2)$ if and only if $L[\mathcal{A}_1, P]$ is $\mathcal{D}(A \cup \mathbb{E})$-separable from $L[\mathcal{A}_2, P]$.*

Let us explain why these two propositions imply Theorem 2. Let $\mathcal{C}, \mathcal{D}$ be quotienting lattices such that $\mathcal{D}$ is smooth and extends $\mathcal{C}$. We show that the second assertion in the theorem holds (the first one is proved similarly).

Consider two NFAs $\mathcal{A}_i = (A, Q_j, \delta_j, I_j, F_j)$ for $j = 1, 2$. We let $k = max(|\delta_1|, |\delta_2|)$. Since $\mathcal{D}$ is smooth, we may compute (in LogSpace) a tagging $P = (\tau : \mathbb{E}^* \to T, G)$ of rank $|G| \geq k$. Then, we may use Proposition 3 to compute (in LogSpace) monoid morphisms recognizing $L[\mathcal{A}_1, P]$ and $L[\mathcal{A}_2, P]$. Finally, by Proposition 4, $L(\mathcal{A}_1)$ is $\mathcal{C}(A)$-separable from $L(\mathcal{A}_2)$ if and only if $L[\mathcal{A}_1, P]$ is $\mathcal{D}(A \cup \mathbb{E})$-separable from $L[\mathcal{A}_2, P]$. Altogether, this construction is a LogSpace reduction to $\mathcal{D}$-separation for monoids which concludes the proof.

3.2 Applications

We now present the two main applications of Theorem 2. We start with the most simple one positive varieties. Indeed, we have the following lemma.

▶ **Lemma 5.** *Let $\mathcal{C}$ be a positive variety. Then, $\mathcal{C}$ is an extension of itself. Moreover, if $Bool(\mathcal{C}) \neq \mathrm{REG}$, then $\mathcal{C}$ is smooth.*

That a positive variety is an extension of itself is immediate (one uses closure under inverse image). The difficulty is to prove smoothness. We may now combine Theorem 2 with Lemma 5 to get the following corollary.

▶ **Corollary 6.** *Let $\mathcal{C}$ be a positive variety such that $Bool(\mathcal{C}) \neq \mathrm{REG}$. There exists a* LogSpace *reduction from $\mathcal{C}$-separation for* NFAs *to $\mathcal{C}$-separation for monoids.*

Corollary 6 implies that for any positive variety $\mathcal{C}$, the complexity of $\mathcal{C}$-separation is the same for monoids and NFAs. We illustrate this with an example: the *star-free languages*.

▶ **Example 7.** Consider the star-free languages (SF): for every alphabet A, $\mathrm{SF}(A)$ is the least set of languages containing all singletons $\{a\}$ for $a \in A$ and closed under Boolean operations and concatenation. It is folklore and simple to verify that SF is a variety. It is known that SF-membership is in NL for monoids (this is immediate from Schützenberger's theorem [19]). On the other hand, SF-membership is PSpace-complete for NFAs. In fact, it is shown in [2] that PSpace-completeness still holds for *deterministic* finite automata (DFAs).

For SF-separation, we may combine Corollary 6 with existing results to obtain that the problem is in EXPTime and PSpace-hard for both NFAs and monoids. Indeed, the EXPTime upper bounds is proved in [14] for monoids and we may lift it to NFAs with Corollary 6. Finally, the PSpace lower bound follows from [2]: SF-membership is PSpace-hard for DFAs. This yields that SF-separation is PSpace-hard for both DFAs and NFAs (by reduction from membership to separation which is easily achieved in LogSpace when starting from a DFA). Using Corollary 6 again, we get that SF-separation is PSpace-hard for monoids as well. ◀

We turn to our second application: finitely based concatenation hierarchies. Consider a finite quotienting Boolean algebra $\mathcal{C}$. We associate another finite quotienting Boolean algebra $\mathcal{C}_{\mathbb{E}}$ which we only define for alphabets of the form $A \cup \mathbb{E}$ (this is harmless: $\mathcal{C}_{\mathbb{E}}$ is used as the output class of our reduction). Let A be an alphabet and consider the morphism $\gamma : (A \cup \mathbb{E})^* \to A^*$ defined by $\gamma(a) = a$ for $a \in A$ and $\gamma(0) = \gamma(1) = \varepsilon$. We define,

$$\mathcal{C}_{\mathbb{E}}(A \cup \mathbb{E}) = \{\gamma^{-1}(L) \mid L \in \mathcal{C}(A)\}$$

It is straightforward to verify that $\mathcal{C}_{\mathbb{E}}$ remains a finite quotienting Boolean algebra. Moreover, we have the following lemma.

▶ **Lemma 8.** *Let $\mathcal{C}$ be a finite quotienting Boolean algebra. For every $n \in \frac{1}{2}\mathbb{N}$, $\mathcal{C}_{\mathbb{E}}[n]$ is smooth and an extension of $\mathcal{C}[n]$.*

In view of Theorem 2, we get the following corollary which provides a generic reduction for levels within finitely based hierarchies.

▶ **Corollary 9.** *Let $\mathcal{C}$ be a finite basis and $n \in \frac{1}{2}\mathbb{N}$. There exists a* LogSpace *reduction from $\mathcal{C}[n]$-separation for* NFAs *to $\mathcal{C}_{\mathbb{E}}[n]$-separation for monoids.*

4 Generic upper bounds for low levels in finitely based hierarchies

In this section, we present generic complexity results for the fixed alphabet separation problem associated to the lower levels in finitely based concatenation hierarchies. More precisely, we show that for every finite basis $\mathcal{C}$ and every alphabet A, $\mathcal{C}[1/2](A)$- and $\mathcal{C}[1](A)$-separation are respectively in NL and in P. These upper bounds hold for both monoids and NFAs: we prove them for monoids and lift the results to NFAs using the reduction of Corollary 9.

▶ Remark. We do **not** present new proofs for the decidability of $\mathcal{C}[1/2]$- and $\mathcal{C}[1]$-separation when $\mathcal{C}$ is a finite quotienting Boolean algebra. These are difficult results which are proved in [17]. Instead, we recall the (inefficient) procedures which were originally presented in [17] and carefully analyze and optimize them in order to get the above upper bounds.

For the sake of avoiding clutter, we fix an arbitrary finite quotienting Boolean algebra $\mathcal{C}$ and an alphabet A for the section.

4.1 Key sub-procedure

The algorithms $\mathcal{C}[1/2](A)$- and $\mathcal{C}[1](A)$-separation presented in [17] are based on a common sub-procedure. This remains true for the improved algorithms which we present in the paper. In fact, this sub-procedure is exactly what we improve to get the announced upper complexity bounds. We detail this point here. Note that the algorithms require considering special monoid morphisms (called "$\mathcal{C}$-compatible") as input. We first define this notion.

$\mathcal{C}$-compatible morphisms. Since $\mathcal{C}$ is finite, one associates a classical equivalence $\sim_{\mathcal{C}}$ defined on A^*. Given $u, v \in A^*$, we write $u \sim_{\mathcal{C}} v$ if and only if $u \in L \Leftrightarrow v \in L$ for all $L \in \mathcal{C}(A)$. Given $w \in A^*$, we write $[w]_{\mathcal{C}} \subseteq A^*$ for its $\sim_{\mathcal{C}}$-class. Since $\mathcal{C}$ is a finite quotienting Boolean algebra, $\sim_{\mathcal{C}}$ is a congruence of finite index for concatenation (see [18] for a proof). Hence, the quotient $A^*/\sim_{\mathcal{C}}$ is a monoid and the map $w \mapsto [w]_{\mathcal{C}}$ a morphism.

Consider a morphism $\alpha : A^* \to M$ into a finite monoid M. We say that α is $\mathcal{C}$-compatible when there exists a *monoid morphism* $s \mapsto [s]_{\mathcal{C}}$ from M to $A^*/\sim_{\mathcal{C}}$ such that for every $w \in A^*$, we have $[w]_{\mathcal{C}} = [\alpha(w)]_{\mathcal{C}}$. Intuitively, the definition means that α "computes" the $\sim_{\mathcal{C}}$-classes of words in A^*. The following lemma is used to compute $\mathcal{C}$-compatible morphisms (note that the LogSpace bound holds because $\mathcal{C}$ and A is fixed).

▶ **Lemma 10.** *Given two morphisms recognizing regular languages $L_1, L_2 \subseteq A^*$ as input, one may compute in* LogSpace *a $\mathcal{C}$-compatible morphism which recognizes both L_1 and L_2.*

In view of Lemma 10, we shall assume in this section without loss of generality that our input in separation for monoids is a single $\mathcal{C}$-compatible morphism recognizing the two languages that need to be separated.

Sub-procedure. Consider two $\mathcal{C}$-compatible morphisms $\alpha : A^* \to M$ and $\beta : A^* \to N$. We say that a subset of N is *good* (for β) when it contains $\beta(A^*)$ and is closed under multiplication. For every good subset S of N, we associate a subset of $M \times 2^N$. We then consider the problem of deciding whether specific elements belong to it (this is the sub-procedure used in the separation algorithms).

▶ Remark. The set $M \times 2^N$ is clearly a monoid for the componentwise multiplication. Hence we may multiply its elements and speak of idempotents in $M \times 2^N$.

An (α, β, S)-*tree* is an unranked ordered tree. Each node x must carry a label $lab(x) \in M \times 2^N$ and there are three possible kinds of nodes:

- **Leaves**: x has no children and $lab(x) = (\alpha(w), \{\beta(w)\})$ for some $w \in A^*$.
- **Binary**: x has exactly two children x_1 and x_2. Moreover, if $(s_1, T_1) = lab(x_1)$ and $(s_2, T_2) = lab(x_2)$, then $lab(x) = (s_1 s_2, T)$ with $T \subseteq T_1 T_2$.
- **S-Operation**: x has a unique child y. Moreover, the following must be satisfied:
  1. The label $lab(y)$ is an idempotent $(e, E) \in M \times 2^N$.
  2. $lab(x) = (e, T)$ with $T \subseteq E \cdot \{t \in S \mid [e]_{\mathcal{C}} = [t]_{\mathcal{C}} \in S\} \cdot E$.

We are interested in deciding whether elements in $M \times 2^N$ are the root label of some computation tree. Observe that computing all such elements is easily achieved with a least fixpoint procedure: one starts from the set of leaf labels and saturates this set with three operations corresponding to the two kinds of inner nodes. This is the approach used in [17] (actually, the set of all root labels is directly defined as a least fixpoint and (α, β, S)-trees are not considered). However, this is costly since the computed set may have exponential size with respect to $|N|$. Hence, this approach is not suitable for getting efficient algorithms. Fortunately, solving $\mathcal{C}[1/2](A)$- and $\mathcal{C}[1](A)$-separation does not require to have the whole set of possible root labels in hand. Instead, we shall only need to consider the elements $(s, T) \in M \times 2^N$ which are the root label of some tree **and** such that T is a **singleton set**. It turns out that these specific elements can be computed efficiently. We state this in the next theorem which is the key technical result and main contribution of this section.

▶ **Theorem 11.** *Consider two $\mathcal{C}$-compatible morphisms $\alpha : A^* \to M$ and $\beta : A^* \to N$ and a good subset $S \subseteq N$. Given $s \in M$ and $t \in N$, one may test in* NL *with respect to $|M|$ and $|N|$ whether there exists an (α, β, S)-tree with root label $(s, \{t\})$.*

Theorem 11 is proved in the full version of the paper. We only present a brief outline which highlights two propositions about (α, β, S)-trees upon which the theorem is based.

We first define a complexity measure for (α, β, S)-trees. Consider two $\mathcal{C}$-compatible morphisms $\alpha : A^* \to M$ and $\beta : A^* \to N$ as well as a good subset $S \subseteq N$. Given an (α, β, S)-tree $\mathbb{T}$, we define the *operational height of* $\mathbb{T}$ as the greatest number $h \in \mathbb{N}$ such that $\mathbb{T}$ contains a branch with h S-operation nodes.

Our first result is a weaker version of Theorem 11. It considers the special case when we restrict ourselves to (α, β, S)-trees whose operational heights are bounded by a constant.

▶ **Proposition 12.** *Let $h \in \mathbb{N}$ be a constant and consider two $\mathcal{C}$-compatible morphisms $\alpha : A^* \to M$ and $\beta : A^* \to N$ and a good subset $S \subseteq N$. Given $s \in M$ and $t \in N$, one may test in* NL *with respect to $|M|$ and $|N|$ whether there exists an (α, β, S)-tree of operational height at most h and with root label $(s, \{t\})$.*

Our second result complements the first one: in Theorem 11, it suffices to consider (α, β, S)-trees whose operational heights are bounded by a constant (depending only on the class $\mathcal{C}$ and the alphabet A which are fixed here). Let us first define this constant. Given a finite monoid M, we define the $\mathcal{J}$-depth of M as the greatest number $h \in \mathbb{N}$ such that one may find h pairwise distinct elements $s_1, \ldots, s_h \in M$ such that for every $i < h$, $s_{i+1} = x s_i y$ for some $x, y \in M$

▶ Remark. The term "$\mathcal{J}$-depth" comes from the Green's relations which are defined on any monoid [4]. We do not discuss this point here.

Recall that the quotient set $A^*/{\sim_{\mathcal{C}}}$ is a monoid. Consequently, it has a $\mathcal{J}$-depth. Our second result is as follows.

▶ **Proposition 13.** *Let $h \in \mathbb{N}$ be the $\mathcal{J}$-depth of $A^*/{\sim_{\mathcal{C}}}$. Consider two $\mathcal{C}$-compatible morphisms $\alpha : A^* \to M$ and $\beta : A^* \to N$, and a good subset $S \subseteq N$. Then, for every $(s, T) \in M \times 2^N$, the following properties are equivalent:*

1. (s, T) *is the root label of some* (α, β, S)*-tree.*
2. (s, T) *is the root label of some* (α, β, S)*-tree whose operational height is at most* h*.*

In view of Proposition 13, Theorem 11 is an immediate consequence of Proposition 12 applied in the special case when h is the $\mathcal{J}$-depth of $A^*/\sim_{\mathcal{C}}$ and $m = 1$.

4.2 Applications

We now combine Theorem 11 with the results of [17] to get the upper complexity bounds for $\mathcal{C}[1/2](A)$- and $\mathcal{C}[1](A)$-separation that we announced at the begging of the section.

Application to $\mathcal{C}[1/2]$. Let us first recall the connection between $\mathcal{C}[1/2]$-separation and (α, β, S)-trees. The result is taken from [17].

▶ **Theorem 14** ([17])**.** *Let $\alpha : A^* \to M$ be a $\mathcal{C}$-compatible morphism and $F_0, F_1 \subseteq M$. Moreover, let $S = \alpha(A^*) \subseteq M$. The two following properties are equivalent:*

- *$\alpha^{-1}(F_0)$ is $\mathcal{C}[1/2]$-separable from $\alpha^{-1}(F_1)$.*
- *for every $s_0 \in F_0$ and $s_1 \in F_1$, there exists no (α, α, S)-tree with root label $(s_0, \{s_1\})$.*

By Theorem 11 and the Immerman–Szelepcsényi theorem (which states that NL = *co*-NL), it is straightforward to verify that checking whether the second assertion in Theorem 14 holds can be done in NL with respect to $|M|$. Therefore, the theorem implies that $\mathcal{C}[1/2](A)$-separation for monoids is in NL. This is lifted to NFAs using Corollary 9.

▶ **Corollary 15.** *For every finite basis $\mathcal{C}$ and alphabet A, $\mathcal{C}[1/2](A)$-separation is in* NL *for both* NFAs *and monoids.*

Application to $\mathcal{C}[1]$. We start by recalling the $\mathcal{C}[1]$-separation algorithm which is again taken from [17]. In this case, we consider an auxiliary sub-procedure which relies on (α, β, S)-trees.

Consider a $\mathcal{C}$-compatible morphism $\alpha : A^* \to M$. Observe that M^2 is a monoid for the componentwise multiplication. We let $\beta : A^* \to M^2$ as the morphism defined by $\beta(w) = (\alpha(w), \alpha(w))$ for every $w \in A^*$. Clearly, β is $\mathcal{C}$-compatible: given $(s, t) \in M^2$, it suffices to define $[(s, t)]_{\mathcal{C}} = [s]_{\mathcal{C}}$. Using (α, β, S)-trees, we define a procedure $S \mapsto \mathit{Red}(\alpha, S)$ which takes as input a good subset $S \subseteq M^2$ (for β) and outputs a subset $\mathit{Red}(\alpha, S) \subseteq S$.

$$\mathit{Red}(\alpha, S) = \{(s, t) \in S \mid (s, \{(t, s)\}) \in M \times 2^{M^2} \text{ is the root label of an } (\alpha, \beta, S)\text{-tree}\} \subseteq S$$

It is straightforward to verify that $\mathit{Red}(\alpha, S)$ remains a good subset of M^2. We now have the following theorem which is taken from [17].

▶ **Theorem 16** ([17])**.** *Let $\alpha : A^* \to M$ be a morphism into a finite monoid and $F_0, F_1 \subseteq M$. Moreover, let $S \subseteq M^2$ be the greatest subset of $\alpha(A^*) \times \alpha(A^*)$ such that $\mathit{Red}(\alpha, S) = S$. Then, the two following properties are equivalent:*

- *$\alpha^{-1}(F_0)$ is $\mathit{Bool}(\mathit{Pol}(\mathcal{C}))$-separable from $\alpha^{-1}(F_1)$.*
- *for every $s_0 \in F_0$ and $s_1 \in F_1$, $(s_0, s_1) \notin S$.*

Observe that Theorem 11 implies that given an arbitrary good subset S of $\alpha(A^*) \times \alpha(A^*)$, one may compute $\mathit{Red}(\alpha, S) \subseteq S$ in P with respect to $|M|$. Therefore, the greatest subset S of $\alpha(A^*) \times \alpha(A^*)$ such that $\mathit{Red}(\alpha, S) = S$ can be computed in P using a greatest fixpoint algorithm. Consequently, Theorem 16 yields that $\mathcal{C}[1](A)$-separation for monoids is in P. Again, this is lifted to NFAs using Corollary 9.

▶ **Corollary 17.** *For every finite basis $\mathcal{C}$ and alphabet A, $\mathcal{C}[1](A)$-separation is in* P *for both* NFAs *and monoids.*

5 The Straubing-Thérien hierarchy

In this final section, we consider one of the most famous concatenation hierarchies: the Straubing-Thérien hierarchy [21, 22]. We investigate the complexity of separation for the levels 3/2 and 2.

▶ Remark. Here, the alphabet is part of the input. For fixed alphabets, these levels can be handled with the generic results presented in the previous section (see Theorem 18 below).

The basis of the Straubing-Thérien hierarchy is the trivial variety ST[0] defined by $\mathrm{ST}[0](A) = \{\emptyset, A^*\}$ for every alphabet A. It is known and simple to verify (using induction) that all half levels are positive varieties and all full levels are varieties.

The complexity of separation for the level one (ST[1]) has already been given a lot of attention. Indeed, this level corresponds to a famous class which was introduced independently from concatenation hierarchies: the piecewise testable languages [20]. It was shown independently in [3] and [11] that ST[1]-separation is in P for NFAs (and therefore for DFAs and monoids as well). Moreover, it was also shown in [5] that the problem is actually P-complete for NFAs and DFAs[1]. Additionally, it is shown in [3] that ST[1/2]-separation is in NL.

In the paper, we are mainly interested in the levels ST[3/2] and ST[2]. Indeed, the Straubing-Thérien hierarchy has a unique property: the generic separation results of [17] apply to these two levels as well. Indeed, these are also the levels 1/2 and 1 in another finitely based hierarchy. Consider the class AT of *alphabet testable languages*. For every alphabet A, $\mathrm{AT}(A)$ is the set of all Boolean combinations of languages A^*aA^* for $a \in A$. One may verify that AT is a variety and that $\mathrm{AT}(A)$ is finite for every alphabet A. Moreover, we have the following theorem which is due to Pin and Straubing [8] (see [18] for a modern proof).

▶ **Theorem 18** ([8]). *For every $n \in \frac{1}{2}\mathbb{N}$, we have $\mathrm{AT}[n] = \mathrm{ST}[n+1]$.*

The theorem implies that $\mathrm{ST}[3/2] = \mathrm{AT}[1/2]$ and $\mathrm{ST}[2] = \mathrm{AT}[1]$. Therefore, the results of [17] yield the decidability of separation for both ST[3/2] and ST[2] (the latter is the main result of [17]). As expected, this section investigates complexity for these two problems.

5.1 The level 3/2

We have the following tight complexity bound for ST[3/2]-separation.

▶ **Theorem 19.** ST[3/2]*-separation is* PSpace*-complete for both* NFAs *and monoids.*

The PSpace upper bound is proved by building on the techniques introduced in the previous section for handling the level 1/2 of an arbitrary finitely based hierarchies. Indeed, we have $\mathrm{ST}[3/2] = \mathrm{AT}[1/2]$ by Theorem 18. However, let us point out that obtaining this upper bound requires some additional work: the results of Section 4 apply to the setting in which the alphabet is fixed, this is not the case here. In particular, this is why we end up with a PSpace upper bound instead of the generic NL upper presented in Corollary 15. The detailed proof is postponed to the full version of the paper.

In this abstract, we focus on proving that ST[3/2]-separation is PSpace-hard. The proof is presented for NFAs: the result can then be lifted to monoids with Corollary 6 since ST[3/2]

[1] Since ST[1] is a variety, P-completeness for ST[1]-separation can also be lifted to monoids using Corollary 6.

is a positive variety. We use a LogSpace reduction from the quantified Boolean formula problem (QBF) which is among the most famous PSpace-complete problems.

We first describe the reduction. For every quantified Boolean formula Ψ, we explain how to construct two languages L_Ψ and L'_Ψ. It will be immediate from the presentation that given Ψ as input, one may compute NFAs for L_Ψ and L'_Ψ in LogSpace. Then, we show that this construction is the desired reduction: Ψ is true if and only if L_Ψ is not ST[3/2]-separable from L'_Ψ.

Consider a quantified Boolean formula Ψ and let n be the number of variables it involves. We assume without loss of generality that Ψ is in prenex normal form and that the quantifier-free part of Ψ is in conjunctive normal form (QBF remains PSpace-complete when restricted to such formulas). That is,

$$\Psi = Q_n\ x_n \cdots Q_1\ x_1\ \varphi$$

where $x_1 \dots x_n$ are the variables of Ψ, $Q_1, \dots, Q_n \in \{\exists, \forall\}$ are quantifiers and φ is a quantifier-free Boolean formula involving the variables $x_1 \dots x_n$ which is in conjunctive normal form.

We describe the two regular languages L_Ψ, L'_Ψ by providing regular expressions recognizing them. Let us first specify the alphabet over which these languages are defined. For each variable x_i occurring in Ψ, we create two letters that we write x_i and $\overline{x_i}$. Moreover, we let,

$$X = \{x_1, \dots, x_n\} \quad \text{and} \quad \overline{X} = \{\overline{x_1}, \dots, \overline{x_n}\}$$

Additionally, our alphabet also contains the following letters: $\#_1, \dots, \#_i, \$$. For $0 \leq i \leq n$, we define an alphabet B_i. We have:

$$B_0 = X \cup \overline{X} \quad \text{and} \quad B_i = X \cup \overline{X} \cup \{\#_1, \dots, \#_i, \$\}$$

Our languages are defined over the alphabet B_n: $L_\Psi, L'_\Psi \subseteq B_n^*$. They are built by induction: for $0 \leq i \leq n$ we describe two languages $L_i, L'_i \subseteq B_i^*$ (starting with the case $i = 0$). The languages L_Ψ, L'_Ψ are then defined as L_n, L'_n.

Construction of L_0, L'_0. The language L_0 is defined as $L_0 = (B_0)^*$. The language L'_0 is defined from the quantifier-free Boolean formula φ. Recall that by hypothesis φ is in conjunctive normal form: $\varphi = \bigwedge_{j \leq k} \varphi_j$ were φ_i is a disjunction of literals. For all $j \leq k$, we let $C_j \subseteq B_0 = X \cup \overline{X}$ as the following alphabet:

- Given $x \in X$, we have $x \in C_j$, if and only x is a literal in the disjunction φ_j.
- Given $\overline{x} \in \overline{X}$, we have $\overline{x} \in C_j$, if and only $\neg x$ is a literal in the disjunction φ_j.

Finally, we define $L'_0 = C_1 C_2 \cdots C_k$.

Construction of L_i, L'_i for $i \geq 1$. We assume that L_{i-1}, L'_{i-1} are defined and describe L_i and L'_i. We shall use the two following languages in the construction:

$$T_i = (\#_i x_i (B_{i-1} \setminus \{\overline{x_i}\})^* \$ x_i)^* \quad \text{and} \quad \overline{T_i} = (\#_i \overline{x_i} (B_{i-1} \setminus \{x_i\})^* \$ \overline{x_i})^*$$

The definition of L_i, L'_i from L_{i-1}, L'_{i-1} now depends on whether the quantifier Q_i is existential or universal.

- If Q_i is an existential quantifier (i.e. $Q_i = \exists$):

$$\begin{aligned} L_i &= (\#_i (x_i + \overline{x_i}) L_{i-1} \$ (x_i + \overline{x_i}))^* \#_i \\ L'_i &= (\#_i (x_i + \overline{x_i}) L'_{i-1} \$ (x_i + \overline{x_i}))^* \#_i \$ \left(T_i \#_i + \overline{T_i} \#_i\right) \end{aligned}$$

- If the Q_i is an universal quantifier (i.e. $Q_i = \forall$):

$$\begin{aligned} L_i &= (\#_i(x_i + \overline{x_i})L_{i-1}\$(x_i + \overline{x_i}))^*\#_i \\ L'_i &= \overline{T_i}\#_i\$(\#_i(x_i + \overline{x_i})L'_{i-1}\$(x_i + \overline{x_i}))^*\#_i\$T_i\#_i \end{aligned}$$

Finally, L_Ψ, L'_Ψ are defined as the languages $L_n, L'_n \subseteq (B_n)^*$. It is straightforward to verify from the definition, than given Ψ as input, one may compute NFAs for L_Ψ and L'_Ψ in LogSpace. Consequently, it remains to prove that this construction is the desired reduction. We do so in the following proposition.

▶ **Proposition 20.** *For every quantified Boolean formula Ψ, Ψ is true if and only if L_Ψ is not* ST[3/2]*-separable from L'_Ψ.*

Proposition 20 is proved by considering a stronger result which states properties of all the languages L_i, L'_i used in the construction of L_Ψ, L'_Ψ (the argument is an induction on i). While we postpone the detailed proof to the full version of the paper, let us provide a sketch which presents this stronger result.

Proof of Proposition 20 (sketch). Consider a quantified Boolean formula Ψ. Moreover, let $B_0, \ldots, B_n$ and $L_i, L'_i \subseteq (B_i)^*$ as the alphabets and languages defined above. The key idea is to prove a property which makes sense for all languages L_i, L'_i. In the special case when $i = n$, this property implies Proposition 20.

Consider $0 \leq i \leq n$. We write Ψ_i for the sub-formula $\Psi_i := Q_i\ x_i \cdots Q_1\ x_1\ \varphi$ (with the free variables $x_{i+1}, \ldots, x_n$). In particular, $\Psi_0 := \varphi$ and $\Psi_n := \Psi$. Moreover, we call "*i-valuation*" a sub-alphabet $V \subseteq B_i$ such that,

1. $\#_1, \ldots, \#_i, \$ \in V$ and $x_1, \overline{x_1}, \ldots, x_i, \overline{x_i} \in V$, and,
2. for every j such that $i < j \leq n$, one of the two following property holds:
   - $x_j \in V$ and $\overline{x_j} \notin V$, or,
   - $x_j \notin V$ and $\overline{x_j} \in V$.

Clearly, an i-valuation corresponds to a truth assignment for all variables x_j such that $j > i$ (i.e. those that are free in Ψ_i): when the first (resp. second) assertion in Item 2 holds, x_j is assigned to $\top$ (resp. $\bot$). Hence, abusing terminology, we shall say that an i-valuation V *satisfies* Ψ_i if Ψ_i is true when replacing its free variables by the truth values provided by V.

Finally, for $0 \leq i \leq n$, if $V \subseteq B_i$ is an i-valuation, we let $[V] \subseteq V^*$ as the following language. Given $w \in V^*$, we have $w \in [V]$ if and only if for every $j > i$ either $x_j \in \mathsf{alph}(w)$ or $\overline{x_j} \in \mathsf{alph}(w)$ (by definition of i-valuations, exactly one of these two properties must hold). Proposition 20 is now a consequence of the following lemma.

▶ **Lemma 21.** *Consider $0 \leq i \leq n$. Then given an i-valuation V, the two following properties are equivalent:*

1. *Ψ_i is satisfied by V.*
2. *$L_i \cap [V]$ is not* ST[3/2]*-separable from $L'_i \cap [V]$.*

Lemma 21 is proved by induction on i using standard properties of the polynomial closure operation (see [18] for example). The proof is postponed to the full version of the paper. Let us explain why the lemma implies Proposition 20.

Consider the special case of Lemma 21 when $i = n$. Observe that $V = B_n$ is an n-valuation (the second assertion in the definition of n-valuations is trivially true since there are no j such that $n < j \leq n$). Hence, since $\Psi = \Psi_n$ and $L_\Psi, L'_\Psi = L_n, L'_n$, the lemma yields that,

1. Ψ is satisfied by V (i.e. Ψ is true).
2. $L_\Psi \cap [V]$ is not ST[3/2]-separable from $L'_\Psi \cap [V]$.

Moreover, we have $[V] = (B_n)^*$ by definition. Hence, we obtain that Ψ is true if and only if L is not ST[3/2]-separable from L' which concludes the proof of Proposition 20. ◀

5.2 The level two

For the level two, there is a gap between the lower and upper bound that we are able to prove. Specifically, we have the following theorem.

▶ **Theorem 22.** ST[2]*-separation is in* EXPTime *and* PSpace*-hard for both* NFAs *and monoids.*

Similarly to what happened with ST[3/2], the EXPTime upper bound is obtained by building on the techniques used in the previous section. Proving PSpace-hardness is achieved using a reduction from ST[3/2]-separation (which is PSpace-hard by Theorem 19). The reduction is much simpler than what we presented for ST[3/2] above. It is summarized by the following proposition.

▶ **Proposition 23.** *Consider an alphabet A and $H, H' \subseteq A^*$. Let $B = A \cup \{\#, \$\}$ with $\#, \$ \notin A$, $L = \#(H'\#(A^*\$\#)^*)^*H\#(A^*\$\#)^* \subseteq B^*$ and $L' = \#(H'\#(A^*\$\#)^*)^* \subseteq B^*$. The two following properties are equivalent:*

1. *H is* ST[3/2]*-separable from H'.*
2. *L is* ST[2]*-separable from L'.*

Proposition 23 is proved using standard properties of the polynomial and Boolean closure operations. The argument is postponed ot the full version of the paper. It is clear than given as input NFAs for two languages H, H', one may compute NFAs for the languages L, L' defined Proposition 23 in LogSpace. Consequently, the proposition yields the desired LogSpace reduction from ST[3/2]-separation for NFAs to ST[2]-separation for NFAs. This proves that ST[2]-separation is PSpace-hard for NFAs (the result can then be lifted to monoids using Corollary 6) since ST[2] is a variety).

6 Conclusion

We showed several results, all of them raising new questions. First we proved that for many important classes of languages (including all positive varieties), the complexity of separation does not depend on how the input languages are represented. A natural question is whether the technique can be adapted to encompass more classes. In particular, one may define more permissive notions of positive varieties by replacing closure under inverse image by weaker notions. For example, many natural classes are *length increasing positive varieties*: closure under inverse image only has to hold for length increasing morphisms (*i.e.*, morphisms $\alpha : A^* \to B^*$ such that $|\alpha(w)| \geq |w|$ for every $w \in A^*$). For example, the levels of another famous concatenation hiearchy, the dot-depth [1] (whose basis is $\{\emptyset, \{\varepsilon\}, A^+, A^*\}$) are length increasing positive varieties. Can our techniques be adapted for such classes? Let us point out that there exists no example of natural class $\mathcal{C}$ for which separation is decidable and strictly harder for NFAs than for monoids. However, there are classes $\mathcal{C}$ for which the question is open (see for example the class of locally testable languages in [10]).

We also investigated the complexity of separation for levels 1/2 and 1 in finitely based concatenation hierarchies. We showed that when the alphabet is fixed, the problems are respectively in NL and P for any such hierarchy. An interesting follow-up question would be to push these results to level 3/2, for which separation is also known to be decidable in any finitely based concatenation hierarchy [9]. A rough analysis of the techniques used in [9] suggests that this requires moving above P.

Finally, we showed that in the famous Straubing-Thérien hierarchy, ST[3/2]-separation is PSpace-complete and ST[2]-separation is in EXPTime and PSpace-hard. Again, a natural question is to analyze ST[5/2]-separation whose decidability is established in [9].

References

1 Janusz A. Brzozowski and Rina S. Cohen. Dot-Depth of Star-Free Events. *Journal of Computer and System Sciences*, 5(1):1–16, 1971.

2 Sang Cho and Dung T. Huynh. Finite automaton aperiodicity is PSPACE-complete. *Theoretical Computer Science*, 88(1):99–116, 1991.

3 Wojciech Czerwiński, Wim Martens, and Tomáš Masopust. Efficient Separability of Regular Languages by Subsequences and Suffixes. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP'13)*, pages 150–161. Springer-Verlag, 2013.

4 James Alexander Green. On the Structure of Semigroups. *Annals of Mathematics*, 54(1):163–172, 1951.

5 Tomás Masopust. Separability by piecewise testable languages is PTIME-complete. *Theoretical Computer Science*, 711:109–114, 2018.

6 Jean-Éric Pin. The Dot-Depth Hierarchy, 45 Years Later. In *The Role of Theory in Computer Science - Essays Dedicated to Janusz Brzozowski*, pages 177–202, 2017.

7 Jean-Éric Pin. Mathematical Foundations of Automata Theory. In preparation, 2018. URL: `https://www.irif.fr/~jep/PDF/MPRI/MPRI.pdf`.

8 Jean-Eric Pin and Howard Straubing. Monoids of upper triangular Boolean matrices. In *Semigroups. Structure and Universal Algebraic Problems*, volume 39 of *Colloquia Mathematica Societatis Janos Bolyal*, pages 259–272. North-Holland, 1985.

9 Thomas Place. Separating Regular Languages with Two Quantifier Alternations. Unpublished, a preliminary version can be found at `https://arxiv.org/abs/1707.03295`, 2018.

10 Thomas Place, Lorijn van Rooijen, and Marc Zeitoun. Separating Regular Languages by Locally Testable and Locally Threshold Testable Languages. In *Proceedings of the 33rd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, FSTTCS'13, pages 363–375, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

11 Thomas Place, Lorijn van Rooijen, and Marc Zeitoun. Separating Regular Languages by Piecewise Testable and Unambiguous Languages. In *Proceedings of the 38th International Symposium on Mathematical Foundations of Computer Science*, MFCS'13, pages 729–740. Springer-Verlag, 2013.

12 Thomas Place and Marc Zeitoun. Separating Regular Languages with First-order Logic. In *Proceedings of the Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic (CSL'14) and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'14)*, pages 75:1–75:10. ACM, 2014.

13 Thomas Place and Marc Zeitoun. The tale of the quantifier alternation hierarchy of first-order logic over words. *SIGLOG News*, 2(3):4–17, 2015.

14 Thomas Place and Marc Zeitoun. Separating Regular Languages with First-Order Logic. *Logical Methods in Computer Science*, 12(1), 2016.

15 Thomas Place and Marc Zeitoun. Adding successor: A transfer theorem for separation and covering. Unpublished, a preliminary version can be found at `http://arxiv.org/abs/1709.10052`, 2017.

16 Thomas Place and Marc Zeitoun. Going higher in the First-order Quantifier Alternation Hierarchy on Words. Unpublished, a preliminary version can be found at `https://arxiv.org/abs/1404.6832`, 2017.

17 Thomas Place and Marc Zeitoun. Separation for Dot-depth Two. In *Proceedings of the 32th Annual ACM/IEEE Symposium on Logic in Computer Science, (LICS'17)*, pages 202–213. IEEE Computer Society, 2017.

18 Thomas Place and Marc Zeitoun. Generic results for concatenation hierarchies. *Theory of Computing Systems (ToCS)*, 2018. Selected papers from CSR'17.

19 Marcel Paul Schützenberger. On Finite Monoids Having Only Trivial Subgroups. *Information and Control*, 8:190–194, 1965.

20 Imre Simon. Piecewise testable events. In *2nd GI Conference on Automata Theory and Formal Languages*, pages 214–222, 1975.

21 Howard Straubing. A Generalization of the Schützenberger Product of Finite Monoids. *Theoretical Computer Science*, 13(2):137–150, 1981.

22 Denis Thérien. Classification of Finite Monoids: The Language Approach. *Theoretical Computer Science*, 14(2):195–208, 1981.

Reducing Transducer Equivalence to Register Automata Problems Solved by "Hilbert Method"

Adrien Boiret
University of Warsaw, Poland
a.boiret@mimuw.edu.pl

Radosław Piórkowski
University of Warsaw, Poland
r.piorkowski@mimuw.edu.pl

Janusz Schmude
University of Warsaw, Poland
j.schmude@mimuw.edu.pl

Abstract

In the past decades, classical results from algebra, including Hilbert's Basis Theorem, had various applications in formal languages, including a proof of the Ehrenfeucht Conjecture, decidability of HDT0L sequence equivalence, and decidability of the equivalence problem for functional tree-to-string transducers.

In this paper, we study the scope of the algebraic methods mentioned above, particularily as applied to the functionality problem for register automata, and equivalence for functional register automata. We provide two results, one positive, one negative. The positive result is that functionality and equivalence are decidable for MSO transformations on unordered forests. The negative result comes from a try to extend this method to decide functionality and equivalence on macro tree transducers. We reduce macro tree transducers equivalence to an equivalence problem for some class of register automata naturally relevant to our method. We then prove this latter problem to be undecidable.

2012 ACM Subject Classification Theory of computation → Transducers

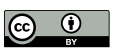
Keywords and phrases formal language, Hilbert's basis theorem, transducers, register automata, equivalence problem, unordered trees, MSO transformations

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.48

Related Version `https://arxiv.org/abs/1806.04361`

Funding This work was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC consolidator grant LIPA, agreement no. 683080). Furthermore, it was partially supported by the NCN grant 2016/21/B/ST6/01505.

Acknowledgements We would like to thank Mikołaj Bojańczyk, from the University of Warsaw, for introducing us to the topic of using the Hilbert Method to decide equivalence of register automata, and for his active participation in the finding of this result and the writing of this paper.



1 Introduction

The study of finite-state machines, such as transducers [15, 8, 14] or register automata [2, 3], and of logic specifications, such as MSO-definable transformations [9], provides a theoretical ground to study document and data processing.

In this paper, we will consider the equivalence problem of functional transducers. We focus on register automata, i.e. transducers that store values in a finite number of registers that can be updated or combined after reading an input symbol. Streaming String Transducers (SST) [2] and Streaming Tree Transducers (STT) [3] are classes of register automata (see for example [4]) where the equivalence is decidable for the *copyless* restriction, i.e. the case where each register update cannot use the same register twice. This restriction makes SST equivalent to MSO-definable string transformations. Macro tree transducers (MTT) [11], an expressive class of tree transducers for which equivalence decidability remains a challenging open problem, can be seen as register automata, whose registers store tree contexts. Although equivalence is not known to be decidable for the whole class, there exists a linear size increase fragment of decidable equivalence, that is equivalent to MSO-definable tree transformations, and can be characterized by a restriction on MTT quite close to copyless [10].

Some equivalence decidability results have been proven on register automata without copyless restrictions [16, 5], by reducing to algebraic problems such as ideal inclusion and by applying Hilbert's Basis Theorem and other classical results of algebraic geometry. In this paper we will refer to this as the "Hilbert Method". This method was used to prove diverse results, dating back to at least the proof of the Ehrenfeucht Conjecture [1], and the sequence problem for HDT0L [13, 12]. It has recently found new applications in formal languages; for example, equivalence was proven decidable for general tree-to-string transducers by seeing them as copyful register automata on words [16].

In this paper, we use an abstraction of these previous applications of the "Hilbert Method" as presented in [6]. We apply these preexisting results to the study of unordered forest transductions – and notably MSO functions. Note that equivalence of MSO-definable transductions on unordered forests is not a straightforward corollary of the ordered case, as the loss of order makes equivalence more difficult to identify. We also try to apply those methods to obtain decidability of MTT equivalence. For unordered forests, we obtain a positive result, showing that register automata on forest contexts with one hole have decidable functionality and equivalence. For the attempt to study MTT, we prove an undecidability result on register automata using polynomials and composition, which means the natural extension of this approach does not yield a definitive answer for the decidability of MTT equivalence.

Layout

Section 2 presents the notions of algebra, register automata, and the notions necessary to use an abstraction of the "Hilbert Method" as presented in [6]. **Section 3** is dedicated to the proof of the positive result that we can apply the "Hilbert Method" to contexts of unordered forests with at most one hole (i.e. the algebra of unordered forests with limited substitution). This provides a class of register automata encompassing MSO functions on unordered forests where functionality is decidable. Finally, **Section 4** describes how applying a method similar as in Section 3 to study MTT equivalence leads to studying register automata on the algebra of polynomials with the substitution operation, a class whose functionality and equivalence we prove to be undecidable.

2 Preliminaries

Algebras. An *algebra* $\mathbf{A} = (A, \rho_1, \dots, \rho_n)$ is a (potentially infinite) set of elements A, and a finite number of operations $\rho_1, \dots, \rho_n$. Each operation is a function $\rho : A^k \to A$ for some $k \in \mathbb{N}$.

Polynomials. For an algebra $\mathbf{A}$ and a set $X = \{x_1, \dots, x_n\}$ of variables, we note $A[X]$ the set of terms over $A \cup X$. A *polynomial function* of $\mathbf{A}$ is a function $f : A^k \to A$. For example on $\mathbf{A} = (\mathbb{Q}, +, \times)$, the term $\times(+(x, 2), +(x, y))$ induces the polynomial function $f : (x, y) \mapsto (x + 2)(x + y)$. The definition of polynomial functions can be extended to functions $f \colon A^k \to A^m$ by product of their output: if $f_1, \dots, f_m$ are polynomial functions from A^k to A, then $f' \colon \overline{a} \in A^k \mapsto (f_1(\overline{a}), \dots, f_m(\overline{a}))$ is a polynomial function from A^k to A^m. Note that polynomial functions are closed under composition.

One can define the *algebra of polynomials over* $\mathbf{A}$ *with variable set* X, denoted $\mathbf{A}[X]$. Its elements are equivalence classes of terms over $A \cup X$ with the operations of $\mathbf{A}$, where two terms are called *equivalent* if they induce identical polynomial functions. $\mathbf{A}[X]$ can be seen as an algebra that subsumes $\mathbf{A}$, with natural definition of operations. A classical example of this construction is the ring of polynomials $(\mathbb{Q}[x], +, \times)$, obtained from the ring $(\mathbb{Q}, +, \times)$.

By adding the substitution operation $(-)[X := (-)]$ to $\mathbf{A}[X]$, we get a new algebra called a *composition algebra of polynomials* and denoted $\mathbf{A}[X]^{\mathsf{subs}}$. Homomorphisms of such algebras are called *composition homomorphisms*. For brevity we write $(-)[x_i := (-)]$ for the substitution of a single $x_i \in X$. Examples of such algebras include well-nested words with a placeholder symbol "?", as used in the registers of Streaming Tree Transducers [3], or tree contexts with variables in their leaves, as used in Macro Tree Transducers [11].

Simulation. Following the abstractions as they are presented in [6] [1], we define simulations between algebras in a way that is relevant to the use of the "Hilbert Method".

▶ **Definition 1.** Let $\mathbf{A}$ and $\mathbf{B}$ be algebras. We say that $\alpha : A \to B^n$ is a *simulation* of $\mathbf{A}$ in $\mathbf{B}$ if for every operation $\rho : A^m \to A$ of $\mathbf{A}$, there is a polynomial function $f : B^{m \times n} \to B^n$ of $\mathbf{B}$ such that $\alpha \circ \rho = f \circ \alpha$, where α is defined from A^m to $B^{m \times n}$ coordinate-wise. If such a simulation α exists, we say that $\mathbf{A}$ is *simulated by* $\mathbf{B}$ ($\mathbf{A} \preceq_{\mathsf{pol}} \mathbf{B}$).

$$\begin{array}{ccc} A^m & \xrightarrow{(\alpha, \cdots, \alpha)} & B^{m \times n} \\ \rho \downarrow & & \downarrow f \\ A & \xrightarrow[\alpha]{} & B^n \end{array}$$

The following lemma states that simulations extend to composition algebras.

▶ **Lemma 2.** *Let* $\boldsymbol{Q}[X] = (\mathbb{Q}[X], +, \times)$. *If* $\mathbf{A} \preceq_{\mathsf{pol}} \boldsymbol{Q}[X]$ *and* A *is an infinite set, then* $\mathbf{A}[Y]^{\mathsf{subs}} \preceq_{\mathsf{pol}} \boldsymbol{Q}[X][Y]^{\mathsf{subs}}$.

[1] As of this version's redaction, Part 11 of [6] is the part relevant for this paper. This and any theorem or page number can change in future versions of [6].

Proof. If there is a simulation α from $\mathbf{A}$ to $\mathbb{Q}[X]$, then α can be extended into a simulation $\widetilde{\alpha}$ from $\mathbf{A}[Y]$ to $\boldsymbol{Q}[X][Y]$ by setting $\alpha(Y) = Y$, and requiring $\widetilde{\alpha}$ to be a homomorphism. It is important to check that $\widetilde{\alpha}$ indeed is a function (i.e. preserves equivalence of terms): if terms $t_1, t_2 \in A[Y]$ induce the same functions on A, then $\widetilde{\alpha}(t_1)(Y)$ and $\widetilde{\alpha}(t_2)(Y)$ are polynomial functions that are equal on $\alpha(A)$. Since $\alpha(A)$ is an infinite subset of $\mathbb{Q}[X]$, $\alpha(t_1)$ and $\alpha(t_2)$ are equal everywhere. The proof of injectivity is straightforward. Let $P(Y), Q(Y) \in A[Y]$ be any two nonequivalent terms. Then there is a tuple $\overline{a}$ of A such that $P(\overline{a}) \neq Q(\overline{a})$. If $\widetilde{\alpha}(P) = \widetilde{\alpha}(Q)$, we would have $\alpha(P(\overline{a})) = \alpha(P)[Y := \overline{a}] = \alpha(Q)[Y := \overline{a}] = \alpha(Q(\overline{a}))$. This would contradict the injectivity of α on A. ◀

Typed Algebras. Some of the algebras we consider are *multi-sorted*, which is to say that their elements are divided between a finite number of types. A *multi-sorted algebra* is an algebra $\mathbf{A} = (A, \rho_1, \dots, \rho_n)$ such that:

- A can be partitioned into $A_1, \dots, A_m$,
- each operation ρ is a function $\rho : A_{i_0} \times \cdots \times A_{i_k} \to A_j$.

To each $a \in A$ we associate a *type*, which is a unique i such that $a \in A_i$. Note that in a multi-sorted algebra $\mathbf{A}$ polynomial functions are typed $f : A_{i_0} \times \cdots \times A_{i_k} \to A_j$, and simulation and substitutions must be defined type-wise.

Register automata. In this paper we will work on register automata that make a single bottom-up pass on an input ranked tree, use a finite set of states, and a finite set of registers with values in $A$for some algebra $\mathbf{A}$. When the automaton reads an input symbol, it updates its register values as a polynomial function of $\mathbf{A}$ applied to the register values in its subtrees. This formalism is already present in the literature: streaming tree transducers [2], for example, are register automata on input words and register values in the algebra of words on an alphabet Σ, with the concatenation operation.

A *signature* Σ is a finite set of symbols a, each with a corresponding finite rank $\mathrm{rk}(a) \in \mathbb{N}$. A *ranked tree* is a term on this signature Σ: if $a \in \Sigma$, $\mathrm{rk}(a) = n$, and $t_1, \dots, t_n$ are trees, then $a(t_1, \dots, t_n)$ is a tree.

▶ **Definition 3.** Let $\mathbf{A} = (A, \rho_1, \dots, \rho_n)$ be an algebra. A *bottom-up register automaton* with values in $\mathbf{A}$ (or $\mathbf{A}$-RA) is a tuple $M = (\Sigma, n, Q, \delta, f_{\mathrm{out}})$, where:

- Σ is a ranked set
- n is the number of A-registers used by M
- Q is a finite set of states
- δ is a finite set of transitions of form $a(q_1, \dots, q_k) \to q, f$ where $a \in \Sigma$ of rank k, $\{q, q_1, \dots, q_k\} \subset Q$, and $f : A^{n \times k} \to A^n$ a polynomial function of $\mathbf{A}$.
- f_{out} is a partial output function that to some states $q \in Q$ associates $f_q : A^n \to A$ a polynomial function of $\mathbf{A}$.

A *configuration* of M is a n-uple $(q, \overline{r})$ where $q \in Q$ is a state and $\overline{r} = (r_1, \dots, r_n) \in A^n$ is a n-uple of register values in A. We define by induction the fact that a tree t can reach a configuration $(q, \overline{r})$, noted $t \to (q, \overline{r})$: If $a \in \Sigma$ of rank k, $a(q_1, \dots, q_k) \to q, f$ a rule of δ, and for $0 \leqslant i \leqslant k$, $t_i \to (q_i, \overline{r}_i)$, then

$$a(t_1, \dots, t_k) \to (q, f(\overline{r}_1, \dots, \overline{r}_k)).$$

M determines a relation $[\![M]\!]$ from trees to values in A. It is defined using f_{out} as a final step: if $f_{\mathrm{out}}(q) = f_q$, and $t \to (q, \overline{r})$, then $f_q(\overline{r}) \in [\![M]\!](t)$.

We say that a **A**-RA is *functional* if $[\![M]\!]$ is a function. We say that a **A**-RA is *deterministic* if for all $a, q_1, \dots, q_k$ there is at most one rule $a(q_1, \dots, q_k) \to q, f$ in δ. Any deterministic **A**-RA is functional.

Note that on a multi-sorted algebra, we further impose that every state q has a certain type $A_{i_1} \times \cdots \times A_{i_n}$, i.e. if $(q, \overline{r})$ is a configuration of M, then $\overline{r} \in A_{i_1} \times \cdots \times A_{i_n}$.

"Hilbert Method". We now describe an abstraction [6] of the classical algebra methods that are used in the literature [16, 5] to decide equivalence of functional register automata over certain algebras (e.g. $(\Sigma^*, \cdot)$) using what we will refer to as the "Hilbert Method". More specifically, as it is always easy to prove the semi-decidability of non-equivalence of functional **A**-RA, by guessing two runs on the same input with different outputs, this method aims to prove that the functionality and equivalence problems over functional **A**-RA are semi-decidable.

This method can be described as a 4-step process:

- Simulate **A** by $\boldsymbol{Q}$, hence reducing **A**-RA equivalence to $\boldsymbol{Q}$-RA equivalence
- Functional $\boldsymbol{Q}$-RA equivalence can be reduced to $\boldsymbol{Q}$-RA zeroness, i.e. checking if a $\boldsymbol{Q}$-RA only outputs 0.
- $\boldsymbol{Q}$-RA zeroness can be reduced to ideal inclusion problem in $\overline{\mathbb{Q}}[X]$, i.e. the ring of polynomials with algebraic numbers as coefficients
- Ideal inclusion problem in $\overline{\mathbb{Q}}[X]$ is decidable

These results exist in the literature. We will provide references as well as an intuition of the main mechanisms in these proofs. For the first point, the reduction from **A**-RA equivalence to $\boldsymbol{Q}$-RA equivalence, an example is provided in [16], where **A** is $(\Sigma^*, \cdot)$. In essence, this part of the method amounts to a simulation as described in Definition 1. If $\mathbf{A} \preceq_{\mathsf{pol}} \mathbf{B}$ with simulation α, then any **A**-RA M can be simulated by a **B**-RA M', in the sense that M outputs $a \in A$ for an input t if and only M' outputs $\alpha(a) \in B^k$ for the same input t. This gives a reduction from **A**-RA equivalence to **B**-RA equivalence.

The second point is presented in the proof of Theorem 11.8 of [6]. If M and M' are two functional $\boldsymbol{Q}$-RA of same domain, using a natural product construction, one can create M'' that runs M and M' in parallel, then computes the difference of outputs between M and M'. Thus M and M' are equivalent iff M'' only outputs 0.

The third point can be found in the proof of Theorem 11.8 of [6]. The idea is to express $\boldsymbol{Q}$-RA zeroness as a set problem (with polynomial grammars as an intermediary in [6]). We want to find for each state q the set X_q of register values that M can hold in state q. These states obey to some inclusion equations: if $a(q_1, \dots, q_k) \to q, f$ is a rule of M, then $f(X_{q_1} \times ... \times X_{q_n}) \subseteq X_q$. Furthermore, if zeroness is true for M, then for every q such that final output function f_q is defined, $f_q(X_q) \subseteq \{0\}$. Interestingly, if such a family of sets of $\mathbb{Q}^n$ $(X_q)_{q \text{ state of } M}$ exists to satisfy those inclusions, then there exists a family of ideals of $\overline{\mathbb{Q}}[X]$, $(S_q)_{q \text{ state of } M}$, that satisfy opposite inclusions (Lemma 11.5 of [6]).

The fourth point uses classical algebra results to find such a family of ideal sets. The proof, as it is presented in Theorem 11.3 of [6], works as follows: Hilbert's Basis Theorem ensures that all families of ideals $(S_q)_{q \text{ state of } M}$ can be enumerated. For each of these families, it can be checked using Groebner Basis whether it respects a set of inclusions or not. Eventually, if a solution exists, it will be found, making this ideal problem, and thus $\boldsymbol{Q}$-RA zeroness, semi-decidable.

For this paper, we point out a few natural extensions to those methods and establish Theorem 4 and Corollary 5 as a basis for our work.

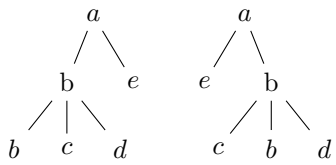


Figure 1 Two representations of the same unordered tree.

The first remark is that one can consider more problems than functional equivalence. Functionality itself can be studied with these methods. It is preserved by simulations, and $\boldsymbol{Q}$-RA functionality can be reduced to zeroness: instead of comparing two functional $\boldsymbol{Q}$-RA in the second point, M'' can run two copies of the same $\boldsymbol{Q}$-RA M and compute the output difference. M is functional iff M'' only outputs 0.

The second remark is that the classical algebra results (Hilbert Basis Theorem, Groebner Basis, algebraic closure of a field...) used in the fourth point extend to any computable field $\mathcal{K}$. In consequence, Theorem 11.8 of [6] holds for any $\mathcal{K}$-RA. Since the polynomial ring $\mathcal{K}[X]$ is a subring of a computable field $\mathcal{K}(X)$ (rational functions over $\mathcal{K}$), it holds for $\mathcal{K}[X]$-RA as well. We therefore state the following theorem.

▶ **Theorem 4.** *Let $\boldsymbol{Q}[X] = (\mathbb{Q}[X], +, \times)$. Functionality of $\boldsymbol{Q}[X]$-RA and equivalence of functional $\boldsymbol{Q}[X]$-RA are decidable.*

The result of Theorem 4 can be extended to other algebras using simulations from algebra to algebra. Indeed, if $\mathbf{A} \preceq_{\mathsf{pol}} \mathbf{B}$, then any $\mathbf{A}$-RA can be simulated by a $\mathbf{B}$-RA, and problems of functionality and equivalence reduce from $\mathbf{A}$-RA to $\mathbf{B}$-RA.

▶ **Corollary 5.** *Let $\mathbf{A}$ be an algebra. If $\mathbf{A} \preceq_{\mathsf{pol}} \boldsymbol{Q}[X]$, then functionality of $\mathbf{A}$-RA and equivalence of functional $\mathbf{A}$-RA are decidable.*

3 Unordered forests are simulated by polynomials

In this section we will show that the unordered tree forests (and more generally – the unordered forest algebra [7] that contains both forests and contexts with one hole) can be simulated in the sense of Definition 1 by polynomials with rational coefficients over a variable x (noted $\mathbb{Q}[x]$) with the operations $+, \times$. This, combined with Corollary 5, implies the decidability of functionality and equivalence for a class of Forests-RA. We then prove that this class can express all MSO-transformations on unordered forests.

An *unordered tree* on a finite signature Σ is an unranked tree (i.e. every node can have arbitrarily many children), but the children of a node form an unordered multiset, rather than an ordered list. For example, the following figure displays two representations of the same unordered tree. An *unordered forest* is a multiset of unordered trees.

Unordered forests can thus be defined as an algebra $\mathbf{UF} = (\mathsf{UF}, +, \{\mathsf{root}_a\})$:

1. UF is the set of unordered forests, including $\emptyset$ – the empty forest;
2. the operations are:
   - binary operation $+$ is the multiset addition,
   - for each letter $a \in \Sigma$, unary operation root_a: if $h = t_1 + \cdots + t_n$, then $\mathsf{root}_a(h) = a(t_1, \ldots, t_n)$.

In the rest of this paper, we will reason with a unary signature (and thus a unique root operation). This is done without loss of generality, as unordered forests on a finite signature

$\Sigma = \{a_1, \dots, a_n\}$ can easily be encoded by forests on a unary signature. To express it as a polynomial simulation, we can say that $\alpha(\emptyset) = \emptyset$, and that for all $1 \leqslant i \leqslant n$

$$\alpha(\mathsf{root}_{a_i}(h)) = \mathsf{root}^i(\mathsf{root}(\emptyset) + \mathsf{root}(\alpha(h)))$$

3.1 Encoding forests into polynomials

This subsection's aim is to prove the following result:

▶ **Proposition 6.** $(\mathsf{UF}, +, \mathsf{root})$ *is simulated by* $(\mathbb{Q}[x], +, \times)$.

To this end we construct an injective homomorphism $\phi : \mathsf{UF} \to \mathbb{Q}[x]$. This ϕ associates injectively to each forest a rational polynomial p. It is important to check that two identical forests with different representations (as in Figure 1) will not obtain different value by ϕ. Furthermore, the operations $+, \mathsf{root}$ must be encoded as ψ_+, ψ_r, two polynomial functions in $(\mathbb{Q}[x], +, \times)$, such that $\phi(h + h') = \psi_+(\phi(h), \phi(h'))$ and $\phi(\mathsf{root}(h)) = \psi_r(\phi(h))$.

Note that the term "polynomial" suffers here from semantic overload. We will take care to differentiate, on one hand, rational polynomials (i.e. the elements of $\mathbb{Q}[x]$, e.g. $2x - 7$), denoted by variants on letters p, q, and on the other hand, polynomial functions on the algebra $(\mathbb{Q}[x], +, \times)$ (e.g. $\psi : (p, q) \mapsto q \times q + 2p$), denoted by variants on the letter ψ.

Since $+$ in **UF** is both associative and commutative, we choose ψ_+ to be multiplication between rational polynomials: $\psi_+ : (p, q) \mapsto p \times q$. This leaves root to encode. To ensure that ϕ is injective, we would like to pick ψ_r so that ϕ sends all $\mathsf{root}(h)$ to pairwise different irreducible polynomials. This is done by picking $\psi_r : p \mapsto 2 + x \times p$ and using the Eisenstein's criterion with prime number 2: if a monic polynomial has all its nonleading coefficients divisible by 2, and the constant coefficient not divisible by 4, then this polynomial is irreducible over $\mathbb{Q}$. From there we define ϕ inductively: $\phi(\emptyset) = 1$, $\phi(h + h') = \phi(h) \times \phi(h')$, and $\phi(\mathsf{root}(h)) = 2 + x \times \phi(h)$. It is clear that ϕ respects the condition of polynomial simulation that any operation of **UF** must be encoded as polynomial operation in $(\mathbb{Q}[x], +, \times)$.

This leads directly to the proof of Proposition 6: ϕ is a simulation from **UF** to $\mathbb{Q}[x]$ as defined in Definition 1. It is injective, the operation $+$ is encoded by the polynomial function $\psi_+ : (p, q) \mapsto p \times q$, and the operation root is encoded by the polynomial function $\psi_r : p \mapsto 2 + x \times p$.

3.2 Extension to contexts

The combination of Corollary 5 and Proposition 6 gives decidability results on the class of **UF**-RA. The transducers of this class read a ranked input, and manipulate registers with values in UF. As an example, an **UF**-RA can read a binary input, and output the unordered forests that it encodes in a "First Child Next Sibling" manner, that is to say the left child in the input corresponds to the child in the output, and the right child in the input corresponds to the brother in the output. Note that this is an adaptation of classical FCNS encoding of unranked **ordered** trees in binary trees, but where the order is forgotten.

This can be described by a one-state one-register **UF**-RA that uses rules of form

$$(a, q, q) \to (q, (x, y) \mapsto \mathsf{root}_a(x) + y).$$

However, **UF**-RA have their restriction: since root and $+$ are the only two operations allowed, registers can only store subtrees to be placed at the bottom of the output. This leaves the class without the ability to combine subtrees of its output as freely as the MSO

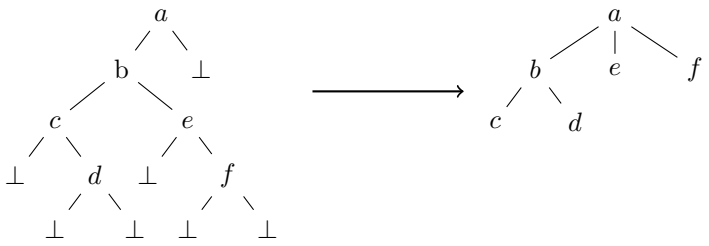


Figure 2 "FCNS" decoding.

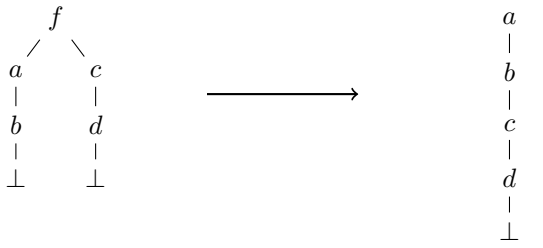
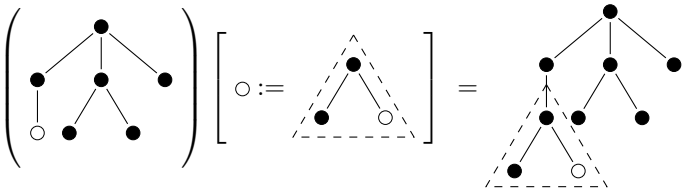


Figure 3 Subtree concatenation.

logic does. As an example, it is impossible to create an **UF**-RA that, if given an input $f(u, v)$ where u and v are two unary subtrees, outputs the subtree u above the subtree v as shown in Figure 3.

To get a more general class of register automata, that can perform such superpositions, we need to allow registers to store contexts, rather than forests. While the use of the Hilbert Methods for algebras of general contexts remains a difficult and interesting open problem, we will show that forest contexts with at most one hole are simulated by polynomials of $\mathbb{Z}[x]$.

We use the unordered version of 2-sorted Forest Algebra [7], consisting of unordered forests of trees and contexts with at most one hole. Since the previous subsection deals with an algebra of forests, to avoid confusion, we will call this the *Unordered Context and Forest* algebra (noted **UCF**). Using the definition of composition algebras, **UCF** is a subset of $\mathbf{UF}[\circ]^{\mathsf{subs}}$, where we impose that the replacable variable $\circ$ occurs at most once.



On this algebra, we will show the following results:

▶ **Theorem 7.** *UCF is simulated by* $(\mathbb{Q}[x], +, \times)$.

▶ **Corollary 8.** *Functionality of UCF-RA and equivalence of functional UCF-RA are decidable.*

Lemma 2 ensures that since $\mathbf{UF} \preceq_{\mathsf{pol}} (\mathbb{Q}[x], +, \times)$, then $\mathbf{UF}[\circ]^{\mathsf{subs}} \preceq_{\mathsf{pol}} \mathbb{Q}[x][y]^{\mathsf{subs}}$, i.e. $\mathbb{Q}[x, y]$ where only y can be substituted. **UCF** is the restriction of $\mathbf{UF}[\circ]^{\mathsf{subs}}$ to its elements with at most one occurrence of $\circ$. This forms a 2-sorted algebra. We consider its natural match in $\mathbb{Q}[x][y]^{\mathsf{subs}}$: Let $\mathbf{A}$ be the 2-sorted algebra:

- The universe is $A := \{p(x) + yq(x) : p, q \in \mathbb{Q}[x]\} \subseteq \mathbb{Q}[x, y]$.
- The types are $A_0 := \mathbb{Q}[x], A_1 := A \setminus A_0$.
- The operations are:
  - multiplication, defined only on pairs of types: $(0, 0), (0, 1), (1, 0)$,
  - $(-)[y := (-)]$.

▶ **Lemma 9.** *UCF is simulated by* **A**.

Proof. We call $\alpha : \mathbf{UF}[\circ]^{\mathsf{subs}} \to \mathbb{Q}[x][y]^{\mathsf{subs}}$ the homomorphism obtained by extending last subsection's ϕ with mapping the substitution variable $\circ$ to y. We restrict α to terms with at most one occurrence of $\circ$. The image of $h \in \mathbf{UCF}$ will then be a term of $\mathbb{Q}[x][y]^{\mathsf{subs}}$ with at most one occurrence of y. If $\circ$ never appears in h, then y never appears in $\alpha(h)$, thus $\alpha(h) \in \mathbf{A}_0$. If $\circ$ appears once in h, then y appears once in $\alpha(h)$, thus $\alpha(h) \in \mathbf{A}_1$. ◀

We now prove that **A** is simulated by $\mathbb{Q}[x]$ *without* substitution.

▶ **Lemma 10.** **A** *is simulated by* $(\mathbb{Q}[x], +, \times)$.

Proof. We will use encoding of **A** in $\mathbb{Q}[x] \times \mathbb{Q}[x]$ given by $p(x) + yq(x) \mapsto (p, q)$. Provided this, we encode operations $+, \times, (-)[y := (-)]$ in a straightforward manner. For example, for the composition operation in **A**, we see that $(p(x) + yq(x))[y := (p'(x) + yq'(x))]$ is equal to $p(x) + p'(x)q(x) + yq'(x)q(x)$. Hence, in pairs of $\mathbb{Q}$[x], $(-)[y := (-)]$ is encoded by $\psi_{(-)[y:=(-)]} : (p, q, p', q') \mapsto (p + p'q, q'q)$. ◀

Since $\preceq_{\mathsf{pol}}$ is a transitive relation, Lemma 10 and Lemma 9 give Theorem 7. Once Theorem 7 is proven, Corollary 5 gives Corollary 8.

Note that this proof extends to contexts with a bounded number of holes. We can add N substitution variables $\circ_1, \ldots, \circ_N$ to **UF**. Lemma 2 gives a homomorphism α_N that ensures $\mathbf{UF}[\circ_1, \ldots, \circ_N]^{\mathsf{subs}} \preceq_{\mathsf{pol}} \mathbb{Q}[x][y_1, \ldots, y_N]^{\mathsf{subs}}$. One could then define contexts with at most M occurrences of variables $\mathbf{UCF}^{\leq \mathrm{M}}$. In a manner similar to Lemma 9, we can find a finitely-sorted algebra that contains $\alpha_N(\mathbf{UCF}^{\leq \mathrm{M}})$, i.e. an algebra of all polynomials of $\mathbb{Q}[x][y_1, \ldots, y_N]$ with a degree $\leqslant M$ regarding the variables $y_1, \ldots, y_N$. Then, in a manner similar to Lemma 10, we can show that finite degree composition can be encoded in $(\mathbb{Q}[x], +, \times)$.

▶ **Corollary 11.** $\boldsymbol{UCF}^{\leq \mathrm{M}}$ *is simulated by* $(\mathbb{Q}[x], +, \times)$. *Functionality of* $\boldsymbol{UCF}^{\leq \mathrm{M}}$*-RA and equivalence of functional* $\boldsymbol{UCF}^{\leq \mathrm{M}}$*-RA are decidable.*

3.3 Encompassing of MSO

Corollary 8 gives decidability results on the class of **UCF**-RA. We motivated this class as a relevant extension of **UF**-RA by exhibiting a transformation (see Figure 3) that required contexts to be expressed. However, this class is not immediately relevant in its properties or expressiveness. In this section, we prove that **UCF**-RA can express strictly more than all MSO-definable transformations on unordered trees. Note that **UCF**-RA define functions from *binary ordered* trees to UCF, not from UF to UF. We say that an **UCF**-RA expresses a function $f : \mathsf{UF} \to \mathsf{UF}$ if for a binary tree t that is the "FCNS" encoding of a forest h, its image for the tree t is $f(h)$.

We briefly present a definition of MSO formulae and transformations. More complete definitions exist elsewhere in the literature (e.g. [9]).

The syntax of monadic second order logic (MSO) is:

$$\phi := \phi \wedge \phi \mid \neg\phi \mid \exists x \phi \mid \exists X \phi \mid x \in X$$

where lower cases x are node variables, and upper cases X are set variables. This syntax is enriched by different relations to describe the structure of the objects we consider:

- For binary trees (BT), we add two relations $\mathsf{Ch}_L(x, y)$ and $\mathsf{Ch}_R(x, y)$ that express that y is the left child (resp. right child) of x.
- For unranked ordered forests (OF), we add $\mathsf{FC}(x, y)$, that expresses that y is the first child of x, and $\mathsf{NS}(x, y)$ that express that y is the brother directly to the right of x.
- For unranked unordered forests (UF), we only add the relation $\mathsf{Ch}(x, y)$, that expresses that y is a child of x. The relation "Sibling" would only be syntactic sugar.

An *MSO-definable transformation* with n copies is a transformation that for each input node x, makes n output nodes $x_1, \dots, x_n$. The presence or absence of an edge in the output are dictated by formulae defining the transformation. A MSO-definable transformation is characterized by its formulae $\varphi_{\mathcal{R},i,j}$ for each $1 \leqslant i, j \leqslant n$, and each structure relation $\mathcal{R}$ (e.g. FC and NS if the output is ordered forests).

For example, if one wanted to reverse left and right children in binary trees, this would be a transformation definable in $\mathrm{MSO}_{BT\to BT}$ with one copy, where $\varphi_{\mathsf{Ch}_L,1,1}(x, y) = \mathsf{Ch}_R(x, y)$, i.e. y_1 is x_1's left child in the output iff y was x's right child in the input, and conversely $\varphi_{\mathsf{Ch}_R,1,1}(x, y) = \mathsf{Ch}_L(x, y)$.

We note that this definition can express transformations between any two tree algebras. For example, the "FCNS" decoding of Figure 2 can be encoded in MSO from binary trees to UF. Since we will use different combinations of input-output in this part, we introduce the notation $\mathrm{MSO}_{\bullet\to\bullet}$ to denote MSO from one type of trees to the other. $\mathrm{MSO}_{BT\to OF}$ designs MSO-definable functions from binary trees to ordered forests, and $\mathrm{MSO}_{UF\to UF}$ designs MSO-definable functions from unordered forests to unordered forests.

▶ **Proposition 12.** *Every function of $MSO_{UF\to UF}$ can be described by an **UCF**-RA.*

The proof we provide to show this Proposition has three arguments:

1. $\mathrm{MSO}_{UF\to UF}$ can be represented by functions of $\mathrm{MSO}_{BT\to OF}$.
2. Bottom-UP Streaming Tree Transducers (STT) [3] describe all functions of $\mathrm{MSO}_{BT\to OF}$.
3. Bottom-Up STT can be expressed as register automata.

From $\mathrm{MSO}_{UF\to UF}$ to $\mathrm{MSO}_{BT\to OF}$. We say that a binary tree t *represents* an unordered forest h if the "FCNS" decoding of t as represented in Figure 2 is h. Note that t is not unique for h, but every t represents a unique h. Similarily, we can say that an unranked ordered forest h *represents* an unordered forest h' if by forgetting the siblings' order in h, we get h'. Once again such an h is not unique for h', but every h represents a unique h'. We can extend this notion to MSO transformation.

▶ **Definition 13.** A function $f \in \mathrm{MSO}_{BT\to OF}$ *represents* a function $f' \in \mathrm{MSO}_{UF\to UF}$ if:

- For every unordered forest h such that f' is defined over h, then there exists at least one binary tree t such that t represents h, and f is defined over t
- For every binary tree t such that f is defined over t, f' is defined over the unordered forests h such that t represents h, and $f(t)$ represents $f'(h)$.

Once again such an f is not unique for f', but every f represents a unique f'. Furthermore, it is always possible to find a representant f for an MSO-definable function f'.

▶ **Lemma 14.** *If $f' \in MSO_{UF\to UF}$, then there exists $f \in MSO_{BT\to OF}$ that represents f'.*

Proof. We start by encoding the input, transforming f' into a function of $\text{MSO}_{BT\to UF}$. To modify f' so that it transforms trees that represent h into $f'(h)$, one has to replace every occurence of $\mathsf{Ch}(x,y)$ into $\varphi_{\mathsf{Ch},i,j}$ by its "FCNS" encoding, i.e. $\exists z \mid \mathsf{Ch}_L(x,z) \wedge \mathsf{Ch}^*_L(z,y)$.

Encoding the output requires to change $\varphi_{\mathsf{Ch},i,j}$ into two relations $\varphi_{\mathsf{FC},i,j}$ and $\varphi_{\mathsf{NS},i,j}$, i.e. to artificially order the siblings of the output forest. To that effect, we note that from the BT-formula $\varphi_{\mathsf{Ch},i,j}$, one can describe in BT-MSO a set $S_{x,i,j} = \{y \mid \varphi_{\mathsf{Ch},i,j}(x,y)\}$. Since $S_{x,i,j}$ is a set of input nodes of a binary tree, it is totally ordered by their occurrence in the infix run. This order can be expressed as a BT-MSO relation. We can then decide to order all the children y_j of the output node x_i.

To find the first child of a node in the output, we say that $\varphi_{\mathsf{FC},i,j}(x,y)$ if j is the first index where $S_{x,i,j} \neq \emptyset$ and y is its first element. Similarily, to find the next sibling of a node in the output, we say that $\varphi_{\mathsf{NS},i,j}(y,z)$ if $\exists x \mid \varphi_{\mathsf{Ch},k,i}(x,y) \wedge \varphi_{\mathsf{Ch},k,j}(x,z)$, and either $i = j$ and y, z are consecutive elements of $S_{x,k,i}$, or y is the last element of $S_{x,k,i}$, z is the first element of $S_{x,k,j}$, and j is the first index bigger than i such that $S_{x,k,j} \neq \emptyset$. ◀

From $\text{MSO}_{BT\to OF}$ to STT to RA. The next step is to use an existing result from the literature [3] that describes a model of transducers that describes all $\text{MSO}_{BT\to OF}$. The formalism in question are *Streaming Tree Transducers* (STT). A STT is an automaton on nested words (words representing trees) that maintains a stack of register configurations. The nesting of the words dictates how this stack behaves: each opening letter $<a$ stores the current variable values in the stack to start with fresh ones, then each closing letter $a>$ uses the current variable values and the top of the stack to generate new values for the registers.

In [3], STT are limited to linear functions for the update of their value. Furthermore, the paper proves that without loss of expression, one can consider *Bottom-Up STT*, where reading an opening symbol $<a$ resets the state as well as the registers. On such STT, the behavior of a STT reading the nested word of a subtree does not depend on what occurs before or after, and its computation behaves like a register automaton reading a tree in a bottom-up manner. We will consider the class of Bottom-Up STT that read nested representations of binary trees (Bottom-Up BT STT).

▶ **Proposition 15.** *Every function of $\text{MSO}_{BT\to UF}$ is described by a Bottom-Up BT STT.*

▶ **Proposition 16.** *Every function of a Bottom-Up BT STT is described by an OCF-RA.*

Proposition 15 comes directly from Theorems 3.7 and 4.6 of [3]: 3.7 explains Bottom-Up BT-STT are as powerful as general BT-STT, and 4.6 states that STT can describe any function of $\text{MSO}_{BT\to UF}$. Proposition 16 is not directly proven in [3] but their definition of Bottom-Up STT is made specifically to that end. We provide more details in the appendix.

End of Proof. To turn that OCF-RA into an **UCF**-RA, we just have to change the ordered concatenation of OCF to the unordered concatenation of **UCF**. By combining Lemma 14, Proposition 15 and Proposition 16, we conclude our proof of Proposition 12.

We note that every MSO-definable function can be described by a **UCF**-RA, however the converse is not true; consider a function that creates output of exponential size (whereas MSO can only describe functions of linear size increase). Consider unary input trees of form $\mathsf{root}_a^n(\bot)$, and a 1-counter **UCF**-RA with rules $\bot \to q(\mathsf{root}())$, and $a(q(h)) \to h + h$. The image doubles in size each time a symbol is read. Unsurprisingly, this counterexample uses the copyful nature of **UCF**-RA, as copyless restrictions tend to limit the expressivity power of register automata to MSO classes [2, 3].

4 On decidability of MTT equivalence. Equivalence of polynomials-RA with composition is undecidable

In this section, we tru to use the "Hilbert Method" to study the equivalence problem on Macro Tree Transducers (MTT) [11]. MTT have numerous definitions. For this paper, we will consider them to be register automata on an algebra of ranked trees with an operation of substitution on the leaves; observe this is exactly $\mathsf{OrderedTrees}[X]^{\mathsf{subs}}$-RA. The algebra $\mathsf{OrderedTrees}$ (ranked trees without substitution on the leaves) can be simulated by words with concatenation (via nested word encoding). Words with concatenation can be encoded by $\mathbb{Q}$ (see, for example, the proof of Corollary 10.11 [6]). Thus, $\mathsf{OrderedTrees} \preceq_{\mathsf{pol}} \mathbb{Q}$. Finally, by Lemma 2, we have that $\mathsf{OrderedTrees}[X]^{\mathsf{subs}} \preceq_{\mathsf{pol}} \mathbb{Q}[X]^{\mathsf{subs}}$. This means that if equivalence is decidable for $\mathbb{Q}[X]^{\mathsf{subs}}$-RA, then MTT equivalence is decidable. Unfortunately, we will show that even with one variable x, the register automata of $\mathbb{Q}[x]^{\mathsf{subs}}$-RA have undecidable functionality and equivalence:

▶ **Theorem 17.** *The functionality problem for $\mathbb{Q}[x]^{\mathsf{subs}}$-RA and equivalence problem for functional $\mathbb{Q}[x]^{\mathsf{subs}}$-RA are undecidable.*

We prove this undecidability result by reducing the reachability problem for 2-counter machines to the equivalence problem on deterministic $\mathbb{Q}[x]^{\mathsf{subs}}$-RA with a monadic input (i.e. that reads words rather than trees). This means that the actual theorem we prove is slightly more powerful than Theorem 17. Its full extent is described in Theorem 20. We recall the definition of a 2-counter machine.

▶ **Definition 18.** A 2-counter machine (2CM) is a pair $M = (Q, \delta)$, where:
- Q is a finite set of states,
- $\delta : Q \times \{0,1\} \times \{0,1\} \to Q \times \{-1,0,1\} \times \{-1,0,1\}$ is a *total* transition function.

A configuration of M is a triplet of one state and two nonnegative integer values (or counters) $(q, c_1, c_2) \in \mathbb{Q} \times \mathbb{N} \times \mathbb{N}$. We describe how to use transitions between configurations: $(q, c_1, c_2) \to (q', c_1', c_2')$ if there exists $(q, b_1, b_2) \to (q', d_1, d_2)$ in δ such that for $i \in \{1, 2\}$: $c_i = 0 \iff b_i = 0$ and $c_i' = c_i + d_i$. Note that to ensure that no register wrongfully goes into the negative, we assume wlog that if there exists $(q, b_1, b_2) \to (q', d_1, d_2)$ in δ, then $d_i = -1 \implies b_i = 1$ (i.e. we can only decrease a non-zero counter).

The *2CM reachability problem* can be expressed as such: starting from an initial configuration $(q_0, 0, 0)$, can we access the state $q_f \in Q$, i.e. is there a configuration (q_f, c_1, c_2) such that $(q_0, 0, 0) \to^* (q_f, c_1, c_2)$. It is well known that 2CM reachability is undecidable.

Reduction of 2CM Reachability to $\mathbb{Q}[x]^{\mathsf{subs}}$-RA Equivalence. Let $M = (Q, \delta)$ be a n-states 2CM. We rename its states $Q = \{1, \dots, n\}$. We consider the 2CM reachability problem in M from state 1 to state n.

We simulate this machine with a $\mathbb{Q}[x]^{\mathsf{subs}}$-RA M'. It will have only one state $q_{M'}$: the configurations (q, c_1, c_2) of M will be encoded in 3 registers of M'. It will work on a signature $\bot \cup \delta$, where $\bot$ is of rank 0 and every transition $(q, b_1, b_2) \to (q', d_1, d_2)$ of δ is a symbol of rank 1. Intuitively, reading a symbol $(q, b_1, b_2) \to (q', d_1, d_2)$ in M' models executing this transition in M. The automaton will have 6 registers: 3 to encode the configurations of M' and 3 containing auxiliary polynomials useful to test if the input sequence of transitions describes a valid computation in M.

We encode the configurations (q, n_1, n_2) in 3 registers as follows:
- register r_q holds the (number of the) current state.
- registers r_{c_1}, r_{c_2} hold n_1, n_2 – the current values of the counters.

We now encode transitions in M as register operations in M'. When reading a transition $(i, b_1, b_2) \to (j, d_1, d_2)$, the update of the configuration is natural. However, we must ensure that we are allowed to use this transition in the current configuration.

To this end, we keep in M' a witness register. Its value will be 0 if and only if the sequence of transitions read as an input does not constitute a valid path in M. To update such a register, when a transition $(i, b_1, b_2) \to (j, d_1, d_2)$ is read, we need to check that $r_q = i$ and that $r_{c_i} = 0 \iff b_i = 0$.

For the state $q \in \{1, \dots, n\}$, we design $P_{q=i}$, a polynomial such that $P_{q=i}(i) \neq 0$ and for every other value $1 \leqslant j \leqslant n$, $P_{q=i}(j) = 0$: $P_{q=i}(x) := \prod_{1 \leqslant j \leqslant n}^{i \neq j} (x - j)$. This approach cannot work for counters, as there is no absolute bound to their value. To remedy that problem, we will design *for each* m a polynomial $P_{c=0}^{\leqslant m}$ such that $P_{c=0}^{\leqslant m}(0) \neq 0$ and for every other value, $1 \leqslant j \leqslant m$, $P_{c=0}^{\leqslant m}(j) = 0$. $P_{c=0}^{\leqslant m}(x) := \prod_{1 \leqslant j \leqslant m} (x - j)$. Intuitively, $P_{c=0}^{\leqslant m}$ works as a test for counters in the m-th step of M, since counters c_1, c_2 cannot exceed the value m at that point. This means that $P_{c=0}^{\leqslant m}$ will have to be stored and updated in a register of its own. To this end, we introduce the last three registers of M':

- the register r_+. After m steps, $r_+ = m$.
- the register r_{zt}. After m steps, $r_{zt} = P_{c=0}^{\leqslant m}$.
- the witness register r_w. After m steps, $r_w \neq 0 \iff$ we read a valid path in M.

We describe how to update the registers of M' when reading an input symbol $(i, b_1, b_2) \to (j, d_1, d_2)$. Note that according to our definition of $\mathbb{Q}[x]^{\mathsf{subs}}$-RA, the new values $\overline{r}'$ are computed as a function of the old value of $\overline{r}$. This means that any value on the right of the assignation symbol $\leftarrow$ is the value before reading $(i, b_1, b_2) \to (j, d_1, d_2)$.

- $r_q \leftarrow j$, $r_{c_1} \leftarrow r_{c_1} + d_1$, $r_{c_2} \leftarrow r_{c_2} + d_2$,
- $r_+ \leftarrow r_+ + 1$, $r_{zt} \leftarrow r_{zt} \times (x - r_+ - 1)$,
- $r_w \leftarrow r_w \times T_q \times T_1 \times T_2$, where:
  - $T_q = (P_{q=i})[x := (r_q)]$,
  - for $i \in \{1, 2\}$, $T_i = \begin{cases} r_{c_i} \text{ if } b_i = 1, \\ (r_{zt})[x := (r_{c_i})] \text{ if } b_i = 0. \end{cases}$

This update strategy ensures that each counter does what we established its role to be. The only register for which this is not trivial is r_w. We show that $r_w = 0$ if and only if we failed to read a proper path in M.

We proceed by induction on the number of steps. The induction hypothesis is that a mistake happened before the m-th step if and only if $r_w = 0$ before reading the m-th symbol. If such is the case, r_w will stay at zero for every subsequent step, as the new value of r_w is always a multiple of the previous ones. If the error occurs exactly at the m-th step, it means that the m-th letter of the input was a transition $(i, b_1, b_2) \to (j, d_1, d_2)$, but r_q was not i (and hence $T_q = (P_{q=i})[x := (r_q)] = 0$), or that for this transition to apply we need the counter c_i to be 0 when it was not (or conversely assumed it > 0 when it was 0). This last case is caught by T_i. By using $T_i = r_{c_i}$, we have $T_i = 0$ exactly when we were wrong. If $b_i = 0$ then we assume $c_i = 0$. We know that $c_i \leq m$, where m is the number of step taken. By using $T_i = (r_{zt})[x := (r_{c_i})] = P_{c=0}^{\leqslant m}(c_i)$, we have that $T_i = 0$ exactly when $0 < c_i \leq m$.

The final step of the reduction comes by picking the output function for the only state of M'. We pick $f(\overline{r}) := (P_{q=i})[x := (r_q)] \times r_w$. The only way for the output to not be 0 is if r_q ends in n (i.e. we reached state n) and if $r_w \neq 0$ (i.e. we used a valid path). In other words, the following Lemma holds.

▶ **Lemma 19.** *$[\![M']\!]$ is the constant 0 function if and only if n is not reachable from 1 in M*

By comparing M' to a $\mathbb{Q}[x]^{\mathsf{subs}}$-RA M_0 performing the constant 0 function, we get that deciding equivalence on functional $\mathbb{Q}[x]^{\mathsf{subs}}$-RA would allow to decide 2CM Reachability. Similarly, running nondeterministically M' and M_0, we get that deciding functionality on $\mathbb{Q}[x]^{\mathsf{subs}}$-RA would allow to decide 2CM Reachability. This leads to the proof of Theorem 17. More than that, we show a more thorough result:

▶ **Theorem 20.** *The equivalence problem for deterministic $\mathbb{Q}[x]^{\mathsf{subs}}$-RA is undecidable, even with a monadic input alphabet. The functionality problem for $\mathbb{Q}[x]^{\mathsf{subs}}$-RA is undecidable, even with a one-letter monadic alphabet.*

The first point is given directly by the point above: M' is a deterministic $\mathbb{Q}[x]^{\mathsf{subs}}$-RA on a monadic alphabet $\bot \cup \delta$, that is to say, the input of M' is a word where each letter is an element of δ.

For the second point, we imagine a slight alteration of this proof where the input alphabet is $\{a, \bot\}$ where $\bot$ is of rank 0 and a of rank 1, that is to say, the input of the $\mathbb{Q}[x]^{\mathsf{subs}}$-RA would be a word of form a^k. In this new version, M' is no longer deterministic, but guesses each time what transition of M to emulate. When M' reads a^k, it either guesses a correct path of length k, or makes a mistake and returns 0. M' is functional iff it cannot guess a run that produces something else than 0, i.e. iff n is not reachable from 1 in M.

5 Conclusion

We use "Hilbert Methods" to study equivalence problems on register automata. To apply these methods to register automata on contexts, we consider algebras with a substitution operation. To show the decidability of equivalence on **UCF**-RA, a class that subsumes MSO-definable transformations in unordered forests, we use the fact that bounded degree substitution can be encoded into $+, \times$ in $\mathbb{Q}[X]$. However, when applying the same method to Macro Tree Transducers, we are led to consider register automata on $\mathbb{Q}[X]^{\mathsf{subs}}$, whose equivalence we prove to be undecidable. In essence, for the "Hilbert Methods" we consider to provide positive results, it seems necessary to limit the use of composition.

Future developments of this work could then consist of finding other acceptable restrictions on the use of composition in $\mathbb{Q}[X]$ that still allows for decidability results in register automata. Another possible avenue is to use the properties of $\preceq_{\mathsf{pol}}$ to prove negative results: if $\mathbf{A} \preceq_{\mathsf{pol}} \mathbf{B}$, and register automata have undecidable problems in $\mathbf{A}$, then this negative results propagates to $\mathbf{B}$. Finally, "Hilbert Methods" can apply to a huge variety of algebras (e.g. **UCF** in this paper or $\mathbb{Q}^n$ in [5]). They provide decidability results on register automata on algebras with nontrivial structure properties like commutativity of operations (e.g. children in **UCF**) that make the usual methods to decide equivalence difficult to apply.

References

1 Michael H. Albert and J. Lawrence. A Proof of Ehrenfeucht's Conjecture. *Theor. Comput. Sci.*, 41:121–123, 1985.

2 Rajeev Alur and Pavol Cerný. Expressiveness of streaming string transducers. In *FSTTCS*, volume 8 of *LIPIcs*, pages 1–12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

3 Rajeev Alur and Loris D'Antoni. Streaming Tree Transducers. *J. ACM*, 64(5):31:1–31:55, 2017. doi:10.1145/3092842.

4 Rajeev Alur, Loris D'Antoni, Jyotirmoy V. Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular Functions and Cost Register Automata. In *LICS*, pages 13–22. IEEE Computer Society, 2013.

5 Michael Benedikt, Timothy Duff, Aditya Sharad, and James Worrell. Polynomial automata: Zeroness and applications. In *LICS*, pages 1–12. IEEE Computer Society, 2017.

6 Mikołaj Bojańczyk. Automata toolbox, May 2018. URL: `https://www.mimuw.edu.pl/~bojan/paper/automata-toolbox-book`.

7 Mikołaj Bojańczyk and Igor Walukiewicz. Forest algebras. In *Logic and Automata*, volume 2 of *Texts in Logic and Games*, pages 107–132. Amsterdam University Press, 2008.

8 Christian Choffrut. Minimizing subsequential transducers: a survey. *Theor. Comput. Sci.*, 292(1):131–143, 2003.

9 Bruno Courcelle. Monadic Second-Order Definable Graph Transductions: A Survey. *Theor. Comput. Sci.*, 126(1):53–75, 1994.

10 Joost Engelfriet and Sebastian Maneth. Macro Tree Translations of Linear Size Increase are MSO Definable. *SIAM J. Comput.*, 32(4):950–1006, 2003.

11 Joost Engelfriet and Heiko Vogler. Macro Tree Transducers. *J. Comput. Syst. Sci.*, 31(1):71–146, 1985.

12 Juha Honkala. A short solution for the HDT0L sequence equivalence problem. *Theor. Comput. Sci.*, 244(1-2):267–270, 2000.

13 Karel Culik II and Juhani Karhumäki. The Equivalence of Finite Valued Transducers (On HDT0L Languages) is Decidable. *Theor. Comput. Sci.*, 47(3):71–84, 1986.

14 Sebastian Maneth. A Survey on Decidable Equivalence Problems for Tree Transducers. *Int. J. Found. Comput. Sci.*, 26(8):1069–1100, 2015. `doi:10.1142/S0129054115400134`.

15 William C. Rounds. Mappings and Grammars on Trees. *Mathematical Systems Theory*, 4(3):257–287, 1970.

16 Helmut Seidl, Sebastian Maneth, and Gregor Kemper. Equivalence of Deterministic Top-Down Tree-to-String Transducers is Decidable. In *FOCS*, pages 943–962. IEEE Computer Society, 2015.

A Appendix

Bottom-Up Streaming Tree Transducers. We go more into detail into *Streaming Tree Transducers* (STT), that read and output nested words. This formalism is central to the proof of Proposition 15, and of Proposition 16.

In intuition, an STT works with a configuration composed of a state, a finite number of typed variables (or registers) that contains nested words with at most one occurrence of a context symbol (this corresponds to the Ordered Forest Algebra (OCF) in the sense of [7]), and a stack containing pairs of stack symbols and variable valuations. The nesting of the words dictates how this stack behaves: each opening letter $<a$ stores the current variable values in the stack to start with fresh ones, then each closing letter $a>$ uses the current variable values and the top of the stack to generate new values for the registers. The operations on nested words that can be performed in such cases correspond to polynomial operations on OCF: one can use concatenation, context application (which translates directly into OCF), or use a constant nested word, that can be simulated by the roots and concatenation: $(<a \circ a>)[\circ := r \cdot r']<b\ b>$ can be seen in forests as $\mathsf{root}_a(r + r') + \mathsf{root}_b()$.

The general definitions are available on [3]. We use specifically *bottom-up* STT, where reading an opening symbol $<a$ resets the state as well as the registers. On such STT, the behavior of a STT reading the nested word of a subtree does not depend on what occurs before or after. The original paper also imposes a single-use restriction, to ensure each operation can use each register only once. We can keep this restriction, but will not need it. We add a few restrictions to this model:

- We do not allow letters beyond nesting letters. In the language of [3] this means we ignore internal transitions.
- The input domain is a language of nested words of binary trees.

We will call this subclass Bottom-Up BT STT. The first result we need comes directly from the results of [3] that states that single-use STT (even limited to bottom-up) describe exactly MSO functions on nested words:

▶ **Proposition 21.** *Every function of $MSO_{BT\to UF}$ is described by a Bottom-Up BT STT.*

From STT to UCF-RA. To complete the proof, we show that if a Bottom-Up BT STT describes f, then we can find a **UCF**-RA that describes the function f' that f represents, by forgetting the order in the output.

▶ **Proposition 22.** *Every function of a Bottom-Up BT STT is described by an OCF-RA.*

Proof. We propose in a figure below the run of a bottom-up STT in a tree $c(t, t')$. The subtrees t and t' are of root a and b. The second line corresponds to its configuration (state q, register values $\overline{r}$), while the third line keeps track of the top symbol of the STT's stack. The state q_0 and register valuation $\overline{r_0}$ are respectively the initial state and register values. The symbol that was at the top of the stack when reaching $<c$ is denoted as λ.

$<c$	$<a$	h	$a>$	$<b$	h'	$b>$	$c>$
$(q_0, \overline{r_0})$	$(q_0, \overline{r_0})$	$\dots (q, \overline{r})$	$(q_1, \overline{r_1})$	$(q_0, \overline{r_0})$	$\dots (q', \overline{r'})$	$(q_2, \overline{r_2})$	
λ	$(p_a, \overline{r_0})$	$\dots (p_a, \overline{r_0})$	λ	$(p_b, \overline{r_1})$	$\dots (p_b, \overline{r_1})$	λ	
	t			t'			

Figure 4 The run of a bottom-up STT on a binary tree $c(t, t')$.

From $(q, \overline{r})$, when we read $a>$, we use a transition depending only on q, p_a to get q_1 and apply to $\overline{r}, \overline{r_0}$ a polynomial function depending only on q, p_a. We note that p_a depends only of a. Similarly, from $(q', \overline{r'})$, when we read $b>$, we use a transition depending only on q', p_b to get q_2 and apply to $\overline{r}, \overline{r_0}$ a polynomial function depending only on q', p_b. We note that p_b depends of b and q_1.

This means that, if we have prior knowledge of a, b – the roots of the left and right child of c – and q, q' – the states reached by our STT after reading the left and right child of c – we have enough information to find the state reached by our STT after reading $c(t, t')$. We can call this state $q_{a,b,q,q'}$. From a, b, q, q', we can also deduce the polynomial function that links $\overline{r}, \overline{r'}$ to $\overline{r_2}$, the values of the registers after reading $c(t, t')$. We call such a function $\phi_{a,b,q,q'}$, and it can be seen as a polynomial function on nested words or on OCF.

To find an OCF-RA that computes this function, we say that an input subtree leads to a state (q, a), where a is the label of its root, and q is the state reached by the STT right before reading the final $a>$. The registers have the same values as the STT's right before reading the final $a>$. The transitions of our OCF-RA will then be of form:

$$c((q, a), (q', b)) \to ((q_{a,b,q,q'}, c), \phi_{a,b,q,q'})$$ ◀