Patricia Bouyer
Lutz Schröder (Eds.)

Foundations of Software Science and Computation Structures

**25th International Conference, FOSSACS 2022
Held as Part of the European Joint Conferences
on Theory and Practice of Software, ETAPS 2022
Munich, Germany, April 2–7, 2022
Proceedings**

ETAPS
EUROPEAN JOINT CONFERENCES ON
THEORY & PRACTICE OF SOFTWARE

Springer

OPEN ACCESS

Lecture Notes in Computer Science　13242

Advanced Research in Computing and Software Science

Subline of Lecture Notes in Computer Science

More information about this series at

Patricia Bouyer · Lutz Schröder (Eds.)

Foundations of Software Science and Computation Structures

25th International Conference, FOSSACS 2022
Held as Part of the European Joint Conferences
on Theory and Practice of Software, ETAPS 2022
Munich, Germany, April 2–7, 2022
Proceedings

Springer

Editors
Patricia Bouyer
Université Paris-Saclay, CNRS,
ENS Paris-Saclay
Gif-sur-Yvette, France

Lutz Schröder
Friedrich-Alexander-Universität Erlangen
Erlangen, Germany

ETAPS Foreword

Welcome to the 25th ETAPS! ETAPS 2022 took place in Munich, the beautiful capital of Bavaria, in Germany.

ETAPS 2022 is the 25th instance of the European Joint Conferences on Theory and Practice of Software. ETAPS is an annual federated conference established in 1998, and consists of four conferences: ESOP, FASE, FoSSaCS, and TACAS. Each conference has its own Program Committee (PC) and its own Steering Committee (SC). The conferences cover various aspects of software systems, ranging from theoretical computer science to foundations of programming languages, analysis tools, and formal approaches to software engineering. Organizing these conferences in a coherent, highly synchronized conference program enables researchers to participate in an exciting event, having the possibility to meet many colleagues working in different directions in the field, and to easily attend talks of different conferences. On the weekend before the main conference, numerous satellite workshops took place that attract many researchers from all over the globe.

ETAPS 2022 received 362 submissions in total, 111 of which were accepted, yielding an overall acceptance rate of 30.7%. I thank all the authors for their interest in ETAPS, all the reviewers for their reviewing efforts, the PC members for their contributions, and in particular the PC (co-)chairs for their hard work in running this entire intensive process. Last but not least, my congratulations to all authors of the accepted papers!

ETAPS 2022 featured the unifying invited speakers Alexandra Silva (University College London, UK, and Cornell University, USA) and Tomáš Vojnar (Brno University of Technology, Czech Republic) and the conference-specific invited speakers Nathalie Bertrand (Inria Rennes, France) for FoSSaCS and Lenore Zuck (University of Illinois at Chicago, USA) for TACAS. Invited tutorials were provided by Stacey Jeffery (CWI and QuSoft, The Netherlands) on quantum computing and Nicholas Lane (University of Cambridge and Samsung AI Lab, UK) on federated learning.

As this event was the 25th edition of ETAPS, part of the program was a special celebration where we looked back on the achievements of ETAPS and its constituting conferences in the past, but we also looked into the future, and discussed the challenges ahead for research in software science. This edition also reinstated the ETAPS mentoring workshop for PhD students.

ETAPS 2022 took place in Munich, Germany, and was organized jointly by the Technical University of Munich (TUM) and the LMU Munich. The former was founded in 1868, and the latter in 1472 as the 6th oldest German university still running today. Together, they have 100,000 enrolled students, regularly rank among the top 100 universities worldwide (with TUM's computer-science department ranked #1 in the European Union), and their researchers and alumni include 60 Nobel laureates. The

local organization team consisted of Jan Křetínský (general chair), Dirk Beyer (general, financial, and workshop chair), Julia Eisentraut (organization chair), and Alexandros Evangelidis (local proceedings chair).

ETAPS 2022 was further supported by the following associations and societies: ETAPS e.V., EATCS (European Association for Theoretical Computer Science), EAPLS (European Association for Programming Languages and Systems), and EASST (European Association of Software Science and Technology).

The ETAPS Steering Committee consists of an Executive Board, and representatives of the individual ETAPS conferences, as well as representatives of EATCS, EAPLS, and EASST. The Executive Board consists of Holger Hermanns (Saarbrücken), Marieke Huisman (Twente, chair), Jan Kofroň (Prague), Barbara König (Duisburg), Thomas Noll (Aachen), Caterina Urban (Paris), Tarmo Uustalu (Reykjavik and Tallinn), and Lenore Zuck (Chicago).

Other members of the Steering Committee are Patricia Bouyer (Paris), Einar Broch Johnsen (Oslo), Dana Fisman (Be'er Sheva), Reiko Heckel (Leicester), Joost-Pieter Katoen (Aachen and Twente), Fabrice Kordon (Paris), Jan Křetínský (Munich), Orna Kupferman (Jerusalem), Leen Lambers (Cottbus), Tiziana Margaria (Limerick), Andrew M. Pitts (Cambridge), Elizabeth Polgreen (Edinburgh), Grigore Roşu (Illinois), Peter Ryan (Luxembourg), Sriram Sankaranarayanan (Boulder), Don Sannella (Edinburgh), Lutz Schröder (Erlangen), Ilya Sergey (Singapore), Natasha Sharygina (Lugano), Pawel Sobocinski (Tallinn), Peter Thiemann (Freiburg), Sebastián Uchitel (London and Buenos Aires), Jan Vitek (Prague), Andrzej Wasowski (Copenhagen), Thomas Wies (New York), Anton Wijs (Eindhoven), and Manuel Wimmer (Linz).

I'd like to take this opportunity to thank all authors, attendees, organizers of the satellite workshops, and Springer-Verlag GmbH for their support. I hope you all enjoyed ETAPS 2022.

Finally, a big thanks to Jan, Julia, Dirk, and their local organization team for all their enormous efforts to make ETAPS a fantastic event.

February 2022

Marieke Huisman
ETAPS SC Chair
ETAPS e.V. President

Preface

This volume contains the papers presented at the 25th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2022), which was held during April 4–6, 2022, in Munich, Germany. The conference is dedicated to foundational research with a clear significance for software science and brings together research on theories and methods to support the analysis, integration, synthesis, transformation, and verification of programs and software systems.

In addition to an invited talk by Nathalie Bertrand (Université de Rennes, Inria, CNRS, and IRISA, France) on "Parameterized verification to the rescue of distributed algorithms", the program consisted of 23 contributed papers, selected from among 77 submissions. Each submission was assessed by three or more Program Committee members. The conference management system EasyChair was used to handle the submissions, to conduct the electronic Program Committee discussions, and to assist with the assembly of the proceedings.

We wish to thank all the authors who submitted papers for consideration, the members of the Program Committee for their conscientious work, and all additional reviewers who assisted the Program Committee in the evaluation process. Finally, we would like to thank the ETAPS organization for providing an excellent environment for FoSSaCS, other conferences, and workshops.

January 2022

Patricia Bouyer
Lutz Schröder

Organization

Program Committee

C. Aiswarya	Chennai Mathematical Institute, India
S. Akshay	Indian Institute of Technology Bombay, India
Carlos Areces	Universidad Nacional de Córdoba, Argentina
Filippo Bonchi	Università di Pisa, Italy
Patricia Bouyer (Chair)	CNRS, LMF, France
Michaël Cadilhac	DePaul University, USA
Ankush Das	Amazon Web Services, USA
Maribel Fernandez	King's College London, UK
Santiago Figueira	Universidad de Buenos Aires, Argentina
Hongfei Fu	Shanghai Jiao Tong University, China
Patricia Johann	Appalachian State University, USA
Ohad Kammar	University of Edinburgh, UK
Shin-ya Katsumata	National Institute of Informatics, Japan
Aleks Kissinger	University of Oxford, UK
Naoki Kobayashi	University of Tokyo, Japan
Orna Kupferman	Hebrew University, Israel
Alexander Kurz	Chapman University, USA
Sławomir Lasota	University of Warsaw, Poland
Annabelle McIver	Macquarie University, Australia
Daniela Petrisan	Université de Paris, IRIF, France
Elaine Pimentel	Universidade Federal do Rio Grande do Norte, Brazil
Jean-Francois Raskin	Université Libre de Bruxelles, Belgium
Jurriaan Rot	Radboud University, The Netherlands
Lutz Schröder (Chair)	Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
Pawel Sobocinski	Tallinn University of Technology, Estonia
Ana Sokolova	Universität Salzburg, Austria
Jiri Srba	Aalborg University, Denmark
James Worrell	University of Oxford, UK

Additional Reviewers

Abriola, Sergio	Balasubramanian, A. R.
Allais, Guillaume	Bansal, Suguman
Alvarez-Picallo, Mario	Barloy, Corentin
Atkey, Robert	Blondin, Michael
Baillot, Patrick	Bodlaender, Hans L.
Balabonski, Thibaut	Boker, Udi

Bollig, Benedikt
Bonomo, Flavia
Bork, Alexander
Bønneland, Frederik M.
Carai, Luca
Carbone, Marco
Chen, Zhenbang
Clemente, Lorenzo
Comfort, Cole
Crubillé, Raphaëlle
Czerwiński, Wojciech
D'Argenio, Pedro R.
Dal Lago, Ugo
Della Penna, Giuseppe
Delzanno, Giorgio
Demri, Stéphane
Devillers, Raymond
DeYoung, Henry
Domínguez, Martín Ariel
Doyen, Laurent
Exibard, Léo
Fervari, Raul
Figueira, Diego
Finkel, Alain
Garner, Richard
Gastin, Paul
Gay, Simon
Genest, Blaise
Gocht, Stephan
Goncharov, Sergey
Grochau Azzi, Guilherme
Grädel, Erich
Hadzihasanovic, Amar
Hague, Matthew
Hedges, Jules
Ho, Hsi-Ming
Hodkinson, Ian
Junges, Sebastian
Kahn, David
Karimov, Toghrul
Kauffman, Sean
Kiefer, Stefan
Klin, Bartek
Koutny, Maciej
Kura, Satoshi
Kuznetsov, Stepan

Lange, Martin
Lewis, Marco
Lorber, Florian
López Franco, Ignacio
Maarand, Hendrik
Maderbacher, Benedikt
Mamouras, Konstantinos
Martens, Wim
Martinez, Maria Vanina
Mathieson, Luke
Matsushita, Yusuke
Meggendorfer, Tobias
Mikulski, Lukasz
Mikučionis, Marius
Moerman, Joshua
Muniz, Marco
Nakazawa, Koji
Nester, Chad
Ockerlund, Kyle
Oualhadj, Youssouf
Padhi, Saswat
Paperman, Charles
Perez, Guillermo
Piedeleu, Robin
Piróg, Maciej
Poças, Diogo
Praveen, M.
Puglisi, Simon
Reynier, Pierre-Alain
Román, Mario
Sacerdoti Coen, Claudio
Saivasan, Prakash
Sangnier, Arnaud
Sankur, Ocan
Sarkar, Saptarshi
Schmid, Todd
Schou, Morten Konggaard
Sharma, Vaibhav
Steinberg, Florian
Sterling, Jonathan
Thejaswini, K. S.
Trotta, Davide
Tull, Sean
Tzevelekos, Nikos
Ulidowski, Irek
van Dijk, Tom

van Glabbeek, Rob
van Heerdt, Gerco
Veltri, Niccolò
Voorneveld, Niels
Vortmeier, Nils
Wagemaker, Jana
Wagner, Dominik
Wang, Di

Wang, Weiyou
Wojtczak, Dominik
Yamakami, Tomoyuki
Yang, Qizhe
Ying, Mingsheng
Ziliani, Beta
Zimmermann, Martin
Žikelić, Djordje

Parameterized Verification to the Rescue of Distributed Algorithms (Abstract of Invited Talk)

Nathalie Bertrand [ORCID]

Univ Rennes, Inria, CNRS, IRISA, France
nathalie.bertrand@inria.fr

Abstract. Distributed computing is everywhere in our daily lives and in advanced technological applications. Bugs in distributed algorithms can have huge consequences, so that already in 2006, Lamport advised: "Model-checking algorithms prior to submitting them for publication should become the norm" [4]. Formal verification techniques indeed avoid tedious and error-prone manual correctness proofs.

Developing formal verification techniques for distributed algorithms is a real challenge, since correctness should typically hold independently of the number of participants. The latter often can be considered, or are by design, anonymous, forming a crowd of identical copies. Since the seminal work of German and Sistla establishing the decidability of parameterized verification for crowds of finite-state machines interacting via rendez-vous [3], the model checking community has been focusing on specific classes of distributed algorithms, and has proposed appropriate crowds models with a decidable parameterized verification problem [1, 2].

In this talk, we will report on recent contributions to the parameterized verification of distributed algorithms.

Keywords: Model checking · Parameterized verification · Distributed algorithms

References

1. Bloem, R., et al.: Decidability of Parameterized Verification. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers (2015). https://doi.org/10.2200/S00658ED1V01Y201508DCT013
2. Esparza, J.: Keeping a crowd safe: on the complexity of parameterized verification (invited talk). In: Proceedings of the 31st International Symposium on Theoretical Aspects of Computer Science (STACS'14). LIPIcs, vol. 25, pp. 1–10. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2014). https://doi.org/10.4230/LIPIcs.STACS.2014.1
3. German, S.M., Sistla, A.P.: Reasoning about systems with many processes. J. ACM **39**(3), 675–735 (1992). https://doi.org/10.1145/146637.146681
4. Lamport, L.: Checking a multithreaded algorithm with +CAL. In: Dolev, S. (ed.) Distributed Computing. DISC 2006. Lecture Notes in Computer Science, vol. 4167, pp. 151–163. Springer, Berlin (2006). https://doi.org/10.1007/11864219_11

Contents

Representing Regular Languages of Infinite Words Using Mod 2 Multiplicity Automata

Dana Angluin[1], Timos Antonopoulos[1](✉), Dana Fisman[2], and Nevin George[1]

[1] Yale University, New Haven, CT, USA
`timos.antonopoulos@yale.edu`
[2] Ben-Gurion University, Beer-Sheva, Israel

Abstract. We explore the suitability of mod 2 multiplicity automata (M2MAs) as a representation for regular languages of infinite words. M2MAs are a deterministic representation that is known to be learnable in polynomial time with membership and equivalence queries, in contrast to many other representations. Another advantage of M2MAs compared to non-deterministic automata is that their equivalence can be decided in polynomial time and complementation incurs only an additive constant size increase. Because learning time is parameterized by the size of the representation, particular attention is focused on the relative succinctness of alternate representations, in particular, LTL formulas and Büchi automata of the types: deterministic, non-deterministic and strongly unambiguous. We supplement the theoretical results of worst case upper and lower bounds with experimental results computed for randomly generated automata and specific families of LTL formulas.

Keywords: Multiplicity Automata · Regular Omega Languages · Büchi Automata · Linear Temporal Logic · Conciseness

1 Introduction

Regular languages of infinite words (or ω-words) play an important role in verification of reactive systems. The question of whether a system S satisfies a specification given by a temporal logic formula φ can be reduced to the question of whether $L(S) \cap L(\neg\varphi)$ is empty, where $L(S)$ is the set of ω-words representing the computation paths of the system S and $L(\neg\varphi)$ is the set of ω-words representing computations that violate φ. Automata are a useful machinery for performing operations on languages such as complementation and intersection, and for deciding properties such as emptiness and equivalence. Many verification tools are implemented using reductions to automata [20].

Regular ω-languages can be represented using various types of automata (e.g. Büchi, Rabin, Parity, etc.). Different automata types differ in their succinctness and in the complexity of performing operations of interest. Non-deterministic Büchi automata (NBAs) are one of the most popular acceptor types for regular ω-languages, mainly due to their simplicity, succinctness, and good complexity

for the emptiness problem. An issue with Büchi automata is that their deterministic version (DBAs) is strictly less expressive: while NBAs accept all regular ω-languages, DBAs recognize only a strict subset thereof. Another issue is that complementation of NBAs is hard; it has a $2^{\Omega(n \log n)}$ lower bound (where n is the number of states) [16]. This motivated the introduction of *complete unambiguous Büchi automata* (CUBA) by Carton and Michel who showed that every regular ω-language can be represented by a CUBA, i.e. there is a way to limit the non-determinism without losing expressiveness [8]. Bousquet and Löding proposed *strongly unambiguous Büchi automata* (SUBA), a slight relaxation of CUBA for which they have shown that equivalence can be decided in polynomial time [6].

The SUBA model was also shown useful in terms of learnability of regular ω-languages — Angluin, Antonopoulos and Fisman have shown that SUBAs are polynomially predictable using membership queries (while NBAs, under plausible cryptographic assumptions, are not) [1]. Their proof makes use of a model of automata called *Mod 2 Multiplicity Automata* (M2MA). Informally, *multiplicity automata* are an algebraic variant of automata that compute functions from finite words to a field $\mathcal{K}$ [4,5], and *M2MAs* are multiplicity automata that work over the field $GF(2) = \{0, 1\}$ where sum and product are computed modulo 2.

In this paper we look at questions concerning the adequacy of M2MAs for representing regular ω-languages. We note that M2MAs operate on finite words, and their use for representing regular ω-languages follows a reduction, by Calbrix, Nivat and Podelski from a regular ω-language L to a regular language $(L)_\$$ of finite words [7]. We thus start by reviewing the succinctness of M2MAs with respect to automata on finite words, particularly of types non-deterministic (NFAs), deterministic (DFAs), and unambiguous (UFAs). We show that M2MAs are more succinct than DFAs and UFAs, whereas with respect to NFAs there are in the worst case exponential gaps in going from M2MAs to NFAs and vice versa.

We also study the complexity of performing basic operations on M2MAs; complementation can be done with an additive constant increase in size, and union and intersection with the product of sizes. There is a known cubic algorithm to minimize a weighted automaton [10,19], which applies to an M2MA and also implies cubic procedures for determining emptiness and equivalence.

We then investigate the succinctness of M2MAs in representing regular ω-languages, by comparing translations from *linear temporal logic* (LTL) formulas and Büchi automata (deterministic, non-deterministic and strongly unambiguous) into M2MAs, DFAs, UFAs, SUBAs and NBAs (where the former three use the $(L)_\$$ representation). The results are summarized in Fig. 3.

To complement the theoretical bounds, we implemented procedures to transform SUBAs to UFAs and UFAs to M2MAs, and to minimize and learn M2MAs, and report estimates of the average size increases in transforming random SUBAs, DBAs, and NBAs to M2MAs. We also determine the minimum dimensions of M2MAs and minimum sizes of DFAs for a few members of three specific families of LTL formulas and compare them with the respective ω-automaton sizes.

2   Preliminaries

For nonnegative integers k and ℓ, $[k..\ell]$ is the set of nonnegative integers n such that $k \leq n \leq \ell$. Given a finite alphabet Σ, Σ^* is the set of finite words over Σ. The length of a word x is $|x|$ and the empty word is ε. $\Sigma^n = \{x \in \Sigma^* \mid |x| = n\}$. The reverse of a word x is x^r. A language L is any subset of Σ^*. The reverse of L, denoted L^r, is $\{x^r \mid x \in L\}$. The *Hankel matrix* of a language L is the infinite matrix whose rows and columns are indexed by elements of Σ^*, where the entry for row x and column y is 1 if $xy \in L$ and 0 if $xy \notin L$.

The set of infinite words (or ω-words) over Σ is the set of all maps from the positive integers to Σ and is denoted Σ^ω. An ω-language is any subset of Σ^ω. For a finite or infinite word w, $w[i]$ denotes the symbol at position i, with indices starting at 1. Concatenation of a finite word x with a finite or infinite word y is denoted xy. The word x is a prefix of xy and the word y is a suffix of xy. The suffix of w starting at position i is denoted $w[i:]$. If $x \in \Sigma^*$ and k is a nonnegative integer, x^k denotes the concatenation of k copies of x, and x^ω denotes the concatenation of x with itself infinitely many times. An ω-word is ultimately periodic if it can be written in the form $u(v)^\omega$ for $u, v \in \Sigma^*$ with $|v| > 0$. If A_1 and A_2 are sets and $S \subseteq A_1 \times A_2$, then we define the projection $\pi_1(S) = \{a_1 \mid (\exists a_2)(a_1, a_2) \in S\}$ and analogously for the projection π_2.

2.1   NFAs, UFAs, DFAs, NBAs, UBAs, SUBAs, and DBAs

A (nondeterministic) finite-state automaton A is a tuple $(\Sigma, Q, I, \Delta, F)$ consisting of a finite alphabet Σ, a finite set Q of states, a set $I \subseteq Q$ of initial states, a set $F \subseteq Q$ of final states, and a transition relation $\Delta \subseteq Q \times \Sigma \times Q$. The transition relation Δ is deterministic if for every state $q \in Q$ and every symbol $\sigma \in \Sigma$, there is at most one state $q' \in Q$ such that $(q, \sigma, q') \in \Delta$. The size of a finite-state automaton is $|Q|$.

For a word w, a run of A on w is a sequence of states $q_0, q_1, \ldots$ such that for each i that indexes a symbol in w, $(q_{i-1}, w[i], q_i) \in \Delta$. Thus, for $w \in \Sigma^*$ a run on w is a sequence of length $|w| + 1$, and for $w \in \Sigma^\omega$, a run on w is an infinite sequence of states. A run on w is *initial* if $q_0 \in I$. A finite run is *final* if $q_{|w|} \in F$, and an infinite run is *final* if there are infinitely many values of i for which $q_i \in F$. Acceptors of languages and ω-languages may be defined using finite-state automata, as follows. In each case, the language of words accepted by an acceptor A is denoted $L(A)$.

A *nondeterministic finite acceptor* (NFA) is a finite-state automaton A that accepts a word $w \in \Sigma^*$ if there exists a run of A on w that is both initial and final. An NFA A is an *unambiguous finite acceptor* (UFA) if for every word $w \in L(A)$ there is exactly one run of A on w that is initial and final. An NFA A is a *deterministic finite acceptor* (DFA) if there is exactly one initial state ($|I| = 1$) and the transition relation Δ is deterministic. The languages over Σ that are accepted by NFAs, UFAs, or DFAs is precisely the regular languages over Σ.

A *nondeterministic Büchi acceptor* (NBA) is a finite-state automaton A that accepts a word $w \in \Sigma^\omega$ if there exists a run of A on w that is both initial and

final. An NBA is an *unambiguous Büchi acceptor* (UBA) if for every $w \in L(A)$, there exists exactly one run of A on w that is initial and final. Bousquet and Löding [6] introduced the concept of a *strongly unambiguous Büchi acceptor* (SUBA), which is an NBA such that for every $w \in \Sigma^\omega$, there is at most one final run of the acceptor on w — note that the condition of being initial is dropped. Thus, every SUBA is a UBA. The ω-languages over Σ that are accepted by NBAs, UBAs, or SUBAs are precisely the regular ω-languages. An NBA is a *deterministic Büchi acceptor* (DBA) if there is exactly one initial state ($|I| = 1$) and the transition relation Δ is deterministic. Every DBA is a UBA, but is not necessarily a SUBA. The ω-languages that are accepted by DBAs are a proper subclass of the class of all regular ω-languages.

For Büchi acceptors, we also consider a generalized version, GNBA, in which the acceptance condition is specified not by a single set of final states, but by a collection $\mathcal{F}$ of sets of final states. For a GNBA, a run $q_0, q_1, \ldots$ is *final* iff for each $F \in \mathcal{F}$, there exist infinitely many indices i such that $q_i \in F$. Applying this generalization to a SUBA yields a GSUBA. There is a standard translation of a GNBA of size n with k sets of final states into an NBA of size kn, in which there are k copies of the GNBA automaton. However, applying this construction to a GSUBA does not in general yield a SUBA.

2.2   LTL formulas

The syntax of *linear temporal logic* (LTL) [18] over a set AP of atomic propositions is given by the following grammar $\varphi ::= p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid (\varphi_1\,\mathcal{U}\,\varphi_2)$ where $p \in AP$ is an atomic proposition.

The semantics of LTL relates ω-words over 2^{AP} to formulas as shown on the right (recall that indexing of words starts at 1). Additional Boolean and temporal connectives are defined in the usual way. In particular $\top$ (*true*) is defined as $p \vee \neg p$, $\Diamond\varphi$ (*eventually* φ) is defined as $(\top\,\mathcal{U}\,\varphi)$ and $\Box\varphi$ (*always* φ) is defined as $\neg\Diamond(\neg\varphi)$.

$$
\begin{aligned}
w &\models p & &\text{iff} & &p \in w[1] \\
w &\models \neg\varphi & &\text{iff} & &w \not\models \varphi \\
w &\models \varphi_1 \wedge \varphi_2 & &\text{iff} & &w \models \varphi_1 \text{ and } w \models \varphi_2 \\
w &\models \bigcirc\varphi & &\text{iff} & &w[2:] \models \varphi \\
w &\models (\varphi_1\,\mathcal{U}\,\varphi_2) & &\text{iff} & &\exists j.\ w[j:] \models \varphi_2 \text{ and} \\
& & & & &\forall i < j.\ w[i:] \models \varphi_1
\end{aligned}
$$

The ω-language of an LTL formula φ, denoted $L(\varphi)$, is the set of ω-words for which it is true. The *size* of an LTL formula φ is the number of distinct subformulas it contains. Every LTL formula represents a regular ω-language (see Section 5). However, not every regular ω-language can be represented by an LTL formula; in particular, the regular ω-languages that can be represented by LTL formulas are noncounting [9].

2.3   M2MAs

A *multiplicity automaton* represents a function mapping Σ^* to elements of a field $\mathcal{K}$. We focus on the case where $\mathcal{K} = \{0, 1\}$ and product and sum are computed modulo 2. A *mod 2 multiplicity acceptor* (M2MA) of dimension d is a tuple $A = (\Sigma, v_I, \{\mu_\sigma\}_{\sigma \in \Sigma}, v_F)$, where Σ is the input alphabet, $v_I \in \mathcal{K}^d$ is the initial

vector, $v_F \in \mathcal{K}^d$ is the final vector, and for each $\sigma \in \Sigma$, μ_σ is a $d \times d$ transition matrix over $\mathcal{K}$, that is, an element of $\mathcal{K}^{d \times d}$.

The vectors v_I and v_F are interpreted as $d \times 1$ column vectors. The transpose operation is denoted by $\top$, and the inner product of two column vectors $v, w \in \mathcal{K}^d$ is denoted $v^\top w$.

To define $L(A)$ we inductively define the matrix μ_x for all $x \in \Sigma^*$. If $x = \varepsilon$, then μ_x is the $d \times d$ identity matrix. If $x = \sigma y$ for some $\sigma \in \Sigma$ and $y \in \Sigma^*$ then $\mu_x = \mu_\sigma \mu_y$. The function $f_A : \Sigma^* \to \mathcal{K}$ computed by A is defined by $f_A(x) = v_I^\top \mu_x v_F$.   A word x is *accepted* by A if $f_A(x) = 1$.

We refer to column vectors $v \in \mathcal{K}^d$ as *states* or *co-states* of A. A state v is *reachable* iff there exists a word $x \in \Sigma^*$ such that $v = (v_I^\top \mu_x)^\top$. A co-state w is *co-reachable* iff there exists a word $x \in \Sigma^*$ such that $w = \mu_x v_F$. For any state v, $L_v(A)$ denotes the language of words accepted by A with its initial vector replaced by v.

We assume standard results from finite dimensional vector spaces. If U is a vector space of dimension k over the field $\{0, 1\}$ then $|U| = 2^k$. If U is a vector subspace of the vector space V, then the *orthogonal complement* of U is the set $U^\perp = \{v \mid v^\top u = 0 \ \forall u \in U\}$, $U^\perp$ is a vector subspace of V which is disjoint from U except for the zero vector, and the dimensions of U and $U^\perp$ sum to the dimension of V.

The following simple lemmas relate M2MAs to UFAs and DFAs, and show that M2MAs accept exactly the regular languages.

Lemma 1. *[Beimel et al. [4]] Let $L \subseteq \Sigma^*$. If L is accepted by a UFA of size n, it is also accepted by an M2MA of dimension n.*

Lemma 2. *Let $L \subseteq \Sigma^*$. If L is accepted by an M2MA of dimension d with R reachable states, then L is also accepted by a DFA of R states. Clearly, $R \leq 2^d$.*

Beimel et al. [4] have shown that there is a polynomial time algorithm to learn an unknown M2MA using equivalence and membership queries.

2.4   Size lower bounds for DFAs, M2MAs and NFAs

Given a language $L \subseteq \Sigma^*$, we define an *observation table* for L as an $\ell \times m$ matrix T of 0's and 1's where each row i is associated with a finite word x_i and each column j is associated with a finite word y_j, and the entry $T_{i,j}$ is 1 if and only if $x_i y_j \in L$. This terminology is derived from its use in algorithms to learn DFAs. An observation table for L is thus a finite submatrix of its Hankel matrix.

Certain properties of observation tables for a language L yield lower bounds on acceptors recognizing L. Recall that the rank of a matrix is the number of linearly independent rows (or columns) it contains.

Lemma 3. *Let T be an observation table for the regular language L with rows associated with finite words x_i for $i = [1..\ell]$ and columns associated with finite words y_j for $j \in [1..m]$. Assume T has n distinct rows and rank d over the field $\{0, 1\}$. Then any DFA to accept L must have at least n states, and any M2MA to accept L must have dimension at least d.*

Proof. Let D be a DFA accepting L. If the rows for x_i and x_k are distinct, then there is a column j on which they differ, that is, $x_i y_j \in L$ iff $x_k y_j \notin L$. Thus, the states of D reached from the initial state on the words x_i and x_k must be different and D has at least n states.

Let M be an M2MA accepting L. Following the argument of Beimel et al. [4], the observation table is a submatrix of the Hankel matrix of the language L, and its rank (modulo 2) is a lower bound for the rank (modulo 2) of the Hankel matrix, which is a lower bound for the size of any M2MA accepting L.   □

For lower bounds for NFAs, we use the concept of covering the observation table by 1-monochromatic rectangles. If R and C are subsets of the indices of the rows and columns (respectively) of a matrix M, then the (R, C)-*rectangle* of M is the matrix obtained from M by deleting those rows whose indices are not in R and those columns whose indices are not in C. The (R, C)-rectangle of a matrix M is v-*monochromatic* iff all of its entries are equal to the value v.

Let M be a matrix of 0 and 1 values. A 1-*rectangle cover* of M is a set $\{(R_s, C_s) \mid s \in [1..t]\}$, of 1-monochromatic rectangles (R_s, C_s) of M such that for every i and j, if $M_{i,j} = 1$ then there exists some $s \in [1..t]$ such that $i \in R_s$ and $j \in C_s$. A *minimum* 1-*rectangle cover* of M is a 1-rectangle cover of M of minimum possible cardinality t.

Lemma 4. *Let T be an $\ell \times m$ observation table for the regular language L. Any NFA M recognizing L must have at least as many states as the cardinality of the minimum 1-rectangle cover of T.*

This is implied by Theorem 5.2.4.10 and Exercise 5.2.5.14 of Hromkovič [12]. For completeness we provide a simple direct proof.

Proof. Let the strings indexing the rows of T be x_i for $i \in [1..\ell]$ and the strings indexing the columns of T be y_j for $j \in [1..m]$. For each state q of M, let R_q be the set of all $i \in [1..\ell]$ such that x_i reaches q from an initial state of M, and let C_q be the set of all $j \in [1..m]$ such that y_j reaches a final state of M from q.

Clearly (R_q, C_q) must be a 1-monochromatic rectangle of T, because if $i \in R_q$ and $j \in C_q$ then $x_i y_j$ is accepted by M and the entry $T_{i,j}$ must be 1. Also, if $T_{i,j} = 1$, then $x_i y_j$ must be accepted by M, so there must exist a state q of M such that x_i reaches q from an initial state of M and y_j reaches a final state of M from q, that is, $i \in R_q$ and $j \in C_q$. Thus, the rectangles (R_q, C_q) for all states q of M form a 1-rectangle covering of T, and the number of states of M is greater than or equal to the cardinality of the minimum 1-rectangle covering of T.   □

Corollary 1. *If L is a regular language with an $n \times n$ observation table T that has exactly one 1 in every row and column, then any DFA, M2MA, or NFA to recognize L must have at least n states.*

As an example of the use of these results, let L be the regular language over $\{a, b, c\}$ consisting of those strings that do not contain any occurrences of the substrings ba or cb, with the observation table for L in Fig. 1. There are 4 different rows, so any DFA to accept L must have at least 4 states. The mod 2 rank of the table is 3 (the first three rows are a row basis) so any M2MA accepting L must have dimension

	ε	a	b
ε	1	1	1
b	1	0	1
c	1	1	0
ba	0	0	0

Fig. 1: Observation table with rank 3.

at least 3. The observation table with rows c and b, and columns a and b is the 2×2 identity matrix, so any NFA to accept L must have at least 2 states. In fact, there is a DFA of 4 states, an M2MA of dimension 3, and an NFA of 2 states accepting L, so for this example, the lower bounds are tight.

3   M2MAs as representations of regular languages

We consider the computational cost and size implications of some common operations and decision questions using M2MAs to represent regular languages.

3.1   M2MAs: procedures for operations and properties

Reverse Given an M2MA A accepting a regular language L, an M2MA A^r accepting the reverse language L^r may be obtained from A by exchanging the initial and final vectors, and transposing each of the transition matrices. Thus, the minimum dimension of an M2MA accepting L is equal to the minimum dimension of an M2MA accepting L^r. Reversing is similarly easy for UFAs and NFAs, but may incur an exponential increase in size for a DFA.

Sum If for $i = 1, 2$, M_i is a multiplicity automaton of dimension d_i computing the function $f_i : \Sigma^* \to \mathcal{K}$, then the sum $f_1 + f_2$ is computed by a multiplicity automaton M of dimension $d_1 + d_2$ constructed as the direct product of M_1 and M_2 as follows. State vectors of M are the concatenation of state vectors of M_1 and M_2, including the initial and final vectors. For each $\sigma \in \Sigma$, the transition matrix μ_σ is a $(d_1 + d_2) \times (d_1 + d_2)$ matrix obtained by putting $(\mu_1)_\sigma$ in the upper left, $(\mu_2)_\sigma$ in the lower right, and setting the remaining entries to 0. This ensures that the state updates of M_1 and M_2 are done in parallel for each symbol, and the output is the sum of the outputs for M_1 and M_2.

Boolean operations For M2MAs, complementation follows directly from the sum construction. If A is an M2MA of dimension d and C is the M2MA of dimension 1 that outputs 1 on every string, then the sum construction with M and C yields an M2MA of dimension $d+1$ that accepts the regular language $\Sigma^* \setminus L(A)$. For DFAs, complementation is size-preserving, while for NFAs, complementation may incur an exponential increase in size.

Given M2MAs A_i of dimension d_i for $i = 1, 2$, the intersection language $L(A_1) \cap L(A_2)$ is accepted by an M2MA of dimension $d_1 \cdot d_2$ obtained from A_1

and A_2 using the Kronecker product of matrices.[3] Union can then be obtained from complementation and intersection.

Minimization, Equivalence, and Emptiness Sakarovitch [10,19] describes a cubic-time algorithm to minimize a weighted automaton with weights from a skew field, which has the following corollary.

Corollary 2 (of Theorem 5.20 in [10]). *Given an M2MA A of dimension d, an M2MA A' of the minimum possible dimension accepting $L(A)$ may be found in time $O(|\Sigma|d^3)$.*

An M2MA recognizes the empty language iff it has dimension 0 when minimized, and the equivalence of two M2MAs may be tested by determining if their sum is the empty language.

3.2   Conciseness comparisons for regular languages

We summarize known results comparing the conciseness of M2MAs with that of DFAs, UFAs and NFAs as representations of regular languages in Fig. 2. The entry for row A and column B is "$-$" if the representation A is an instance of the representation B, otherwise, starting with a machine of size n in the representation A, how large must an equivalent machine in the representation B be in the worst case? The entry $2^{\Theta(n)}$ means that there is a lower bound of 2^{cn} and an upper bound of 2^{dn} for positive constants c and d.

We briefly explain the entries in the table. A DFA is also a UFA and an NFA, and a UFA is also an NFA. A DFA or UFA of size n can be converted to an equivalent M2MA of dimension n (Lemma 1). The subset construction to determinize an NFA of size n yields a DFA (and therefore also a UFA or M2MA) of size at most 2^n. An M2MA of dimension n can be converted to a DFA (or UFA or NFA) of size

	DFA	UFA	NFA	M2MA
DFA	$-$	$-$	$-$	n
UFA	$2^{\Theta(n)}$	$-$	$-$	n
NFA	$2^{\Theta(n)}$	$2^{\Theta(n)}$	$-$	$2^{\Theta(n)}$
M2MA	$2^{\Theta(n)}$	$2^{\Theta(n)}$	$2^{\Theta(n)}$	$-$

Fig. 2: Worst-case size bounds for representations of regular languages.

at most 2^n (Lemma 2). The language $B_n = \Sigma^* \cdot 1 \cdot \Sigma^n$, for $\Sigma = \{0,1\}$, consisting of binary strings with a 1 located $n+1$ symbols before the end is accepted by a UFA of size $n+2$ (and therefore also an NFA of size $n+2$ and an M2MA of dimension $n+2$), but requires at least 2^{n+1} states for any DFA that accepts it. For the problem of converting an NFA to an M2MA, Kaznatcheev and Panangaden [13] consider the language $L_n = \Sigma^* \left((0\Sigma^{n-1}1) + (1\Sigma^{n-1}0) \right) \Sigma^*$ for $\Sigma = \{0,1\}$, and show that L_n is recognized by an NFA of $2n+2$ states, but that any M2MA to recognize L_n must have dimension at least 2^n. By Lemma 1, this lower bound applies also to UFAs.

For the problem of converting an M2MA to an NFA, Kaznatcheev and Panangaden [13] give a family of languages $\{L_n\}$ such that L_n is recognized by an

---

[3] If A is an $m \times n$ matrix and B is a $p \times q$ matrix, then the Kronecker product $A \otimes B$ is the $pm \times qn$ block matrix, with blocks of size B, where the block-matrix at position (i,j) is $a_{ij}B$ [17, Def 1.2.1].

M2MA of dimension $n + 2$, and prove that any NFA to recognize L_n must have at least $2^{n/2} - 2$ states. Here we provide a simpler proof of a stronger lower bound. Let L_n be the language recognized by the M2MA given in Fig. 1 of the paper of Kaznatcheev and Panangaden. This M2MA accepts a word iff the number of indices i such that both $w[i]$ and $w[i + n]$ is 1, is odd.

Lemma 5. *Any NFA to recognize L_n must have at least 2^{n-1} states.*

Proof. The language L_n has an observation table T_n of dimension $2^n \times 2^n$, in which the rows and columns are indexed by strings $x, y \in \{0, 1\}^n$. We view strings in $\{0, 1\}^n$ as vectors of length n over the field $\{0, 1\}$, so that the entry corresponding to the pair (x, y) is the inner product of the vectors x and y, that is $x^\top y$. Note that the inner product $x^\top y$ is 1 iff the number of indices i such that both $xy[i]$ and $xy[i+n]$ is 1, is odd. The lower bound of $2^n - 1$ then follows from Lemma 4 and the following Lemma. □

Lemma 6. *The minimum 1-rectangle covering of the observation table T_n just defined has cardinality $2^n - 1$.*

Proof. For the upper bound it suffices to consider a 1-rectangle covering of T_n consisting of pairs (R, C) where R is the singleton index of a nonzero row and C consists of the indices of the occurrences of 1 in that row.

If $x \in \{0, 1\}^n$ is the zero vector, then $x^\top y$ is 0 for all vectors y; otherwise, $x^\top y = 1$ for exactly half the vectors y, that is, for 2^{n-1} columns of T_n. Hence, T_n contains exactly $2^{n-1}(2^n - 1)$ entries of value 1. We now show that any 1-monochromatic rectangle (R, C) of T_n has at most 2^{n-1} entries of 1, which shows that a minimum 1-rectangle covering of T_n must have cardinality at least $2^n - 1$.

Let (R, C) be any 1-monochromatic rectangle of T_n. Let U be the vector subspace spanned by the vectors x corresponding to indices in R, and let B be a basis for U whose indices are drawn from R. Let $k = |B|$, so that $|U| = 2^k$. Every element of U is a sum of elements of B, but a sum of an even number of elements of B will be 0 in all the columns with indices in C, so R can contain the indices of at most half the elements of U, that is, $|R| \leq 2^{k-1}$.

Let $S = \{v \mid u^\top v = 1 \ \forall u \in B\}$, the set of vectors whose inner product with all elements of B is 1; clearly, $|C| \leq |S|$. We use inclusion/exclusion to find the cardinality of S, as follows.

$$|S| = 2^n - |\bigcup_{u \in B} \{v \mid u^\top v = 0\}|$$

$$= 2^n - |\bigcup_{C \subseteq B} C^\perp|$$

$$= 2^n - k2^{n-1} + \binom{k}{2}2^{n-2} - \ldots (-1)^k 2^{n-k}$$

$$= 2^n \cdot (1 - \frac{1}{2})^k$$

$$= 2^{n-k}$$

Thus, $|C| \leq 2^{n-k}$. Then $|R \times C| \leq 2^{k-1} \cdot 2^{n-k} = 2^{n-1}$, concluding the proof. □

4   Representing regular omega-languages using regular languages

In the preliminaries we discussed NBAs, SUBAs and DBAs, and LTL formulas as representations of regular ω-languages. Here we explain that M2MAs and other automata over finite words can also be used to represent regular ω-languages.

A regular ω-language is uniquely determined by the set of ultimately periodic ω-words it contains. Let L be a regular ω-language and let $\$$ be a symbol not in the alphabet of L. To represent the set of ultimately periodic words in L, Calbrix, Nivat and Podelski [7] introduced the related language of finite words $L_\$ = \{u\$v \mid u(v)^\omega \in L\}$ and proved that it is regular.

Thus a regular ω-language L can be represented by an acceptor for the regular language $L_\$$, for example, a DFA, UFA, NFA or M2MA. The representation of $L_\$$ by an M2MA was used by Angluin, Antonopoulos, and Fisman [1] in showing that regular ω-languages are polynomially predictable with membership queries as a function of the size of the smallest SUBA accepting the language.

We note that if for $i = 1, 2$, A_i is an M2MA of dimension d_i accepting $(L_i)_\$$ for the regular ω-language L_i, then there is an M2MA of dimension $d_1 \cdot d_2$ accepting $(L_1 \cap L_2)_\$$, and an M2MA of dimension $d_1 + 3$ accepting $(\Sigma^\omega \setminus L_1)_\$$. The former follows by the intersection result for M2MAs, and the latter follows by the sum result applied to A_1 and the dimension 3 M2MA that accepts the set $\{u\$v \mid u \in \Sigma^*, v \in \Sigma^+\}$.

5   Conciseness comparisons for regular omega-languages

We present known and new results comparing the conciseness of M2MAs with that of several other representations of regular ω-languages, summarized in Fig. 3. The entry for row A and column B gives upper (above) and lower (below) bounds on the worst case increase in size for a representation of type A of size or dimension n to an equivalent representation of type B. The entry is "$-$" if a representation of type A is an instance of a representation of type B. The entries for the columns for DFA, UFA, M2MA, and NFA are for the language $L_\$$. An arrow indicates that the (lower or upper) bound is derived from a related (lower or upper) bound in the table. For example, the upper bound for the row DBA and columns UFA, M2MA and NFA are derived from the upper bound for the row DBA and column DFA. We now discuss the entries.

5.1   Size increases for LTL formulas

Upper bounds

There is a "classic" algorithm, described by Baier and Katoen [3, Chapter 5], to translate an LTL formula of size n into a GNBA of size 2^n with at most n sets of final states, which then yields an NBA of size at most $n2^n$. This shows that every LTL formula represents a regular ω-language, and gives an upper bound for translating an LTL formula to an NBA. Another algorithm to translate LTL formulas into NBAs is given by Gerth, Peled, Vardi and Wolper [11].

	DFA	UFA	M2MA	NFA	(G)SUBA	NBA
LTL	$2^{2^{O(n)}}$ via UFA $\rightarrow$	$2^{O(n)}$ Cor.3 $\rightarrow$	$\leftarrow$ $2^{\Omega(n)}$ Thm. 2	$\leftarrow$ $2^{\Omega(n)}$ Thm. 2	$(2^n, n)$ Prop. 1 $\rightarrow$	$n2^n$ [3] $2^{\Omega(n)}$ [3]
DBA	$n + n3^{n^2}$ [14] $2^{\Omega(n \log n)}$ [2]	$\leftarrow$ $\rightarrow$	$\leftarrow$ $2^{\Omega(n)}$ Thm. 3	$\leftarrow$ $2^{\Omega(n)}$ Thm. 3	$\downarrow$ $2^{\Omega(n)}$ [6]	$-$ $-$
NBA	$2^n + 2^n3^{n^2}$ [14] $\uparrow$	$\leftarrow$ $\uparrow$	$\leftarrow$ $\uparrow$	$n + n3^{n^2}$ [14] $\uparrow$	$(12n)^n$ [8] $\uparrow$	$-$ $-$
SUBA	$\uparrow$ $2^{\Omega(n)}$ [1]	$2n^2 + n$ [6] $\rightarrow$	$\leftarrow$ $2n^2 - n + 2$ Thm. 4	$\leftarrow$ $2n^2 - n + 2$ Thm. 4	$-$ $-$	$-$ $-$

Fig. 3: Worst-case size bounds for representations of regular ω-languages.

Concerning the classic translation algorithm, Bousquet and Löding [6] give a brief argument and state that "Hence the automaton that is constructed in this standard way is strongly unambiguous." Wilke [21] states that "Every temporal formula with n subformulas can be translated into an equivalent backwards deterministic generalized Büchi automaton with at most 2^n states and as many Büchi sets as there are subformulas with leading temporal operator F (eventually) or U (until)." To clarify these earlier statements, we reformulate them in our terminology. This gives an upper bound for transforming an LTL formula to a GSUBA.

Proposition 1. *Let ϕ be an LTL formula of size n with temporal operators next and until, with m until subformulas. Applying the classic translation algorithm to ϕ yields a GSUBA of size 2^n with m sets of final states.*

Proof. Baier and Katoen [3] show that the algorithm yields a GNBA M of the given size in which each state corresponds to an assignment of true or false to every subformula of ϕ. Moreover, if the ω-word w is accepted from a state q, then q assigns true to each subformula ψ of ϕ iff ψ is true for w. Hence there is at most one state of M from which the ω-word w is accepted, and thus M is also GSUBA. □

To get an upper bound for translation of LTL formulas to UFAs, M2MAs, and NFAs, we would like to use the property of being strongly unambiguous. However, if the resulting GSUBA has more than one set of final states, transforming it in the usual way into an NBA does not in general yield a SUBA. Instead, we generalize to GSUBAs the method of Bousquet and Löding [6] for transforming a SUBA accepting L into a UFA accepting $L_\$$.

Theorem 1. *There is an algorithm to transform a GSUBA of size n with m sets of final states accepting L into a UFA of size $2^m n^2 + n$ accepting $L_\$$. It runs in time polynomial in n and 2^m.*

Proof. Let L be accepted by the GSUBA $M = (\Sigma, Q, I, \Delta, \mathcal{F})$ with $n = |Q|$ and $m = |\mathcal{F}|$. We index the elements of $\mathcal{F}$ as F_i for $i \in [1..m]$. Bousquet and Löding [6, Lemma 1] show that $u(v)^\omega$ is accepted by a SUBA iff there exists a state q reachable from an initial state on reading u, such that on the word v there is a computation path that loops from q back to q while passing through an accepting state. For the GSUBA M, the condition is that the computation path that loops from q back to q must pass through at least one state from each F_i for $i \in [1..m]$.

We define an NFA $M' = (\Sigma', Q', I', \Delta', F')$ as follows. The alphabet is $\Sigma' = \Sigma \cup \{\$\}$. The state set is $Q' = Q \cup Q_1$, where $Q_1 = \{(q_1, q_2, S) \mid q_1, q_2 \in Q, S \subseteq [1..m]\}$. The initial states are $I' = I$. The transition relation is $\Delta' = \Delta \cup \Delta_1 \cup \Delta_2$, where Δ_1 is the set of all triples $((q_1, q_2, S), \sigma, (q_1', q_2', S'))$ such that $q_1' = q_1$, $(q_2, \sigma, q_2') \in \Delta$, and $S' = S \cup T$, where $T = \{i \in [1..m] \mid q_2' \in F_i\}$. And Δ_2 contains all triples $(q, \$, (q, q, \emptyset))$ such that $q \in Q$. The set of final states F' is the set of triples (q_1, q_2, S) such that $S = [1..m]$ and $q_1 = q_2$.

Then M' has $2^m n^2 + n$ states, and can be constructed in time polynomial in n and 2^m given the GSUBA M. On an input $u\$v$, the NFA M' behaves like M on the word u, reaching some state q. Then on the symbol $\$$, M' transitions to the state $(q, q, \emptyset)$, recording the state q reached after reading u. As M' continues reading v, the first component remembers q while the second component transitions as in M. The third component, S, records the set of indices of those final sets F_i that have been visited in the processing of v. The input $u\$v$ is accepted by M' iff there is a state q of M reachable from a state of I on input u such that there exists a computation path in M on input v from q to q that visits at least one state in F_i for every $i \in [1..m]$. Thus M' accepts $L_\$$. (Note that the set S generalizes the single bit used in Bousquet and Löding's construction.)

To see that M' is a UFA, we note that if there are two different accepting computations in M' for $u\$v$, then these may be used to construct two different accepting computations in M for $u(v)^\omega$, contradicting the fact that M is a GSUBA. $\square$

The entry in Fig. 3 for row LTL and column UFA is then justified by the following.

Corollary 3. *Let ϕ be an LTL formula of size n with temporal operators next and until, with m until subformulas. Then there is a UFA of size $2^{2n+m} + 2^n$ to accept $L(\phi)_\$$.*

For transforming LTL to DFA, we have only the doubly-exponential bound for transforming an LTL formula to a UFA and the UFA to DFA.

Lower bounds

We first generalize Lemma 3 to DBAs and Lemma 4 to NBAs. An *observation*

table for an ω-language L is a matrix $T \in \{0,1\}^{\ell \times m}$ with rows indexed by finite words x_i for $i \in [1..\ell]$ and columns indexed by ω-words y_j for $j \in [1..m]$ such that $T_{i,j} = 1$ iff $x_i y_j \in L$. Then we have the following, proved analogously to Lemma 3 and Lemma 4.

Lemma 7. *Let T be an observation table for the ω-language L. If T has n distinct rows, then any DBA accepting L has at least n states.*

Lemma 8. *Let T be an observation table for the ω-language L. If the minimum 1-cover of T has cardinality n, then any NBA to recognize L has at least n states.*

Baier and Katoen [3, Theorem 5.4.2] give a lower bound for a family of LTL formulas ϕ_n of size $\text{poly}(n)$ for which equivalent NBAs must have at least 2^n states. Below we give a simplified and slightly strengthened version of their lower bound, which also applies to M2MAs or NFAs for $L_\$$.

Theorem 2. *For every positive integer n there exists an LTL formula ψ_n of size at most $2n + 6$ such that any NBA accepting $L(\psi_n)$ must have size at least 2^n. Any NFA or M2MA accepting $L(\psi_n)_\$$ must have size or dimension at least 2^n.*

Proof. Let p be a propositional variable. For any positive integer n we define the LTL formula $\quad \psi_n = \Box(p \rightarrow \bigcirc^n(p)) \wedge (\neg p \rightarrow \bigcirc^n(\neg p))$. We use $\bigcirc^n$ to represent the composition of $\bigcirc$ with itself n times, so $\bigcirc^3(p)$ abbreviates $\bigcirc(\bigcirc(\bigcirc(p)))$. The formula ψ_n has size $2n+6$. Let the symbols 0 and 1 represent the assignment of false and true to p. Then $L(\psi_n)$ is the language of ω-words w over $\{0,1\}$ such that for some $x \in \Sigma^n$, $w = x^\omega$.

For $L(\psi_n)_\$$, let $x_1, x_2, \ldots, x_{2^n}$ be any total ordering of all the elements of $\{0,1\}^n$, and consider the observation table T with rows corresponding to x_i and columns corresponding to $\$x_i$ for $i \in [1..2^n]$. Clearly, there is exactly one 1 in row x_i, in the column $\$x_i$, so this observation table is the $2^n \times 2^n$ identity matrix, which has rank 2^n, and any NFA or M2MA accepting $L(\psi_n)_\$$ must have size at least 2^n by Corollary 1.

For the lower bound on NBAs, we observe that if we instead index the columns of T with $(x_i)^\omega$, it becomes an observation table for the ω-language $L(\psi_n)$, and remains the $2^n \times 2^n$ identity matrix, which implies that any NBA accepting $L(\phi_n)$ must have at least 2^n states, by Lemma 8. $\square$

5.2   Size increases for DBAs, NBAs, SUBAs

Upper bounds

For an NBA of n states accepting L, Calbrix, Nivat and Podelski [7] show that there is a DFA of $2^n + 2^{2n^2+n}$ states to accept $L_\$$. Kuperberg, Pinault and Pous [14] give a more concise construction that yields for $L_\$$ an NFA of size $n + n3^{n^2}$ and a DFA of size $2^n + 2^n 3^{n^2}$. For the conversion of an NBA of n states to a SUBA, Carton and Michel provide the upper bound of $(12n)^n$ [8].

Starting with a DBA instead of an NBA, the NFA construction of Kuperberg, Pinault and Pous is fully deterministic, so the upper bound of $n + n3^{n^2}$ holds for transforming a DBA into a DFA. Bousquet and Löding [6] show that a SUBA of n states accepting the ω-language L may be transformed into a UFA of $2n^2 + n$ states accepting $L_\$$.

Lower bounds

For transforming a DBA for L into a DFA for $L_\$$, Angluin and Fisman [2] prove that for every n there is a DBA of $n + 2$ states accepting a language L such that no DFA of fewer than $n!$ states accepts $L_\$$. For transforming a DBA into a UFA, M2MA or NFA, we prove the following result.

Theorem 3. *For every even positive integer n there is an ω-language L_n that is accepted by a DBA of $n + 5$ states such that any UFA, NFA or M2MA to accept $(L_n)_\$$ must have size or dimension at least $\binom{n}{n/2}$, which is $\sim 2^n/\sqrt{\pi n/2}$.*

Proof (Sketch). The proof uses a modification of the DBAs in the construction by Angluin and Fisman [2]. Here we sketch the main idea and give an example. Let $n = 2k$ for some nonnegative integer k, let $\Sigma_{2k} = \{\sigma_1, \ldots, \sigma_{2k}\}$ and let Σ be $\Sigma_{2k} \cup \{0, L, E, F\}$. Consider the regular ω-language defined by the ω-regular expression $\left(\cup_{\sigma \in \Sigma \setminus \{0\}} (\sigma \cdot (\Sigma \setminus \{\sigma\})^* \cdot \sigma)\right)^\omega$, which is accepted by a DBA with $2k + 5$ states. Given two subsets C and D of Σ_{2k}, each of size k, we define words u_C and v_D such that $(u_C \cdot v_D)^\omega$ is in the language if and only if $C = D$. The main idea behind the construction is that v_D forces each symbol σ_D in $\Sigma_{2k} \setminus D$ to be followed by the character 0. Thus, if the string preceding (and including) an occurrence of such a symbol σ_D is described by the (unambiguous) regular expression $\left(\bigcup_{\sigma \in \Sigma \setminus \{0\}} \sigma \cdot (\Sigma \setminus \{\sigma\})^* \cdot \sigma\right)^*$, then the symbol 0 that follows cannot be properly consumed, resulting in the ω-word being not in the language. We construct the words u_C and v_D in such a way that this can happen if and only if such a symbol $\sigma_D \in \Sigma_{2k} \setminus D$ is also in C. Since C and D are subsets of Σ_{2k}, each of size k, this happens exactly when $C \neq D$. There is therefore an observation table with rows indexed by $\$u_C$ for all subsets C of size k and whose columns are indexed by v_D for all subsets D of size k, and where each entry, corresponding to row and column subsets C and D respectively, is 1 if and only if $C = D$. By Corollary 1, the result follows. □

Example. Let $\Sigma_{2k} = \{1, 2, 3, 4\}$, let Σ be $\Sigma_{2k} \cup \{0, L, E, F\}$, let $C = \{2, 3\}$, and let $D = \{2, 4\}$. Then u_C, v_C and v_D are defined on the right. Then $(u_C \cdot v_C)^\omega$ is in the language, whereas $(u_C \cdot v_D)^\omega$ is not (since $C \neq D$).

$$u_C = F \cdot 2 \cdot F \cdot 2 \cdot 3 \cdot 2 \cdot 3 \cdot L \cdot 3$$
$$v_C = L \cdot E \cdot 1 \cdot 0 \cdot 4 \cdot 0 \cdot E$$
$$v_D = L \cdot E \cdot 1 \cdot 0 \cdot 3 \cdot 0 \cdot E$$

For the lower bound on transforming a DBA into a SUBA, Bousquet and Löding [6] show that for every positive integer n there exists an ω-language that is accepted by a DBA with $n + 1$ states, and cannot be accepted by a SUBA with fewer than 2^{n-1} states.

For transforming a SUBA into a DFA, Angluin, Antonopoulos and Fisman [1, Theorem 5] give a family of ω-languages such that L_n is accepted by a SUBA of size $4n + 5$, but any DFA to accept $(L_n)_\$$ or its reverse must have size at least 2^n. For transforming a SUBA into a UFA, M2MA or NFA, we prove the following asymptotically tight lower bound.

Theorem 4. *For every positive integer m greater than 3, there is an ω-language L that is accepted by a SUBA with m states, but no M2MA of dimension less than $2m^2 - m + 2$ or NFA or UFA of size less than $2m^2 - m + 2$ accepts $(L)_\$$.*

Proof (Sketch). For every $n \in \mathbb{N}$ we define L_n to be the regular ω-language over $\Sigma = \{a, b, c\}$ given by the expression $((cc \cdot b^n)^* \cdot aa \cdot b^n)^\omega$. This language is accepted by a SUBA S_n, with $m = n + 3$ states. We construct a specific observation table M for the language $(L_n)_\$$. We then show that any 1-rectangle cover of M is of size at least $2m^2 - m + 2$, which implies by Lemma 4 that the number of states of any NFA (or UFA) for the language $(L_n)_\$$ is at least $2m^2 - m + 2$. We further show that the rank of M is $2m^2 - m + 2$, and by Lemma 3, obtain that the dimension of any M2MA for this language is also at least $2m^2 - m + 2$. □

6 Empirical results

We report typical size increases in going from a random SUBA, DBA or NBA acceptor for a regular ω-language L to a minimized M2MA (and DFA, in the case of a SUBA) for $L_\$$. We also report computed sizes of minimized M2MAs and DFAs for $L(\phi_n)_\$$ for members of particular families $\{\phi_n\}$ of LTL formulas. Code is available in the GitHub repository:
https://github.com/nevingeorge/Learning_Automata.
    For the generation of random SUBAs, DBAs or NBAs, our procedure is as follows. Given parameters n, f, and t we generate a transition relation on n states (random reverse-deterministic for a SUBA, random deterministic for a DBA, and all possible transitions for an NBA), select f of the n states at random to be final, and randomly remove t of the transitions. The resulting transition relation is trimmed to remove non-live states and their transitions. The trimmed acceptor may have fewer than n states.
    If the goal is a SUBA, using the criterion of Wilke [21], we check that there do not exist two different states q_1 and q_2 and a nonempty finite word v such that for $i = 1, 2$, there is a loop on v from q_i to q_i that passes through a final state. If the acceptor fails this test, it is rejected, and the procedure is repeated until a SUBA is successfully generated.

6.1 SUBAs to minimized M2MAs and DFAs

For random SUBAs to minimized M2MAs, we first generate a random SUBA with $\Sigma = \{a, b, c\}$, $n \in \{5, 10, 15\}$, $t \in \{[1, 5], [2, 10], [18, 22]\}$ (resp.), and $f = 2$ or $f = 3$ with equal probability. We then convert it into a UFA using the algorithm of Bousquet and Löding [6], and minimize the equivalent M2MA.

Minimized M2MA Dimension
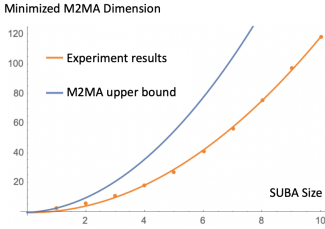
Fig. 4: Random SUBAs to minimized M2MAs
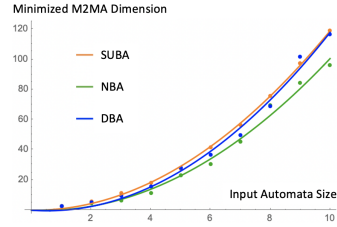
Fig. 5: Random SUBAs, NBAs, and DBAs to minimized M2MAs
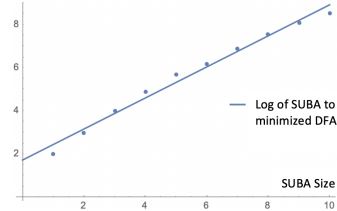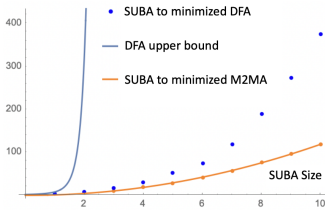
Fig. 6: Random SUBAs to minimized DFAs

We performed the above process on approximately $220,000$ randomly generated SUBAs.

Fig. 4 is a plot of the average minimized M2MA dimension for each trimmed SUBA size from 1 to 10. Upon performing quadratic regression, we obtain the orange curve $1.212n^2 - .2248n$, and the blue curve is the theoretical upper bound of $2n^2 + n$ given in Fig. 3. The quadratic fit has a R^2 of 0.9996 while a linear fit has a R^2 of 0.9370, suggesting that the growth is indeed quadratic. This curve satisfies the theoretical upper bound of $2n^2 + n$, and suggests that the lower bound of $\Omega(n^2)$ holds on average.

For random SUBAs to minimized DFAs, we also calculated the number of reachable states of each minimized M2MA. This is the number of states in the equivalent minimized DFA, by a property of the minimization algorithm of Corollary 2. From Fig. 3, the lower bound in going from a SUBA to a DFA is $2^{\Omega(n)}$, and the upper bound is $2^n + 2^n 3^{n^2}$.

In the left graph in Fig. 6, the blue data points representing the results of the SUBA to DFA experiment grow much more sharply than the results of the SUBA to M2MA experiment, so it is clear that a SUBA can be represented more concisely as an M2MA than as a DFA on average. Upon taking the log (base 2), we obtain a roughly linear fit as seen in the right graph with equation $.7196n + 1.738$ and a R^2 of .9841, suggesting that on average the growth is exponential. The standard deviation and range of converted DFA sizes was large for this conversion, making it difficult to make firm claims about the growth. However, the data suggests that the exponential lower bound likely holds on average, and that in general the upper bound of $2^n + 2^n 3^{n^2}$ is a severe overestimate.

6.2   NBAs and DBAs to minimized M2MAs

For NBAs and DBAs, a minimized M2MA is computed using the M2MA learning algorithm of Beimel et al. [4], which makes membership and equivalence queries to the NBA or DBA. Instead of exact equivalence queries, we use approximate equivalence queries, implemented by testing membership agreement on a sample of randomly generated ultimately periodic words. Thus, the dimension of the learned M2MA may be an underestimate of the true minimum dimension of an M2MA for $L_\$$.

For the NBA/DBA to M2MA experiments, we generated approximately 1000 random NBAs/DBAs with $\Sigma = \{a, b, c\}$, $n \in \{5, \ldots, 10\}$, $t \in [0, n]$ for DBAs and t in ranges within $[90, 680]$ for NBAs, and $f = 2$ or $f = 3$ with equal probability. For the approximate equivalence queries, we tested 1000 random ultimately periodic words of length at most 25. The results of the experiments can be seen in Fig. 5. The fitted NBA and DBA curves are quadratic with equations $1.096n^2 - .8947n$ and $1.318n^2 - 1.392n$, respectively. The quadratic fits for the NBA and DBA results have a R^2 of .9954 and .9961, respectively, while linear fits have a R^2 of .9227 and .9118, respectively. These experiments have limitations: the use of approximate equivalence queries, the small sample size (because of the time requirements of the learning algorithm), and the large standard deviation and range of converted M2MA sizes. However, the results from all three conversions are very similar, suggesting that in these conditions, SUBAs, NBAs, and DBAs don't vary significantly on average with respect to their equivalent M2MA representations.

6.3   LTL formulas to minimized M2MAs

Random LTL formulas seem not to provide much insight, so we consider specific families of LTL formulas: bounded request/grant formulas and two families based on the hierarchy of Manna and Pnueli [15], namely obligation and reactivity formulas. Empirically, for each of the first few members of each family we calculate the minimum dimension of an M2MA and the minimum size of a DFA accepting the corresponding $L_\$$ language, and use the online tool provided by the Spot website (https://spot.lrde.epita.fr/) to find an ω-language acceptor for the corresponding L. (Omitted Spot entries exceeded the limit on calculation time.)

The canonical request/grant formula is of the form $\Box(p \to \Diamond(q))$, which asserts that whenever a request (p) is made, it is eventually granted (q). In the bounded version, a number of steps n is specified, and the assertion is that the request is granted within n steps. Thus, for each natural number n, we have a formula $R_n = \Box(p \to (q \vee \bigcirc(q) \vee \bigcirc^2(q) \vee \ldots \vee \bigcirc^n(q)))$. The table in Fig. 7a gives the resulting sizes and dimensions for n from 0 to 5. It is reasonable to conjecture $n + 1$ for the size of a DBA, $n^2 + 3n + 3$ for the minimum dimension of an M2MA, and $2n^2 + 3n + 4$ for the minimum size of a DFA representing R_n.

The family of obligation formulas we consider is: $F_n = \wedge_{i=1}^{n}(\Box p_i \vee \Diamond q_i)$. Using conjunction and minimization, we calculate the minimum dimension M2MA (and minimum size DFA) for $L_\$$ for these formulas for n up to 5. The table in Fig. 7b

n	DBA	M2MA	DFA
0	1	3	4
1	2	7	9
2	3	13	18
3	4	21	31
4	5	31	48
5	6	43	69

(a) R_n sizes.

n	DBA	M2MA	DFA
1	3	7	9
2	9	19	23
3	27	55	63
4	81	163	179
5	–	487	519

(b) F_n sizes.

n	GNBA	M2MA	DFA
1	(4,1)	5	6
2	(10, 2)	11	12
3	(28, 3)	29	30
4	–	83	84
5	–	245	246

(c) G_n sizes.

Fig. 7: Size or dimension of acceptors for families of LTL formulas.

shows the results. It is reasonable to conjecture 3^n for the size of a DBA, $2 \cdot 3^n + 1$ for the minimum dimension of an M2MA, and $2 \cdot 3^n + 2^n + 1$ for the minimum size of a DFA to represent F_n.

The family of reactivity formulas we consider is: $G_n = \wedge_{i=1}^{n}(\Box\Diamond p_i \vee \Diamond\Box q_i)$. We proceed as for the obligation formulas, with the results shown in the table in Fig. 7c. Note that these formulas cannot be represented by DBAs, but are instead represented by GNBAs, which may have multiple sets of final states. For example, the entry $(10, 2)$ indicates a GNBA with 10 states and 2 sets of final states. A reasonable conjecture in this case is $(3^n + 1, n)$ for the size of a GNBA, $3^n + 2$ for the minimum dimension of an M2MA, and $3^n + 3$ for the minimum size of a DFA representing G_n.

In these cases, the minimum dimension of an M2MA (and size of a DFA) appears to grow at most as a polynomial in the size of an ω-language acceptor, quadratically for the bounded request/grant family, and linearly for the obligation and reactivity families.

7   Summary and conclusions

We provide a survey of size relations of M2MAs as a representation of regular languages and regular ω-languages, as well as empirical results for several of these relations. New theoretical results include an improvement of the lower bound for transforming an M2MA to an NFA, an upper bound of $2^{O(n)}$ for the translation of an LTL formula of size n to a UFA, NFA, or M2MA, a lower bound of $2^{\Omega(n)}$ for the translation of a DBA of n states to an M2MA or NFA, and an asymptotically optimal lower bound of $2n^2 - n + 2$ for the translation of a SUBA of n states to an M2MA or NFA.

M2MAs have many advantages as a representation for regular ω-languages: determinism, succinct complementation, and polynomial time algorithms for minimization, equivalence testing, and learning with membership and equivalence queries. M2MAs are as succinct as DFAs, sometimes exponentially more so, and deserve further study.

References

1. Angluin, D., Antonopoulos, T., Fisman, D.: Strongly unambiguous Büchi automata are polynomially predictable with membership queries. In: 28th EACSL Annual Conference on Computer Science Logic, CSL. pp. 8:1–8:17 (2020)
2. Angluin, D., Fisman, D.: Learning regular omega languages. Theor. Comput. Sci. **650**, 57–72 (2016)
3. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press (2008)
4. Beimel, A., Bergadano, F., Bshouty, N.H., Kushilevitz, E., Varricchio, S.: Learning functions represented as multiplicity automata. J. ACM **47**(3), 506–530 (May 2000)
5. Bergadano, F., Varricchio, S.: Learning behaviors of automata from multiplicity and equivalence queries. SIAM J. Comput. **25**(6), 1268–1280 (1996)
6. Bousquet, N., Löding, C.: Equivalence and inclusion problem for strongly unambiguous Büchi automata. In: Language and Automata Theory and Applications, 4th International Conference, LATA. Proceedings. pp. 118–129 (2010). https://doi.org/10.1007/978-3-642-13089-2_10
7. Calbrix, H., Nivat, M., Podelski, A.: Ultimately periodic words of rational w-languages. In: Proceedings of the 9th International Conference on Mathematical Foundations of Programming Semantics. pp. 554–566. Springer-Verlag (1994)
8. Carton, O., Michel, M.: Unambiguous Büchi automata. Theor. Comput. Sci. **297**(1-3), 37–81 (2003). https://doi.org/10.1016/S0304-3975(02)00618-7
9. Diekert, V., Gastin, P.: First-order definable languages. In: Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]. pp. 261–306 (2008)
10. Droste, M., Kuich, W., Vogler, H. (eds.): Handbook of Weighted Automata, chap. 4: Rational and Recognizable Series, by Jaques Sakarovitch, pp. 105–174. Springer-Verlag Berlin Heidelberg (2009)
11. Gerth, R., Peled, D., Vardi, M., Wolper, P.: Simple on-the-fly automatic verification of linear temporal logic. In: Protocol Specification, Testing and Verification XV. PSTV 1995. Springer (1996). https://doi.org/10.1007/978-0-387-34892-6_1
12. Hromkovič, J.: Communication Complexity and Parallel Computing. Springer-Verlag Berlin Heidelberg (1997), (There is also 2013 edition.)
13. Kaznatcheev, A., Panangaden, P.: Weighted automata are compact and actively learnable. Information Processing Letters **171** (2021), (The authors were apparently unaware of prior results on learning multiplicity automata by Beimel et al. and others.)
14. Kuperberg, D., Pinault, L., Pous, D.: Coinductive algorithms for Büchi automata. In: Developments in Language Theory - 23rd International Conference, DLT Proceedings. pp. 206–220 (2019)
15. Manna, Z., Pnueli, A.: A hierarchy of temporal properties (invited paper, 1989). In: Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing. p. 377–410. PODC '90, Association for Computing Machinery (1990). https://doi.org/10.1145/93385.93442
16. Michel, M.: Complementation is much more difficult with automata on infinite words. In: Manuscript, CNET (1988)
17. Moser, B.K.: Linear algebra and related introductory topics. In: Linear Models, A Mean Model Approach, A volume in Probability and Mathematical Statistics. pp. 1–22 (1996)
18. Pnueli, A.: The temporal logic of programs. In: FOCS. pp. 46–57 (1977)
19. Sakarovitch, J.: Elements of Automata Theory. Cambridge University Press, USA (2009)

20. Vardi, M.Y.: An automata-theoretic approach to linear temporal logic. In: Logics for Concurrency - Structure versus Automata (8th Banff Higher Order Workshop, Banff, Canada, August 27 - September 3, 1995, Proceedings). pp. 238–266 (1995). https://doi.org/10.1007/3-540-60915-6_6
21. Wilke, T.: ω-automata. CoRR **abs/1609.03062** (2016), http://arxiv.org/abs/1609.03062

Limits and difficulties in the design of under-approximation abstract domains⋆

Flavio Ascari✉ , Roberto Bruni , and Roberta Gori

Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo 3, Pisa, Italy,
`flavio.ascari@phd.unipi.it`, {`roberto.bruni,roberta.gori`}`@unipi.it`

Abstract. Static analyses are mostly designed to show the *absence of bugs*: if the analysis reports no alarms then the program won't exhibit any unwanted behaviours. To this aim they manipulate over-approximations of program semantics and, inevitably, they often report some *false* alarms. Recently, O'Hearn proposed Incorrectness Logic, that is based on under-approximations, as a formal method to *find bugs* that only reports *true* alarms. In this paper we aim to answer one important question raised by O'Hearn, namely which role can Abstract Interpretation play for the development of under-approximate tools for bug catching. In principle, Abstract Interpretation based static analyses can be defined for computing over-approximations as well as under-approximations, but in practice, most techniques exploited the former while few attempts developed the latter. To show why it is difficult to design effective under-approximation abstract domains, we first propose the new definitions of *non emptying functions* and *highly surjective function family* and then we formally prove the limits of under-approximation analysis by showing the non existence of abstract domains able to approximate such functions in a non trivial way. Our results outline the limits of under-approximation Abstract Interpretation and clarify, for the first time, why over- and under-approximation analyzers exhibited such a different development.

Keywords: Abstract Interpretation, Under-approximation, Abstract domains, Impossibility results

1 Introduction

Static program analyses are techniques used to infer properties of programs directly from their source code, without executing them. They have been studied and successfully applied for over 50 years [12,3,13,1,10,17,18,22,23,4] to produce effective methods and tools to support the development of correct software. For all these years, the main focus of static analysis was to prove the absence of bugs by computing over-approximations (supersets of all possible behaviours) of the semantics of programs: the absence of unwanted behaviour

---

⋆ Research supported by MIUR PRIN Project 201784YSZ5 *ASPRA–Analysis of Program Analyses*.

in the over-approximation guarantees the correctness of the program. However, over-approximations cannot be used to expose bugs, since any alert raised by the analyser may be caused by the over-approximation rather than by the program, i.e. it can be a so called false alarm. From the point of view of a software developer, false alarms are undesirable because they undermine the credibility and usefulness of the analysis. In principle, there is a symmetrical approach to static analysis, that is to compute an under-approximation of the semantics, i.e., a subset of all possible behaviours of a program. Dually to over-approximations, under-approximations can then expose defects in the code, while they are unable to show their absence.

Early works on static analysis, like Hoare logic [13], focused on over-approximation to prove the absence of errors, and maybe their influence directed the focus toward over-approximations. Recently O'Hearn argued for the relevance of bug catching with respect to correctness proofs and proposes the Incorrectness Logic [19], a dual version of Hoare logic thought from the ground up for under-approximation. He also advocates for a similar change of perspective in the static analyses approach.

For instance, consider the simple code

```
for(i = 0; i < 5; ++i) sum += 1000 / (2 * i) + 100 / (2 * i - 5);
```

An abstract analysis based on the domain Int of intervals allows to over-approximate the set of possible values each variable can take as the smallest interval that contains such values. When applied the above program, the analysis may detect that the value of variable i is between 0 and 4 within the body of the loop, so that the arithmetic expression 2 * i is then over-approximated by the interval $[0, 8]$ while 2 * i - 5 by the interval $[-5, 3]$. This raises two warnings for possible division by zero, since it seems that both arithmetic expression may assume the value 0. It is worth noting that while the warning on the first expression is a true alarm, the warning on the second one is a false alarm. On the contrary, an analysis based on under-approximation will never raise a warning for the second expression since no value of i can cause an error in this case, However, not all under-approximations will detect the problem with 2 * i, because any subset of $\{0, 2, 4, 6, 8\}$ is a valid under-approximation, including e.g. $\{2, 4, 6, 8\}$.

The Problem: Abstract Interpretation [6,22,4] is a general framework to define sound analyses based on constructive approximations that found its way through many aspects of modern computer science, such as verification, optimization, security and program transformation. Given its broad applicability, in his paper on Incorrectness Logic [19], O'Hearn leaves as an open question whether Abstract Interpretation could "*eventually play a guiding and explanatory role for a wide range of static and dynamic under-approximate tools for bug catching, similar to what it already does for over-approximate analyses*". The goal of this work is to investigate this topic. The results we have achieved will establish that under-approximation based Abstract Interpretation analyses have serious intrinsic limitations, and therefore our contribution can be read as a negative answer, even if we will then discuss how to overcome some limits.

Related Work: In their first works on Abstract Interpretation [6], Cousot and Cousot introduced the formal theory that could be used to define either over- or under-approximations. However, while the former has been extensively studied, there have been only sparse studies on the latter. Bourdoncle [2] proposed abstract debugging using over-approximation domains, but acknowledged that under-approximation ones could be better suited. Lev-Ami et al. [14] proposed to use complements of over-approximation domains to infer sufficient precondition for program correctness. For the same goal, Miné [15] used directly over-approximation domains, giving up the best abstraction and handling the choice of a maximal one with heuristics. To infer necessary condition for incorrectness, a problem similar to O'Hearn's but studied for a different goal, Cousot et al. [9,8] use Abstract Interpretation techniques but on boolean formulas, hence bypassing the issue of defining an abstract domain. Schmidt [24] uses higher-order domains, defining abstract states with meaning "there exists a value satisfying this over-approximation property", hence giving rise to an under-approximation of over-approximations. In conclusion, all the above approaches design under-approximation domains starting from over-approximation ones, and, to the extent of our knowledge, there are no abstract domains thought from the ground up for under-approximation. So the question whether it is possible to design an abstract domain for computing under-approximations naturally arises.

Contributions: We believe the absence of under-approximation abstract domains to be caused by intrinsic difficulties in their design. In this article, we determine and explain the reasons behind these difficulties. In the following we point out some intuitive asymmetries that suggest why under-approximations are not as immediate to use as over-approximations for program analysis.

While over- and under-approximation can be thought as dual theories, they have a deep asymmetry when dealing with the semantics of basic constructs of the language, the so called basic transfer functions. For instance, given an over-approximation abstract domain, we can define an under-approximation domain by taking the opposite interpretation of abstract elements: the idea is that an abstract element represents all concrete elements that may not be present in the set of possible values. As a consequence of being an under-approximation, this means that all the other concrete elements (the complement of the set) *must* be actual values. Considering the abstract domain of (complemented) intervals, it happens, e.g. that an arithmetic expression such as a sum of variables is often under-approximated as the whole $\mathbb{Z}$. It is also worth noticing that, while basic transfer functions are the same, over-approximation abstract domains are closed under intersection, while under-approximation abstract domains are closed under union and can grow large very easily.

Another asymmetry we point out is the handling of divergence. Divergence is represented in over- and under-approximation by the same abstract element $\bot$, but note that $\bot$ as an under-approximations also represents the absence of information (dually to $\top$ in over-approximations). This becomes a problem since many concrete functions are strict, that is, when applied to a non-terminating expression, they also fail to terminate (they return $\bot$ if one argument is $\bot$), and,

to be a correct under-approximation, also the corresponding abstract function needs to be strict. This implies that whenever the analysis can't determine any meaningful information at some program point, it has to propagate this absence of information along all program paths, at least until a join in the control flow is found. So "recovery" from $\bot$, that is, producing a result different from $\bot$, once we start with it, is very hard in an under-approximation. Note that, on the contrary, "recovery" from $\top$ in an over-approximation is quite easier, e.g. by a constant assignment.

The previous arguments are substantiated by formal impossibility results for building meaningful under-approximation abstract domains. First, we introduce the new definition of *non emptying function*, describing functions that don't tamper the analysis and we prove that no abstract domain for integers can be constructed that makes all sums non emptying. Second, we propose two generalizations (one local and one global) of the result for integers domains to arbitrary concrete domains and function families, by introducing the notion of *highly surjective function family*, of which sums are an instance. The local condition applies to each function in the family, while the global condition is a property of the whole family. Finally, we study hypothesis for the existence of abstract domains making all functions in a family non emptying to show first that the hypothesis of high surjectivity is tight, and then that further conditions on the function family must hold.

Structure of the paper: In Section 2 we introduce the notation used in the rest of the paper and recall the basics of Abstract Interpretation for over- and under-approximations. In Section 3 we apply our idea to the concrete domain of integers to show that, under some simple conditions, no under-approximation abstract domain can exist. In Section 4 we extend the result obtained for integers to arbitrary concrete domains and function families. In Section 5 we show that the hypothesis of high surjectivity is needed and explore other requirements for the function family. Section 6 contains some concluding remarks and an outline of future research directions. Due to space limitation, only informal proof sketches are included in this proceedings.

2   Background

Notation. We let $\mathcal{P}(S)$ denote the powerset of the set S and $\mathrm{id}_S : S \to S$ be the identity function on a set S. We omit subscripts when obvious from the context. If $f : S \to T$ is a function, then we overload the symbol f to denote also its additive extension $f : \mathcal{P}(S) \to \mathcal{P}(T)$ defined as $f(X) = \{f(x) \,|\, x \in X\}$ for any $X \subseteq S$. We say a function $f : S \to S$ is *acyclic* if, for any element $x \in S$ and any $n > 0$, we have $f^n(x) \neq x$, where f^n denotes composition of f with itself n times. In ordered structures, such as posets and lattices, we usually denote the ordering with $\preceq$, least upper bounds (lubs) with $\sqcup$, greatest lower bounds (glbs) with $\sqcap$, least element with $\bot$, greatest element with $\top$. If $\preceq$ is an order relation, $\succeq$ is the opposite relation, defined as $s \succeq t$ if and only if $t \preceq s$. We write just

S for the poset $(S, \preceq)$ whenever the order relation $\preceq$ is known from the context and we use S^{op} to denote the opposite poset $(S, \succeq)$: hence S^{op} denotes the same set as S, but S^{op} comes equipped with the opposite ordering relation $\succeq$. Given a poset T and two functions $f, g : S \to T$, the notation $f \preceq g$ means that, for all $s \in S$, $f(s) \preceq g(s)$. Any powerset is a complete lattice with ordering given by the inclusion relation. In this case, we use standard symbols $\subseteq$, $\cup$, etc.

Abstract Interpretation. Abstract Interpretation [6,7,16] is a general framework to define sound-by-construction static analyses, with the main idea of approximating the program semantics on some abstract domain A instead of working on the concrete domain C. The main tool used to study Abstract Interpretations are Galois connections. Given two complete lattices C and A, a pair of monotone functions $\alpha : C \to A$ and $\gamma : A \to C$ define a Galois connection (GC) when

$$\forall c \in C, a \in A. \quad \alpha(c) \preceq a \iff c \preceq \gamma(a)$$

and we denote it with $\langle C \xrightarrow[\alpha]{\gamma} A \rangle$. We call C and A, respectively, the concrete and the abstract domain, α is the abstraction function and γ is the concretization function. In any GC, $\text{id}_C \preceq \gamma \circ \alpha$, $\alpha \circ \gamma \preceq \text{id}_A$, γ preserves glbs and α preserves lubs. In particular, this means that $\gamma(\top_A) = \top_C$ and dually $\alpha(\bot_C) = \bot_A$.

A GC in which $\alpha \circ \gamma = \text{id}_A$ is called Galois insertion (GI), and if this is the case also α is onto and γ is injective. By this last property, there is a bijection between A and $\gamma(A)$, and using this isomorphism, whenever we consider a GI we identify A and its γ-image so that A becomes a subset of C and $\gamma = \text{id}_A$, written as $\langle C \xleftrightarrow[\alpha]{} A \rangle$. A GI is said to be trivial if A is the concrete domain or it only contains $\top_C$.

Given a monotone function $f : C \to C$ and a GC $\langle C \xrightarrow[\alpha]{\gamma} A \rangle$, a function $f^{\sharp} : A \to A$ is a correct (or sound) approximation of f if $\alpha \circ f \preceq f^{\sharp} \circ \alpha$. Its best correct approximation (bca) is $f^A = \alpha \circ f \circ \gamma$, and it is the most precise of all the correct approximation of f.

As an example, let us consider $C = \mathcal{P}(\mathbb{Z})$ be the powerset of integers and $A = \text{Int}$ be the abstract domain of intervals [6]. Elements of Int are finite intervals $[n, m]$ with $n \leq m$, or infinite intervals of the form $[-\infty, m]$ or $[n, \infty]$, together with the empty interval $\bot$. The top element is $[-\infty, \infty]$. Intervals are ordered by inclusion, the concretisation function γ is defined as usual, while the abstraction function α maps a set of integers to the smallest interval that contains it. If $f(x) = |x|$ is the absolute value function, one of its sound abstractions is $f^{\sharp}([n, m]) = [0, \max(|n|, |m|)]$ because the interval $[0, \max(|n|, |m|)]$ always contains the entire set $f(S)$ when $n = \min(S)$ and $m = \max(S)$. However this is not the best possible abstraction: for instance on $S = \{1\}$ this yields $[0, 1]$ while $f(S) = \{1\}$. Actually the best correct abstraction f^A is computed as

$$f^A([n, m]) = \alpha \circ f \circ \gamma([n, m]) = \begin{cases} [0, \max(|n|, |m|)] & \text{if } n \leq 0 \leq m \\ [n, m] & \text{if } 0 < n \\ [-m, -n] & \text{if } m < 0 \end{cases}$$

2.1   Under-approximation Galois Connections

The definition of GC is not symmetric in γ and α: it favours over-approximation, and is not suited to describe under-approximations. This can be more easily seen from the property $\mathrm{id}_C \preceq \gamma \circ \alpha$, that means the abstraction $\gamma(\alpha(c))$ of a concrete element c is greater than (ie. an over-approximation of) c itself. For this reason we introduce the notion of under-approximation Galois connection (UGC). Formally, an UGC is just a GC between A and C, in the reverse order, or equivalently a GC in which we replaced C and A with C^{op} and A^{op}. However, we believe this definition to allow a better notation, helping the reader's intuition. Given two complete lattices C and A, a pair of monotone functions $\alpha : C \to A$, $\gamma : A \to C$ defines an UGC between C and A when

$$\forall c \in C, a \in A. \quad a \preceq \alpha(c) \iff \gamma(a) \preceq c$$

and we denote such UGC with $\langle C \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} A \rangle$. Note the different positions of arrows and their super/subscripts when compared with a GC $\langle C \underset{\alpha}{\overset{\gamma}{\rightleftarrows}} A \rangle$. The difference

(a) Over-approximation GC   (b) Under-approximation GC

Fig. 1: Sketches of GC and UGC

between a GC and an UGC is sketched in Figure 1: in the GC (on the left) γ is above and α below, while in the UGC (on the right) the two are reversed. Using the duality observed above, from standard properties of GCs we get, reversing inequalities, that $\gamma \circ \alpha \preceq \mathrm{id}_C$, $\mathrm{id}_A \preceq \alpha \circ \gamma$, γ preserves lubs and α preserves glbs. Moreover, an under-approximation Galois insertion (UGI) is an UGC in which $\alpha \circ \gamma = \mathrm{id}_A$, and has the properties of α being onto and γ being injective, making the same identification of A with $\gamma(A)$ possible, written as $\langle C \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} A \rangle$. In particular, this means that in an UGI on a concrete powerset $\langle \mathcal{P}(C) \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} A \rangle$, for all $a, a' \in A$, $\gamma(a \cup a') = a \cup a'$, that is A is closed under union.

   Dually to standard, over-approximation GCs, given a monotone function $f : C \to C$ and an UGC $\langle C \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} A \rangle$, a function $f^\flat : A \to A$ is a correct (or sound)

abstraction of f if $\alpha \circ f \succeq f^\flat \circ \alpha$. Again, $f^A = \alpha \circ f \circ \gamma$ is the best correct approximation of f.

As an example, let us take again $C = \mathcal{P}(\mathbb{Z})$ and $A = \mathsf{Int}_0$ be the set of integer intervals around 0, ie. $\mathsf{Int}_0 = \{I \in \mathsf{Int} \mid 0 \in I\} \cup \{\bot\}$. This is an under-approximation abstract domain because it contains $\bot$ and is closed under union: the union of intersecting intervals is an interval too, and all elements of Int_0 intersects at 0. If again $f(x) = |x|$ is the absolute value function, its bca f^A is $f^A([n, m]) = [0, \max(|n|, |m|)]$ since it's always the case that $n \le 0 \le m$.

3   Integer Domains

In this section we focus on under-approximations of integer domains and prove that any under-approximation abstract domain will mostly return trivial analyses for programs that include sums inside arithmetic expressions.

To this aim, we introduce the concept of *non emptying function*.

Definition 1 (Non emptying function). *Let $\langle C \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} A \rangle$ be an UGC, $f : C \to C$ a monotone function and $f^A = \alpha \circ f \circ \gamma$ its bca. We say that f is* non emptying *(in A) if, for any concrete value c, $\alpha(c) \ne \bot$ and $\alpha(f(c)) \ne \bot$ imply $f^A(\alpha(c)) \ne \bot$.*

Remember that $\bot$ does not give any interesting information in the under-approximation setting, because it can mean divergence as well as complete loss of precision. On the contrary, any abstract element different than $\bot$ means "something" interesting. The rationale behind the definition of non emptying function is that if the analysis starts from something ($\alpha(c) \ne \bot$) and it can find something ($\alpha(f(c)) \ne \bot$) then it will find at least one of the possible results ($f^A(\alpha(c)) \ne \bot$), thus not falling to $\bot$ and avoiding the issues discussed in the Introduction. The meaning of Definition 1 is illustrated by the following toy example.

Example 2. Consider the simple imperative fragment

```
if (x ≠ 0) then { while (x < 10) { y := 7 / x; x := x + 1; } }
```

where a careless programmer used the condition $\mathtt{x} \ne 0$ instead of the expected $\mathtt{x} > 0$: on any initial state where x is negative the program incurs a division by 0 error.

For the analysis, suppose $\mathtt{x}$ is an integer value and consider the domain $\mathsf{Int}_{01} = \{I \in \mathsf{Int} \mid 0 \in I \vee 1 \in I\} \cup \{\bot\}$, a variation of Int_0 such that each interval in Int_{01} must contain at least one of 0 and 1. By an argument similar to that for Int_0 it can be shown that Int_{01} is closed under union (since 0 and 1 are consecutive values in the integer domain), and thus is an under-approximation domain.

Assume to start the analysis in this domain with the initial condition $[-1; 10]$ for variable x: remember that this being an under-approximation analysis, the abstract state $[-1; 10]$ means that x may assume all the values in that interval at the beginning of the code fragment. In the concrete execution, the filter $x \ne 0$ then produces the concrete set of values $c = \{-1, 1, 2, \dots, 10\}$, but the abstract

interpreter must abstract this to its largest subset that is an interval containing 0 or 1, that is $[1; 10]$. The abstract analysis of the cycle then proceeds straightforwardly, finding $\bot$ after one iteration of the loop body (since after the increment the set of values for x is $\{2, 3, \ldots, 11\}$ that is abstracted to $\bot$ because it doesn't contain neither 0 nor 1) and so the abstract fixpoint of the loop $[1; 10]$. This yields no error, even though the concrete execution starting at $x = -1$ does indeed fail after one iteration. The issue here is that the semantics f of the increment $x :=$ $x + 1$ is not non emptying in Int_{01}: on the concrete value $c = \{-1, 1, 2, \ldots, 10\}$, its input in this program, we have $\alpha(f(c)) = \alpha(\{0, 2, 3, \ldots, 11\}) = [0] \neq \bot$ but $f^A(\alpha(c)) = f^A([1; 10]) = \alpha(f(\gamma([1; 10]))) = \alpha(\{2, 3, \ldots, 11\}) = \bot$.

For the remainder of the paper we assume a set of concrete *values* C, an UGI $\langle \mathcal{P}(C) \overset{\alpha}{\underset{\gamma}{\rightleftarrows}} A \rangle$ with concrete domain $\mathcal{P}(C)$, and we say an element $S \in \mathcal{P}(C)$ is *representable* if it belongs to A, or equivalently if $\alpha(S) = S$.

Definition 3. *Let $S \subseteq C$ be a subset of C. We say that $d \in C$ is* representable with S *if $S \cup \{d\}$ is representable. We call $R(S)$ the set of elements of C representable with S, ie.*

$$R(S) = \{d \in C \,|\, \alpha(\{d\} \cup S) = \{d\} \cup S\}$$

For the sake of brevity, we shall write R for $R(\emptyset)$, the set of representable values of C, and $R(c)$ for $R(\{c\})$ where $c \in C$ is any concrete value. The following is a technical lemma valid for non emptying functions, that explains the role played by Definition 1 in proving all our negative results (Propositions 7, 10 and Theorems 12, 15).

Lemma 4. *Let $f : C \to C$ be non emptying, $c \in R$ and the pair $\{c, \bar{c}\}$ be not representable, ie. $\bar{c} \notin R(c)$. If $f(\bar{c}) \in R$ then also $f(c) \in R$.*

The main proof line of all our impossibility results is the same, and exploit this Lemma. All our results requires the size of the abstract domain to be comparable with that of the set of concrete values C (whose powerset $\mathcal{P}(C)$ is the concrete domain), and this in turn implies that representable elements are few. Then, assuming that all functions in a certain family are non emptying, we use repeatedly Lemma 4 to get many new representable elements, thus finding a contradiction. The key issues in the proofs are two: first, it must be possible to apply Lemma 4; second, all the new representable elements obtained applying it must be different from one another. In the following, we present some sets of conditions that are able to guarantee these two points, hence getting hypothesis for non existence of under-approximation abstract domain.

3.1   Infinite Integer Domain

As a first example, we consider the infinite domain $\mathcal{P}(\mathbb{Z})$ of integers.

Assumption 5 *We assume that an abstract domain A, to be feasible for analyses, must be at most countable.*

We make this assumption because we want to represent abstract elements with an amount of bits comparable with that of concrete *values*, to have a complexity comparable with a single concrete execution of the program and not exponentially larger. Thus, we require the size of the abstract domain to be that of $\mathbb{Z}$, the set of values handled by the program, and not the concrete domain $\mathcal{P}(\mathbb{Z})$. Many abstract domains satisfy it, for instance intervals, octagons and polyhedrons with at most n edges, for any n; some, such as general polyhedrons, don't, but they also exhibit a worst case exponential cost.

Based on Assumption 5, we prove a simple cardinality estimate that is used, as anticipated before, to prove that there are few representable elements.

Lemma 6. *For any fixed subset $S \subseteq \mathbb{Z}$, $R(S)$ is finite.*

The result for integers now shows that no under-approximation abstract domain makes all sums non emptying. The idea of the proof is to define an infinite sequence of representable elements, that is in contradiction with the previous lemma that says that R is finite. In order to define such a sequence, we want to use Lemma 4: we start from an initial representable n_0 and from a value $\bar{n}$ not representable with it, then find a non-emptying f that maps $\bar{n}$ into n_0, so that $f(\bar{n})$ is representable and we can then apply the lemma to get the new representable element $f(n_0)$. We then iterate this procedure, changing f, to build the infinite sequence. We believe the hypothesis that there exists an initial representable value is not very restrictive since initializations like x = 0 must be abstracted to $\bot$ if 0 is not representable.

Proposition 7. *Let $\langle \mathcal{P}(\mathbb{Z}) \overset{\alpha}{\rightleftarrows} A \rangle$ be an UGI, and assume that there is an integer n_0 that is representable. Then it can't be the case that all the functions of the form $f_n(x) = x + n$ are non emptying in A.*

The meaning of this proposition for program analysis is the fact that a domain small enough (by Assumption 5) is probably unable to deduce meaningful informations on an integer domain: if it doesn't contain representable singletons it must abstract to $\bot$ any variable initialization, and otherwise it can't be non emptying for all sums, hence getting $\bot$ when values are manipulated using this operation. In both cases, because of strictness, the abstract $\bot$ is propagated along program paths, yielding it as the final result of the analysis, that means exactly it can't determine any information. This issue is not bound to manifest for all programs, but for any domain there exists programs for which it does.

3.2   Finite Integer Domain

An analogous result can be obtained for a finite integer domain $\mathcal{P}([-N; N])$, where N is some big integer. This concrete domain models machine integers, that are constrained within an interval, so we assume that operations are performed in machine arithmetic, that is wrapping around in case of overflows. This is modelled working modulo $2N + 1$, the length of the interval, and taking the unique representative of each congruence class in the interval $[-N, N]$ of interest.

It is worth noting that the interval is taken symmetric around 0 to simplify notation, but there is no conceptual difficulty in using an asymmetric one.

Assumption 8 *We assume that an abstract domain A, to be feasible, must have a cardinality that is polynomial in N.*

This assumption guarantees that the number of bits required to represent an abstract element is linear in that for concrete elements so that, again, the cost of the analysis is polynomial and not exponential in that of a concrete execution.

In the following we'll use asymptotic notation for some quantities. For this to be completely formal we should define a sequence of abstract domain A_N, each one for the concrete domain $\mathcal{P}([-N, N])$, then define a sequence of values for each quantity we want to estimate, and take the limit of this sequence for N going to infinity. However we do believe all these formal details would clutter notation, making hard to get insight. For this reason, we avoid all this, just (ab)using the intuitive meaning associated with the notation.

The next lemma is analogous to Lemma 6 in proving that some sets are small under Assumption 8 on the cardinality of A.

Lemma 9. *For any fixed subset $S \subseteq \mathbb{Z}$, $|R(S)| = O(\log(N))$.*

The following proposition uses the same proof line as Proposition 7 above: we define a sequence of representable elements, and prove that they are too many since, by the previous lemma, R is quite small.

Proposition 10. *Let $\langle \mathcal{P}([-N, N]) \overset{\alpha}{\underset{}{\rightleftarrows}} A \rangle$ be an under-approximation Galois insertion, and assume that there is an integer n_0 that is representable. Then it can't be the case that all the functions of the form $f_n(x) = x + n$ (modulo $2N + 1$) are non emptying in A.*

4   Arbitrary domains

The definition of non emptying function is fully general and not limited to the concrete integer domain, hence we use it to propose conditions that are independent of the concrete domain. In this section, we deal with an infinite set C of concrete values, and an UGI $\langle \mathcal{P}(C) \overset{\alpha}{\underset{}{\rightleftarrows}} A \rangle$. Again, we take the Assumption 5 on the size of A. Under this assumption we can prove again Lemma 6, that doesn't depend on the specific integer domain considered in the previous section.

All conditions we propose in this section are mainly on the family of functions considered and not on the abstract domain. The reason for this is that first we fix a function family, corresponding to a program, and then we look for a domain well suited to analyse the specific family at hand. In other words, the family is given by the applicative context, while the domain can be adapted to it.

Definition 11 (Highly surjective function family). *Given a family F of functions from C to itself and an element $c \in C$, let*

$$P(c) = \{d \in C \mid \exists f \in F.\ f(d) = c\}$$

be the set of preimages of *c, elements of C that can be mapped to c by a function in F. We say that the family F is* highly surjective *if $P(c)$ is infinite for any possible choice of $c \in C$.*

This property is needed together with Lemma 6 to apply Lemma 4 and get a new representable element: since there are infinite preimages of c but $R(c)$ is finite, there are elements $\bar{c} \in P(c)$ not in $R(c)$; then by definition of $P(c)$ there is an f such that $f(\bar{c}) = c \in R$, so we can apply the lemma to get $f(c) \in R$. The reason for requiring $f(\bar{c}) = c$ instead of just in R is that, at the beginning of the proof, we only assume R to contain one element, hence the two conditions are equivalent. Starting from this basic idea, we present two set of sufficient conditions to prove the non existence of any under-approximation abstract domain.

4.1   Local Requirements for Impossibility

The first set of conditions we propose is in a sense more "local", in that it requires conditions on each function in the family F independently on the other.

Theorem 12. *Let F be an highly surjective function family from C to itself such that all functions $f \in F$ are either injective or acyclic. Assume also that R isn't empty. Then A can't be non emptying for all $f \in F$.*

In the previous section we developed an ad hoc proof for the family of sums over integers, but the same result can also be obtained as an application of this theorem: if $C = \mathbb{Z}$ and $F = \{\lambda x.x + n \,|\, n \in \mathbb{Z}\}$, the family is highly surjective (actually $P(c) = \mathbb{Z}$ for all c) and all these functions are injective, so it meets the hypothesis of the theorem. Another example are rational or real numbers, with sums or products

Example 13. Take $C = \mathbb{Q} \setminus \{0\}$ and $F = \{\lambda x.x \cdot q \,|\, q \in \mathbb{Q} \setminus \{0\}\}$. The family is highly surjective since $P(c) = \mathbb{Q} \setminus \{0\}$ for all c, and all these functions are invertible, hence injective.

A possibly more interesting example of application is to floating-point numbers as described by the IEEE Standard.

Example 14. Take $C = \mathcal{F} \setminus \{0\}$ the set of non-zero floating-point numbers that can be represented with a fixed number of significant digits, say t bits, but with an arbitrary precision exponent. We make the choice of infinite precision exponents and finite number of significant digits in order to have an infinite domain, as required by the theorem, but also preserve characteristics of floating-point arithmetic.

Let $\cdot$ and $\odot$ denote respectively real product and its floating-point approximation, and consider the function family $F = \{\lambda x.x \odot y \,|\, y \in C\}$. The function family is highly surjective, eg. considering that all numbers with the same significant digits as a floating-point x but different exponent can be mapped into x multiplying them by 1 times the difference of exponents. For the second condition, if $y = \pm 1$ we have that the function $\lambda x.x \odot y$ is invertible, hence injective. Otherwise, assume without loss of generality that $y > 1$ (other cases are analogous),

and by contradiction assume it has a cycle $f^n(x_0) = x_0$. By monotonicity of $\odot$ we have $f(x) = x \odot y \geq x \odot 1 = x$, hence $x_0 \leq f(x_0) \leq f^2(x_0) \leq \cdots \leq f^n(x_0) = x_0$ so all the elements of the cycle are equal, in particular $f(x_0) = x_0$. However, if $y \neq 1$, the product $x \odot y$ is never equal to x, that is a contradiction. Hence the function is acyclic. This means F meets hypothesis of Theorem 12, hence no abstract domain on floating-point numbers can be non emptying for all multiplications.

4.2   Global Requirements for Impossibility

The second set of conditions we propose is "global", in the sense that it requires the family F to satisfy a property as a whole.

Theorem 15. *Let F be an highly surjective function family from C in itself such that*

- *for all pair of elements $c, d \in C$ there exists at most a finite amount of $f \in F$ such that $f(d) = c$*
- *for all pair of an element $c \in C$ and a function $f \in F$, there exists at most a finite amount of elements $d \in C$ such that $f(d) = c$*

Assume also that R isn't empty. Then A can't be non emptying for all $f \in F$.

Again this result can be used to prove the impossibility of building an abstract domain for integers that is non emptying for all sums, or for floating-point numbers.

Example 16. Take $C = \mathcal{F} \setminus \{0\}$ the set of non-zero floating-point numbers with t bits significands and arbitrary precision exponents, and $F = \{\lambda x.x \odot y \mid y \in \mathcal{F} \setminus \{0\}\}$. As observed in Example 14 this family is highly surjective. Fixed now two floating-point numbers x, y, and letting **u** be the machine precision of floating-point arithmetic, we have that $y = f(x) = x \odot z$ only if

$$\left| \frac{y - (x \cdot z)}{x \cdot z} \right| < \mathbf{u}$$

that is

$$\left| \frac{y}{x} \right| \frac{1}{1 + \mathbf{u}} < |z| < \left| \frac{y}{x} \right| \frac{1}{1 - \mathbf{u}}$$

This is a bounded interval since $x \neq 0$, and hence contains only a finite amount of floating-point numbers. Analogously, fixed a floating-point y and a function $f(x) = x \odot z$, we have that $y = x \odot z$ only if $|x|$ belong to a bounded interval, that contains a finite amount of floating-point numbers. So, by means of Theorem 15 above, we proved again that no abstract domain on floating-point numbers can be non emptying for all multiplications.

5   On the necessity of high surjectivity hypothesis

Both sets of conditions we proposed in this section require the function family to be highly surjective. This turns out to be necessary in order to prove that no under-approximation abstract domain exists:

Proposition 17. *For any fixed family F of functions from C to itself that is not highly surjective, there exists an abstract domain A_F for $\mathcal{P}(C)$ such that*

- *A_F is finite*
- *all functions $f \in F$ are non emptying in A_F*

Moreover, the proof of this proposition is constructive, and we present an example of such construction in the following.

Example 18. Fix the pair of functions $f(x) = x - 1$ and $g(x) = x - 2$ on $\mathbb{Z}$. The family $F = \{f, g\}$ is clearly not highly surjective, so we build an under-approximation abstract domain for which these functions are non emptying. First, take an integer n_0 such that $P(n_0)$ (computed with respect to F) is finite. With this F, any integer is fine, so let us fix $n_0 = 0$.

The set of preimages of 0 is $P(0) = \{1, 2\}$. We define the abstract domain A_F as

$$A_F = \{\emptyset\} \cup \{X \cup \{0\} \mid X \subseteq P(0)\} = \{\emptyset, \{0\}, \{0, 1\}, \{0, 2\}, \{0, 1, 2\}\}$$

In this abstract domain, a set is abstracted to $\emptyset$ if and only if it doesn't contain 0 since all elements of A_F but $\emptyset$ contains 0 and the abstraction of a set must be a subset of that set.

To check that f is non emptying in A_F fix a set $S \subseteq \mathbb{Z}$. If $\alpha(S) = \emptyset$ the non emptying condition is vacuously true, so assume this is not the case, that is equivalent to $0 \in S$. Analogously, if $\alpha(f(S)) = \emptyset$ the condition is true, so assume $0 \in f(S)$ or, equivalently, $1 \in S$. Using these two we get

$$
\begin{aligned}
f^A(\alpha(S)) &= \alpha(f(\alpha(S))) && \text{[def. of } f^A] \\
&\supseteq \alpha(f(\alpha(\{0, 1\}))) && [\alpha, f \text{ monotone, } S \supseteq \{0, 1\}] \\
&= \alpha(f(\{0, 1\})) && [\alpha(\{0, 1\}) = \{0, 1\}] \\
&= \alpha(\{-1, 0\}) = \{0\} && \text{[def. of } f \text{ and } \alpha]
\end{aligned}
$$

The check for g is analogous.

Even though this proposition defines an under-approximation abstract domain, it shouldn't be interpreted as a positive result since the resulting domain is almost a power set and hence too large to be feasible in practice. Instead, the proposition should be regarded as a way to show that one of the hypothesis required in the previous theorems is tight and can't be weakened. In particular, since these kind of results need high surjectivity, they are ill suited when the focus is on a single function.

This proposition can be generalized to consider sets $S \subseteq C$ whose preimages are finite, but a little care is needed when lifting the definition of preimages to sets of values: a preimage is a set for which there exists a function that maps it to S, not the union of the preimages of elements in S:

$$P(S) = \{T \subseteq C \mid \exists f \in F.f(T) = S\}$$

Using this definition, the proposition generalizes straightforwardly:

Proposition 19. *Let F be a family of functions from C in itself, and assume there is a set $S_0 \subseteq C$ such that $P(S_0)$ is finite. Then there exists a finite abstract domain A_F for $\mathcal{P}(C)$ such that all functions $f \in F$ are non emptying in A_F.*

This proposition may for instance be applied to the concrete domain of finite lists to show that a natural function family to consider can't be used to prove non existence of under-approximation domains using non emptying functions.

Example 20. Fix the concrete domain C as the set of all lists of finite length over a finite, non-empty alphabet Γ, i.e. $C = \Gamma^*$. For $\alpha \in \Gamma^*$ a finite string, let

$$\mathrm{concat}_\alpha(\beta) = \alpha\beta$$

the function that prefix α to its argument. The family

$$F = \{\mathrm{concat}_\alpha \mid \alpha \in \Gamma^*\}$$

is not highly surjective, because fixed a string γ only its prefixes can be mapped into it by a function in F, and they are a finite amount. Hence we can define an under-approximation abstract domain for which all these functions are non emptying by means of Proposition 19. Such domains are defined with a construction similar to that of Example 18, and in particular, if ϵ is the empty list, considering the set $S_0 = \{\epsilon\}$ whose preimage is only S_0 itself, the construction yields

$$A_F = \{\emptyset, \{\epsilon\}\}$$

It's easy to check that all functions concat_α are non emptying in this abstract domain.

The previous proposition focuses on preimages, stating that if there is a concrete element that has a finite amount of them then it is possible to define an under-approximation domain. A natural dual of this proposition can be formulated in terms of images. For a subset $S \subseteq C$, the set of its images is

$$I(S) = \{f(S) \mid f \in F\}$$

This definition is exactly dual to that of preimages, and can actually be used to formulate a similar result.

Proposition 21. *Let F be a family of total functions (ie. if $S \neq \emptyset$ then $f(S) \neq \emptyset$) from $\mathcal{P}(C)$ in itself, and assume there is a non empty set $S_0 \subseteq C$ such that $I(S_0)$ is finite. Then there exists a finite abstract domain A_F such that all functions $f \in F$ are non emptying in A_F.*

Even though this proposition introduces the technical hypothesis that all $f \in F$ are total, we don't believe this to be very restrictive because these theorems are intended to be applied when F is a family of basic transfer functions, that seldom introduce divergence: in programming languages this is often caused by control-flow constructs. An application of this proposition is again on lists, to rule out another natural function family.

Example 22. Fix again $C = \Gamma^*$, and consider functions $\mathrm{drop}_n : \Gamma^* \to \Gamma^*$ that, taken a list, drop its first n elements and return the resulting list. If the input list is shorter than n, the output of drop_n is the empty list ϵ. The function family

$$F = \{\mathrm{drop}_n \,|\, n \in \mathbb{N}\}$$

is highly surjective since, for any fixed list $\alpha \in \Gamma^*$ and any n, we can extend α with any n character, and map this list to α with drop_n. However, images through this function family are finite:

$$I(\alpha) = \{\mathrm{drop}_n(\alpha) \,|\, n \in \mathbb{N}\}$$

that is finite since it's the set of all tails of α. Hence by Proposition 21 we can define an under-approximation abstract domain such that all functions drop_n are non emptying. Again, these domains are constructed from sets S_0 with a finite amount of images, and considering $S_0 = \{\epsilon\}$, that satisfies $I(S_0) = \{\epsilon\}$, it yields

$$A_F = \{\emptyset, \{\epsilon\}\}$$

Again it can be easily checked that all functions drop_n are non emptying in A_F.

These two propositions consider opposite situations in which it is possible to define an under-approximation domain: the former requires to be able to go backward using F in infinitely many ways, while the latter to go forward. This often isn't the case in the presence of "boundaries" in the concrete domain, that are points with respect to which functions tend to walk either up or away: for instance, ϵ is such a point with finite strings because concat functions go away from it while drop go towards. Another example of such boundary is 0 in the domain of integers $\mathbb{Z}$ with respect to multiplications and (rounded) divisions: the former increase absolute value, moving away from 0 (even though 0 itself is never a preimage), while the latter decrease it. Also considering a function family made of both kind of functions doesn't work: a slight adaptation of the constructions for the two propositions above shows that, if F can be partitioned in two subfamilies, each satisfying the hypothesis of one of the two propositions, then there exists an under-approximation abstract domain. An example of this is in the set of finite lists, taking as F both concat and drop functions. The construction then yields exactly $A_F = \{\emptyset, \{\epsilon\}\}$, for which all these functions are non emptying, as shown in Examples 20 and 22. In light of these observations, in order to apply effectively the definition of non emptying function to prove non existence of abstract domains, for all possible boundaries there is the need for a function that is able to both enter and exit it. This happens for integers, since there is no boundary, but doesn't for finite lists, with $\{\epsilon\}$ being often either a sink or a source for many functions on lists.

6   Conclusions and Future Works

Until recently, the focus of formal static analyses has been on over-approximation to prove program correctness, but many tools based on this theory are instead deployed to catch bugs [23,10]. Incorrectness Logic promoted the study of a theory for under-approximation to give a formal basis to a new class of tools. This has seldom been done in the last few decades, especially in the framework of Abstract Interpretation. In our work, we point out some asymmetries between over- and under-approximation in Abstract Interpretation, and why those are an obstacle to the design of abstract domains. We have identified functions as the main difference, because they remain the same in both over- and under-approximation thus preventing one theory to be obtained simply as a dual of the other. Handling of divergence is another critical issue. Building on those ideas, we have proposed the new (to the extent of our knowledge) definition of *non emptying function* and studied how it can be used to prove non existence of under-approximation abstract domains. We have presented some general results, and applied them to integer and floating point domains to conclude that, under some assumptions, there are no useful under-approximation domains. Then, we have found conditions under which there do exist under-approximation abstract domains, showing that some of the hypothesis required in our theorems are very tight. However, because of the scarcity of works in this direction, we believe there are many possible subjects for future research.

Under-approximation abstract domains must be closed under union, but known abstract domains are rarely such. However disjunctive completion [11], a known domain transformer, refines any abstract domain in a union-closed one. This has been studied for over-approximation in order to improve precision at the expense of increased complexity. A solution to keep the analysis feasible is to use heuristics to prune disjunctions, trading back complexity for precision, but making the analysis possible for under-approximations. Moreover, practical tools based on the theory of Incorrectness Logic already use heuristic to drop logical disjunctions [19], so taking inspiration from them may be effective also for Abstract Interpretation.

In their recent work, Raad et al. [20] study incorrectness separation logic, the join of separation logic [21] and Incorrectness Logic. They notice that the original separation logic doesn't distinguish a pointer known to be dangling from one about which it has no information, and they introduce a new kind of heap assertion for dangling pointers. This issue is reminiscent of the difference between divergence and no information we incur into in Abstract Interpretation. This may suggest the introduction of a similar distinction also in under-approximation domains, but a new point different from $\perp$ describing divergence needs a concretization, and no such element exists in a power set other than $\emptyset$. However, in Abstract Interpretation it happens at times that more general concrete domains allow more flexibility in the abstraction (eg. as proposed for higher-order functional languages [5]), so it may be worth to investigate the possibility to change the concrete domain to account for this new point.

All our results depend on the existence of a representable value. This assumption is motivated by the analysis performed, but is not a requirement of Abstract Interpretation itself. A way to remove this hypothesis may be to consider representable sets of minimal cardinality because functions defined as additive extensions don't increase cardinality, so they might take the place of singletons. The technical issue is if and how Lemma 4 can be generalized, but we believe it may be possible to relax that hypothesis about singletons.

We have discussed the finite domain of integers at the end of Section 3, but all our general results deal with infinite concrete domains. Both theorems rely on cardinality estimates essentially based on the fact that arbitrary combinations of finite numbers is still finite, hence less than the cardinality of the concrete domain. However, with a finite concrete domain those would be replaced by combinations of logarithmic factors, which may become equal to the size of the concrete domain. For finite domains we can prove a result reminiscent of Theorem 15, but this topic requires thorough investigation to understand the new issues and possibilities they open up.

References

1. Boulanger, J.L. (ed.): Static Analysis of Software: The Abstract Interpretation. Wiley (2011)
2. Bourdoncle, F.: Abstract debugging of higher-order imperative languages. SIGPLAN Not. **28**(6), 46–55 (Jun 1993). https://doi.org/10.1145/173262.155095
3. Calcagno, C., Distefano, D., Dubreil, J., Gabi, D., Hooimeijer, P., Luca, M., O'Hearn, P.W., Papakonstantinou, I., Purbrick, J., Rodriguez, D.: Moving fast with software verification. In: Proc. NFM'15. LNCS, vol. 9058, pp. 3–11. Springer (2015). https://doi.org/10.1007/978-3-319-17524-9_1
4. Cousot, P.: Principles of Abstract Interpretation. MIT Press (2021)
5. Cousot, P., Cousot, R.: Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and per analysis of functional languages). In: Proceedings of 1994 IEEE International Conference on Computer Languages (ICCL'94). pp. 95–112 (1994). https://doi.org/10.1109/ICCL.1994.288389
6. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. p. 238–252. POPL '77, Association for Computing Machinery, New York, NY, USA (1977). https://doi.org/10.1145/512950.512973
7. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: Proceedings of the 6th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. p. 269–282. POPL '79, Association for Computing Machinery, New York, NY, USA (1979). https://doi.org/10.1145/567752.567778

8. Cousot, P., Cousot, R., Fähndrich, M., Logozzo, F.: Automatic inference of necessary preconditions. In: Giacobazzi, R., Berdine, J., Mastroeni, I. (eds.) Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7737, pp. 128–148. Springer (2013). https://doi.org/10.1007/978-3-642-35873-9_10

9. Cousot, P., Cousot, R., Logozzo, F.: Precondition inference from intermittent assertions and application to contracts on collections. In: Jhala, R., Schmidt, D.A. (eds.) Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6538, pp. 150–168. Springer (2011). https://doi.org/10.1007/978-3-642-18275-4_12

10. Distefano, D., Fähndrich, M., Logozzo, F., O'Hearn, P.W.: Scaling static analyses at Facebook. Commun. ACM **62**(8), 62–70 (2019). https://doi.org/10.1145/3338112

11. Filé, G., Ranzato, F.: Improving abstract interpretations by systematic lifting to the powerset. In: Proceedings of the 1994 International Symposium on Logic Programming. p. 655–669. ILPS '94, MIT Press, Cambridge, MA, USA (1994)

12. Floyd, R.W.: Assigning meanings to programs. Proceedings of Symposium on Applied Mathematics **19**, 19–32 (1967)

13. Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM **12**(10), 576–580 (Oct 1969). https://doi.org/10.1145/363235.363259

14. Lev-Ami, T., Sagiv, M., Reps, T., Gulwani, S.: Backward analysis for inferring quantified preconditions. Tr-2007-12-01, Tel Aviv University (2007)

15. Miné, A.: Backward under-approximations in numeric abstract domains to automatically infer sufficient program conditions. Sci. Comput. Program. **93**, 154–182 (Nov 2014). https://doi.org/10.1016/j.scico.2013.09.014

16. Miné, A.: Tutorial on static inference of numeric invariants by abstract interpretation. Found. Trends Program. Lang. **4**(3–4), 120–372 (Dec 2017). https://doi.org/10.1561/2500000034

17. Nielson, F., Nielson, H., Hankin, C.: Principles of Program Analysis. Springer (2010). https://doi.org/10.1007/978-3-662-03811-6

18. O'Hearn, P.W.: Continuous reasoning: Scaling the impact of formal methods. In: Proc. LICS'18. p. 13–25. ACM (2018). https://doi.org/10.1145/3209108.3209109

19. O'Hearn, P.W.: Incorrectness logic. Proc. ACM Program. Lang. **4**(POPL) (Dec 2019). https://doi.org/10.1145/3371078

20. Raad, A., Berdine, J., Dang, H.H., Dreyer, D., O'Hearn, P.W., Villard, J.: Local reasoning about the presence of bugs: Incorrectness separation logic. In: Lahiri, S.K., Wang, C. (eds.) Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12225, pp. 225–252. Springer (2020). https://doi.org/10.1007/978-3-030-53291-8_14

21. Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: 17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings. pp. 55–74. IEEE Computer Society (2002). https://doi.org/10.1109/LICS.2002.1029817

22. Rival, X., Yi, K.: Introduction to Static Analysis – An Abstract Interpretation Perspective. MIT Press (2020)

23. Sadowski, C., Aftandilian, E., Eagle, A., Miller-Cushon, L., Jaspan, C.: Lessons from building static analysis tools at Google. Commun. ACM **61**(4), 58–66 (Mar 2018). https://doi.org/10.1145/3188720

24. Schmidt, D.A.: A calculus of logical relations for over- and underapproximating static analyses. Sci. Comput. Program. **64**(1), 29–53 (2007). https://doi.org/10.1016/j.scico.2006.03.008

On probability-raising causality
in Markov decision processes [*]

Christel Baier ✉[iD], Florian Funke [iD], Jakob Piribauer ✉[iD], and Robin Ziemek ✉[iD]

Technische Universität Dresden
{christel.baier, florian.funke,
jakob.piribauer,robin.ziemek}@tu-dresden.de

Abstract. The purpose of this paper is to introduce a notion of causality in Markov decision processes based on the probability-raising principle and to analyze its algorithmic properties. The latter includes algorithms for checking cause-effect relationships and the existence of probability-raising causes for given effect scenarios. Inspired by concepts of statistical analysis, we study quality measures (recall, coverage ratio and f-score) for causes and develop algorithms for their computation. Finally, the computational complexity for finding optimal causes with respect to these measures is analyzed.

1 Introduction

As modern software systems control more and more aspects of our everyday lives, they grow increasingly complex. Even small changes to a system might cause undesired or even disastrous behavior. Therefore, the goal of modern computer science does not only lie in the development of powerful and versatile systems, but also in providing comprehensive techniques to understand these systems. In the area of formal verification, counterexamples, invariants and related certificates are often used to provide a verifiable justification that a system does or does not behave according to a specification (see e.g., [30,16,32]). These, however, provide only elementary insights on the system behavior. Thus, there is a growing demand for a deeper understanding on *why* a system satisfies or violates a specification and *how* different components influence the performance. The analysis of causal relations between events occurring during the execution of a system can lead to such understanding. The majority of prior work in this direction relies on causality notions based on Lewis' counterfactual principle [29] stating the effect would not have occurred if the cause would not have happened. A prominent formalization of the counterfactual principle is given by Halpern and Pearl [21] via structural equation models. This inspired formal definitions of causality and related notions of blameworthiness and responsibility in Kripke and game structures (see, e.g., [15,11,14,40,19,41,7]).

In this work, we approach the concept of causality in a probabilistic setting, where we focus on the widely accepted *probability-raising principle* which has its roots in

Table 1. Complexity results for MDPs and Markov chains (MC) with fixed effect set

| | for fixed set Cause | | find optimal cause | |
	check PR condition	compute quality values (recall, covratio, f-score)	covratio-optimal = recall-optimal	f-score-optimal
SPR	$\in P$	poly-time	poly-time	poly-space poly-time for MC threshold problem $\in NP \cap coNP$
GPR	$\in PSPACE$ and $\in P$ for MC	poly-time	poly-space threshold problems $\in PSPACE$ and NP-hard and NP-complete for MC	

philosophy [38,39,18,22] and has been refined by Pearl [35] for causal and probabilistic reasoning in intelligent systems. The different notions of probability-raising cause-effect relations discussed in the literature share the following two main principles:

(C1) Causes raise the probabilities for their effects, informally expressed by the requirement "$\Pr(\text{effect}|\text{cause}) > \Pr(\text{effect})$".
(C2) Causes must happen before their effects.

Despite the huge amount of work on probabilistic causation in other disciplines, research on probability-raising causes in the context of formal methods is comparably rare and has concentrated on Markov chains (see, e.g., [24,25,6] and the discussion of related work in Section 3.2). To the best of our knowledge, probabilistic causation for probabilistic operational models with nondeterminism has not been studied before.

We formalize the principles (C1) and (C2) for Markov decision processes (MDPs), a standard operational model combining probabilistic and non-deterministic behavior, and concentrate on reachability properties where both cause and effect are given as sets of states. Condition (C1) can be interpreted in two natural ways in this setting: On one hand, the probability-raising property can be locally required for each element of the cause. Such causes are called *strict probability-raising (SPR) causes* in our framework. This interpretation is especially suited when the task is to identify system states that have to be avoided for lowering the effect probability. On the other hand, one might want to treat the cause set globally as a unit in (C1) leading to the notion of *global probability-raising (GPR) cause*. Considering the cause set as a whole is better suited when further constraints are imposed on the candidates for cause set. This might apply, e.g., when the set of non-terminal states of the given MDP is partitioned into sets of states S_i under the control of an agent i, $1 \leqslant i \leqslant k$. For the task to identify which agent's decisions cause the effect only the subsets of $S_1, \ldots, S_k$ are candidates for causes. Furthermore, global causes are more appropriate when causes are used for monitoring purposes under partial observability constraints as then the cause candidates are sets of indistinguishable states.

Different causes for an effect according to our definition can differ substantially regarding how well they predict the effect and how well the executions exhibiting the cause cover the executions showing the effect. Taking inspiration from measures used in statistical analysis (see, e.g., [36]), we introduce quality measures that allow us to compare causes and to look for optimal causes: The *recall* captures the probability that the effect is indeed preceded by the cause. The *coverage-ratio* quantifies the fraction of

the probability that cause and effect are observed and the probability that the effect but not the cause is observed. Finally, the *f-score*, a widely used quality measure for binary classifiers, is the harmonic mean of recall and precision, i.e., the probability that the cause is followed by the effect.

Contributions. The goal of this work are the mathematical and algorithmic foundations of probabilistic causation in MDPs based on (C1) and (C2). We introduce strict and global probability-raising causes in MDPs (Section 3). Algorithms are provided to check whether given cause and effect sets satisfy (one of) the probability-raising conditions (Section 4.1 and 4.2) and to check the existence of causes for a given effect (Section 4.1). In order to evaluate the coverage properties of a cause, we subsequently introduce the above-mentioned quality measures (Section 5.1). We give algorithms for computing these values for given cause-effect relations (Section 5.2) and characterize the computational complexity of finding optimal causes with respect to the different measures (Section 5.3). Table 1 summarizes our complexity results. An extended version of this paper containing the omitted proofs can be found in [8].

2   Preliminaries

Throughout the paper, we will assume some familiarity with basic concepts of Markov decision processes. Here, we only present a brief summary of the notations used in the paper. For more details, we refer to [37,9,23].

A *Markov decision process (MDP)* is a tuple $\mathcal{M} = (S, Act, P, \text{init})$ where S is a finite set of states, Act a finite set of actions, $\text{init} \in S$ the initial state and $P : S \times Act \times S \to [0,1]$ the transition probability function such that $\sum_{t \in S} P(s, \alpha, t) \in \{0,1\}$ for all states $s \in S$ and actions $\alpha \in Act$. An action α is *enabled* in state $s \in S$ if $\sum_{t \in S} P(s, \alpha, t) = 1$. We define $Act(s) = \{\alpha \mid \alpha \text{ is enabled in } s\}$. A state t is *terminal* if $Act(t) = \emptyset$. A Markov chain (MC) is a special case of an MDP where Act is a singleton (we then write $P(s, u)$ rather than $P(s, \alpha, u)$). A *path* in an MDP $\mathcal{M}$ is a (finite or infinite) alternating sequence $\pi = s_0 \alpha_0 s_1 \alpha_1 s_2 \cdots \in (S \times Act)^* \cup (S \times Act)^\omega$ such that $P(s_i, \alpha_i, s_{i+1}) > 0$ for all indices i. A path is called maximal if it is infinite or finite and ends in a terminal state. An MDP can be interpreted as a Kripke structure in which transitions go from states to probability distributions over states.

A *(randomized) scheduler* $\mathfrak{S}$ is a function that maps each finite non-maximal path $s_0 \alpha_0 \ldots \alpha_{n-1} s_n$ to a distribution over $Act(s_n)$. $\mathfrak{S}$ is called deterministic if $\mathfrak{S}(\pi)$ is a Dirac distribution for all finite non-maximal paths π. If the chosen action only depends on the last state of the path, $\mathfrak{S}$ is called *memoryless*. We write MR for the class of memoryless (randomized) and MD for the class of memoryless deterministic schedulers. *Finite-memory* schedulers are those that are representable by a finite-state automaton.

The scheduler $\mathfrak{S}$ of $\mathcal{M}$ induces a (possibly infinite) Markov chain. We write $\Pr^{\mathfrak{S}}_{\mathcal{M},s}$ for the standard probability measure on measurable sets of maximal paths in the Markov chain induced by $\mathfrak{S}$ with initial state s. If φ is a measurable set of maximal paths, then $\Pr^{\max}_{\mathcal{M},s}(\varphi)$ and $\Pr^{\min}_{\mathcal{M},s}(\varphi)$ denote the supremum resp. infimum of the probabilities for φ under all schedulers. We use the abbreviation $\Pr^{\mathfrak{S}}_{\mathcal{M}} = \Pr^{\mathfrak{S}}_{\mathcal{M},\text{init}}$ and notations $\Pr^{\max}_{\mathcal{M}}$ and $\Pr^{\min}_{\mathcal{M}}$ for extremal probabilities. Analogous notations will be used for expectations. So, if f is a random variable, then, e.g., $E^{\mathfrak{S}}_{\mathcal{M}}(f)$ denotes the expectation of f under $\mathfrak{S}$ and

$E_{\mathcal{M}}^{\max}(f)$ its supremum over all schedulers. We use LTL-like temporal modalities such as $\Diamond$ (eventually) and U (until) to denote path properties. For $X, T \subseteq S$ the formula $X \cup T$ is satisfied by paths $\pi = s_0 s_1 \ldots$ such that there exists $j \geqslant 0$ such that for all $i < j : s_i \in X$ and $s_j \in T$ and $\Diamond T = S \cup T$. It is well-known that $\Pr_{\mathcal{M}}^{\min}(X \cup T)$ and $\Pr_{\mathcal{M}}^{\max}(X \cup T)$ and corresponding optimal MD-schedulers are computable in polynomial time.

If $s \in S$ and $\alpha \in Act(s)$, then (s, α) is said to be a state-action pair of $\mathcal{M}$. An *end component* (EC) of an MDP $\mathcal{M}$ is a strongly connected sub-MDP containing at least one state-action pair. ECs will be often identified with the set of their state-action pairs. An EC $\mathcal{E}$ is called maximal (abbreviated MEC) if there is no proper superset $\mathcal{E}'$ of (the set of state-action pairs of) $\mathcal{E}$ which is an EC.

3 Strict and global probability-raising causes

We now provide formal definitions for cause-effect relations in MDPs which rely on the probability-raising (PR) principle as stated by (C1) and (C2) in the introduction. We focus on the case where both causes and effects are state properties, i.e., sets of states.

In the sequel, let $\mathcal{M} = (S, Act, P, init)$ be an MDP and $\mathsf{Eff} \subseteq S \setminus \{init\}$ a nonempty set of terminal states. (As the effect set is fixed, for the analysis of cause-effect relationships in $\mathcal{M}$ it suffices to assume all effect states are terminal by (C2).) Furthermore, we may assume that every state $s \in S$ is reachable from init.

We consider here two variants of the probability-raising condition: the global setting treats the set Cause as a unit, while the strict view requires the probability-raising condition for all states in Cause individually.

Definition 1 (Global and strict probability-raising cause (GPR/SPR cause)). *Let* $\mathcal{M}$ *and* Eff *be as above and* Cause *a nonempty subset of* $S \setminus \mathsf{Eff}$. *Then,* Cause *is said to be a GPR cause for* Eff *iff the following two conditions (G) and (M) hold:*

(G) For each scheduler $\mathfrak{S}$ *where* $\Pr_{\mathcal{M}}^{\mathfrak{S}}(\Diamond \mathsf{Cause}) > 0$:

$$\Pr_{\mathcal{M}}^{\mathfrak{S}}(\,\Diamond \mathsf{Eff} \mid \Diamond \mathsf{Cause}\,) \;>\; \Pr_{\mathcal{M}}^{\mathfrak{S}}(\Diamond \mathsf{Eff}). \qquad\qquad \text{(GPR)}$$

(M) For each $c \in \mathsf{Cause}$, *there is a scheduler* $\mathfrak{S}$ *with* $\Pr_{\mathcal{M}}^{\mathfrak{S}}((\neg\mathsf{Cause}) \cup c) > 0$.

Cause *is called an* SPR cause *for* Eff *iff (M) and the following condition (S) hold:*

(S) For each state $c \in \mathsf{Cause}$ *and each scheduler* $\mathfrak{S}$ *where* $\Pr_{\mathcal{M}}^{\mathfrak{S}}((\neg\mathsf{Cause}) \cup c) > 0$:

$$\Pr_{\mathcal{M}}^{\mathfrak{S}}(\,\Diamond \mathsf{Eff} \mid (\neg\mathsf{Cause}) \cup c\,) \;>\; \Pr_{\mathcal{M}}^{\mathfrak{S}}(\Diamond \mathsf{Eff}). \qquad\qquad \text{(SPR)}$$

Condition (M) can be seen as a minimality requirement as states $c \in \mathsf{Cause}$ which are not accessible from init without traversing other states in Cause could be omitted without affecting the true positives (events where an effect state is reached after visiting a cause state, "covered effects") or false negatives (events where an effect state is reached without visiting a cause state before, "uncovered effect"). More concretely, whenever a set $C \subseteq S \setminus \mathsf{Eff}$ satisfies conditions (G) or (S) then the set Cause of states $c \in C$ where $\mathcal{M}$ has a path from init satisfying $(\neg C) \cup c$ is a GPR resp. an SPR cause.

3.1   Examples and simple properties of probability-raising causes

We first observe that SPR/GPR causes cannot contain the initial state init, since otherwise an equality instead of an inequality would hold in (GPR) and (SPR). Furthermore as a direct consequence of the definitions and using the equivalence of the LTL formulas $\Diamond$Cause and $(\neg$Cause$)\,\mathsf{U}\,$Cause we obtain:

Lemma 1 (Singleton PR causes). *If* Cause *is a singleton then* Cause *is a SPR cause for* Eff *if and only if* Cause *is a GPR cause for* Eff.

As the event $\Diamond$Cause is a disjoint union of all events $(\neg$Cause$)\,\mathsf{U}\,c$ with $c \in$ Cause, the probability for covered effects $\mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}}(\,\Diamond$Eff $|\,\Diamond$Cause $)$ is a weighted average of the probabilities $\mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}}(\,\Diamond$Eff $|\,(\neg$Cause$)\,\mathsf{U}\,c\,)$ for $c \in$ Cause. This yields:

Lemma 2 (Strict implies global). *Every SPR cause for* Eff *is a GPR cause for* Eff.

Example 1 (Non-strict GPR cause). Consider the Markov chain $\mathcal{M}$ depicted below where the nodes represent states and the directed edges represent transitions labeled with their respective probabilities. Let Eff $= \{$eff$\}$. Then, $\mathrm{Pr}_{\mathcal{M}}(\Diamond$Eff$) = \frac{1}{3} + \frac{1}{3}\cdot\frac{1}{4} + \frac{1}{12} = \frac{1}{2}$, $\mathrm{Pr}_{\mathcal{M}}(\Diamond$Eff$|\Diamond c_1) = \mathrm{Pr}_{\mathcal{M},c_1}(\Diamond$eff$) = 1$ and $\mathrm{Pr}_{\mathcal{M}}(\Diamond$Eff$|\Diamond c_2) = \mathrm{Pr}_{\mathcal{M},c_2}(\Diamond$eff$) = \frac{1}{4}$. Thus, $\{c_1\}$ is both an SPR and a GPR cause for Eff, while $\{c_2\}$ is not. The set Cause $= \{c_1, c_2\}$ is a non-strict GPR cause for Eff as:

$$\mathrm{Pr}_{\mathcal{M}}(\,\Diamond\mathsf{Eff} \mid \Diamond\mathsf{Cause}\,) = (\tfrac{1}{3} + \tfrac{1}{3}\cdot\tfrac{1}{4})/(\tfrac{1}{3} + \tfrac{1}{3}) = (\tfrac{5}{12})/(\tfrac{2}{3}) = \tfrac{5}{8} > \tfrac{1}{2} = \mathrm{Pr}_{\mathcal{M}}(\Diamond\mathsf{Eff}).$$

The second condition (M) is obviously fulfilled. Non-strictness follows from the fact that the SPR condition does not hold for state c_2. $\triangleleft$
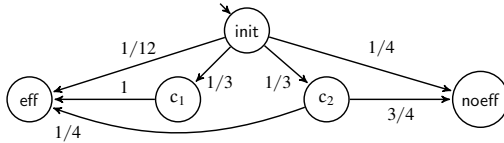
Example 2 (Probability-raising causes might not exist). PR causes might not exist, even if $\mathcal{M}$ is a Markov chain. This applies, e.g., to the Markov chain $\mathcal{M}$ with two states init and eff where $\mathsf{P}(\mathsf{init},\mathsf{eff}) = 1$ and the effect set Eff $= \{$eff$\}$. The only cause candidate is the singleton $\{$init$\}$. However, the strict inequality in (GPR) or (SPR) does not hold for Cause $= \{$init$\}$. The same phenomenon occurs if all non-terminal states of a Markov chain reach the effect states with the same probability. In such cases, however, the non-existence of PR causes is well justified as the events $\Diamond$Eff and $\Diamond$Cause are stochastically independent for every set Cause $\subseteq S \setminus$ Eff. $\triangleleft$

Remark 1 (Memory needed for refuting PR condition). Let $\mathcal{M}$ be the MDP in Figure 1, where the notation is similar to Example 1 with the addition of actions α, β and γ. Let Cause $= \{c\}$ and Eff $= \{$eff$\}$. Only state s has a nondeterministic choice. Cause is not an PR cause. To see this, regard the deterministic scheduler $\mathfrak{T}$ that schedules β only for the first visit of s and α for the second visit of s. Then:

$$\mathrm{Pr}_{\mathcal{M}}^{\mathfrak{T}}(\Diamond\mathsf{eff}) = \tfrac{1}{2}\cdot\tfrac{1}{2} + \tfrac{1}{2}\cdot\tfrac{1}{2}\cdot 1 \cdot \tfrac{1}{4} = \tfrac{5}{16} > \tfrac{1}{4} = \mathrm{Pr}_{\mathcal{M}}^{\mathfrak{T}}(\Diamond\mathsf{eff}|\Diamond c)$$
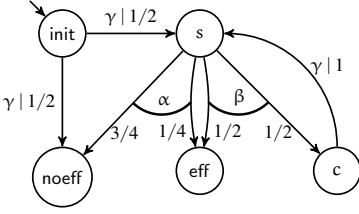
Fig. 1. MDP $\mathcal{M}$ from Remark 1
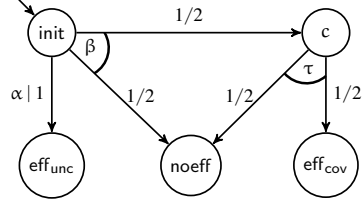
Fig. 2. MDP $\mathcal{M}$ from Remark 2

Denote the MR schedulers reaching c with positive probability as $\mathfrak{S}_\lambda$ with $\mathfrak{S}_\lambda(s)(\alpha) = \lambda$ and $\mathfrak{S}_\lambda(s)(\beta) = 1-\lambda$ for some $\lambda \in [0,1[$. Then, $\mathrm{Pr}_{\mathcal{M},s}^{\mathfrak{S}_\lambda}(\Diamond\mathsf{eff}) > 0$ and:

$$\mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}_\lambda}(\Diamond\mathsf{eff}) \;=\; \tfrac{1}{2}\cdot\mathrm{Pr}_{\mathcal{M},s}^{\mathfrak{S}_\lambda}(\Diamond\mathsf{eff}) \;<\; \mathrm{Pr}_{\mathcal{M},s}^{\mathfrak{S}_\lambda}(\Diamond\mathsf{eff}) \;=\; \mathrm{Pr}_{\mathcal{M},c}^{\mathfrak{S}_\lambda}(\Diamond\mathsf{eff}) \;=\; \mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}_\lambda}(\Diamond\mathsf{eff}|\Diamond c)$$

Thus, the SPR/GPR condition holds for Cause and Eff under all memoryless schedulers reaching Cause with positive probability, although Cause is not an PR cause. ◁

Remark 2 (Randomization needed for refuting PR condition). Consider the MDP $\mathcal{M}$ of Figure 2. Let $\mathsf{Eff} = \{\mathsf{eff}_{unc}, \mathsf{eff}_{cov}\}$ and $\mathsf{Cause} = \{c\}$. The two MD-schedulers $\mathfrak{S}_\alpha$ and $\mathfrak{S}_\beta$ that select α resp. β for the initial state init are the only deterministic schedulers. As $\mathfrak{S}_\alpha$ does not reach c, it is irrelevant for the SPR or GPR condition. $\mathfrak{S}_\beta$ satisfies (SPR) and (GPR) as $\mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}_\beta}(\Diamond\mathsf{Eff}|\Diamond c) = \tfrac{1}{2} > \tfrac{1}{4} = \mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}_\beta}(\Diamond\mathsf{Eff})$. The MR scheduler $\mathfrak{T}$ which selects α and β with probability $\tfrac{1}{2}$ in init reaches c with positive probability and violates (SPR) and (GPR) as $\mathrm{Pr}_{\mathcal{M}}^{\mathfrak{T}}(\Diamond\mathsf{Eff}|\Diamond c) = \tfrac{1}{2} < \tfrac{5}{8} = \tfrac{1}{2} + \tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2} = \mathrm{Pr}_{\mathcal{M}}^{\mathfrak{T}}(\Diamond\mathsf{Eff})$. ◁

Remark 3 (Cause-effect relations for regular classes of schedulers). The definitions of PR causes in MDPs impose constraints for all schedulers reaching a cause state. This condition is fairly strong and might lead to the phenomenon that no PR cause exists. However, replacing $\mathcal{M}$ with an MDP resulting from the synchronous parallel composition of $\mathcal{M}$ with a deterministic finite automaton representing a regular constraint on the scheduled state-action sequences (e.g., "alternate between actions α and β in state s" or "take α on every third visit to state s and actions β or γ otherwise") leads to a weaker notion of PR causality. This can be useful to obtain more detailed information on cause-effect relationships in special scenarios. For example at design time where multiple scenarios (regular classes of schedulers) are considered or for a post-hoc analysis. For the later, one seeks causes of an occurred effect and the information about the scheduled actions is either extractable from log files or gathered by a monitor. ◁

Remark 4 (Action causality and other forms of PR causality). Our notions of PR causes are purely state-based with conditions that compare probabilities under the same scheduler. However, in combination with model transformations, the proposed notions are also applicable for reasoning about other forms of PR causality.

Suppose, the task is to check whether taking action α in state s raises the effect probabilities compared to never scheduling α in state s. Let $\mathcal{M}_0$ and $\mathcal{M}_1$ be copies of $\mathcal{M}$ with the following modifications: In $\mathcal{M}_0$, the only enabled action of state s is α, while

in $\mathcal{M}_1$ the enabled actions of state s are the elements of $Act_{\mathcal{M}}(s) \setminus \{\alpha\}$. Let now $\mathcal{N}$ be the MDP whose initial state has a single enabled action and moves with probability $1/2$ to $\mathcal{M}_0$ and $\mathcal{M}_1$. Then, action α raises the effect probability in $\mathcal{M}$ iff the initial state of $\mathcal{M}_0$ consitutes an SPR cause in $\mathcal{N}$. This idea can be generalized to check whether scheduler classes satisfying a regular constraint have higher effect probability compared to all other schedulers. In this case, we can deal with an MDP $\mathcal{N}$ as above where $\mathcal{M}_0$ and $\mathcal{M}_1$ are defined as the synchronous product of deterministic finite automata and $\mathcal{M}$.   ◁

3.2   Related work

Previous work in the direction of probabilistic causation in stochastic operational models has mainly concentrated on Markov chains. Kleinberg [24,25] introduced *prima facie causes* in finite Markov chains where both causes and effects are formalized as PCTL state formulae, and thus they can be seen as sets of states as in our approach. The correspondence of Kleinberg's PCTL constraints for prima facie causes and the strict probability-raising condition formalized using conditional probabilities has been worked out in the survey article [5]. Our notion of SPR causes corresponds to Kleinberg's prima facie causes, except for the minimality condition (M). Ábrahám et al [1] introduces a hyperlogic for Markov chains and gives a formalization of probabilistic causation in Markov chains as a hyperproperty, which is consistent with Kleinberg's prima facie causes, and with SPR causes up to minimality. Cause-effect relations in Markov chains where effects are ω-regular properties have been introduced in [6]. The notion relies on the strict probability-raising condition, but requires completeness in the sense that every path where the effect occurs has a prefix in the cause set. The paper [6] permits a non-strict inequality in the SPR condition with the consequence that causes always exist, which is not the case for our notions.

The survey article [5] introduces notions of global probability-raising causes for Markov chains where causes and effects can be path properties. [5]'s notion of *reachability causes* in Markov chains directly corresponds to our notion GPR causes, the only difference being that [5] deals with a relaxed minimality condition and requires that the cause set is reachable without visiting an effect state before. The latter is inherent in our approach as we suppose that all states are reachable and the effect states are terminal.

To the best of our knowledge, probabilistic causation in MDPs has not been studied before. The only work in this direction we are aware of is the recent paper by Dimitrova et al [17] on a hyperlogic, called PHL, for MDPs. While the paper focuses on the foundation of PHL, it contains an example illustrating how action causality can be formalized as a PHL formula. Roughly, the presented formula expresses that taking a specific action α increases the probability for reaching effect states. Thus, it also relies on the probability-raising principle, but compares the "effect probabilities" under different schedulers (which either schedule α or not) rather than comparing probabilities under the same scheduler as in our PR condition. However, as Remark 4 argues, to some extent our notions of PR causes can reason about action causality as well.

There has also been work on causality-based explanations of counterexamples in probabilistic models [27,28]. The underlying causality notion of this work, however, relies on the non-probabilistic counterfactual principle rather than the probability-raising

condition. The same applies to the notions of forward and backward responsibility in stochastic games in extensive form introduced in the recent work [7].

4   Checking the existence of PR causes and the PR conditions

We now turn to algorithms for checking whether a given set Cause is an SPR or GPR cause for Eff. As condition (M) of SPR and GPR causes is verifiable by standard model checking techniques in polynomial time, we concentrate on checking the probability-raising conditions (SPR) and (GPR). For Markov chains, both (SPR) and (GPR) can be checked in polynomial time by computing the corresponding probabilities. So, the interesting case is checking the PR conditions in MDPs.

 We start by stating that for the SPR and GPR condition, it suffices to consider schedulers minimizing the probability to reach an effect state from every cause state.

Notation 1 (MDP with minimal effect probabilities from cause candidates). If $C \subseteq$ S then we write $\mathcal{M}_{[C]}$ for the MDP resulting from $\mathcal{M}$ by removing all enabled actions of the states in C. Instead, $\mathcal{M}_{[C]}$ has a new action γ that is enabled exactly in the states $s \in C$ with the transition probabilities $P_{\mathcal{M}_{[C]}}(s, \gamma, \mathsf{eff}) = \mathrm{Pr}^{\min}_{\mathcal{M},s}(\Diamond\mathsf{Eff})$ and $P_{\mathcal{M}_{[C]}}(s, \gamma, \mathsf{noeff}) = 1 - \mathrm{Pr}^{\min}_{\mathcal{M},s}(\Diamond\mathsf{Eff})$. Here, eff is a fixed state in Eff and noeff a (possibly fresh) terminal state not in Eff. We write $\mathcal{M}_{[c]}$ if $C = \{c\}$ is a singleton.

Lemma 3. *Let* $\mathcal{M} = (S, Act, P, \mathsf{init})$ *be an MDP and* $\mathsf{Eff} \subseteq S$ *a set of terminal states. Let* Cause $\subseteq S \setminus \mathsf{Eff}$. *Then,* Cause *is an SPR cause (resp. a GPR cause) for* Eff *in* $\mathcal{M}$ *if and only if* Cause *is an SPR cause (resp. a GPR cause) for* Eff *in* $\mathcal{M}_{[\mathsf{Cause}]}$.

4.1   Checking the strict probability-raising condition and the existence of causes

The basis of both checking the existence of PR causes or checking the SPR condition for a given cause candidate is the following polynomial time algorithm to check whether the SPR condition holds in a given state c of $\mathcal{M}$ for all schedulers $\mathfrak{S}$ with $\mathrm{Pr}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond c) > 0$:

Algorithm 2. Input: state $c \in S$, set of terminal states $\mathsf{Eff} \subseteq S$.
Task: Decide whether (SPR) holds in c for all schedulers $\mathfrak{S}$.
 Compute $w_c = \mathrm{Pr}^{\min}_{\mathcal{M},c}(\Diamond\mathsf{Eff})$ and $q_s = \mathrm{Pr}^{\max}_{\mathcal{M}_{[c]},s}(\Diamond\mathsf{Eff})$ for each state s in $\mathcal{M}_{[c]}$.

1. If $q_{\mathsf{init}} < w_c$, then return "yes, (SPR) holds for c".
2. If $q_{\mathsf{init}} > w_c$, then return "no, (SPR) does not hold for c".
3. Suppose $q_{\mathsf{init}} = w_c$. Let $A(s) = \{\alpha \in Act_{\mathcal{M}_{[c]}}(s) \mid q_s = \sum_{t \in S_{[c]}} P_{\mathcal{M}_{[c]}}(s, \alpha, t) \cdot q_t\}$ for each non-terminal state s. Let $\mathcal{M}^{\max}_{[c]}$ denote the sub-MDP of $\mathcal{M}_{[c]}$ induced by the state-action pairs (s, α) where $\alpha \in A(s)$.
   3.1 If c is reachable from init in $\mathcal{M}^{\max}_{[c]}$, then return "no, (SPR) does not hold for c".
   3.2 If c is not reachable from init in $\mathcal{M}^{\max}_{[c]}$, then return "yes, (SPR) holds for c".

Lemma 4. *Algorithm 2 is sound and runs in polynomial time.*

Soundness. Let $\mathcal{N} = \mathcal{M}_{[c]}$. Soundness is obvious in case 1. For case 2, consider a real number λ with $1 > \lambda > \frac{w_c}{q_{init}}$ and MD-schedulers $\mathfrak{T}$ and $\mathfrak{S}$ realizing $\mathrm{Pr}_{\mathcal{N},s}^{\mathfrak{T}}(\Diamond\mathsf{Eff}) = q_s$ and $\mathrm{Pr}_{\mathcal{N}}^{\mathfrak{S}}(\Diamond c) > 0$ for all states s. We can combine $\mathfrak{T}$ and $\mathfrak{S}$ to a new MR-scheduler $\mathfrak{U}$ with the property that $\mathrm{Pr}_{\mathcal{N}}^{\mathfrak{U}}(\Diamond t) = \lambda\mathrm{Pr}_{\mathcal{N}}^{\mathfrak{T}}(\Diamond t) + (1-\lambda)\mathrm{Pr}_{\mathcal{N}}^{\mathfrak{S}}(\Diamond t)$ for all terminal states t and for $t = c$. Then, $\mathfrak{U}$ witnesses a violation of (SPR). For case 3.1 consider an MD-scheduler $\mathfrak{S}$ of $\mathcal{M}_{[c]}^{\max}$ where c is reachable from init via a $\mathfrak{S}$-path and $\mathrm{Pr}_{\mathcal{N},s}^{\mathfrak{S}}(\Diamond\mathsf{Eff}) = q_s$ for all states s. Then, (SPR) does not hold for c in the scheduler $\mathfrak{S}$. In case 3.2 we have $\mathrm{Pr}_{\mathcal{N}}^{\mathfrak{S}}(\Diamond c) = 0$ for all schedulers $\mathfrak{S}$ for $\mathcal{N}$ with $\mathrm{Pr}_{\mathcal{N}}^{\mathfrak{S}}(\Diamond\mathsf{Eff}) = q_{init} = w_c$. But then $\mathrm{Pr}_{\mathcal{N}}^{\mathfrak{S}}(\Diamond c) > 0$ implies $\mathrm{Pr}_{\mathcal{N}}^{\mathfrak{S}}(\Diamond\mathsf{Eff}) < w_c$ as required in (SPR).   □

By applying Algorithm 2 to all states $c \in \mathsf{Cause}$ and standard algorithms to check the existence of a path satisfying $(\neg\mathsf{Cause})\,\mathsf{U}\,c$ for every state $c \in \mathsf{Cause}$, we obtain:

Theorem 3 (Checking SPR causes). *The problem "given $\mathcal{M}$, Cause and Eff, check whether Cause is a SPR cause for Eff in $\mathcal{M}$" is solvable in polynomial-time.*

Remark 5 (Memory requirements for refuting the SPR property). As the soundness proof for Algorithm 2 shows: If Cause does not satisfy the SPR condition, then there is an MR-scheduler $\mathfrak{S}$ for $\mathcal{M}_{[\mathsf{Cause}]}$ witnessing the violation of (SPR). Scheduler $\mathfrak{S}$ corresponds to a finite-memory (randomized) scheduler $\mathfrak{T}$ with two memory cells for $\mathcal{M}$: "before Cause" (where $\mathfrak{T}$ behaves as $\mathfrak{S}$) and "after Cause" (where $\mathfrak{T}$ behaves as an MD-scheduler minimizing the effect probability form every state).   ◁

Lemma 5 (Criterion for the existence of probability-raising causes). *Let $\mathcal{M}$ be an MDP and Eff a nonempty set of states. Then Eff has an SPR cause in $\mathcal{M}$ iff Eff has a GPR cause in $\mathcal{M}$ iff there is a state $c_0 \in S \setminus \mathsf{Eff}$ such that the singleton $\{c_0\}$ is an SPR cause (and therefore a GRP cause) for Eff in $\mathcal{M}$. In particular, the existence of SPR/GPR causes can be checked with Algorithm 2 in polynomial time.*

4.2   Checking the global probability-raising condition

Theorem 4. *The problem "given $\mathcal{M}$, Cause and Eff, check whether Cause is a GPR cause for Eff in $\mathcal{M}$" is solvable in polynomial space.*

In order to provide an algorithm, we perform a model transformation after which the violation of (GPR) by a scheduler $\mathfrak{S}$ can be expressed solely in terms of the expected frequencies of the state-action pairs of the transformed MDP under $\mathfrak{S}$. This allows us to express the existence of a scheduler witnessing the non-causality of Cause in terms of the satisfiability of a quadratic constraint system. Then we can restrict the quantification in (G) to MR-schedulers in the transformed model. We trace back the memory requirements to $\mathcal{M}_{[\mathsf{Cause}]}$ and to the original MDP $\mathcal{M}$ yielding the second main result. Still, memory can be necessary to witness non-causality (Remark 1).

Theorem 5. *Let $\mathcal{M}$ be an MDP with effect set Eff as before and Cause a set of non-effect states such that condition (M) holds. If Cause is not a GPR cause for Eff, then there is an MR-scheduler for $\mathcal{M}_{[\mathsf{Cause}]}$ refuting the GPR condition for Cause in $\mathcal{M}_{[\mathsf{Cause}]}$ and a finite-memory scheduler for $\mathcal{M}$ with two memory cells refuting the GPR condition for Cause in $\mathcal{M}$.*

The remainder of this section is concerned with the proofs of Theorem 4 and Theorem 5. We suppose that both the effect set Eff and the cause candidate Cause are fixed disjoint subsets of the state space of the MDP $\mathcal{M}$ and that Cause satisfies (M).

Checking the GPR condition (Proof of Theorem 4). The first step is a polynomial-time model transformation which permits to make the following assumptions when checking the GPR condition of Cause for Eff.

(A1) $\mathsf{Eff} = \{\mathsf{eff}_{\mathsf{unc}}, \mathsf{eff}_{\mathsf{cov}}\}$ consists of two terminal states.

(A2) For every state $c \in \mathsf{Cause}$, there is only a single enabled action, say $Act(c) = \{\gamma\}$, and there exists $w_c \in [0,1] \cap \mathbb{Q}$ such that $P(c, \gamma, \mathsf{eff}_{\mathsf{cov}}) = w_c$ and $P(c, \gamma, \mathsf{noeff}_{\mathsf{fp}}) = 1 - w_c$ where $\mathsf{noeff}_{\mathsf{fp}}$ is a terminal non-effect state and $\mathsf{noeff}_{\mathsf{fp}}$ and $\mathsf{eff}_{\mathsf{cov}}$ are only accessible via the γ-transition from the states $c \in \mathsf{Cause}$.

(A3) $\mathcal{M}$ has no end components and there is a further terminal state $\mathsf{noeff}_{\mathsf{tn}}$ and an action τ such that $\tau \in Act(s)$ implies $P(s, \tau, \mathsf{noeff}_{\mathsf{tn}}) = 1$.

Intuitively, $\mathsf{eff}_{\mathsf{cov}}$ stands for covered effects ("Eff after Cause") and can be seen as a true positive, while $\mathsf{eff}_{\mathsf{unc}}$ represents the uncovered effects ("Eff without preceding Cause") and corresponds to a false negative. Let $\mathfrak{S}$ be a scheduler in $\mathcal{M}$. Note that $\mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}}((\neg \mathsf{Cause})\,U\,\mathsf{Eff}) = \mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}}(\Diamond \mathsf{eff}_{\mathsf{unc}})$ and $\mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}}(\Diamond(\mathsf{Cause} \wedge \Diamond \mathsf{Eff})) = \mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}}(\Diamond \mathsf{eff}_{\mathsf{cov}})$. As the cause states can not reach each other we also have $\mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}}((\neg \mathsf{Cause})\,U\,c) = \mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}}(\Diamond c)$ for each $c \in \mathsf{Cause}$. The intuitive meaning of $\mathsf{noeff}_{\mathsf{fp}}$ is a false positive ("no effect after Cause"), while $\mathsf{noeff}_{\mathsf{tn}}$ stands for true negatives where neither the effect nor the cause is observed. Note that $\mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}}(\Diamond(\mathsf{Cause} \wedge \neg \Diamond \mathsf{Eff})) = \mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}}(\Diamond \mathsf{noeff}_{\mathsf{fp}})$ and $\mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}}(\neg \Diamond \mathsf{Cause} \wedge \neg \Diamond \mathsf{Eff})) = \mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}}(\Diamond \mathsf{noeff}_{\mathsf{tn}})$.

Justification of assumptions (A1)-(A3): We justify the assumptions as we can transform $\mathcal{M}$ into a new MDP of the same asymptotic size satisfying the above assumptions. Thanks to Lemma 3, we may suppose that $\mathcal{M} = \mathcal{M}_{[\mathsf{Cause}]}$ (see Notation 1) without changing the satisfaction of the GPR condition. We then may rename the effect state eff and the non-effect state noeff reachable from Cause into $\mathsf{eff}_{\mathsf{cov}}$ and $\mathsf{noeff}_{\mathsf{fp}}$, respectively. Furthermore, we collapse all other effect states into a single state $\mathsf{eff}_{\mathsf{unc}}$ and all true negative states into $\mathsf{noeff}_{\mathsf{tn}}$. Similarly, by renaming and possibly duplicating terminal states we also suppose that $\mathsf{noeff}_{\mathsf{fp}}$ has no other incoming transitions than the γ-transitions from the states in Cause. This ensures (A1) and (A2). For (A3) consider the set T of terminal states in the MDP obtained so far. We remove all end components by switching to the MEC-quotient [2], i.e., we collapse all states that belong to the same MEC $\mathcal{E}$ into a single state $s_{\mathcal{E}}$ while ignoring the actions inside $\mathcal{E}$. Additionally, we add a fresh τ-transition from the states $s_{\mathcal{E}}$ to $\mathsf{noeff}_{\mathsf{tn}}$ (i.e., $P(s_{\mathcal{E}}, \tau, \mathsf{noeff}_{\mathsf{tn}}) = 1$). The τ-transitions from states $s_{\mathcal{E}}$ to $\mathsf{noeff}_{\mathsf{tn}}$ mimic cases where schedulers of the original MDP eventually enter an end component and stay there forever with positive probability.

With assumptions (A1)-(A3), the GPR condition can be reformulated as follows:

Lemma 6. *Under assumptions (A1)-(A3),* Cause *satisfies the GPR condition if and only if for each scheduler $\mathfrak{S}$ with $\mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}}(\Diamond \mathsf{Cause}) > 0$ the following condition holds:*

$$\mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}}(\Diamond \mathsf{Cause}) \cdot \mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}}(\Diamond \mathsf{eff}_{\mathsf{unc}}) \;<\; \big(1 - \mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}}(\Diamond \mathsf{Cause})\big) \cdot \sum_{c \in \mathsf{Cause}} \mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}}(\Diamond c) \cdot w_c \quad \text{(GPR-1)}$$

With assumptions (A1)-(A3), a terminal state of $\mathcal{M}$ is reached almost surely under any scheduler after finitely many steps in expectation. Given a scheduler $\mathfrak{S}$ for $\mathcal{M}$, the expected frequencies (i.e., expected number of occurrences in maximal paths) of state action-pairs (s, α), states $s \in S$ and state-sets $T \subseteq S$ under $\mathfrak{S}$ are defined by:

$$freq_{\mathfrak{S}}(s, \alpha) \stackrel{\text{def}}{=} E_{\mathcal{M}}^{\mathfrak{S}}(\text{number of visits to s in which } \alpha \text{ is taken})$$

$$freq_{\mathfrak{S}}(s) \stackrel{\text{def}}{=} \sum\nolimits_{\alpha \in Act(s)} freq_{\mathfrak{S}}(s, \alpha), \qquad freq_{\mathfrak{S}}(T) \stackrel{\text{def}}{=} \sum\nolimits_{s \in T} freq_{\mathfrak{S}}(s).$$

Let T be one of the sets $\{eff_{cov}\}$, $\{eff_{unc}\}$, Cause, or a singleton $\{c\}$ with $c \in$ Cause. As T is visited at most once during each run of $\mathcal{M}$ (assumptions (A1) and (A2)), we have $Pr_{\mathcal{N}}^{\mathfrak{S}}(\Diamond T) = freq_{\mathfrak{S}}(T)$ for each scheduler $\mathfrak{S}$. This allows us to express the violation of the GPR condition in terms of a quadratic constraint system over variables for the expected frequencies of state-action pairs in the following way:

Let *StAct* denote the set of state-action pairs in $\mathcal{M}$. We consider the following constraint system over the variables $x_{s,\alpha}$ for each $(s, \alpha) \in$ *StAct* where we use the short form notation $x_s = \sum_{\alpha \in Act(s)} x_{s,\alpha}$:

$$x_{s,\alpha} \geq 0 \qquad \text{for all } (s, \alpha) \in \text{\textit{StAct}} \tag{1}$$

$$x_{\text{init}} = 1 + \sum_{(t,\alpha) \in \text{\textit{StAct}}} x_{t,\alpha} \cdot P(t, \alpha, \text{init}) \tag{2}$$

$$x_s = \sum_{(t,\alpha) \in \text{\textit{StAct}}} x_{t,\alpha} \cdot P(t, \alpha, s) \qquad \text{for all } s \in S \setminus \{\text{init}\} \tag{3}$$

Using well-known results for MDPs without ECs (see, e.g., [23, Theorem 9.16]), given a vector $x \in \mathbb{R}^{\text{\textit{StAct}}}$, then x is a solution to (1) and the balance equations (2) and (3) if and only if there is a (possibly history-dependent) scheduler $\mathfrak{S}$ for $\mathcal{M}$ with $x_{s,\alpha} = freq_{\mathfrak{S}}(s, \alpha)$ for all $(s, \alpha) \in$ *StAct* if and only if there is an MR-scheduler $\mathfrak{S}$ for $\mathcal{M}$ with $x_{s,\alpha} = freq_{\mathfrak{S}}(s, \alpha)$ for all $(s, \alpha) \in$ *StAct*.

The violation of (GPR-1) in Lemma 6 and the condition $Pr_{\mathcal{M}}^{\mathfrak{S}}(\Diamond \text{Cause}) > 0$ can be reformulated in terms of the frequency-variables as follows where x_{Cause} is an abbreviation for $\sum_{c \in \text{Cause}} x_c$:

$$x_{\text{Cause}} \cdot x_{\text{eff}_{\text{unc}}} \geq (1 - x_{\text{Cause}}) \cdot \sum_{c \in \text{Cause}} x_c \cdot w_c \tag{4}$$

$$x_{\text{Cause}} > 0 \tag{5}$$

Lemma 7. *Under assumptions (A1)-(A3), the set* Cause *is not a GPR cause for* Eff *in* $\mathcal{M}$ *iff the constructed quadratic system of inequalities (1)-(5) has a solution.*

Proof of Theorem 4. The existence of a solution to the quadratic system of inequalities (Lemma 7) can straight-forwardly be formulated as a sentence in the language of the existential theory of the reals. The system of inequalities can be constructed from $\mathcal{M}$, Cause, and Eff in polynomial time. Its solvability is decidable in polynomial space as the decision problem of the existential theory of the reals is in PSPACE [13]. □

Memory requirements of schedulers in the original MDP (Proof of Theorem 5).
As stated above, every solution to the linear system of inequalities (1), (2), and (3)
corresponds to expected frequencies of state-action pairs of an MR-scheduler in the
transformed model satisfying (A1)-(A3). Hence:

Corollary 1. *Under assumptions (A1)-(A3),* Cause *is no GPR cause for* Eff *iff there
exists an MR-scheduler* $\mathfrak{T}$ *with* $\mathrm{Pr}_{\mathcal{M}}^{\mathfrak{T}}(\Diamond\mathsf{Cause}) > 0$ *violating the GPR condition.*

The model transformation we used for assumptions (A1)-(A3), however, does affect
the memory requirements of schedulers. We may further restrict the MR-schedulers
necessary to witness non-causality under assumptions (A1)-(A3). For the following
lemma, recall that τ is the action of the MEC quotient used for the extra transition
from states representing MECs to a new trap state (see also assumption (A3)).

Lemma 8. *Assume (A1)-(A3). Given an MR-scheduler* $\mathfrak{U}$ *with* $\mathrm{Pr}_{\mathcal{M}}^{\mathfrak{U}}(\Diamond\mathsf{Cause}) > 0$ *that
violates* (GPR)*, an MR-scheduler* $\mathfrak{T}$ *with* $\mathfrak{T}(s)(\tau) \in \{0,1\}$ *for each state* s *with* $\tau \in Act(s)$
that satifies $\mathrm{Pr}_{\mathcal{M}}^{\mathfrak{T}}(\Diamond\mathsf{Cause}) > 0$ *and violates* (GPR) *is computable in polynomial time.*

The condition that τ only has to be scheduled with probability 0 or 1 in each state
is the key to transfer the sufficiency of MR-schedulers to the MDP $\mathcal{M}_{[\mathsf{Cause}]}$. This fact
is of general interest as well and stated in the following theorem where τ again is the
action added to move from a state $s_{\mathcal{E}}$ to the new trap state in the MEC-quotient.

Theorem 6. *Let* $\mathcal{M}$ *be an MDP with pairwise disjoint action sets for all states. Then,
for each MR-scheduler* $\mathfrak{S}$ *for the MEC-quotient of* $\mathcal{M}$ *with* $\mathfrak{S}(s_{\mathcal{E}})(\tau) \in \{0,1\}$ *for each
MEC* $\mathcal{E}$ *of* $\mathcal{M}$ *there is an MR-scheduler* $\mathfrak{T}$ *for* $\mathcal{M}$ *such that every action* α *of* $\mathcal{M}$ *that does
not belong to an MEC of* $\mathcal{M}$*, has the same expected frequency under* $\mathfrak{S}$ *and* $\mathfrak{T}$.

Proof sketch. The crux are cases where $\mathfrak{S}(s_{\mathcal{E}})(\tau) = 0$, which requires to traverse the
MEC $\mathcal{E}$ of $\mathcal{M}$ in a memoryless way such that all actions leaving $\mathcal{E}$ have the same ex-
pected frequency under $\mathfrak{T}$ and $\mathfrak{S}$. First, we construct a finite-memory scheduler $\mathfrak{T}'$ that
always leaves each such end component according to the distribution given by $\mathfrak{S}(s_{\mathcal{E}})$.
By [23, Theorem 9.16], we then conclude that there is an MR-scheduler $\mathfrak{T}$ under which
the expected frequencies of all state-action pairs are the same as under $\mathfrak{T}'$.   □

Proof of Theorem 5. The model transformation establishing assumptions (A1)-(A3) re-
sults in the MEC-quotient of $\mathcal{M}_{[\mathsf{Cause}]}$ up to the renaming and collapsing of terminal
states. By Corollary 1 and Theorem 6, we conclude that Cause is not a GPR cause for
Eff in $\mathcal{M}$ iff there is a MR-scheduler $\mathfrak{S}$ for $\mathcal{M}_{[\mathsf{Cause}]}$ with $\mathrm{Pr}_{\mathcal{M}_{[\mathsf{Cause}]}}^{\mathfrak{S}}(\Diamond\mathsf{Cause}) > 0$ that
violates (GPR). As in Remark 5, $\mathfrak{S}$ can be extended to a finite-memory randomized
scheduler $\mathfrak{T}$ for $\mathcal{M}$ with two memory cells.   □

Remark 6 (On lower bounds on GPR checking). Solving systems of quadratic inequal-
ities with linear side constraints is NP-hard in general (see, e.g., [20]). For convex prob-
lems, in which the associated symmetric matrix in the quadratic inequality has only
non-negative eigenvalues, the problem is, however, solvable in polynomial time [26].
Unfortunately, the quadratic constraint system given by (1)-(5) is not of this form. Even
if Cause is a singleton $\{c\}$ and the variable $x_{\mathsf{eff}_{\mathsf{unc}}}$ is forced to take a constant value y by
(1)-(3), i.e., by the structure of the MDP, the inequality (4) takes the form:

$$x_c \cdot w_c - x_c^2 \cdot (w_c + y) \leqslant 0 \qquad (*)$$

Here, the 1×1-matrix $(-w_c - y)$ has a negative eigenvalue. Although it is not ruled out that (1)-(5) belongs to another class of efficiently solvable constraint systems, the NP-hardness result in [33] for the solvability of quadratic inequalities of the form (*) with linear side constraints might be an indication for the computational difficulty. $\quad \lhd$

5 Quality and optimality of causes

The goal of this section is to identify notions that measure how "good" causes are and to present algorithms to determine good causes according to proposed quality measures. We have seen so far that small (singleton) causes are easy to determine (see Section 4.1). Moreover, it is easy to see that the proposed existence-checking algorithm can be transformed such that it returns a singleton (strict or global) probability-raising cause $\{c_0\}$ with maximal *precision*, i.e., a state c_0 where $\inf_{\mathfrak{S}} \Pr^{\mathfrak{S}}_{\mathcal{M}}(\lozenge \mathsf{Eff} | \lozenge c_0) = \Pr^{\min}_{\mathcal{M}, c_0}(\lozenge \mathsf{Eff})$ is maximal. On the other hand, singleton or small cause sets might have poor coverage in the sense that the probability of paths which reach an effect state without visiting a cause state before ("uncovered effects") can be large. This motivates the consideration of quality notions for causes that incorporate how well effect scenarios are covered. We take inspiration of quality measures that are considered in statistical analysis (see e.g. [36]). This includes the *recall* as a measure for the relative coverage (proportion of covered effects among all effect scenarios), the *coverage ratio* (quotient of covered and uncovered effects) as well as the *f-score*. The f-score is a standard measure for classifiers defined by the harmonic mean of precision and recall. It can be seen as a compromise to achieve both good precision and good recall.

Throughout this section, we assume as before an MDP $\mathcal{M} = (S, Act, P, \mathsf{init})$ and a set $\mathsf{Eff} \subseteq S$ are given where all effect states are terminal. Furthermore, we suppose that all states $s \in S$ are reachable from init.

5.1 Quality measures for causes

In statistical analysis, the precision of a classifier with binary outcomes ("positive" or "negative") is defined as the ratio of all true positives among all positively classified elements, while its recall is defined as the ratio of all true positives among all actual positive elements. Translated to our setting, we consider classifiers induced by a given cause set Cause that return "positive" for sample paths in case that a cause state is visited and "negative" otherwise. The intuitive meaning of true positives and false negatives is as explained after Definition 1. The meaning of true negatives and false positives is analogous. We use $\mathsf{tp}^{\mathfrak{S}}$ for the probability for true positives under $\mathfrak{S}$. The notations $\mathsf{fp}^{\mathfrak{S}}$, $\mathsf{fn}^{\mathfrak{S}}$, $\mathsf{tn}^{\mathfrak{S}}$ have analogous meanings.

With this interpretation of causes as binary classifiers in mind, the recall and precision and coverage ratio of a cause set Cause *under a scheduler* $\mathfrak{S}$ is defined as follows (assuming $\Pr^{\mathfrak{S}}_{\mathcal{M}}(\lozenge \mathsf{Eff}) > 0$ resp. $\Pr^{\mathfrak{S}}_{\mathcal{M}}(\lozenge \mathsf{Cause}) > 0$ resp. $\Pr^{\mathfrak{S}}_{\mathcal{M}}\big((\neg\mathsf{Cause}) \cup \mathsf{Eff}\big) > 0$):

$$precision^{\mathfrak{S}}(\mathsf{Cause}) \;=\; \Pr^{\mathfrak{S}}_{\mathcal{M}}(\,\lozenge\mathsf{Eff} \mid \lozenge\mathsf{Cause}\,) = \tfrac{\mathsf{tp}^{\mathfrak{S}}}{\mathsf{tp}^{\mathfrak{S}} + \mathsf{fp}^{\mathfrak{S}}}$$

$$recall^{\mathfrak{S}}(\mathsf{Cause}) \;=\; \Pr^{\mathfrak{S}}_{\mathcal{M}}(\,\lozenge\mathsf{Cause} \mid \lozenge\mathsf{Eff}\,) = \tfrac{\mathsf{tp}^{\mathfrak{S}}}{\mathsf{tp}^{\mathfrak{S}} + \mathsf{fn}^{\mathfrak{S}}}$$

$$covrat^{\mathfrak{S}}(\mathsf{Cause}) = \frac{\mathrm{Pr}^{\mathfrak{S}}_{\mathcal{M}}\big(\Diamond(\mathsf{Cause} \wedge \Diamond \mathsf{Eff})\big)}{\mathrm{Pr}^{\mathfrak{S}}_{\mathcal{M}}\big((\neg\mathsf{Cause})\,\mathsf{U}\,\mathsf{Eff}\big)} = \frac{\mathsf{tp}^{\mathfrak{S}}}{\mathsf{fn}^{\mathfrak{S}}}.$$

For the coverage ratio, if $\mathrm{Pr}^{\mathfrak{S}}_{\mathcal{M}}\big((\neg\mathsf{Cause})\,\mathsf{U}\,\mathsf{Eff}\big) = 0$ and $\mathrm{Pr}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond\mathsf{Cause}) > 0$ we define $covrat^{\mathfrak{S}}(\mathsf{Cause}) = +\infty$. Finally, the f-score of Cause *under a scheduler* $\mathfrak{S}$ is defined as the harmonic mean of the precision and recall (assuming $\mathrm{Pr}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond\mathsf{Cause}) > 0$, which implies $\mathrm{Pr}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond\mathsf{Eff}) > 0$ as Cause is a PR cause):

$$fscore^{\mathfrak{S}}(\mathsf{Cause}) \;\overset{\text{def}}{=}\; 2 \cdot \frac{precision^{\mathfrak{S}}(\mathsf{Cause}) \cdot recall^{\mathfrak{S}}(\mathsf{Cause})}{precision^{\mathfrak{S}}(\mathsf{Cause}) + recall^{\mathfrak{S}}(\mathsf{Cause})}$$

If, however, $\mathrm{Pr}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond\mathsf{Eff}) > 0$ and $\mathrm{Pr}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond\mathsf{Cause}) = 0$ we define $fscore^{\mathfrak{S}}(\mathsf{Cause}) = 0$.

Quality measures for cause sets. Let Cause be a PR cause. The recall of Cause measures the relative coverage in terms of the worst-case conditional probability for covered effects (true positives) among all scenarios where the effect occurs.

$$recall(\mathsf{Cause}) \;=\; \inf_{\mathfrak{S}} \; recall^{\mathfrak{S}}(\mathsf{Cause}) \;=\; \mathrm{Pr}^{\min}_{\mathcal{M}}\big(\,\Diamond\mathsf{Cause} \mid \Diamond\mathsf{Eff}\,\big)$$

when ranging over all schedulers $\mathfrak{S}$ with $\mathrm{Pr}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond\mathsf{Eff}) > 0$. Likewise, the coverage ratio and f-score of Cause are defined by the worst-case coverage ratio resp. f-score (when ranging over schedulers for which $covrat^{\mathfrak{S}}(\mathsf{Cause})$ resp. $fscore^{\mathfrak{S}}(\mathsf{Cause})$ is defined):

$$covrat(\mathsf{Cause}) \;=\; \inf_{\mathfrak{S}} \; covrat^{\mathfrak{S}}(\mathsf{Cause}), \quad fscore(\mathsf{Cause}) \;=\; \inf_{\mathfrak{S}} \; fscore^{\mathfrak{S}}(\mathsf{Cause})$$

5.2   Computation schemes for the quality measures for fixed cause set

For this section, we assume a fixed PR cause Cause is given and address the problem to compute its quality values. Since all quality measures are preserved by the switch from $\mathcal{M}$ to $\mathcal{M}_{[\mathsf{Cause}]}$ as well as the transformations of $\mathcal{M}_{[\mathsf{Cause}]}$ to an MDP that satisfies conditions (A1)-(A3) of Section 4.2, we may assume that $\mathcal{M}$ satisfies (A1)-(A3).

While efficient computation methods for $recall(\mathsf{Cause})$ are known from literature (see [10,31] for poly-time algorithms to compute conditional reachability probabilities), we are not aware of known concepts that are applicable for computing the coverage ratio or the f-score. Indeed, both are efficiently computable:

Theorem 7. *The values covrat(Cause) and fscore(Cause) and corresponding worst-case schedulers are computable in polynomial time.*

By definition, the value $covrat(\mathsf{Cause})$ is the infimum over a quotient of reachability probabilities for disjoint sets of terminal states. While this is not the case for the f-score, we can express $fscore(\mathsf{Cause})$ in terms of the supremum of such a quotient. More precisely, under assumptions (A1)-(A3) and assuming $fscore(\mathsf{Cause}) > 0$, we have:

$$fscore(\mathsf{Cause}) = \frac{2}{X+2} \quad \text{where} \quad X = \sup_{\mathfrak{S}} \frac{\mathrm{Pr}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond\mathsf{noeff}_{\mathsf{fp}}) + \mathrm{Pr}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond\mathsf{eff}_{\mathsf{unc}})}{\mathrm{Pr}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond\mathsf{eff}_{\mathsf{cov}})}$$

where $\mathfrak{S}$ ranges over all schedulers with $\mathrm{Pr}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond\mathsf{eff}_{\mathsf{cov}}) > 0$. Furthermore, we have $fscore(\mathsf{Cause}) = 0$ if and only if $recall(\mathsf{Cause}) = 0$ if and only if there exists a scheduler $\mathfrak{S}$ satisfying $\mathrm{Pr}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond\mathsf{Eff}) > 0$ and $\mathrm{Pr}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond\mathsf{Cause}) = 0$.

So, the remaining task to prove Theorem 7 is a generally applicable technique for computing extremal ratios of reachability probabilities in MDPs without ECs.

Max/min ratios of reachability probabilities for disjoint sets of terminal states.
Suppose we are given an MDP $\mathcal{M} = (S, Act, P, \text{init})$ without ECs and disjoint subsets
$U, V \subseteq S$ of terminal states. Given a scheduler $\mathfrak{S}$ with $\Pr_{\mathcal{M}}^{\mathfrak{S}}(\lozenge V) > 0$ we define:

$$ratio_{\mathcal{M}}^{\mathfrak{S}}(U, V) \ = \ \Pr_{\mathcal{M}}^{\mathfrak{S}}(\lozenge U) / \Pr_{\mathcal{M}}^{\mathfrak{S}}(\lozenge V)$$

The goal is to compute the extremal values: $ratio_{\mathcal{M}}^{\min}(U, V) = \inf_{\mathfrak{S}} ratio_{\mathcal{M}}^{\mathfrak{S}}(U, V)$ and
$ratio_{\mathcal{M}}^{\max}(U, V) = \sup_{\mathfrak{S}} ratio_{\mathcal{M}}^{\mathfrak{S}}(U, V)$ where $\mathfrak{S}$ ranges over all schedulers such that
$\Pr_{\mathcal{M}}^{\mathfrak{S}}(\lozenge V) > 0$. For their computation, we rely on a polynomial reduction to the classi-
cal *stochastic shortest path problem* [12]. For this, consider the MDP $\mathcal{N}$ arising from
$\mathcal{M}$ by adding reset transitions from all terminal states $t \in S \backslash V$ to init. Thus, exactly the
V-states are terminal in $\mathcal{N}$. The MDP $\mathcal{N}$ might contain ECs, which, however, do not in-
tersect with V. We equip $\mathcal{N}$ with the weight function that assigns 1 to all states in U and
0 to all other states. For a scheduler $\mathfrak{T}$ with $\Pr_{\mathcal{N}}^{\mathfrak{T}}(\lozenge V) = 1$, let $E_{\mathcal{N}}^{\mathfrak{T}}(\boxplus V)$ be the expected
accumulated weight until reaching V under $\mathfrak{T}$. Let $E_{\mathcal{N}}^{\min}(\boxplus V) = \inf_{\mathfrak{T}} E_{\mathcal{N}}^{\mathfrak{T}}(\boxplus V)$ and
$E_{\mathcal{N}}^{\max}(\boxplus V) = \sup_{\mathfrak{T}} E_{\mathcal{N}}^{\mathfrak{T}}(\boxplus V)$, where $\mathfrak{T}$ ranges over all schedulers with $\Pr_{\mathcal{N}}^{\mathfrak{T}}(\lozenge V) = 1$.
We can rely on known results [12,3,4] to obtain that both $E_{\mathcal{N}}^{\min}(\boxplus V)$ and $E_{\mathcal{N}}^{\max}(\boxplus V)$
are computable in polynomial time. As $\mathcal{N}$ has only non-negative weights, $E_{\mathcal{N}}^{\min}(\boxplus V)$
is finite and a corresponding MD-scheduler with minimal expectation exists. If $\mathcal{N}$ has
an EC containing at least one U-state, which is the case iff $\mathcal{M}$ has a scheduler $\mathfrak{S}$ with
$\Pr_{\mathcal{M}}^{\mathfrak{S}}(\lozenge U) > 0$ and $\Pr_{\mathcal{M}}^{\mathfrak{S}}(\lozenge V) = 0$, then $E_{\mathcal{N}}^{\max}(\boxplus V) = +\infty$. Otherwise, $E_{\mathcal{N}}^{\max}(\boxplus V)$ is fi-
nite and the maximum is achieved by an MD-scheduler as well.

Theorem 8. *Let $\mathcal{M}$ be an MDP without ECs and U, V disjoint sets of terminal states in
$\mathcal{M}$, and let $\mathcal{N}$ be the constructed MDP as above. Then, $ratio_{\mathcal{M}}^{\min}(U, V) = E_{\mathcal{N}}^{\min}(\boxplus V)$ and
$ratio_{\mathcal{M}}^{\max}(U, V) = E_{\mathcal{N}}^{\max}(\boxplus V)$. Thus, both values are computable in polynomial time, and
there is an MD-scheduler minimizing $ratio_{\mathcal{M}}^{\mathfrak{S}}(U, V)$, and an MD-scheduler maximizing
$ratio_{\mathcal{M}}^{\mathfrak{S}}(U, V)$ if $ratio_{\mathcal{M}}^{\max}(U, V)$ is finite.*

Proof of Theorem 7. Using assumptions (A1)-(A3), we obtain that $covrat(\mathsf{Cause}) =$
$ratio_{\mathcal{M}}^{\min}(U, V)$ where $U = \{\mathsf{eff}_{\mathsf{cov}}\}$, $V = \{\mathsf{eff}_{\mathsf{unc}}\}$. Similarly, with $U = \{\mathsf{noeff}_{\mathsf{fp}}, \mathsf{eff}_{\mathsf{unc}}\}$,
$V = \{\mathsf{eff}_{\mathsf{cov}}\}$, we get $fscore(\mathsf{Cause}) = 0$ if $ratio_{\mathcal{M}}^{\max}(U, V) = +\infty$ and $fscore(\mathsf{Cause}) =$
$2/(ratio_{\mathcal{M}}^{\max}(U, V) + 2)$ otherwise. Thus, the claim follows from Theorem 8. $\qquad\square$

5.3   Quality-optimal probability-raising causes

An SPR cause Cause is called *recall-optimal* if $recall(\mathsf{Cause}) = \max_C recall(C)$ where
C ranges over all SPR causes. Likewise, *ratio-optimality* resp. *f-score-optimality* of
Cause means maximality of $covrat(\mathsf{Cause})$ resp. $fscore(\mathsf{Cause})$ among all SPR causes.
Recall-, ratio- and f-score-optimality for GPR causes are defined accordingly.

Lemma 9. *Let Cause be an SPR or a GPR cause. Then, Cause is recall-optimal if and
only if Cause is ratio-optimal.*

Recall- and ratio-optimal SPR causes. The techniques of Section 4.1 yield an algo-
rithm for generating a canonical SPR cause with optimal recall and ratio. To see this,
let $\mathcal{C}$ denote the set of states that constitute a singleton SPR cause. The canonical cause
$\mathsf{CanCause}$ is defined as the set of states $c \in \mathcal{C}$ such that there is a scheduler $\mathfrak{S}$ with
$\Pr_{\mathcal{M}}^{\mathfrak{S}}((\neg \mathcal{C}) \, U \, c) > 0$. Obviously, $\mathcal{C}$ and $\mathsf{CanCause}$ are computable in polynomial time.

Theorem 9. *If* $\mathcal{C} \neq \varnothing$ *then* CanCause *is a ratio- and recall-optimal SPR cause.*

This is not true for the f-score. To see this, Consider the Markov chain on the right hand side. We have CanCause $= \{s_1\}$, which has *precision*(CanCause) $= \frac{3}{4}$ and *recall*(CanCause) $= \frac{3}{8}/(\frac{1}{4} + \frac{3}{8}) = \frac{3}{5}$. But the SPR cause $\{s_2\}$ has better f-score as its precision is 1 and it has the same recall as CanCause.
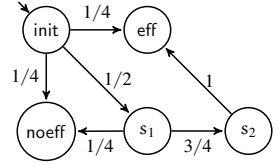
F-score-optimal SPR cause. From Section 5.2, we see that f-score-optimal SPR causes in MDPs can be computed in polynomial space by computing the f-score for all potential SPR causes one by one in polynomial time (Theorem 7). As the space can be reused after each computation, this results in polynomial space. For Markov chains, we can do better and compute an f-score-optimal SPR cause in polynomial time via a polynomial reduction to the stochastic shortest path problem:

Theorem 10. *In Markov chains that have SPR causes, an f-score-optimal SPR cause can be computed in polynomial time.*

Proof. We regard the given Markov chain $\mathcal{M}$ as an MDP with a singleton action set $Act = \{\alpha\}$. As $\mathcal{M}$ has SPR causes, the set $\mathcal{C}$ of states that constitute a singleton SPR cause is nonempty. We may assume that $\mathcal{M}$ has no non-trivial (i.e., cyclic) bottom strongly connected components as we may collapse them. Let $w_c = \Pr_{\mathcal{M},c}(\Diamond \mathsf{Eff})$. We switch from $\mathcal{M}$ to a new MDP $\mathcal{K}$ with state space $\mathsf{S}_{\mathcal{K}} = \mathsf{S} \cup \{\mathsf{eff}_{\mathsf{cov}}, \mathsf{noeff}_{\mathsf{fp}}\}$ with fresh states $\mathsf{eff}_{\mathsf{cov}}$ and $\mathsf{noeff}_{\mathsf{fp}}$ and the action set $Act_{\mathcal{K}} = \{\alpha, \gamma\}$. The MDP $\mathcal{K}$ arises from $\mathcal{M}$ by adding (i) for each state $c \in \mathcal{C}$ a fresh state-action pair (c, γ) with $\mathsf{P}_{\mathcal{K}}(c, \gamma, \mathsf{eff}_{\mathsf{cov}}) = w_c$ and $\mathsf{P}_{\mathcal{K}}(c, \gamma, \mathsf{noeff}_{\mathsf{fp}}) = 1 - w_c$ and (ii) reset transitions to init with action label α from the new state $\mathsf{noeff}_{\mathsf{fp}}$ and all terminal states of $\mathcal{M}$, i.e., $\mathsf{P}_{\mathcal{K}}(\mathsf{noeff}_{\mathsf{fp}}, \alpha, \mathsf{init}) = 1$ and $\mathsf{P}_{\mathcal{K}}(s, \alpha, \mathsf{init}) = 1$ for $s \in \mathsf{Eff}$ or if s is a terminal non-effect state of $\mathcal{M}$. So, exactly $\mathsf{eff}_{\mathsf{cov}}$ is terminal in $\mathcal{K}$, and $Act_{\mathcal{K}}(c) = \{\alpha, \gamma\}$ for $c \in \mathcal{C}$, while $Act_{\mathcal{K}}(s) = \{\alpha\}$ for all other states s. Intuitively, taking action γ in state $c \in \mathcal{C}$ selects c to be a cause state. The states in Eff represent uncovered effects in $\mathcal{K}$, while $\mathsf{eff}_{\mathsf{cov}}$ stands for covered effects.

We assign weight 1 to all states in $\mathsf{U} = \mathsf{Eff} \cup \{\mathsf{noeff}_{\mathsf{fp}}\}$ and weight 0 to all other states of $\mathcal{K}$. Let $\mathsf{V} = \{\mathsf{eff}_{\mathsf{cov}}\}$. Then, $\mathsf{f} = \mathrm{E}_{\mathcal{K}}^{\min}(\boxplus \mathsf{V})$ and an MD-scheduler $\mathfrak{S}$ for $\mathcal{K}$ such that $\mathrm{E}_{\mathcal{K}}^{\mathfrak{S}}(\boxplus \mathsf{V}) = \mathsf{f}$ are computable in polynomial time. Let $\mathcal{C}_{\gamma}$ denote the set of states $c \in \mathcal{C}$ where $\mathfrak{S}(c) = \gamma$ and let Cause be the set of states $c \in \mathcal{C}_{\gamma}$ where $\mathcal{M}$ has a path satisfying $(\neg \mathcal{C}_{\gamma}) \mathsf{U} c$. Then, Cause is an SPR cause of $\mathcal{M}$. With arguments as in Section 5.2 we obtain *fscore*(Cause) $= 2/(\mathsf{f}+2)$. It remains to show that Cause is f-score-optimal. Let C be an arbitrary SPR cause. Then, $\mathsf{C} \subseteq \mathcal{C}$. Let $\mathfrak{T}$ be the MD-scheduler for $\mathcal{K}$ that schedules γ in C and α for all other states of $\mathcal{K}$. Then, *fscore*(C) $= 2/(\mathsf{f}^{\mathfrak{T}}+2)$ where $\mathsf{f}^{\mathfrak{T}} = \mathrm{E}_{\mathcal{K}}^{\mathfrak{T}}(\boxplus \mathsf{V})$. Hence, $\mathsf{f} \leqslant \mathsf{f}^{\mathfrak{T}}$, which yields *fscore*(Cause) $\geqslant$ *fscore*(C). $\square$

The naïve adaption of the construction presented in the proof of Theorem 10 for MDPs would yield a stochastic game structure where the objective of one player is to minimize the expected accumulated weight until reaching a target state. Although algorithms for *stochastic shortest path (SSP) games* are known [34], they rely on assumptions on the game structure which would not be satisfied here. However, for the

threshold problem *SPR-f-score* where inputs are an MDP $\mathcal{M}$, Eff and $\vartheta \in \mathbb{Q}_{\geqslant 0}$ and the task is to decide the existence of an SPR cause whose f-score exceeds ϑ, we can establish a polynomial reduction to SSP games, which yields an NP∩coNP upper bound:

Theorem 11. *The decision problem SPR-f-score is in* NP∩coNP.

Proof sketch. Given an MDP $\mathcal{M}$, an effect set Eff, and $\vartheta \in \mathbb{Q}$, we construct an SSP game [34] after a series of model transformations ensuring (i) that terminal states are reached almost surely and (ii) that Eff is reached with positive probability under all schedulers. Condition (i) is established by a standard MEC-quotient construction. To establish condition (ii), we provide a construction that forces schedulers to leave an initial sub-MDP in which the minimal probability to reach Eff is 0. This construction – unlike the MEC-quotient – affects the possible combinations of probability values with which terminal states and potential cause states can be reached, but the existence of an SPR cause satisfying the f-score-threshold condition is not affected.

The underlying idea of the construction of the game shares similarities with the MDP constructed in the proof of Theorem 10: Player 0 takes the role to select potential cause states while player 1 takes the role of a scheduler in the transformed MDP. Using the observation that for each cause C, $fscore(\mathsf{C}) > \vartheta$ iff

$$2(1-\vartheta)\mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}}(\lozenge\mathsf{C} \wedge \lozenge\mathsf{Eff}) - \vartheta\mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}}(\neg\lozenge\mathsf{C} \wedge \lozenge\mathsf{Eff}) - \vartheta\mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}}(\lozenge\mathsf{C} \wedge \neg\lozenge\mathsf{Eff}) > 0 \qquad (\times)$$

for all schedulers $\mathfrak{S}$ for $\mathcal{M}$ with $\mathrm{Pr}_{\mathcal{M}}^{\mathfrak{S}}(\lozenge\mathsf{Eff}) > 0$, weights are assigned to Eff-states and other terminal states depending on whether player 0 has chosen to include a state to the cause beforehand. In the resulting SSP game, both players have optimal MD-strategies [34]. Given such strategies ζ for player 0 and $\mathfrak{S}$ for player 1, the resulting expected accumulated weight agrees with the left-hand side of $(\times)$ when considering $\mathfrak{S}$ as a scheduler for the transformed MDP and the cause C induced by the states that ζ chooses to belong to the cause. Thus, player 0 wins the constructed game iff an SPR cause with f-score above the threshold ϑ exists. The existence of optimal MD-strategies for both players allows us to decide this threshold problem in NP and coNP.   □

Optimality and threshold constraints for GPR causes. Computing optimal GPR causes for either quality measure can be done in polynomial space by considering all cause candidates, checking the GPR condition in polynomial space (Theorem 4) and computing the corresponding quality measure in polynomial time (Section 5.2). However, we show that no polynomial-time algorithms can be expected as the corresponding threshold problems are NP-hard. Let GPR-covratio (resp. GPR-recall, GPR-f-score) denote the decision problems: Given $\mathcal{M}$, Eff and $\vartheta \in \mathbb{Q}$, decide whether there exists a GPR cause with coverage ratio (resp. recall, f-score) at least ϑ.

Theorem 12. *The problems GPR-covratio, GPR-recall and GPR-f-score are NP-hard and belong to PSPACE. For Markov chains, all three problems are NP-complete. NP-hardness even holds for tree-like Markov chains.*

Proof sketch. NP-hardness is established via a polynomial reduction from the knapsack problem. Membership to NP for Markov chains resp. to PSPACE = NPSPACE for MDPs is obvious as we can guess nondeterministically a cause candidate and then check (i) the GPR condition in polynomial time (Markov chains) resp. polynomial space (MDPs) and (ii) the threshold condition in polynomial time (see Section 5.2).   □

6 Conclusion

The goal of the paper was to formalize the probability-raising principle in MDPs and related quality notions for PR causes as well as studying fundamental algorithmic problems for them. We considered the strict (local) and the global view. Our results indicate that GPR causes are more general and leave more flexibility to achieve better accuracy, while algorithmic reasoning about SPR causes is simpler.

Existential definition of SPR/GPR causes. The proposed definition of PR causes relies on a universal quantification over all relevant schedulers. However, another approach could be via existential quantification, i.e. there is a scheduler $\mathfrak{S}$ such that (GPR) or resp. (SPR) hold. The resulting notion of causality yields fairly the same results (up to $\mathrm{Pr}_{\mathcal{M},\mathbf{c}}^{\max}(\Diamond\mathsf{Eff})$ instead of $\mathrm{Pr}_{\mathcal{M},\mathbf{c}}^{\min}(\Diamond\mathsf{Eff})$ etc). A canonical existential SPR cause can be defined in analogy to the universal case and shown to be recall- and ratio-optimal (cf. Theorem 9). The problem to find an existential f-score-optimal SPR cause is even simpler and solvable in polynomial time as the construction presented in the proof of Theorem 10 can be adapted for MDPs (thanks to the simpler nature of $\max_C \sup_{\mathfrak{S}} fscore^{\mathfrak{S}}(C)$ compared to $\max_C \inf_{\mathfrak{S}} fscore^{\mathfrak{S}}(C)$). However, NP-hardness for the existence of GPR causes with threshold constraints for the quality carries over to the existential definition (as NP-hardness holds for Markov chains, Theorem 12).

Non-strict inequality in the PR conditions. Our notions of PR causes are in line with the classical approach of probability-raising causality in literature with strict inequality in the PR condition. This has the consequence that causes might not exist (see Example 2). The switch to a relaxed definition of PR causes with non-strict inequality seems to be a minor change that identifies more sets as causes. Indeed, the proposed algorithms for checking the SPR and GPR condition (Section 4) can easily be modified for the relaxed definition. While this leads to a questionable notion of causality (e.g., {init} would always be a recall- and ratio-optimal SPR cause under the relaxed definition), it could be useful in combination with other side constraints. E.g., requiring the relaxed PR condition for all schedulers which reach a cause state with positive probability and requiring the existence of a scheduler where the PR condition with strict inequality holds might be a useful alternative definition that agrees with Def. 1 for Markov chains.

Relaxing the minimality condition (M). As many causality notions of the literature include some minimality constraint, we included condition (M). However, (M) could be dropped without affecting the algorithmic results presented here. This can be useful when the task is to identify components or agents that are responsible for the occurrences of undesired effects. In these cases the cause candidates are fixed (e.g., for each agent $\mathfrak{i}$, the set of states controlled by agent $\mathfrak{i}$), but some of them might violate (M).

Future directions include PR causality when causes and effects are path properties and the investigation of other quality measures for PR causes inspired by other indices for binary classifiers used in machine learning or customized for applications of cause-effect reasoning in MDPs. More sophisticated notions of probabilistic backward causality and considerations on PR causality with external interventions as in Pearl's do-calculus [35] are left for future work.

References

1. Ábrahám, E., Bonakdarpour, B.: HyperPCTL: A temporal logic for probabilistic hyperproperties. In: McIver, A., Horváth, A. (eds.) 15th International Conference on Quantitative Evaluation of Systems (QEST). Lecture Notes in Computer Science, vol. 11024, pp. 20–35. Springer (2018), https://doi.org/10.1007/978-3-319-99154-2_2

2. de Alfaro, L.: Formal Verification of Probabilistic Systems. Phd thesis, Stanford University, Stanford, USA (1997), https://wcl.cs.rpi.edu/pilots/library/papers/TAGGED/4375-deAlfaro(1997)-FormalVerificationofProbabilisticSystems.pdf

3. de Alfaro, L.: Computing minimum and maximum reachability times in probabilistic systems. In: Baeten, J.C.M., Mauw, S. (eds.) 10th International Conference on Concurrency Theory (CONCUR). Lecture Notes in Computer Science, vol. 1664, pp. 66–81. Springer (1999), https://doi.org/10.1007/3-540-48320-9_7

4. Baier, C., Bertrand, N., Dubslaff, C., Gburek, D., Sankur, O.: Stochastic shortest paths and weight-bounded properties in Markov decision processes. In: Dawar, A., Grädel, E. (eds.) 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018. pp. 86–94. ACM (2018), https://doi.org/10.1145/3209108.3209184

5. Baier, C., Dubslaff, C., Funke, F., Jantsch, S., Majumdar, R., Piribauer, J., Ziemek, R.: From verification to causality-based explications (invited talk). In: Bansal, N., Merelli, E., Worrell, J. (eds.) 48th International Colloquium on Automata, Languages, and Programming, (ICALP). LIPIcs, vol. 198, pp. 1:1–1:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021), https://doi.org/10.4230/LIPIcs.ICALP.2021.1

6. Baier, C., Funke, F., Jantsch, S., Piribauer, J., Ziemek, R.: Probabilistic causes in Markov chains. CoRR **abs/2104.13604** (2021), https://arxiv.org/abs/2104.13604, accepted for publication at ATVA'21.

7. Baier, C., Funke, F., Majumdar, R.: A game-theoretic account of responsibility allocation. In: Zhou, Z. (ed.) 30th International Joint Conference on Artificial Intelligence (IJCAI). pp. 1773–1779. ijcai.org (2021), https://doi.org/10.24963/ijcai.2021/244

8. Baier, C., Funke, F., Piribauer, J., Ziemek, R.: On probability-raising causality in markov decision processes (2022), https://arxiv.org/abs/2201.08768

9. Baier, C., Katoen, J.P.: Principles of Model Checking (Representation and Mind Series). The MIT Press, Cambridge, MA (2008)

10. Baier, C., Klein, J., Klüppelholz, S., Märcker, S.: Computing conditional probabilities in Markovian models efficiently. In: Ábrahám, E., Havelund, K. (eds.) 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, vol. 8413, pp. 515–530. Springer (2014), https://doi.org/10.1007/978-3-642-54862-8_43

11. Beer, I., Ben-David, S., Chockler, H., Orni, A., Trefler, R.J.: Explaining counterexamples using causality. Formal Methods in System Design **40**(1), 20–40 (2012), https://doi.org/10.1007/s10703-011-0132-2

12. Bertsekas, D.P., Tsitsiklis, J.N.: An analysis of stochastic shortest path problems. Mathemathics of Operations Research **16**(3), 580–595 (1991)

13. Canny, J.F.: Some algebraic and geometric computations in PSPACE. In: 20th Annual ACM Symposium on Theory of Computing (STOC). pp. 460–467. ACM (1988)

14. Chockler, H.: Causality and responsibility for formal verification and beyond. In: First Workshop on Causal Reasoning for Embedded and safety-critical Systems Technologies (CREST). EPTCS, vol. 224, pp. 1–8 (2016), https://doi.org/10.4204/EPTCS.224.1

15. Chockler, H., Halpern, J.Y., Kupferman, O.: What causes a system to satisfy a specification? ACM Transactions on Computational Logic **9**(3), 20:1–20:26 (2008)

16. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. MIT Press (1999)

17. Dimitrova, R., Finkbeiner, B., Torfah, H.: Probabilistic hyperproperties of Markov decision processes. In: Hung, D.V., Sokolsky, O. (eds.) 18th International Symposium on Automated Technology for Verification and Analysis (ATVA). Lecture Notes in Computer Science, vol. 12302, pp. 484–500. Springer (2020), https://doi.org/10.1007/978-3-030-59152-6_27

18. Eells, E.: Probabilistic Causality. Cambridge Studies in Probability, Induction and Decision Theory, Cambridge University Press (1991)

19. Friedenberg, M., Halpern, J.Y.: Blameworthiness in multi-agent settings. In: 33rd Conference on Artificial Intelligence (AAAI). pp. 525–532. AAAI Press (2019), https://doi.org/10.1609/aaai.v33i01.3301525

20. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)

21. Halpern, J.Y., Pearl, J.: Causes and explanations: A structural-model approach: Part 1: Causes. In: 17th Conference in Uncertainty in Artificial Intelligence (UAI). pp. 194–202 (2001)

22. Hitchcock, C.: Probabilistic causation. In: Hájek, A., Hitchcock, C. (eds.) The Oxford Handbook of Probability and Philosophy, pp. 815–832. Oxford University Press (2016)

23. Kallenberg, L.: Lecture Notes Markov Decision Problems - version 2020 (02 2020)

24. Kleinberg, S., Mishra, B.: The temporal logic of causal structures. In: 25th Conference on Uncertainty in Artificial Intelligence (UAI). pp. 303–312 (2009)

25. Kleinberg, S.: Causality, Probability and Time. Cambridge University Press (2012)

26. Kozlov, M.K., Tarasov, S.P., Khachiyan, L.G.: The polynomial solvability of convex quadratic programming. USSR Computational Mathematics and Mathematical Physics **20**(5), 223–228 (1980)

27. Kuntz, M., Leitner-Fischer, F., Leue, S.: From probabilistic counterexamples via causality to fault trees. In: Flammini, F., Bologna, S., Vittorini, V. (eds.) 30th International Conference on Computer Safety, Reliability, and Security (SAFECOMP). Lecture Notes in Computer Science, vol. 6894, pp. 71–84. Springer (2011), https://doi.org/10.1007/978-3-642-24270-0_6

28. Leitner-Fischer, F.: Causality Checking of Safety-Critical Software and Systems. Ph.D. thesis, University of Konstanz, Germany (2015), http://kops.uni-konstanz.de/handle/123456789/30778

29. Lewis, D.: Counterfactuals and comparative possibility. Journal of Philosophical Logic **2**(4), 418–446 (1973)

30. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems: Safety. Springer-Verlag (1995)

31. Märcker, S.: Model checking techniques for design and analysis of future hardware and software systems. Ph.D. thesis, TU Dresden, Germany (2020), https://d-nb.info/1232958204

32. Namjoshi, K.S.: Certifying model checkers. In: 13th International Conference on Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 2102, pp. 2–13. Springer (2001), https://doi.org/10.1007/3-540-44585-4_2

33. Pardalos, P.M., Vavasis, S.A.: Quadratic programming with one negative eigenvalue is np-hard. Journal of Global optimization **1**(1), 15–22 (1991)

34. Patek, S.D., Bertsekas, D.P.: Stochastic shortest path games. SIAM Journal on Control and Optimization **37**(3), 804–824 (1999)

35. Pearl, J.: Causality. Cambridge University Press, 2nd edn. (2009)

36. Powers, D.: Evaluation: From precision, recall and f-factor to ROC, informedness, markedness & correlation. Mach. Learn. Technol. **2** (01 2008)

37. Puterman, M.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., New York, NY (1994)

38. Reichenbach, H.: The Direction of Time. Dover Publications (1956)

39. Suppes, P.: A Probabilistic Theory of Causality. Amsterdam: North-Holland Pub. Co. (1970)
40. Yazdanpanah, V., Dastani, M.: Distant group responsibility in multi-agent systems. In: Baldoni, M., Chopra, A.K., Son, T.C., Hirayama, K., Torroni, P. (eds.) 19th International Conference on Princiles and Practice of Multi-Agent Systems (PRIMA). Lecture Notes in Computer Science, vol. 9862, pp. 261–278. Springer (2016), https://doi.org/10.1007/978-3-319-44832-9_16
41. Yazdanpanah, V., Dastani, M., Jamroga, W., Alechina, N., Logan, B.: Strategic responsibility under imperfect information. In: Elkind, E., Veloso, M., Agmon, N., Taylor, M.E. (eds.) 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS). pp. 592–600. International Foundation for Autonomous Agents and Multiagent Systems (2019), http://dl.acm.org/citation.cfm?id=3331745

Parameterized Analysis of Reconfigurable Broadcast Networks⋆

A. R. Balasubramanian[1] , Lucie Guillou[2] , and Chana Weil-Kennedy[3](✉)

[1] Technical University of Munich `bala.ayikudi@tum.de`
[2] ENS Rennes `lucie.guillou@ens-rennes.fr`
[3] Technical University of Munich `chana.weilkennedy@in.tum.de`

Abstract. Reconfigurable broadcast networks (RBN) are a model of distributed computation in which agents can broadcast messages to other agents using some underlying communication topology which can change arbitrarily over the course of executions. In this paper, we conduct *parameterized analysis* of RBN. We consider cubes, (infinite) sets of configurations in the form of lower and upper bounds on the number of agents in each state, and we show that we can evaluate boolean combinations over cubes and reachability sets of cubes in PSPACE. In particular, reachability from a cube to another cube is a PSPACE-complete problem.

To prove the upper bound for this parameterized analysis, we prove some structural properties about the reachability sets and the *symbolic graph* abstraction of RBN, which might be of independent interest. We justify this claim by providing two applications of these results. First, we show that the almost-sure coverability problem is PSPACE-complete for RBN, thereby closing a complexity gap from a previous paper [3]. Second, we define a computation model using RBN, à la population protocols, called RBN protocols. We characterize precisely the set of predicates that can be computed by such protocols.

Keywords: Broadcast networks · Parameterized reachability · Almost-sure coverability · Asynchronous shared-memory systems

1 Introduction

Reconfigurable broadcast networks (RBN) [8,10] are a formalism for modelling distributed systems in which a set of anonymous, finite-state agents execute the same underlying protocol and broadcast messages to their neighbors according to an underlying communication topology. The communication topology is reconfigurable, meaning that the set of neighbors of an agent can change arbitrarily over the course of an execution. Parameterized verification of these networks concerns itself with proving that a given property is correct, irrespective of the number of participating agents. Dually, it can be viewed as the problem of finding an

execution of some number of agents which violates a given property. Ever since their introduction within this context [10], RBN have been studied extensively, with various results on (parameterized) reachability and coverability [8,10,3,7], along with various extensions using probabilities and clocks [5,4].

In this paper, we first consider the *cube-reachability* problem for RBN, in which we are given two (possibly infinite) sets of configurations $\mathcal{C}$ and $\mathcal{C}'$ (called *cubes*), each of them defined by lower and upper bounds on the number of agents in each state, and we must decide if there is a configuration in $\mathcal{C}$ which can reach some configuration in $\mathcal{C}'$. The cube-reachability question covers parameterized reachability and coverability problems, and as explained in [3], also covers the parameterized reachability problem for a generalized model of RBN called *RBN with leaders*. Moreover, a sub-problem of cube-reachability has already been studied for RBN in [8]. The authors show that this sub-problem is PSPACE-complete. One of the results in our paper is that the entire cube-reachability problem is PSPACE-complete, hence extending the sub-problem considered in [8], while still retaining the same complexity upper bound.

In fact, our main result, which we call the PSPACE Theorem, is a more general result. It subsumes the above result for cube-reachability and allows for more complex parameterized analysis of RBN. The PSPACE Theorem roughly states that any boolean combination of atoms can be evaluated in PSPACE, where an atom is a finite union of cubes or the reachability set of a finite union of cubes (i.e. *post** or *pre**). To prove the PSPACE Theorem, we first consider the so called *symbolic graph* of a RBN ([8], Section 5). We prove some structural properties about these graphs, using results from [8]. Next, using these structural properties, we show that the set of reachable configurations of a cube $\mathcal{C}$ can be expressed as a finite union of cubes, each having a norm exponentially bounded in the size of the given RBN and $\mathcal{C}$. This result then allows us to give an on-the-fly exploration algorithm for proving the PSPACE Theorem.

We believe that the PSPACE Theorem and the results leading to it that we have proven in this paper have further applications to problems concerning RBN. To justify this claim, we provide two applications. First, we show that the almost-sure coverability problem for RBN is PSPACE-complete, thereby closing a complexity gap from a previous paper ([3], Section 5.3). Second, we define a computation model using RBN, called RBN protocols, which is similar in spirit to the population protocols model [1,2]. We characterize precisely the set of predicates that can be computed using RBN protocols. This result generalizes the corresponding result for IO protocols, which are a sub-class of population protocols that can be simulated by RBN protocols, as shown in ([3], Section 6.2).

Finally, by the reduction given in ([3], Section 4.2), our results on cube-reachability and almost-sure coverability can be transferred to another model of distributed computation called asynchronous shared memory systems (ASMS), giving a PSPACE-completeness result for both of these problems. This solves an open problem from ([6], Section 6).

To summarize, we have shown that many important parameterized problems of RBN can be solved in PSPACE, that the sub-problem of the cube-reachability

problem defined in [8] can be generalized while retaining the same upper bounds, and that the almost-sure coverability problems for RBN and ASMS are PSPACE-complete, thereby solving open problems from [3,6]. We believe that our other results might be of independent interest, and we provide an application by introducing RBN protocols and characterizing the set of predicates that they can compute.

The paper is organized as follows. Section 2 contains preliminaries, including the definition of RBN. Section 3 defines the symbolic graph of a RBN, and proves the properties of this graph needed to derive our main result. Section 4 contains the main result that a host of parameterized problems over cubes, including cube-reachability, is PSPACE-complete for RBN. Finally, Sections 5 and 6 give applications of our main results: Section 5 solves the complexity gap for the almost-sure coverability problem, and Section 6 introduces RBN protocols and characterizes their expressive power. Due to lack of space, full proofs of some of the results can be found in the long version.

2 Preliminaries

The definitions and notations in this section are taken from [3].

2.1 Multisets

A *multiset* on a finite set E is a mapping $C: E \to \mathbb{N}$, i.e. for any $e \in E$, $C(e)$ denotes the number of occurrences of element e in C. We let $\mathbb{M}(E)$ denote the set of all multisets on E. Let $\langle e_1, \ldots, e_n \rangle$ denote the multiset C such that $C(e) = |\{j \mid e_j = e\}|$. We sometimes write multisets using set-like notation. For example, $\langle 2 \cdot a, b \rangle$ and $\langle a, a, b \rangle$ denote the same multiset. Given $e \in E$, we denote by $\boldsymbol{e}$ the multiset consisting of one occurrence of element e, that is $\langle e \rangle$. Operations on $\mathbb{N}$ like addition or comparison are extended to multisets by defining them component wise on each element of E. Subtraction is allowed as long as each component stays non-negative. We call $|C| \stackrel{\text{def}}{=} \sum_{e \in E} C(e)$ the *size* of C.

2.2 Reconfigurable Broadcast Networks

Reconfigurable broadcast networks (RBN) are networks consisting of finite-state, anonymous agents and a communication topology which specifies for every pair of processes, whether or not there is a communication link between them. During a single step, a single agent can broadcast a message which is received by all of its neighbors, after which both the agent and its neighbors change their state according to some transition relation. Further, in between two steps, the communication topology can change in an arbitrary manner. For the problems that we consider in this paper, it is easier to forget the communication topology and define the semantics of an RBN directly in terms of collections of agents.

Definition 1. *A reconfigurable broadcast network is a tuple* $\mathcal{R} = (Q, \Sigma, \delta)$ *where Q is a finite set of states, Σ is a finite alphabet and $\delta \subseteq Q \times \{!a, ?a \mid a \in \Sigma\} \times Q$ is the transition relation.*

If $(p, !a, q)$ (resp. $(p, ?a, q)$) is a transition in δ, we will denote it by $p \xrightarrow{!a} q$ (resp. $p \xrightarrow{?a} q$). A *configuration* C of a RBN $\mathcal{R}$ is a multiset over Q, which intuitively counts the number of processes in each state. Given a letter $a \in \Sigma$ and two configurations C and C' we say that there is a *step* $C \xrightarrow{a} C'$ if there exists a multiset $\{t, t_1, \ldots, t_k\}$ of δ for some $k \geq 0$ satisfying the following: $t = p \xrightarrow{!a} q$, each $t_i = p_i \xrightarrow{?a} q_i$, $C \geq \boldsymbol{p} + \sum_i \boldsymbol{p_i}$, and $C' = C - \boldsymbol{p} - \sum_i \boldsymbol{p_i} + \boldsymbol{q} + \sum_i \boldsymbol{q_i}$. We sometimes write this as $C \xrightarrow{t + t_1, \ldots, t_n} C'$ or $C \xrightarrow{a} C'$. Intuitively it means that a process at the state p broadcasts the message a and moves to q, and for each $1 \leq i \leq k$, there is a process at the state p_i which receives this message and moves to q_i. We denote by $\xrightarrow{*}$ the reflexive and transitive closure of the step relation. A *run* is then a sequence of steps.
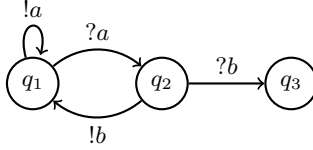
Fig. 1. An RBN $\mathcal{R}$ with three states.

Let $\mathcal{R} = (Q, \Sigma, \delta)$ be an RBN. Given configurations C and C', we say C' is *reachable* from C if $C \xrightarrow{*} C'$. We say C' is *coverable* from C if there exists C'' such that $C \xrightarrow{*} C''$ and $C'' \geq C'$. The *reachability* problem consists of deciding, given a RBN $\mathcal{R}$ and configurations C, C', whether C' is reachable from C in $\mathcal{R}$. The *coverability* problem consists of deciding, given a RBN $\mathcal{R}$ and configurations C, C', whether C' is coverable from C in $\mathcal{R}$. Let $\mathcal{S}$ be a set of configurations. The *predecessor set* of $\mathcal{S}$ is $pre^*(\mathcal{S}) \stackrel{\text{def}}{=} \{C' | \exists C \in \mathcal{S} . C' \xrightarrow{*} C\}$, and the *successor set* of $\mathcal{S}$ is $post^*(\mathcal{S}) \stackrel{\text{def}}{=} \{C | \exists C' \in \mathcal{S} . C' \xrightarrow{*} C\}$.

Example 1. Figure 1 illustrates a RBN $\mathcal{R} = (Q, \Sigma, \delta)$ with $Q = \{q_1, q_2, q_3\}$. Configuration $\{3 \cdot q_1\}$ can reach $\{2 \cdot q_1, q_3\}$ in two steps. First, a process broadcasts a, the two other processes receive it and move to q_2. Then, one of the processes in q_2 broadcasts b and moves to q_1, while the other one receives b and moves to q_3. Notice that $\{q_3\}$ is only coverable from a configuration $\{k \cdot q_1\}$ if $k \geq 3$.

2.3   Cubes and Counting Sets

Given a finite set Q, a *cube* $\mathcal{C}$ is a subset of $\mathbb{M}(Q)$ described by a lower bound $L: Q \to \mathbb{N}$ and an upper bound $U: Q \to \mathbb{N} \cup \{\infty\}$ such that $\mathcal{C} = \{C : L \leq C \leq$

U}. Abusing notation, we identify the set $\mathcal{C}$ with the pair (L, U). Notice that since $U(q)$ can be ∞ for some state q, a cube can contain an infinite number of configurations. All the results in this paper are true irrespective of whether the constants in a given input cube are encoded in unary or binary.

A finite union of cubes $\bigcup_{i=1}^{m}(L_i, U_i)$ is called a *counting constraint* and the set of configurations $\bigcup_{i=1}^{m} \mathcal{C}_i$ it describes is called a *counting set*. Notice that two different counting constraints may describe the same counting set. For example, let $Q = \{q\}$ and let $(L, U) = (1, 3)$, $(L', U') = (2, 4)$, $(L'', U'') = (1, 4)$. The counting constraints $(L, U) \cup (L', U')$ and (L'', U'') define the same counting set. It is easy to show (see also Proposition 2 of [11]) that counting constraints and counting sets are closed under Boolean operations.

Norms. Let $\mathcal{C} = (L, U)$ be a cube. Let $\|\mathcal{C}\|_l$ be the the sum of the components of L. Let $\|\mathcal{C}\|_u$ be the sum of the finite components of U if there are any, and 0 otherwise. The *norm* of $\mathcal{C}$ is the maximum of $\|\mathcal{C}\|_l$ and $\|\mathcal{C}\|_u$, denoted by $\|\mathcal{C}\|$. We define the norm of a counting constraint $\Gamma = \bigcup_{i=1}^{m} \mathcal{C}_i$ as $\|\Gamma\| \stackrel{\text{def}}{=} \max_{i \in [1,m]} \{\|\mathcal{C}_i\|\}$. The norm of a counting set $\mathcal{S}$ is the smallest norm of a counting constraint representing $\mathcal{S}$, that is, $\|\mathcal{S}\| \stackrel{\text{def}}{=} \min_{\mathcal{S} = [\![\Gamma]\!]} \{\|\Gamma\|\}$. Proposition 5 of [11] entails the following results for the norms of the union, intersection and complement.

Proposition 1. *Let $\mathcal{S}_1, \mathcal{S}_2$ be counting sets. The norms of the union, intersection and complement satisfy:* $\|\mathcal{S}_1 \cup \mathcal{S}_2\| \leq \max\{\|\mathcal{S}_1\|, \|\mathcal{S}_2\|\}$, $\|\mathcal{S}_1 \cap \mathcal{S}_2\| \leq \|\mathcal{S}_1\| + \|\mathcal{S}_2\|$, *and* $\|\overline{\mathcal{S}_1}\| \leq |Q| \cdot \|\mathcal{S}_1\| + |Q|$.

Reachability. The reachability problem can be generalized to the *cube-reachability* problem which consists of deciding, given an RBN $\mathcal{R}$ and two cubes $\mathcal{C}, \mathcal{C}'$, whether there exists configurations $C \in \mathcal{C}$ and $C' \in \mathcal{C}'$ such that C' is reachable from C in $\mathcal{R}$. If this is the case, we say $\mathcal{C}'$ is reachable from $\mathcal{C}$. The *counting set-reachability* problem asks, given an RBN $\mathcal{R}$ and two counting sets $\mathcal{S}, \mathcal{S}'$, whether there exists cubes $\mathcal{C} \in \mathcal{S}$ and $\mathcal{C}' \in \mathcal{S}'$ such that $\mathcal{C}'$ is reachable from $\mathcal{C}$ in $\mathcal{R}$. We define *cube-coverability* and *counting set-coverability* in an analoguous way.

Remark 1. In the paper [8], the authors define a sub-class of the cube-reachability problem, which is called the *unbounded initial cube-reachability* problem in [3]. More precisely, the sub-class considered in [8] is the following: We are given an RBN and two cubes $\mathcal{C} = (L, U)$ and $\mathcal{C}' = (L', U')$ with the special property that $L(q) = 0$ and $U(q) \in \{0, \infty\}$ for every state q. We then have to decide if $\mathcal{C}$ can reach $\mathcal{C}'$. This problem was shown to be PSPACE-complete ([8], Theorem 5.5), whenever the numbers in the input are given in unary. As we shall show later in this paper, the cube-reachability problem itself is in PSPACE, even when the input numbers are encoded in binary, thereby generalizing the upper bound results given in that paper.

3   Reachability sets of counting sets

In this section, we set the stage for proving the main result of this paper. This main result is given in two stages: First, we show that given a RBN with state set Q and a counting set $\mathcal{S}$, the set $post^*(\mathcal{S})$ is also a counting set and $\|post^*(\mathcal{S})\| \leq 2^{p(\|\mathcal{S}\| \cdot |Q|)}$ where p is some fixed polynomial. Using this, we then prove that a host of cube-parameterized problems for RBN can be solved in PSPACE.

The rest of this section is organized as follows: To prove the first result, we recall the notion of a *symbolic graph* of a RBN from [8]. In the symbolic graph, each node is a *symbolic configuration* of the RBN, which intuitively represents an infinite set of configurations in which the number of agents is fixed in some states, and arbitrarily big in the others. Next, by exploiting the special structure of the symbolic graph, we prove some properties which allow us to show that whenever two nodes in this graph are reachable, they are reachable by a path having a special structure. Finally, using these properties and the connection between symbolic configurations and configurations of the RBN, we prove the desired first result. Once we have shown the first result, we then show how the PSPACE Theorem can be obtained from it.

Throughout this section, we fix an RBN $\mathcal{R} = (Q, \Sigma, \delta)$.

3.1   Symbolic graph

In this subsection, we recall the notion of a *symbolic graph* of an RBN from [8]. Here, for the sake of convenience, we define it in a slightly different way, but the underlying notion is the same as [8]. Throughout this subsection and the next, we fix a number $k \in \mathbb{N}$.

The *symbolic graph of index k* associated with the RBN $\mathcal{R}$ is an edge-labelled graph $\mathcal{G}_k = (N, E, L)$ where $N = \mathbb{M}_k(Q) \times 2^Q$ is the set of nodes. Here $\mathbb{M}_k(Q)$ denotes the set of multisets on Q of size at most k. E is the set of edges and $L : E \to \Sigma$ is the labelling function. Each node of $\mathcal{G}_k$ is also called a *symbolic configuration*. Intuitively, in each symbolic configuration (v, S), the multiset v (called the *concrete part*) is used to keep track of a fixed set of at most k agents, and the subset S (called the *abstract part*) is used to keep track of the *support* of the remaining agents.

Let $\theta = (v, S)$ and $\theta' = (v', S')$ be two symbolic configurations. There is an edge labelled by a between θ and θ' if and only if the following is satisfied: There exists a transition $(q, !a, q') \in \delta$ such that at least one of the following two conditions holds

- *(Broadcast from v)* There exists a multiset of transitions $\wr(p_1, ?a, p_1'), \ldots, (p_l, ?a, p_l')\wr$ such that $v' = v - \sum_i \boldsymbol{p_i} + \sum_i \boldsymbol{p_i'} - \boldsymbol{q} + \boldsymbol{q'}$, and for each $q_s \in Q$:
  - If $q_s \in S' \setminus S$ then there exists $q_s' \in S$ and $(q_s', ?a, q_s) \in R$,
  - If $q_s \in S \setminus S'$ then there exists $q_s' \in S'$ and $(q_s, ?a, q_s') \in R$.
- *(Broadcast from S)* There exists a multiset of transitions $\wr(p_1, ?a, p_1'), \ldots, (p_l, ?a, p_l')\wr$ such that $v' = v - \sum_i \boldsymbol{p_i} + \sum_i \boldsymbol{p_i'}$, $q \in S, q' \in S'$, and for each $q_s \in Q \setminus \{q, q'\}$:

- if $q_s \in S' \setminus S$ then there exists $q'_s \in S$ and $(q'_s, ?a, q_s) \in R$,
- if $q_s \in S \setminus S'$ then there exists $q'_s \in S'$ and $(q_s, ?a, q'_s) \in R$.

An edge labelled by a between θ and θ' is denoted by $\theta \rightsquigarrow^a_{\mathcal{G}_k} \theta'$. The relation $\rightsquigarrow^*_{\mathcal{G}_k}$ is the reflexive and transitive closure of $\rightsquigarrow_{\mathcal{G}_k} := \cup_{a \in \Sigma} \rightsquigarrow^a_{\mathcal{G}_k}$. Whenever the index k is clear, we will drop the subscript $\mathcal{G}_k$ from these notations.

Remark 2. Let $\theta = (v, S), \theta' = (v', S')$ be two symbolic configurations. By construction, θ can only reach θ' if $|v| = |v'|$.

To give an intuition behind the edges in $\mathcal{G}_k$, recall the intuition that in a symbolic configuration, the concrete part is used to keep track of a fixed set of at most k processes and the abstract part is used to keep track of the support of the remaining processes. The first condition for the existence of an edge asserts the following: 1) In the concrete part, some process broadcasts the message a and some subset of processes receive a, 2) In the abstract part, any new state added or any old state deleted comes because of receiving a. The second condition asserts exactly the same, except we now require the process broadcasting the message a to be from the abstract part.

The symbolic graph of index k can be thought of as an abstraction of the set of configurations of $\mathcal{R}$, where only a fixed number of processes are explicitly represented and the rest are abstracted by means of their support alone. To formalize this, given a symbolic configuration $\theta = (v, S)$, we let $[\![\theta]\!]$ denote the following (infinite) set of configurations: $C \in [\![\theta]\!]$ if and only if $C(q) = v(q)$ for $q \notin S$ and $C(q) \geq v(q)$ for $q \in S$.
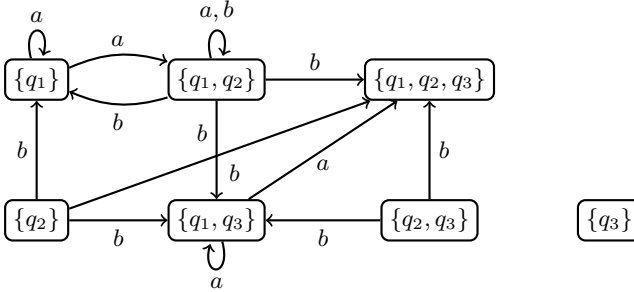
Fig. 2. Symbolic graph $\mathcal{G}_0$ of index 0 of the RBN of Example 1.

Example 2. The symbolic graph $\mathcal{G}_0$ of index 0 of the RBN of Example 1 is illustrated in Figure 2. At this index, the graph only keeps track of a subset $S \subseteq Q$, and the edges correspond to broadcasts from S. Consider the edges from $\{q_1\}$. The self-loop corresponds to a broadcast of a that is not received. The edge to $\{q_1, q_2\}$ corresponds to a broadcast of a received by at least one process

in q_1. There is no edge from $\{q_3\}$ because there is no broadcast transition from q_3.

We then have the following lemma, which asserts that runs between two configurations in an RBN induce corresponding runs in the symbolic graph. The proof of the lemma is easily obtained from the definition of the symbolic graph.

Lemma 1. *Let C, C' be two configurations of $\mathcal{R}$ such that $C \xrightarrow{a} C'$. Then, for every θ such that $C \in [\![\theta]\!]$, there exists θ' such that $C' \in [\![\theta']\!]$ and $\theta \rightsquigarrow^a \theta'$.*

3.2   Properties of the symbolic graph

In this subsection, we prove some properties of the symbolic graph (of any index k). The first two properties that we prove exhibit some structural properties on the paths of the symbolic graph. The next two properties relate paths over the symbolic graph to runs over the configurations of the given RBN. These four properties will ultimately lead us to prove our two main contributions in the next section.

First property: Monotonicity. Let $k \in \mathbb{N}$ and let $\mathcal{G}_k$ be the symbolic graph of index k associated with $\mathcal{R}$. The first key property of $\mathcal{G}_k$ is the following property, which we call *monotonicity*.

Proposition 2. *Let $\theta = (v, S)$ and $\theta' = (v', S')$ be symbolic configurations of $\mathcal{G}_k$. Then the following are true:*

- *If $Z \subseteq S$ and $\theta \rightsquigarrow^a \theta'$, then $(v, S) \rightsquigarrow^a (v', Z \cup S')$.*
- *If $Z \subseteq Q$ and $\theta \rightsquigarrow^a \theta'$, then $(v, Z \cup S) \rightsquigarrow^a (v', Z \cup S')$.*

Proof. The two points follow immediately from the definition of $\rightsquigarrow^a$. $\qquad \square$

Second property: Normal Form. To state the second property, we first need a small definition.

Definition 2. *Let $(v_0, S_0) \rightsquigarrow \cdots \rightsquigarrow (v_m, S_m)$ a path in $\mathcal{G}_k$. A pair of indices $0 \le i < j \le m$ is called a* bad pair *if $(S_i \setminus S_{i+1}) \cap S_j \ne \emptyset$. A path is said to be in* normal form *if it contains no bad pairs, i.e., for all $0 \le i < m$ and any $j > i$, $(S_i \setminus S_{i+1}) \cap S_j = \emptyset$.*

Intuitively, a path is in normal form if during each step, the states that disappear from the abstract part never reappear again. The following lemma asserts that whenever there is a path between two symbolic configurations, then there is a path between them that is in normal form.

Lemma 2. *Let θ, θ' be symbolic configurations of $\mathcal{G}_k$ such that there is a path between θ and θ' of length m. Then, there is a path in normal form between θ and θ' of length m.*

Proof Sketch. Let $\theta = \theta_0 \rightsquigarrow \theta_1 \rightsquigarrow \theta_2 \rightsquigarrow \ldots \theta_{m-1} \rightsquigarrow \theta_m = \theta'$ be the path between θ and θ'. We proceed by induction on m. The claim is clearly true for $m = 0$. Suppose $m > 0$ and the claim is true for $m - 1$. By induction hypothesis, we can assume that the path $\theta_0 \rightsquigarrow \theta_1 \rightsquigarrow \ldots \rightsquigarrow \theta_{m-1}$ is already in normal form.

Let each $\theta_i = (v_i, S_i)$. Let l be the number of bad pairs in the path between θ_0 and θ_m. If $l = 0$, then the path is already in normal form and we are done. Suppose $l > 0$ and let (w, w') be a bad pair. Since the path between θ_0 and θ_{m-1} is already in normal form, it has to be the case that $w' = m$. Hence, we have $Z := (S_w \setminus S_{w+1}) \cap S_m \neq \emptyset$.

By Proposition 2, the following is a valid path: $(v_w, S_w) \rightsquigarrow (v_{w+1}, S_{w+1} \cup Z) \rightsquigarrow (v_{w+2}, S_{w+2} \cup Z) \ldots (v_{m-1}, S_{m-1} \cup Z) \rightsquigarrow (v_m, S_m \cup Z) = (v_m, S_m)$. Let $\theta'_j := \theta_j$ if $j \leq w$ and $(v_j, S_j \cup Z)$ otherwise. Hence, we get a path $\theta'_0 \rightsquigarrow \theta'_1 \rightsquigarrow \ldots \theta'_{m-1} \rightsquigarrow \theta'_m$.

Let each $\theta'_e = (v'_e, S'_e)$ and let $0 \leq i < j \leq m-1$. By a case analysis on where i and j are relative to the index w, we can prove that $(S'_i \setminus S'_{i+1}) \cap S'_j = \emptyset$. Having proved this, it is then clear by construction, that this new path from $\theta'_0 := \theta_0$ to $\theta'_m := \theta_m$ has at most $l - 1$ bad pairs only. Hence, we now have a path from θ_0 to θ_m such that the prefix of length $m - 1$ is in normal form and the number of bad pairs has been strictly reduced to $l - 1$. Repeatedly applying this procedure leads to a path in normal form between θ_0 and θ_m. $\square$

Third property: Refinement. Before we state the third property, we need a small definition. Recall that, given a symbolic configuration $\theta = (v, S)$, the set $\llbracket \theta \rrbracket$ denotes the set of configurations C such that $C(q) = v(q)$ if $q \notin S$ and $C(q) \geq v(q)$ otherwise. The following definition refines the set $\llbracket \theta \rrbracket$.

Definition 3. *Given a symbolic configuration $\theta = (v, S)$ and a number $N \in \mathbb{N}$, let $\llbracket \theta \rrbracket_N$ denote the set of configurations C such that $C(q) = v(q)$ if $q \notin S$ and $C(q) \geq v(q) + N$ otherwise. Note that $\llbracket \theta \rrbracket = \llbracket \theta \rrbracket_0$.*

This definition along with the above two properties now enable us to prove the third property. It roughly states that if a symbolic configuration θ' can be reached from another symbolic configuration θ, then there is a "small" N such that any configuration in $\llbracket \theta' \rrbracket_N$ can be reached from some configuration in $\llbracket \theta \rrbracket$.

Theorem 1. *Let θ, θ' be symbolic configurations of $\mathcal{G}_k$ such that $\theta \rightsquigarrow^* \theta'$. Then there exists $N \leq k \times (2k)^{|Q|} \times (|Q| + 1)^{|Q|+1} + 1$ such that for all $C' \in \llbracket \theta' \rrbracket_N$, there exists $C \in \llbracket \theta \rrbracket$ such that $C \xrightarrow{*} C'$.*

Proof Sketch. Suppose $\theta \rightsquigarrow^* \theta'$. If the length of the path is 0, then there is nothing to prove. Hence, we restrict ourselves to the case when the length of the path is bigger than 0. By Lemma 2, there is a path in normal from from θ to θ' (say) $\theta = \theta_0 \rightsquigarrow \theta_1 \rightsquigarrow \theta_2 \ldots \theta_{m-1} \rightsquigarrow \theta_m = \theta'$ with each $\theta_i := (v_i, S_i)$.

Let $N_0 = 0$ and let $N_i = (N_{i-1} + 1) \cdot (|S_{i-1} \setminus S_i| + 1)$ for every $1 \leq i \leq m$. In Lemma 5.3 of [8] (more precisely in its proof, in Lemma 6 of the long version [9]), the following fact has been proved:

For every $1 \leq i \leq m$ and for every $C' \in [\![\theta_i]\!]_{N_i+1}$, there exists $C \in [\![\theta_{i-1}]\!]_{N_{i-1}+1}$ such that $C \xrightarrow{*} C'$.

This immediately proves that for all $C' \in [\![\theta']\!]_{N_m+1}$, there exists $C \in [\![\theta]\!]$ such that $C \xrightarrow{*} C'$. If we prove $N_m \leq k \times (2k)^{|Q|} \times (|Q|+1)^{|Q|+1}$, then the proof of the theorem will be complete.

Notice that if $(v, \emptyset) \rightsquigarrow (v', S')$ is an edge in $\mathcal{G}_k$ then $S' = \emptyset$. This fact, along with the definition of a path in normal form, allows us to easily conclude that the number of indices i such that $|S_{i-1} \setminus S_i| > 0$ is at most $|Q|$. It then follows that except for at most $|Q|$ indices, each index N_i is obtained from N_{i-1} by simply adding 1 and in the remaining indices, N_i is obtained from N_{i-1} by adding 1 and then multiplying by a number which is at most $|Q| + 1$. Using this, we can deduce that the maximum value for N_m is at most $(m - |Q| + 1)|Q|(|Q|+1)^{|Q|}$. Since m is itself the length of the path between θ_0 and θ_m, m is upper bounded by the number of symbolic configurations in $\mathcal{G}_k$ which is at most $k \times k^{|Q|} \times 2^{|Q|}$. Overall we get that $N_m \leq k \times (2k)^{|Q|} \times (|Q|+1)^{|Q|+1}$. □

Remark 3. A similar result was proved in Lemma 5.3 of [8], but there it was just stated that there exists an N satisfying this property. Moreover from the proof of that lemma, only a doubly exponential bound on N could be inferred.

Fourth property: Compatibility. To describe the fourth property, we need the following notion of order on configurations, *relative* to a given symbolic configuration.

Definition 4. *Let $\theta = (v, S)$ be a symbolic configuration, and let C, C' be two configurations of $\mathcal{R}$. We define an order $\preceq_\theta$ such that $C \preceq_\theta C'$ if and only if $C, C' \in [\![\theta]\!]$, and $\forall q \in S, C(q) \leq C'(q)$.*

This definition enables us to state our next property, which we dub *compatibility*. It intuitively says that the order that we have defined is, in some sense, compatible with the edges of the symbolic configurations.

Lemma 3. *Let θ be a symbolic configuration of $\mathcal{G}_k$, and let C, C' be two configurations of $\mathcal{R}$. If $C \in [\![\theta]\!]$ and $C \xrightarrow{*} C'$, then there exists a symbolic configuration θ' such that 1) $C' \in [\![\theta']\!]$, 2) $\theta \rightsquigarrow^* \theta'$ and 3) for all C'_1 such that $C'_1 \succeq_{\theta'} C'$, there exists $C_1 \in [\![\theta]\!]$ such that $C_1 \xrightarrow{*} C'_1$.*

Proof. Let θ be a symbolic configuration and C, C' be configurations such that $C \in [\![\theta]\!]$ and $C \xrightarrow{*} C'$. Let $C = C_0 \rightarrow \cdots \rightarrow C_{m-1} \rightarrow C_m = C'$ denote the run between C and C'. We prove the property by induction on m. For $m = 0$, we have $C = C'$. The property is easily seen to hold with $\theta' = \theta$.

Suppose now that $m \geq 1$, and that the property holds for all $n \leq m$. By induction hypothesis, for the configuration C_{m-1}, there exists a symbolic configuration θ_{m-1} satisfying the property, in particular $\theta \rightsquigarrow^* \theta_{m-1}$. Since $C_{m-1} \xrightarrow{a} C_m$ for some $a \in \Sigma$, by Lemma 1, there exists a symbolic configuration θ_m such that $C_m \in [\![\theta_m]\!]$, and $\theta_{m-1} \rightsquigarrow^a \theta_m$. Using $\theta \rightsquigarrow^* \theta_{m-1}$, we obtain that $\theta \rightsquigarrow^* \theta_m$.

Let $\theta_{m-1} = (v_{m-1}, S_{m-1})$ and $\theta_m = (v_m, S_m)$. Let $C'_m \in [\![\theta_m]\!]$ be such that $C'_m \succeq_{\theta_m} C_m$. We will construct a configuration $C'_{m-1} \in [\![\theta_{m-1}]\!]$ such that $C'_{m-1} \succeq_{\theta_{m-1}} C_{m-1}$ and $C'_{m-1} \xrightarrow{*} C'_m$. If we construct such a configuration, then by induction hypothesis, there is a $C_1 \in [\![\theta]\!]$ such that $C_1 \xrightarrow{*} C'_{m-1} \xrightarrow{*} C'_m$, which will conclude the proof.

Let $C'_{m-1}(q) = C_{m-1}(q)$ for all $q \notin S_{m-1}$. To define C'_{m-1} on S_{m-1}, we first define a mapping $pred$ from states in S_m to states of $S_{m-1} \cup \overline{S_{m-1}} = Q$ as follows. Given $q' \in S_m$:

- If $q' \in S_{m-1}$, $pred(q') = q'$;
- If $q' \notin S_{m-1}$, by definition of edges in the symbolic graph, there exists $q \in S_{m-1}$ such that $(q, ?a, q')$ is a transition. Then $pred(q') = q$ for one (arbitrary but fixed) such q.

By definition, $C'_m(q) = C_m(q)$ for all $q \notin S_m$. For all $q \in S_m$, let $n_q = C'_m(q) - C_m(q)$. Intuitively, we want to place these n_q processes in the right places of C'_{m-1} so that $C'_{m-1} \to C'_m$. For all $q \in S_{m-1}$, let $C'_{m-1}(q) = C_{m-1}(q) + \sum_{q' \in S_m, pred(q')=q} n_{q'}$. By definition, $C'_{m-1} \succeq_{\theta_{m-1}} C_{m-1}$. So all that remains is to prove that $C'_{m-1} \xrightarrow{*} C'_m$.

Let $C_{m-1} \xrightarrow{t+t_1,\dots,t_n} C_m$ where $t = (p, !a, p')$ and each $t_i = (p_i, ?a, p'_i)$. If we let $S_m \setminus S_{m-1} = \{q'_1, \dots, q'_w\}$, then by definition there is a transition $t'_i := (pred(q'_i), ?a, q'_i)$ for each i. Additionally, $C'_{m-1}(pred(q'_i)) \geq C_{m-1}(pred(q'_i)) + n_{q'_i}$. This allows us to do $C'_{m-1} \xrightarrow{t+t_1,\dots,t_n, n_{q'_1} \cdot t'_1, n_{q'_2} \cdot t'_2, \dots, n_{q'_w} \cdot t'_w} C'_m$, which concludes the proof.

$\square$

4   The PSPACE Theorem

In this section, we prove our two main contributions. First, we show that given a cube $\mathcal{C}$, $post^*(\mathcal{C})$ is a counting set of bounded size. Using this, we show our main result: any boolean combination of atoms can be evaluated in PSPACE, where an atom is a counting set or the reachability set of a counting set. We call this the PSPACE *Theorem*. The intuition behind the PSPACE Theorem is that the norms of the counting sets obtained by such combinations are "small", and so we only need to examine small configurations to verify them, thus yielding a PSPACE algorithm for checking correctness. In particular, the PSPACE Theorem will show that the cube-reachability problem is in PSPACE. We fix an arbitrary RBN $\mathcal{R} = (Q, \Sigma, \delta)$ for the rest of the section.

We start by drawing links between cubes and symbolic configurations.

- Given a symbolic configuration $\theta = (v, S)$, we let $\mathcal{C}_\theta$ be the cube (L, U) where $L = v$, and $U(q) = v(q)$ if $q \notin S$ and $U(q) = \infty$ otherwise. Then $\mathcal{C}_\theta = [\![\theta]\!]$.
- Given a cube $\mathcal{C} = (L, U)$, we define $\Delta_\mathcal{C}$ to be the set of symbolic configurations $\theta = (v, S)$ with $S = \{q \mid U(q) = \infty\}$ and $L(q) \leq v(q) \leq U(q)$ if $q \notin S$ and $v(q) = L(q)$ otherwise. Then $[\![\Delta_\mathcal{C}]\!] = \mathcal{C}$.

Notice that the set $\Delta_\mathcal{C}$ is included in the symbolic graph of index $2\|\mathcal{C}\|$. Indeed, if $\mathcal{C} = (L, U)$ and $(v, S) \in \Delta_\mathcal{C}$, then $|v| \leq |L| + |U_f|$ where $U_f(q) = 0$ if $U(q) = \infty$ and $U_f(q) = U(q)$ otherwise. Since $\|\mathcal{C}\| = \max(|L|, |U_f|)$, we have the desired result. By Remark 2, we know that symbolic configurations in the graph of index $2\|\mathcal{C}\|$ can only reach symbolic configurations which are also in the graph of index $2\|\mathcal{C}\|$.

Lemma 4. *Given a cube $\mathcal{C}$, the sets $\Delta_\mathcal{C}$ and $post^*(\Delta_\mathcal{C})$ are included in the symbolic graph of index $2\|\mathcal{C}\|$.*

There are only a finite number of symbolic configurations in the graph of a given index. Therefore $post^*(\Delta_\mathcal{C})$ is a finite set of symbolic configurations θ. It follows that $[\![post^*(\Delta_\mathcal{C})]\!]$ is the finite union of the cubes $\mathcal{C}_\theta$, and thus a counting set.

Unfortunately, it is in general not the case that $post^*(\mathcal{C}) = [\![post^*(\Delta_\mathcal{C})]\!]$, which would close our argument. However, we will show that for each symbolic configuration θ in $post^*(\Delta_\mathcal{C})$, there is a counting set $\mathcal{S}_\theta \subseteq [\![\theta]\!]$ such that the finite union of these counting sets is equal to $post^*(\mathcal{C})$. This will then show our first important result, namely that the reachability set of a counting set is also a counting set with "small" norm.

Theorem 2. *Let $\mathcal{C}$ be a cube. Then $post^*(\mathcal{C})$ is a counting set and*

$$\|post^*(\mathcal{C})\| \in O((\|\mathcal{C}\| \cdot |Q|)^{|Q|+2})$$

The same holds for pre^* by using the given RBN with reversed transitions.

Proof. We start by defining a counting set $\mathcal{M}$ of configurations, which we will then prove to be equal to $post^*(\mathcal{C})$. Given a symbolic configuration θ of $post^*(\Delta_\mathcal{C})$, we define the set $\min(\theta, \mathcal{C})$ to be the set of configurations $C \in [\![\theta]\!]$ such that C is minimal for the order $\preceq_\theta$ over the configurations of $post^*(\mathcal{C})$, i.e.

$$\min(\theta, \mathcal{C}) = \min_{\preceq_\theta} \{C \in [\![\theta]\!] \mid C \in post^*(\mathcal{C})\}$$

We can now define $\mathcal{M}$ to be the following set

$$\mathcal{M} = \bigcup_{\theta \in post^*(\Delta_\mathcal{C})} \bigcup_{C \in \min(\theta, \mathcal{C})} \mathcal{C}_C^\theta,$$

where $\mathcal{C}_C^\theta$ is the cube $\mathcal{C}_{(C,S)}$ for S such that $\theta = (v, S)$. Since $\mathcal{M}$ is a finite union of cubes, it is a counting set.

We show that $post^*(\mathcal{C}) \subseteq \mathcal{M}$. Let $C \in post^*(\mathcal{C})$. There exists $C_0 \in \mathcal{C}$ such that $C_0 \xrightarrow{*} C$, and there exists $\theta_0 \in \Delta_\mathcal{C}$ such that $C_0 \in [\![\theta_0]\!]$. Applying Lemma 1, we obtain the existence of $\theta \in post^*(\theta_0) \subseteq post^*(\Delta_\mathcal{C})$ such that $C \in [\![\theta]\!]$. Now, there exists a configuration $C' \in \min(\theta, \mathcal{C})$ such that $C' \preceq_\theta C$. By definition of $\mathcal{C}_{C'}^\theta$, C is in $\mathcal{C}_{C'}^\theta$ and thus in $\mathcal{M}$.

Now we show that $\mathcal{M} \subseteq post^*(\mathcal{C})$. Let $C \in \mathcal{M}$. By definition, there must be a symbolic configuration $\theta \in post^*(\Delta_\mathcal{C})$ and a configuration $C' \in post^*(\mathcal{C})$ such

that $C' \preceq_\theta C$. By the Compatibility Lemma (Lemma 3), C is in $post^*(\mathcal{C})$ as well.

All that remains is to bound the norm of $\mathcal{M}$. To do this, let $\theta = (v, S) \in post^*(\Delta_\mathcal{C})$ and let $C \in \min(\theta, \mathcal{C})$. If we bound the norm of $\mathcal{C}_C^\theta$ by the desired quantity, then the proof will be complete. Noticing that $\|\mathcal{C}_C^\theta\| = |C|$, it suffices to bound $|C|$ by the desired quantity, which is what we shall do now.

By Theorem 1 and Lemma 4, there exists an $N \leq 2\|\mathcal{C}\| \times (4\|\mathcal{C}\|)^{|Q|} \times (|Q| + 1)^{|Q|+1}$ such that $[\![post^*(\Delta_\mathcal{C})]\!]_N \subseteq post^*([\![\Delta_\mathcal{C}]\!]) = post^*(\mathcal{C})$. By definition of C, there must be a smallest N' such that $C(q) \leq v(q) + N'$ for every state q. If $N' > N$, then let C_N be the configuration given by $C_N(q) = \min(C(q), v(q)+N)$. We get that $C_N \in [\![\theta]\!]_N \subseteq [\![post^*(\Delta_\mathcal{C})]\!]_N \subseteq post^*(\mathcal{C})$, and so $C_N \preceq_\theta C$ and $C_N \in post^*(\mathcal{C})$, which is a contradiction to the minimality of C. Hence $N' \leq N$ and so $|C| \leq |v| + |Q| \cdot N$. Since $\theta = (v, S)$ is in $post^*(\Delta_\mathcal{C})$, by Lemma 4, we have that $|v| \leq 2\|\mathcal{C}\|$. Substituting the upper bounds for $|v|$ and N in the inequality $|C| \leq |v| + |Q| \cdot N$ then gives the required upper bound for $|C|$, thereby finishing the proof.

This result also holds for $pre^*(\mathcal{C})$. If $\mathcal{R} = (Q, \Sigma, R)$ is the given RBN, consider the "reverse" RBN $\mathcal{R}_r$, defined as $\mathcal{R} = (Q, \Sigma, R_r)$ where R_r has a transition $(q, \star a, q')$ for $\star \in \{!, ?\}$ iff R_r has a transition $(q', \star a, q)$. Notice that $\mathcal{R}_r$ is still an RBN and that $post^*(\mathcal{C})$ in $\mathcal{R}$ is equal to $pre^*(\mathcal{C})$ in $\mathcal{R}_r$. □

Recall that counting sets are closed under boolean operations. With the above theorem, plus the fact that counting sets are finite unions of cubes, we obtain the following closure result.

Corollary 1 (Closure). *Counting sets are closed under $post^*$, pre^* and boolean operations.*

We are now ready to show our main result, the PSPACE Theorem. We show that there exist PSPACE algorithms to evaluate boolean combinations over counting sets and reachability set of counting sets. This result and its proof are adapted from a similar result for population protocols in [12].

Given a counting constraint Γ, we let $[\Gamma]$ denote the counting set described by Γ. To state our result, we first define some "nice" expressions.

Definition 5. *A* nice *expression is any expression that is constructed by the following syntax:*

$$E := \Gamma \mid post^*(\Gamma) \mid pre^*(\Gamma) \mid E \cap E \mid E \cup E \mid \overline{E}$$

where Γ is any counting constraint.

If E is a nice expression, then the size *of E, denoted by $|E|$, is defined as follows:*

- *If $E = \Gamma$ or $post^*(\Gamma)$ or $pre^*(\Gamma)$, then $|E| = 1$;*
- *If $E = E_1 \cup E_2$ or $E = E_1 \cap E_2$, then $|E| = |E_1| + |E_2|$;*
- *If $E = \overline{E_1}$, then $|E| = |E_1| + 1$.*

The set of configurations that is described by a nice expression E can be defined in a straightforward manner, and is denoted as $[E]$.

Notice that any nice expression E is a counting constraint, and $[E]$ is a counting set, by the Closure Corollary 1.

Theorem 3 (PSPACE Theorem). *Let E be a nice expression and let N be the maximum norm of the counting constraints appearing in E. Then* $[E]$ *is a counting set of norm at most exponential in* $N, |E|$ *and* $|Q|$. *Further, the membership and emptiness problems for* $[E]$ *are in* PSACE.

Proof. Recall that $[E]$ is a counting set , by the Closure Corollary (Corollary 1). The exponential bounds for the norms follow immediately from Proposition 1 and Theorem 2. The membership complexity for union, intersection and complement is easy to see. Without loss of generality it suffices to prove that membership in $post^*(\Gamma)$ is in PSPACE, where Γ is a counting constraint.

By Savitch's Theorem NPSPACE=PSPACE, so we provide a nondeterministic algorithm. Given (C, Γ), we want to decide whether $C \in post^*(\Gamma)$. The algorithm first guesses a configuration $C_0 \in \Gamma$ of the same size as C, verifies that C_0 belongs to Γ, and then simply guesses an execution starting at C_0, step by step. The algorithm stops if either the configuration reached at some step is C, or if it has guessed more steps than the number of configurations of size $|C|$. This concludes the discussion regarding the membership complexity.

To see that checking emptiness of E is in PSPACE, notice that if E is nonempty, then it has an element of size at most $\|E\|$. We can guess such an element C in polynomial space (by representing each coefficient in binary), and verify that C is indeed in E by means of the PSPACE membership algorithm. $\square$

This result is a powerful tool which can be used to prove that a host of problems are in PSPACE for RBN. For instance, the cube-reachability problem for cubes $\mathcal{C}$ and $\mathcal{C}'$ is just checking if $post^*(\mathcal{C}) \cap \mathcal{C}'$ is empty, which by the PSPACE Theorem can be done in PSPACE. Combining this with Remark 1, we obtain the following result.

Theorem 4. *Cube-reachability is* PSPACE-*complete for RBN.*

By the reduction given in Section 4.2 of [3], this result also proves that cube-reachability is PSPACE-complete for asynchronous shared-memory systems (ASMS), which is another model of distributed computation where agents communicate by a shared register. Due to lack of space, we defer a discussion of this result to the appendix.

We will demonstrate further applications of the PSPACE Theorem in the next section.

5   Application 1: Almost-sure coverability

Having presented our PSPACE Theorem and the closure property for reachability sets of counting sets, we now provide two applications. For the first one, we

consider the *almost-sure coverability* problem for RBN. Using our new results, we prove that this problem is PSPACE-complete.

The rest of the section is as follows: We first recall the definition of the almost-sure coverability problem, give a characterization of it in terms of counting sets and then prove PSPACE-completeness. Throughout this section, we fix a RBN $\mathcal{R} = (Q, \Sigma, \delta)$ with two special states $init, fin \in Q$, which will respectively be called the initial and final states.

5.1 The almost-sure coverability problem

Let $\uparrow fin$ denote the set of all configurations C of $\mathcal{R}$ such that $C(fin) \geq 1$. For any $k \geq 1$, we say that the configuration $\wr k \cdot init \wr$ *almost-surely covers fin* if and only if $post^*(\wr k \cdot init \wr) \subseteq pre^*(\uparrow fin)$. The reason behind calling this the almost-sure coverability relation is that the definition given here is equivalent to covering the state fin from $\wr k \cdot init \wr$ with probability 1 under a probabilistic scheduler which picks agents uniformly at random at each step.

The number k is called a *cut-off* if one of the following is true: Either, 1) for all $h \geq k$, the configuration $\wr h \cdot init \wr$ almost-surely covers fin, in which case k is called a *positive* cut-off; or, 2) for all $h \geq k$, the configuration $\wr h \cdot init \wr$ does not almost-surely cover fin, in which case k is called a *negative* cut-off. The following was proved in Theorem 9 of [3].

Theorem 5. *Given an RBN with two states init, fin, a cut-off always exists. Whether the cut-off is positive or negative can be decided in* EXPSPACE.

Our main result of this section is that

Theorem 6. *Deciding whether the cut-off of a given RBN is positive or negative is* PSPACE*-complete. Moreover, a given RBN always has a cut-off which is at most exponential in its number of states.*

5.2 A characterization of almost-sure coverability

We now rewrite the definition of almost-sure coverability in terms of counting sets. Let $[init]$ be the cube such that $L(q) = U(q) = 0$ if $q \neq init$ and $L(init) = 0, U(init) = \infty$. Notice that by definition, $\uparrow fin$ is a cube. We now consider the set of configurations defined by $\mathcal{S} := post^*([init]) \cap pre^*(\uparrow fin)$. By our PSPACE Theorem 3, $\mathcal{S}$ is a counting set such that the norm of $\mathcal{S}$ is at most $2^{p(|Q|)}$ for some fixed polynomial p. We now claim the following.

Theorem 7. $\mathcal{R}$ *has a positive cut-off if and only if* $\mathcal{S}$ *is finite. Moreover,* $|Q| \cdot |\mathcal{S}|$ *is an upper bound on the size of the cut-off for* $\mathcal{R}$ *and so* $\mathcal{R}$ *has a cut-off which is exponential in its number of states.*

Proof. Let N be the norm of $\mathcal{S}$. Suppose $\mathcal{S}$ is finite. If $C \in \mathcal{S}$, then $\sum_{q \in Q} C(q) \leq |Q| \cdot N$. So, if C is any configuration of size $h > |Q| \cdot N$ such that $C \in post^*(\wr h \cdot init \wr)$ then $C \in pre^*(\uparrow fin)$. Hence, $|Q| \cdot N$ is a positive cut-off for $\mathcal{R}$.

Suppose $\mathcal{S}$ is infinite, and let $\cup_i \mathcal{C}_i$ be a counting constraint for $\mathcal{S}$ whose norm is N. Then there must exist an index i with $\mathcal{C}_i := (L, U)$ and a state p such that $U(p) = \infty$. For each $h \geq N$, consider the configuration C_h given by $C_h(q) = L(q)$ if $q \neq p$ and $C_h(p) = h$. Notice that $C_h \in \mathcal{S}$ and so $C_h \in post^*([init]) \cap pre^*(\uparrow fin)$. Hence, for every $h \geq |Q| \cdot N$, we have exhibited a configuration of size h, reachable from $\langle h \cdot init \rangle$ but from which fin is not coverable. Thus N is a negative cut-off for $\mathcal{R}$. $\qquad\square$

Remark 4. Notice that we have shown that if $\mathcal{S}$ is finite, then $\mathcal{R}$ has a positive cut-off and if $\mathcal{S}$ is infinite, then $\mathcal{R}$ has a negative cut-off. This gives an alternative proof of the fact that a cut-off always exists for a given RBN.

5.3   PSPACE-completeness of the almost-sure coverability problem

Because of Theorem 7, we now have the following result.

Lemma 5. *Deciding whether the cut-off of a given RBN is positive or negative can be done in* PSPACE.

Proof Sketch. By Theorem 7, it follows that a given RBN has a negative cut-off iff $\mathcal{S} = post^*([init]) \cap \overline{pre^*(\uparrow fin)}$ is infinite. We have already seen that $\mathcal{S}$ is a counting set such that the norm of $\mathcal{S}$ is at most $N := 2^{p(|Q|)}$ for some fixed polynomial p.

Let $\cup_i \mathcal{C}_i$ be a counting constraint for $\mathcal{S}$ which minimizes its norm and let each $\mathcal{C}_i = (L_i, U_i)$. Hence, $L_i(q) \leq N$ for every state q. Further, $\mathcal{S}$ is infinite iff there is an index i and a state q such that $U_i(q) = \infty$. Using these two facts, we can then show that $\mathcal{S}$ is infinite iff there is a state q and a configuration $C \in \mathcal{S}$ such that $C(q') \leq N$ for every $q' \neq q$ and $C(q) = N + 1$.

Hence, to check if $\mathcal{S}$ is infinite, we just have to guess a state q and a configuration C such that $C(q') \leq N$ for every $q' \neq q$ and $C(q) = N + 1$ and check if $C \in \mathcal{S}$. Since guessing C can be done in polynomial space (by representing every number in binary), by the PSPACE Theorem (Theorem 3), we can check if $C \in \mathcal{S}$ in polynomial space as well, which concludes the proof of the theorem. $\qquad\square$

We also have the accompanying hardness result.

Lemma 6. *Deciding whether the cut-off of a given RBN is positive or negative is* PSPACE*-hard.*

Similar to the cube-reachability problem, our result on almost-sure coverability also applies to the related model of ASMS. This solves an open problem from [6]. For lack of space, we once again defer this discussion to the appendix.

6   Application 2: Computation by RBN

In this section we give another application of our results. We introduce a model of computation using RBN called *RBN protocols*. We take inspiration from the

extensively-studied model of population protocols [1,2,12]. The reader can consult the above references for more details on population protocols.

In our model, reconfigurable networks of identical, anonymous agents interact to compute a predicate $\varphi : \mathbb{N}^k \to \{0,1\}$. We show that RBN protocols compute exactly the threshold predicates, which we will define more formally below.

6.1   RBN Protocols

We introduce our computation model. The notation mimics that of [13].

Definition 6. *An* RBN protocol *is a tuple* $\mathcal{P} = (Q, \Sigma, \delta, I, O)$ *where* (Q, Σ, δ) *is an RBN,* $I = \{q_1, \ldots, q_k\}$ *is a set of* input states, *and* $O : Q \to \{0,1\}$ *is an* output function.

Configurations and *runs* of $\mathcal{P}$ are the same as that of the underlying RBN. A configuration C is called a *0-consensus* (respectively a *1-consensus*) if $C(q) > 0$ implies $O(q) = 0$ (respectively $O(q) = 1$). For $b \in \{0,1\}$, a b-consensus C is *stable* if every configuration reachable from C is also a b-consensus. A run $C_0 \to C_1 \to C_2 \cdots$ of $\mathcal{P}$ is *fair* if it is finite and cannot be extended by any step, or if it is infinite and the following condition holds for all configurations C, C': if $C \to C'$ and $C = C_i$ for infinitely many $i \geq 0$, then the step $C \to C'$ appears infinitely along the run. In other words, if a fair run reaches a configuration infinitely often, then all the configurations reachable in a step from that configuration will be reached infinitely often from it.

A fair run $C_0 \to C_1 \to \ldots$ *converges to* b if there is $i \geq 0$ such that C_j is a b-consensus for every $j \geq i$. For every $\boldsymbol{v} \in \mathbb{N}^k$, let $C_{\boldsymbol{v}}$ be the configuration given by $C_{\boldsymbol{v}}(q_i) = \boldsymbol{v}_i$ for every $q_i \in I$, and $C_{\boldsymbol{v}}(q) = 0$ for every $q \in Q \setminus I$. We call $C_{\boldsymbol{v}}$ the *initial configuration for input* $\boldsymbol{v}$. The protocol $\mathcal{P}$ *computes the predicate* $\varphi \colon \mathbb{N}^k \to \{0,1\}$, if for every $\boldsymbol{v} \in \mathbb{N}^k$, every fair run starting at $C_{\boldsymbol{v}}$ converges to $\varphi(\boldsymbol{v})$.
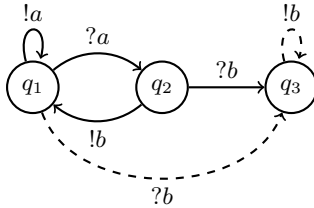
Fig. 3. An RBN protocol $\mathcal{P}$.

Example 3. Adding the dashed line transitions to the RBN of Example 1 yields the RBN protocol $\mathcal{P} = (Q, \Sigma, \delta, I, O)$ illustrated in Figure 3. The initial state is

q_1, i.e. $I = \{q_1\}$, and the output function is defined such that $O(q_1) = O(q_2) = 0$ and $O(q_3) = 1$. If there is a process in q_3, it can "attract" the rest of the processes there using the new dashed transitions. As with the RBN of Example 1, a process can be put in q_3 starting from the initial configuration $\wr k \cdot q_1 \wr$ if and only if $k \geq 3$. This RBN protocol computes the predicate $x \geq 3$: if there are less than 3 processes originally in q_1 then they stay in states with output 0, and if there are more, then in a fair run a process eventually enters q_3, and eventually the others follow, thus converging to 1.

6.2   Expressivity

In this section, we show that RBN protocols compute exactly the predicates definable by counting sets. A predicate $\varphi : \mathbb{N}^k \to \{0, 1\}$ is *definable by counting sets* if for every $b \in \{0, 1\}$, the sets $\{\boldsymbol{v} \mid \varphi(\boldsymbol{v}) = b\}$ are counting sets.

For $b \in \{0, 1\}$, define the following sets of configurations:

- Let $\mathcal{C}_b$ be the set of b-consensus configurations.
- Let $\mathcal{ST}_b$ be the set $\overline{pre^* \left(\overline{\mathcal{C}_b} \right)}$ of stable b-consensuses. These are the configurations from which one can reach only b-consensuses.
- Let $\mathcal{I}_b$ be the set of initial configurations $C_{\boldsymbol{v}}$ for inputs $\boldsymbol{v}$ such that $\varphi(\boldsymbol{v}) = b$.

The next lemma states that every predicate computed by a protocol is definable by counting sets.

Lemma 7. *Let $\mathcal{P}$ be a RBN protocol that computes the predicate $\varphi : \mathbb{N}^k \to \{0, 1\}$. Then for every $b \in \{0, 1\}$, the sets $\mathcal{I}_b, \mathcal{C}_b$ and $\mathcal{ST}_b$ are all counting sets. This entails that φ is definable by counting sets.*

Proof Sketch. Fix a $b \in \{0, 1\}$. It is easy to see that $\mathcal{C}_b$ is a cube. Unraveling the definitions of $\mathcal{I}_b$ and $\mathcal{ST}_b$, we can express them in terms of $\mathcal{C}_b$ by using boolean operations and pre^*. By the Closure Corollary (Corollary 1), they are counting sets. Set $\{\boldsymbol{v} \mid \varphi(\boldsymbol{v}) = b\}$ is simply $\mathcal{I}_b$ restricted to I, and so we are done. □

The next lemma states the converse result. It essentially uses the fact that there is a sub-class of population protocols called IO protocols which compute exactly the predicates definable by counting sets (Theorem 7 and Theorem 39 of [2,13]), and that IO protocols are a sub-class of RBN (Section 6.2 of [3]).

Lemma 8. *Let $\varphi : \mathbb{N}^k \to \{0, 1\}$ be a predicate definable by counting sets. Then there exists a RBN protocol computing φ.*

By Lemma 7 and Lemma 8, we get our result.

Theorem 8. *RBN protocols compute exactly the predicates definable by counting sets.*

Acknowledgements

References

1. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. Distributed Comput. **18**(4), 235–253 (2006). https://doi.org/10.1007/s00446-005-0138-3
2. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distributed Comput. **20**(4), 279–304 (2007). https://doi.org/10.1007/s00446-007-0040-2
3. Balasubramanian, A.R., Weil-Kennedy, C.: Reconfigurable broadcast networks and asynchronous shared-memory systems are equivalent. In: Ganty, P., Bresolin, D. (eds.) Proceedings 12th International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2021, Padua, Italy, 20-22 September 2021. EPTCS, vol. 346, pp. 18–34 (2021). https://doi.org/10.4204/EPTCS.346.2
4. Bertrand, N., Fournier, P.: Parameterized verification of many identical probabilistic timed processes. In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS. pp. 501–513 (2013). https://doi.org/10.4230/LIPIcs.FSTTCS.2013.501
5. Bertrand, N., Fournier, P., Sangnier, A.: Playing with probabilities in reconfigurable broadcast networks. In: Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS. pp. 134–148 (2014). https://doi.org/10.1007/978-3-642-54830-7_9
6. Bouyer, P., Markey, N., Randour, M., Sangnier, A., Stan, D.: Reachability in networks of register protocols under stochastic schedulers. In: Chatzigiannakis, I., Mitzenmacher, M., Rabani, Y., Sangiorgi, D. (eds.) 43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy. LIPIcs, vol. 55, pp. 106:1–106:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016). https://doi.org/10.4230/LIPIcs.ICALP.2016.106
7. Chini, P., Meyer, R., Saivasan, P.: Liveness in broadcast networks. In: Atig, M.F., Schwarzmann, A.A. (eds.) Networked Systems - 7th International Conference, NETYS 2019, Marrakech, Morocco, June 19-21, 2019, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11704, pp. 52–66. Springer (2019). https://doi.org/10.1007/978-3-030-31277-0_4
8. Delzanno, G., Sangnier, A., Traverso, R., Zavattaro, G.: On the complexity of parameterized reachability in reconfigurable broadcast networks. In: D'Souza, D., Kavitha, T., Radhakrishnan, J. (eds.) IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India. LIPIcs, vol. 18, pp. 289–300. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2012), https://doi.org/10.4230/LIPIcs.FSTTCS.2012.289
9. Delzanno, G., Sangnier, A., Traverso, R., Zavattaro, G.: On the complexity of parameterized reachability in reconfigurable broadcast networks. *Long version* (2012), https://www.irif.fr/~sangnier/publis/DSTZ-FSTTCS12-long.pdf
10. Delzanno, G., Sangnier, A., Zavattaro, G.: Parameterized verification of ad hoc networks. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010 - Concurrency Theory, 21st International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6269, pp. 313–327. Springer (2010). https://doi.org/10.1007/978-3-642-15375-4_22
11. Esparza, J., Ganty, P., Majumdar, R., Weil-Kennedy, C.: Verification of immediate observation population protocols. In: CONCUR. LIPIcs, vol. 118, pp. 31:1–31:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2018). https://doi.org/10.4230/LIPIcs.CONCUR.2018.31

12. Esparza, J., Jaax, S., Raskin, M.A., Weil-Kennedy, C.: The complexity of verifying population protocols. Distributed Comput. **34**(2), 133–177 (2021). https://doi.org/10.1007/s00446-021-00390-x

13. Esparza, J., Raskin, M.A., Weil-Kennedy, C.: Parameterized analysis of immediate observation petri nets. In: Donatelli, S., Haar, S. (eds.) Application and Theory of Petri Nets and Concurrency - 40th International Conference, PETRI NETS 2019, Aachen, Germany, June 23-28, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11522, pp. 365–385. Springer (2019). https://doi.org/10.1007/978-3-030-21571-2_20

Separators in Continuous Petri Nets⋆

Michael Blondin[1]✉ 🆔 and Javier Esparza[2] 🆔

[1] Université de Sherbrooke, Sherbrooke, Canada
michael.blondin@usherbrooke.ca
[2] Technical University of Munich, Munich, Germany
esparza@in.tum.de

Abstract. Leroux has proved that unreachability in Petri nets can be witnessed by a Presburger separator, i.e. if a marking $\boldsymbol{m}_{\mathrm{src}}$ cannot reach a marking $\boldsymbol{m}_{\mathrm{tgt}}$, then there is a formula φ of Presburger arithmetic such that: $\varphi(\boldsymbol{m}_{\mathrm{src}})$ holds; φ is forward invariant, i.e., $\varphi(\boldsymbol{m})$ and $\boldsymbol{m} \to \boldsymbol{m}'$ imply $\varphi(\boldsymbol{m}')$; and $\neg\varphi(\boldsymbol{m}_{\mathrm{tgt}})$ holds. While these separators could be used as explanations and as formal certificates of unreachability, this has not yet been the case due to their (super-)Ackermannian worst-case size and the (super-)exponential complexity of checking that a formula is a separator. We show that, in continuous Petri nets, these two problems can be overcome. We introduce locally closed separators, and prove that: (a) unreachability can be witnessed by a locally closed separator computable in polynomial time; (b) checking whether a formula is a locally closed separator is in NC (so, simpler than unreachablity, which is P-complete).

Keywords: Petri net · continuous reachability · separators · certificates.

1 Introduction

Petri nets form a widespread formalism of concurrency with several applications ranging from the verification of concurrent programs to the analysis of chemical systems. The reachability problem — which asks whether a a marking $\boldsymbol{m}_{\mathrm{src}}$ can reach another marking $\boldsymbol{m}_{\mathrm{tgt}}$ — is fundamental as a plethora of problems, such as verifying safety properties, reduce to it (e.g. [13,11,2]).

Leroux has shown that unreachability in Petri nets can be witnessed by a Presburger *separator*, i.e., if a marking $\boldsymbol{m}_{\mathrm{src}}$ cannot reach a marking $\boldsymbol{m}_{\mathrm{tgt}}$, then there exists a formula φ of Presburger arithmetic such that: $\varphi(\boldsymbol{m}_{\mathrm{src}})$ holds; φ is forward invariant, i.e., $\varphi(\boldsymbol{m})$ and $\boldsymbol{m} \to \boldsymbol{m}'$ imply $\varphi(\boldsymbol{m}')$; and $\varphi(\boldsymbol{m}_{\mathrm{tgt}})$ does not hold [14]. Intuitively, φ "separates" $\boldsymbol{m}_{\mathrm{tgt}}$ from the set of markings reachable from $\boldsymbol{m}_{\mathrm{src}}$. Leroux's result leads to a very simple algorithm to decide the Petri net reachability problem, consisting of two semi-algorithms; the first one explores the markings reachable from $\boldsymbol{m}_{\mathrm{src}}$, and halts if and when it hits $\boldsymbol{m}_{\mathrm{tgt}}$, while the

second enumerates formulas from Presburger arithmetic, and halts if and when it hits a separator.

Separators can be used as *explanations* and as formal *certificates*. Verifying a safety property can be reduced to proving that a target marking (or set of markings) is not reachable from a source marking, and a separator is an invariant of the system that *explains* why the property holds. Further, if a reachability tool produces separators, then the user can check that the properties of a separator indeed hold, and so trust the result even if they do not trust the tool (e.g., because it has not been verified, or is executed on a remote faster machine). Yet, in order to be useful as explanations and certificates, separators have to satisfy two requirements: (1) they should not be too large, and (2) checking that a formula is a separator should have low complexity, and in particular lower complexity than deciding reachability. This does not hold, at least in the worst-case, for the separators of [14]: In the worst case, the separator has super-Ackermannian size in the Petri net size (a consequence of the fact that the reachability problem is Ackermann-complete [16,15,7]) and the complexity of the check is super-exponential.

In this paper, we show that, unlike the above, *continuous* Petri nets do have separators satisfying properties (1) and (2). Continuous Petri nets are a relaxation of the standard Petri net model, called *discrete* in the following, in which transitions are allowed to fire "fluidly": instead of firing once, consuming i_p tokens from each input place p and adding o_q tokens to each output place q, a transition can fire α times for any nonnegative real number α, consuming and adding $\alpha \cdot i_p$ and $\alpha \cdot o_q$ tokens, respectively. Continuous Petri nets are interesting in their own right [8], and moreover as an overapproximation of the discrete model. In particular, if $\boldsymbol{m}_{\text{tgt}}$ is not reachable from $\boldsymbol{m}_{\text{src}}$ under the continuous semantics, then it is also not under the discrete one. As reachability in continuous Petri nets is P-complete [12], and so drastically more tractable than discrete reachability, this approximation is used in many tools for the verification of discrete Petri nets, VAS, or multiset rewriting systems (e.g. [5,4,10]).

It is easy to see that unreachability in continuous Petri nets can be witnessed by separators expressible in linear arithmetic (the first-order theory of the reals with addition and order). Indeed, Blondin et al. show in [5] that the continuous reachability relation is expressible by an existential formula $reach(\boldsymbol{m}, \boldsymbol{m}')$ of linear arithmetic, from which we can obtain a separator for any pair of unreachable markings. To wit, for all markings $\boldsymbol{m}_{\text{src}}$ and $\boldsymbol{m}_{\text{tgt}}$, if $\boldsymbol{m}_{\text{tgt}}$ is not reachable from $\boldsymbol{m}_{\text{src}}$, then the formula $sep_{\boldsymbol{m}_{\text{src}}}(\boldsymbol{m}) \coloneqq \neg reach(\boldsymbol{m}_{\text{src}}, \boldsymbol{m})$ is a separator. Further, $reach(\boldsymbol{m}, \boldsymbol{m}')$ has only linear size. However, these separators do not satisfy property (2) unless P = NP. Indeed, while the reachability problem for continuous Petri nets is P-complete [12], checking if a formula of linear arithmetic is a separator is coNP-hard, even for quantifier-free formulas in disjunctive normal form, a very small fragment. So, the separators arising from [5] cannot be directly used as certificates.

In this paper, we overcome this problem. We identify a class of *locally closed separators*, satisfying the following properties: unreachability can always be wit-

nessed by locally closed separators; locally closed separators can be constructed in polynomial time; and checking whether a formula is a locally closed separator is computationally easier than deciding unreachability. Let us examine the last claim in more detail. While the reachability problem for continuous Petri nets is decidable in polynomial time, it is still time consuming for larger models, which can have tens of thousands of nodes. Indeed, for a Petri net with n places and m transitions, the algorithm of [12] requires to solve $\mathcal{O}(m^2)$ linear programming problems in n variables, each of them with up to m constraints. Moreover, since the problem is P-complete, it is unlikely that a parallel computer can significantly improve performance. We prove that, on the contrary, checking if a formula is a locally closed separator is in NC rather than P-complete, and so efficiently parallelizable. Further, the checking algorithm only requires to solve linear programming problems in *a single* variable.

The paper is organized as follows. Section 2 introduces terminology, and defines separators (actually, a slightly different notion called bi-separators). Section 3 recalls the characterization of the reachability relation given by Fraca and Haddad in [12], and derives a characterization of *un*reachability suitable for finding bi-separators. Section 4 shows that checking the separators derivable from [5] is coNP-hard, and introduces locally closed bi-separators. Sections 5 and 6 show that locally closed bi-separators satisfy the aforementioned properties (1) and (2). Finally, Section 7 shows that all our results can be extended to separators that separate two sets of markings instead of singletons.

2   Preliminaries

Numbers, vectors and relations. We write $\mathbb{N}$, $\mathbb{R}$ and $\mathbb{R}_+$ to denote the naturals (including 0), reals, and non-negative reals (including 0). Let S be a finite set. We write $\boldsymbol{e}_s$ to denote the unit vector $\boldsymbol{e}_s \in \mathbb{R}^S$ such that $\boldsymbol{e}_s(s) = 1$ and $\boldsymbol{e}_s(t) = 0$ for all $s, t \in S$ such that $t \neq s$. Given $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^S$, we write $\boldsymbol{x} \sim_S \boldsymbol{y}$ to indicate that $\boldsymbol{x}(s) \sim \boldsymbol{y}(s)$ for all $s \in S$, where $\sim$ is a total order such as $\leq$. We define the *support* of a vector $\boldsymbol{x} \in \mathbb{R}^S$ as $\mathrm{supp}(\boldsymbol{x}) := \{s \in S : \boldsymbol{x}(s) > 0\}$. We write $\boldsymbol{x}(S) := \sum_{s \in S} \boldsymbol{x}(s)$. The *transpose* of a binary relation $\mathcal{R}$ is $\mathcal{R}^{\mathsf{T}} := \{(y, x) : (x, y) \in \mathcal{R}\}$.

Petri nets. A *Petri net*[3] is a tuple $\mathcal{N} = (P, T, F)$ where P and T are disjoint finite sets, whose elements are respectively called *places* and *transitions*, and where $F = (\mathbf{F}_-, \mathbf{F}_+)$ with $\mathbf{F}_-, \mathbf{F}_+ : P \times T \to \mathbb{N}$. For every $t \in T$, vectors $\Delta_t^-, \Delta_t^+ \in \mathbb{N}^P$ are respectively defined as the column of $\mathbf{F}_-$ and $\mathbf{F}_+$ associated to t, i.e. $\Delta_t^- := \mathbf{F}_- \cdot \boldsymbol{e}_t$ and $\Delta_t^+ := \mathbf{F}_+ \cdot \boldsymbol{e}_t$. A *marking* is a vector $\boldsymbol{m} \in \mathbb{R}_+^P$. We say that transition t is α-*enabled* if $\boldsymbol{m} \geq \alpha \Delta_t^-$ holds. If this is the case, then t can be α-*fired* from $\boldsymbol{m}$, which leads to marking $\boldsymbol{m}' := \boldsymbol{m} - \alpha \Delta_t^- + \alpha \Delta_t^+$, which we denote $\boldsymbol{m} \xrightarrow{\alpha t} \boldsymbol{m}'$. A transition is *enabled* if it is α-enabled for some real number

---

[3] In this work, "Petri nets" stands for "continuous Petri nets". In other words, we will consider standard Petri nets, but equipped with a *continuous* reachability relation. We will work over the reals, but note that it is known that working over the rationals is equivalent. For decidability issues, we will assume input numbers to be rationals.

$\alpha > 0$. We define $\mathbf{F} := \mathbf{F}_+ - \mathbf{F}_-$ and $\Delta_t := \mathbf{F} \cdot e_t$. In particular, $\boldsymbol{m} \xrightarrow{\alpha t} \boldsymbol{m}'$ implies $\boldsymbol{m}' = \boldsymbol{m} + \alpha \Delta_t$. For example, for the Petri net of Figure 1:

$$\{p_1 \mapsto 2, p_2 \mapsto 0, p_3 \mapsto 0, p_4 \mapsto 0\} \xrightarrow{(1/2)t_1} \{p_1 \mapsto 3/2, p_2 \mapsto 1/2, p_3 \mapsto 0, p_4 \mapsto 0\}.$$

Moreover, w.r.t. to orderings $p_1 < \cdots < p_4$ (rows) and $t_1 < \cdots < t_4$ (columns):

$$\mathbf{F}_- = \begin{bmatrix} 1 & 2 & 2 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad \mathbf{F}_+ = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{F} = \begin{bmatrix} -1 & -2 & -1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$
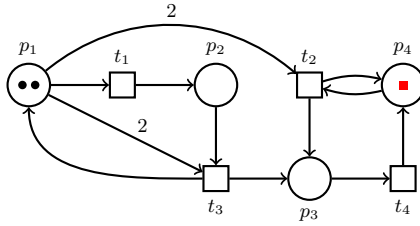
Fig. 1. A Petri net and two markings $\boldsymbol{m}_{\mathrm{src}} = \{p_1 \mapsto 2, p_2 \mapsto 0, p_3 \mapsto 0, p_4 \mapsto 0\}$ (black circles) and $\boldsymbol{m}_{\mathrm{tgt}} = \{p_1 \mapsto 0, p_2 \mapsto 0, p_3 \mapsto 0, p_4 \mapsto 1\}$ (colored squares).

A sequence $\sigma = \alpha_1 t_1 \cdots \alpha_n t_n$ is a *firing sequence* from $\boldsymbol{m}_{\mathrm{src}}$ to $\boldsymbol{m}_{\mathrm{tgt}}$ if there are markings $\boldsymbol{m}_0, \ldots, \boldsymbol{m}_n$ satisfying $\boldsymbol{m}_{\mathrm{src}} = \boldsymbol{m}_0 \xrightarrow{\alpha_1 t_1} \boldsymbol{m}_1 \cdots \xrightarrow{\alpha_n t_n} \boldsymbol{m}_n = \boldsymbol{m}_{\mathrm{tgt}}$. We write $\boldsymbol{m}_0 \xrightarrow{\sigma} \boldsymbol{m}_n$. We say that $\boldsymbol{m}_{\mathrm{src}}$ *enables* σ, and that $\boldsymbol{m}_{\mathrm{tgt}}$ enables σ *backwards*, or *backward-enables* σ. The *support* of σ is the set $\{t_1, \ldots, t_n\}$. For example, for the Petri net of Figure 1, we have $\boldsymbol{m}_{\mathrm{src}} \xrightarrow{\sigma} \boldsymbol{m}_{\mathrm{tgt}}$ where

$$\boldsymbol{m}_{\mathrm{src}} = \{p_1 \mapsto 2, p_2 \mapsto 0, p_3 \mapsto 0, p_4 \mapsto 0\},$$
$$\boldsymbol{m}_{\mathrm{tgt}} = \{p_1 \mapsto 0, p_2 \mapsto 0, p_3 \mapsto 0, p_4 \mapsto 1\},$$
$$\sigma = (1/2)t_1 \ (1/2)t_3 \ (1/2)t_4 \ (1/2)t_2 \ (1/2)t_4.$$

Let $U \subseteq T$. We write $\boldsymbol{m} \to^U \boldsymbol{m}'$ to denote that $\boldsymbol{m} \xrightarrow{\alpha t} \boldsymbol{m}'$ for some $\alpha > 0$ and $t \in U$, and $\to^{U^*}$ for the transitive and reflexive closure of $\to^U$. We simply write $\to$ and $\to^*$ when $U = T$. The Petri net $\mathcal{N}_U$ is obtained by removing transitions $T \setminus U$ from $\mathcal{N}$. In particular, $\boldsymbol{m} \to^{U^*} \boldsymbol{m}'$ holds in $\mathcal{N}$ iff $\boldsymbol{m} \to^* \boldsymbol{m}'$ holds in $\mathcal{N}_U$.

The *transpose* of $\mathcal{N} = (P, T, (\mathbf{F}_-, \mathbf{F}_+))$ is $\mathcal{N}^{\mathsf{T}} := (P, T, (\mathbf{F}_+, \mathbf{F}_-))$. We have $\boldsymbol{m}_{\mathrm{src}} \xrightarrow{\sigma} \boldsymbol{m}_{\mathrm{tgt}}$ in $\mathcal{N}$ iff $\boldsymbol{m}_{\mathrm{tgt}} \xrightarrow{\tau} \boldsymbol{m}_{\mathrm{src}}$ in $\mathcal{N}^{\mathsf{T}}$, where τ is the reverse of σ. For $U \subseteq T$, we write U^{T} to denote U in the context of $\mathcal{N}^{\mathsf{T}}$. This way, when we write, e.g. $\to^U$ and $\to^{U^{\mathsf{T}}}$, it is clear that we respectively refer to $\mathcal{N}$ and $\mathcal{N}^{\mathsf{T}}$.

Linear arithmetic and Farkas' lemma. An *atomic proposition* is a linear inequality of the form $\boldsymbol{a}\boldsymbol{x} \leq b$ or $\boldsymbol{a}\boldsymbol{x} < b$, where b and the components of $\boldsymbol{a}$ are over

$\mathbb{R}$. Such a proposition is *homogeneous* if $b = 0$. A *linear formula* is a first-order formula over atomic propositions with variables ranging over $\mathbb{R}_+$ (the classical definition uses $\mathbb{R}$, but in our context variables will encode markings.) The *solutions* of a linear formula φ, denoted $[\![\varphi]\!]$, are the assignments to the free variables of φ that satisfy φ. A linear formula is *homogeneous* if all of its atomic propositions are homogeneous. For every formula $\varphi(\boldsymbol{x}, \boldsymbol{y})$ where $\boldsymbol{x}$ and $\boldsymbol{y}$ have the same arity, we write φ^{T} to denote the formula that syntactically swaps $\boldsymbol{x}$ and $\boldsymbol{y}$, so that $[\![\varphi^{\mathsf{T}}]\!] = [\![\varphi]\!]^{\mathsf{T}}$. Throughout the paper, we will use Farkas' lemma, a fundamental result of linear arithmetic that rephrases the absence of solution to a system into the existence of one for another system:

Lemma 1 (Farkas' lemma). *Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$. The formula $\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b}$ has no solution iff $\mathbf{A}^{\mathsf{T}}\boldsymbol{y} = \mathbf{0} \wedge \boldsymbol{b}^{\mathsf{T}}\boldsymbol{y} < 0 \wedge \boldsymbol{y} \geq \mathbf{0}$ has a solution.*

2.1 Separators and bi-separators

Let us fix a Petri net $\mathcal{N} = (P, T, F)$ and two markings $\boldsymbol{m}_{\mathrm{src}}, \boldsymbol{m}_{\mathrm{tgt}} \in \mathbb{R}_+^P$.

Definition 1. *A* separator *for $(\boldsymbol{m}_{src}, \boldsymbol{m}_{tgt})$ is a linear formula φ over $\mathbb{R}_+^P$ such that: (1) $\boldsymbol{m}_{src} \in [\![\varphi]\!]$; (2) φ is* forward invariant, *i.e., $\boldsymbol{m} \in [\![\varphi]\!]$ and $\boldsymbol{m} \to \boldsymbol{m}'$ implies $\boldsymbol{m}' \in [\![\varphi]\!]$; and (3) $\boldsymbol{m}_{tgt} \notin [\![\varphi]\!]$.*

It follows immediately from the definition that if there exists a separator φ for $(\boldsymbol{m}_{\mathrm{src}}, \boldsymbol{m}_{\mathrm{tgt}})$, then $\boldsymbol{m}_{\mathrm{src}} \not\to^* \boldsymbol{m}_{\mathrm{tgt}}$. Thus, in order to show that $\boldsymbol{m}_{\mathrm{src}} \not\to^* \boldsymbol{m}_{\mathrm{tgt}}$ in $\mathcal{N}$, we can either give a separator for $(\boldsymbol{m}_{\mathrm{src}}, \boldsymbol{m}_{\mathrm{tgt}})$ w.r.t. $\mathcal{N}$, or a separator for $(\boldsymbol{m}_{\mathrm{tgt}}, \boldsymbol{m}_{\mathrm{src}})$ w.r.t. $\mathcal{N}^{\mathsf{T}}$. Let us call them *forward* and *backward* separators. Loosely speaking, a forward separator shows that $\boldsymbol{m}_{\mathrm{tgt}}$ is not among the markings reachable from $\boldsymbol{m}_{\mathrm{src}}$, and a backward separator shows that $\boldsymbol{m}_{\mathrm{src}}$ is not among the markings backward-reachable from $\boldsymbol{m}_{\mathrm{tgt}}$. Bi-separators are formulas from which we can easily obtain forward and backward separators. The symmetry w.r.t. forward and backward reachability make them easier to handle.

Definition 2. *A linear formula φ over $(\mathbb{R}_+^P)^2$ is* forward invariant *if $(\boldsymbol{m}, \boldsymbol{m}') \in [\![\varphi]\!]$ and $\boldsymbol{m}' \to \boldsymbol{m}''$ imply $(\boldsymbol{m}, \boldsymbol{m}'') \in [\![\varphi]\!]$;* backward invariant *if $(\boldsymbol{m}', \boldsymbol{m}'') \in [\![\varphi]\!]$ and $\boldsymbol{m} \to \boldsymbol{m}'$ imply $(\boldsymbol{m}, \boldsymbol{m}'') \in [\![\varphi]\!]$; and* bi-invariant *if it is forward and backward invariant. A* bi-separator *for $(\boldsymbol{m}_{src}, \boldsymbol{m}_{tgt})$ is a bi-invariant linear formula φ s.t. $(\boldsymbol{m}_{src}, \boldsymbol{m}_{src}) \in [\![\varphi]\!]$, $(\boldsymbol{m}_{tgt}, \boldsymbol{m}_{tgt}) \in [\![\varphi]\!]$ and $(\boldsymbol{m}_{src}, \boldsymbol{m}_{tgt}) \notin [\![\varphi]\!]$.*

The following proposition shows how to obtain separators from bi-separators.

Proposition 1. *Let φ be a bi-separator for $(\boldsymbol{m}_{src}, \boldsymbol{m}_{tgt})$. The following holds:*

- *$\psi(\boldsymbol{m}) := \varphi(\boldsymbol{m}_{src}, \boldsymbol{m})$ is a separator for $(\boldsymbol{m}_{src}, \boldsymbol{m}_{tgt})$ in $\mathcal{N}$;*
- *$\psi'(\boldsymbol{m}) := \varphi(\boldsymbol{m}, \boldsymbol{m}_{tgt})$ is a separator for $(\boldsymbol{m}_{tgt}, \boldsymbol{m}_{src})$ in $\mathcal{N}^{\mathsf{T}}$.*

Proof. It suffices to prove the first statement, the second is symmetric.

It is the case that $\boldsymbol{m}_{\mathrm{src}} \in [\![\psi]\!]$ and $\boldsymbol{m}_{\mathrm{tgt}} \notin [\![\psi]\!]$ as $(\boldsymbol{m}_{\mathrm{src}}, \boldsymbol{m}_{\mathrm{src}}) \in [\![\varphi]\!]$ and $(\boldsymbol{m}_{\mathrm{src}}, \boldsymbol{m}_{\mathrm{tgt}}) \notin [\![\varphi]\!]$. It remains to show that ψ is forward invariant. Let $\boldsymbol{m} \in [\![\psi]\!]$ and $\boldsymbol{m} \xrightarrow{\alpha t} \boldsymbol{m}'$. Since $(\boldsymbol{m}_{\mathrm{src}}, \boldsymbol{m}) \in [\![\varphi]\!]$ and φ is forward invariant, it is the case that $(\boldsymbol{m}_{\mathrm{src}}, \boldsymbol{m}') \in [\![\varphi]\!]$. Hence, $\boldsymbol{m}' \in [\![\psi]\!]$ as desired. $\qquad\square$

3   A characterization of unreachability

In [12], Fraca and Haddad gave the following characterization of the reachability relation in continuous Petri nets:

Theorem 1 ([12]). *Let $\mathcal{N} = (P, T, F)$ be a Petri net, let $U \subseteq T$, and let $\boldsymbol{m}_{src}, \boldsymbol{m}_{tgt} \in \mathbb{R}_+^P$. It is the case that $\boldsymbol{m}_{src} \to^{U^*} \boldsymbol{m}_{tgt}$ iff there exists $S \subseteq U$ such that the following conditions hold:*

1. *some vector $\boldsymbol{x} \in \mathbb{R}_+^T$ with support S satisfies $\boldsymbol{m}_{src} + \mathbf{F}\boldsymbol{x} = \boldsymbol{m}_{tgt}$,*
2. *some firing sequence σ with support S is enabled at $\boldsymbol{m}_{src}$, and*
3. *some firing sequence τ with support S is backward-enabled at $\boldsymbol{m}_{tgt}$.*

Furthermore, these conditions can be checked in polynomial time.

Theorem 1 has the following form, where P_1, P_2 and P_3 stand for the conditions of 1., 2., and 3.:

$$\boldsymbol{m}_{\mathrm{src}} \to^{U^*} \boldsymbol{m}_{\mathrm{tgt}} \iff \exists S \subseteq U : (\exists \boldsymbol{x} : P_1(S, \boldsymbol{x})) \wedge (\exists \sigma : P_2(S, \sigma) \wedge (\exists \tau : P_3(S, \tau)).$$

Therefore, $\boldsymbol{m}_{\mathrm{src}} \not\to^{U^*} \boldsymbol{m}_{\mathrm{tgt}}$ holds iff

$$\forall S \subseteq U : (\forall \boldsymbol{x} : \neg P_1(S, \boldsymbol{x})) \vee (\forall \sigma : \neg P_2(S, \sigma)) \vee (\forall \tau : \neg P_3(S, \tau)).$$

To obtain a witness of unreachability for a given $S \subseteq U$, we replace each universally quantified disjunct by an existentially quantified equivalent one. For conditions 2. and 3., the solution (implicitly given in [12]) is formulated in Proposition 2. Given a set of places X, let $^\bullet X$ (resp. $X^\bullet$) be the set of transitions t such that $\mathbf{F}_+(p, t) > 0$ (resp. $\mathbf{F}_-(p, t) > 0$) for some $p \in X$. A *siphon* of $\mathcal{N}$ is a subset Q of places such that $^\bullet Q \subseteq Q^\bullet$. A *trap* is a subset R of places such that $R^\bullet \subseteq {}^\bullet R$. Informally, empty siphons remain empty, and marked traps remain marked. Formally, if $\boldsymbol{m} \to \boldsymbol{m}'$, then $\boldsymbol{m}(Q) = 0$ implies $\boldsymbol{m}'(Q) = 0$, and $\boldsymbol{m}(R) > 0$ implies $\boldsymbol{m}'(R) > 0$. We have:

Proposition 2 ([12]). *Let $\mathcal{N} = (P, T, F)$ be a Petri net, let $S \subseteq T$, and let $\boldsymbol{m} \in \mathbb{R}_+^P$. The following statements hold:*

- *No firing sequence with support S is enabled at $\boldsymbol{m}$ iff there exists a siphon Q of $\mathcal{N}_S$ such that $Q^\bullet \neq \emptyset$ satisfies $\boldsymbol{m}(Q) = 0$;*
- *No firing sequence with support S is backward-enabled at $\boldsymbol{m}$ iff there exists a trap R of $\mathcal{N}_S$ such that $^\bullet R \neq \emptyset$ satisfies $\boldsymbol{m}(R) = 0$.*

So the universal statements "no firing sequence . . . is enabled/backward-enabled . . ." are replaced by existential statements "there exists a siphon/trap . . . ". The if-direction of the proposition is easy to prove. A siphon Q of $\mathcal{N}_S$ satisfies $Q^\bullet \subseteq S$. Since Q is empty at $\boldsymbol{m}$, if we only fire transitions from S then Q remains empty, and so no transition of $Q^\bullet$ ever becomes enabled. So transitions of $Q^\bullet$ can only fire after transitions that do not belong to S have fired first. But no such firing sequence has support S, and we are done. The case of traps is analogous. For the only-if direction we refer the reader to [12].

For condition 1. of Theorem 1, we obtain a solution in terms of *exclusion functions*.

Definition 3. *Let $\mathcal{N} = (P, T, F)$ be a Petri net, let $\boldsymbol{m}_{src}, \boldsymbol{m}_{tgt} \in \mathbb{R}_+^P$ and let $S \subseteq S' \subseteq T$. An* exclusion function for (S, S') *is a function $f \colon \mathbb{R}_+^P \to \mathbb{R}$ s.t.*

1. *$\boldsymbol{m} \xrightarrow{s} \boldsymbol{m}'$ implies $f(\boldsymbol{m}) \leq f(\boldsymbol{m}')$ for all $s \in S'$; and*
2. *either $f(\boldsymbol{m}_{src}) > f(\boldsymbol{m}_{tgt})$, or $f(\boldsymbol{m}_{src}) = f(\boldsymbol{m}_{tgt})$ and there exists $s \in S$ such that $\boldsymbol{m} \xrightarrow{s} \boldsymbol{m}'$ implies $f(\boldsymbol{m}) < f(\boldsymbol{m}')$.*

An exclusion function for S *is an exclusion function for (S, S).*

An exclusion function for S excludes the existence of a firing sequence from $\boldsymbol{m}_{src}$ to $\boldsymbol{m}_{tgt}$ with support S, i.e., witnesses that condition 1 of Theorem 1 fails. To see why, call $f(\boldsymbol{m})$ the *value* of m. By definition of f, either $\boldsymbol{m}_{tgt}$ has lower value than $\boldsymbol{m}_{src}$ but no transition of S decreases it, or $\boldsymbol{m}_{src}$ and $\boldsymbol{m}_{tgt}$ have the same value but no transition of S decreases it, and at least one increases it. So it is impossible to reach $\boldsymbol{m}_{tgt}$ from $\boldsymbol{m}_{src}$ by firing *all* and *only* the transitions of S. Let us apply exclusion functions and Proposition 2 to an example.

Example 1. Consider the Petri net of Figure 1, but with $\boldsymbol{m}_{tgt} := \{p_1 \mapsto 0, p_2 \mapsto 0, p_3 \mapsto 1, p_4 \mapsto 0\}$ as target. We prove $\boldsymbol{m}_{src} \not\rightarrow^* \boldsymbol{m}_{tgt}$. For the sake of contradiction, assume $\boldsymbol{m}_{src} \rightarrow^{U^*} \boldsymbol{m}_{tgt}$ for some $U \subseteq T$. We proceed in several steps:

- *Claim: $t_4 \notin U$.* The function $f(\boldsymbol{m}) := \boldsymbol{m}(p_4)$ is an exclusion function for T. Indeed, since no transition decreases the number of tokens of p_4, $\boldsymbol{m} \xrightarrow{t} \boldsymbol{m}'$ implies $f(\boldsymbol{m}) \leq f(\boldsymbol{m}')$ for every transition $t \in T$. Furthermore, $f(\boldsymbol{m}_{src}) = 0 = f(\boldsymbol{m}_{tgt})$, and, since t_4 adds tokens to p_4, $\boldsymbol{m} \xrightarrow{t_4} \boldsymbol{m}'$ implies $f(\boldsymbol{m}) < f(\boldsymbol{m}')$. It follows that no firing sequence from $\boldsymbol{m}_{src}$ to $\boldsymbol{m}_{tgt}$ can fire t_4.

- *Claim: $t_2 \notin U$.* The set $Q := \{p_4\}$ is a siphon of $\mathcal{N}_{T \setminus \{t_4\}}$ (but not of $\mathcal{N}$). Since $\boldsymbol{m}_{src}(Q) = 0$, it is impossible to use transitions of $\mathcal{N}_{T \setminus \{t_4\}}$ that consume from Q, i.e. transitions of $Q^\bullet = \{t_2\}$.

- *Claim: $t_1, t_3 \notin U$.* The set $R := \{p_1, p_2\}$ is a trap of $\mathcal{N}_{T \setminus \{t_2, t_4\}}$ (but not of $\mathcal{N}_{T \setminus \{t_4\}}$). Since $\boldsymbol{m}_{tgt}(R) = 0$, it is impossible to reach $\boldsymbol{m}_{tgt}$ using transitions of $\mathcal{N}_{T \setminus \{t_2, t_4\}}$ that produce in R, i.e. transitions of $^\bullet R = \{t_1, t_3\}$.

By the claims, $U = \emptyset$, hence we reach the contradiction $\boldsymbol{m}_{src} = \boldsymbol{m}_{tgt}$. □

Proposition 4 below shows that condition 1. of Theorem 1 fails if and only if there is an exclusion function for S (actually, a slightly more general result). We need the following consequence of Farkas' lemma:

Proposition 3. *The system $\exists \boldsymbol{x} \geq \boldsymbol{0} : \mathbf{A}\boldsymbol{x} = \boldsymbol{b} \wedge S \subseteq \mathrm{supp}(\boldsymbol{x}) \subseteq S'$ has no solution iff this system has some: $\exists \boldsymbol{y} : \mathbf{A}^\mathsf{T}\boldsymbol{y} \geq_{S'} \boldsymbol{0} \wedge \boldsymbol{b}^\mathsf{T}\boldsymbol{y} \leq 0 \wedge \boldsymbol{b}^\mathsf{T}\boldsymbol{y} < \sum_{s \in S}(\mathbf{A}^\mathsf{T}\boldsymbol{y})_s$.*

Proposition 4. *Let $\mathcal{N} = (P, T, F)$ be a Petri net, let $\boldsymbol{m}_{src}, \boldsymbol{m}_{tgt} \in \mathbb{R}_+^P$, and let $S \subseteq S' \subseteq T$. No vector $\boldsymbol{x} \in \mathbb{R}_+^T$ satisfies $S \subseteq \mathrm{supp}(\boldsymbol{x}) \subseteq S'$ and $\boldsymbol{m}_{src} + \mathbf{F}\boldsymbol{x} = \boldsymbol{m}_{tgt}$ iff there exists a linear exclusion function for (S, S').*

Proof. Assume no such $\boldsymbol{x} \in \mathbb{R}_+^T$ exists. Let $\boldsymbol{b} := \boldsymbol{m}_{\text{tgt}} - \boldsymbol{m}_{\text{src}}$. By Proposition 3, there exists $\boldsymbol{y} \in \mathbb{R}^P$ such that: $\mathbf{F}^\mathsf{T} \boldsymbol{y} \geq_{S'} \boldsymbol{0} \wedge \boldsymbol{b}^\mathsf{T} \boldsymbol{y} \leq 0 \wedge \boldsymbol{b}^\mathsf{T} \boldsymbol{y} < \sum_{s \in S} (\mathbf{F}^\mathsf{T} \boldsymbol{y})_s$. We show that $f(\boldsymbol{k}) := \boldsymbol{y}^\mathsf{T} \boldsymbol{k}$ is a linear exclusion function for (S, S').

1. We have $f(\boldsymbol{m}_{\text{tgt}}) - f(\boldsymbol{m}_{\text{src}}) = \boldsymbol{y}^\mathsf{T} \boldsymbol{m}_{\text{tgt}} - \boldsymbol{y}^\mathsf{T} \boldsymbol{m}_{\text{src}} = \boldsymbol{y}^\mathsf{T} (\boldsymbol{m}_{\text{tgt}} - \boldsymbol{m}_{\text{src}}) = \boldsymbol{y}^\mathsf{T} \boldsymbol{b} = \boldsymbol{b}^\mathsf{T} \boldsymbol{y} \leq 0$, and hence $f(\boldsymbol{m}_{\text{tgt}}) \leq f(\boldsymbol{m}_{\text{src}})$.

2. Let $\boldsymbol{m} \xrightarrow{\lambda s} \boldsymbol{m}'$ with $s \in S'$ and $\lambda \in \mathbb{R}_+$. We have $\boldsymbol{m}' = \boldsymbol{m} + \lambda \mathbf{F} \boldsymbol{e}_s$. Thus: $f(\boldsymbol{m}') = \boldsymbol{y}^\mathsf{T} \boldsymbol{m}' = \boldsymbol{y}^\mathsf{T} \boldsymbol{m} + \lambda (\boldsymbol{y}^\mathsf{T} \mathbf{F}) \boldsymbol{e}_s = \boldsymbol{y}^\mathsf{T} \boldsymbol{m} + \lambda (\mathbf{F}^\mathsf{T} \boldsymbol{y})^\mathsf{T} \boldsymbol{e}_s \geq \boldsymbol{y}^\mathsf{T} \boldsymbol{m} = f(\boldsymbol{m})$, where the inequality follows from $\lambda > 0$, $\mathbf{F}^\mathsf{T} \boldsymbol{y}, \geq_{S'} \boldsymbol{0}$ and $s \in S'$.

3. Recall that $\boldsymbol{b}^\mathsf{T} \boldsymbol{y} \leq 0$ and $\sum_{s \in S} (\mathbf{F}^\mathsf{T} \boldsymbol{y})_s > \boldsymbol{b}^\mathsf{T} \boldsymbol{y}$. If the latter sum equals zero, then $\boldsymbol{b}^\mathsf{T} \boldsymbol{y} < 0$, and hence we are done since $f(\boldsymbol{m}_{\text{tgt}}) - f(\boldsymbol{m}_{\text{src}}) = \boldsymbol{b}^\mathsf{T} \boldsymbol{y} < 0$. Otherwise, we have $\sum_{s \in S} (\mathbf{F}^\mathsf{T} \boldsymbol{y})_s > 0$ since $S \subseteq S'$ and $\mathbf{F}^\mathsf{T} \boldsymbol{y} \geq_{S'} \boldsymbol{0}$. Therefore, there exists a transition $s \in S$ such that $(\mathbf{F}^\mathsf{T} \boldsymbol{y})_s > 0$. Let $\boldsymbol{m} \xrightarrow{s} \boldsymbol{m}'$. We have $\boldsymbol{m}' = \boldsymbol{m} + \lambda \mathbf{F} \boldsymbol{e}_s$ for some $\lambda > 0$. Thus, $f(\boldsymbol{m}') = \boldsymbol{y}^\mathsf{T} \boldsymbol{m} + \lambda (\mathbf{F}^\mathsf{T} \boldsymbol{y})^\mathsf{T} \boldsymbol{e}_s > \boldsymbol{y}^\mathsf{T} \boldsymbol{m} = f(\boldsymbol{m})$, where the inequality holds by $\lambda > 0$ and $(\mathbf{F}^\mathsf{T} \boldsymbol{y})_s > 0$. □

Putting together Proposition 4 with Theorem 1 and Proposition 2, we obtain the following characterization of unreachability.

Proposition 5. *Let $\mathcal{N} = (P, T, F)$ be a Petri net, let $U \subseteq T$, and $\boldsymbol{m}_{src}, \boldsymbol{m}_{tgt} \in \mathbb{R}_+^P$. It is the case that $\boldsymbol{m}_{src} \not\to^{U^*} \boldsymbol{m}_{tgt}$ iff for every $S \subseteq U$:*

1. *there exists an exclusion function for S, or*
2. *there exists a siphon Q of $\mathcal{N}_S$ such that $Q^\bullet \neq \emptyset$ and $\boldsymbol{m}_{src}(Q) = 0$, or*
3. *there exists a trap R of $\mathcal{N}_S$ such that $^\bullet R \neq \emptyset$ and $\boldsymbol{m}_{tgt}(R) = 0$.*

This proposition shows that, for all supports S, we can produce a witness of unreachability as an exclusion function, a siphon, or a trap. In the next section, we transform these witnesses into separators useful as certificates.

4   Separators as certificates

Let $\mathcal{N} = (P, T, F)$ be a Petri net and let $\boldsymbol{m}_{\text{src}}, \boldsymbol{m}_{\text{tgt}} \in \mathbb{R}_+^P$ be two markings of $\mathcal{N}$. From [5], one can easily show that if $\boldsymbol{m}_{\text{src}} \not\to^* \boldsymbol{m}_{\text{tgt}}$, then there is a separator for $(\boldsymbol{m}_{\text{src}}, \boldsymbol{m}_{\text{tgt}})$. Indeed, [5, Prop. 3.2] shows that there exists an existential formula ψ of linear arithmetic such that $\boldsymbol{m} \to^* \boldsymbol{m}'$ iff $(\boldsymbol{m}, \boldsymbol{m}') \in [\![\psi]\!]$. Thus, the formula $\varphi(\boldsymbol{m}) := \psi(\boldsymbol{m}_{\text{src}}, \boldsymbol{m})$ is a separator.

However, φ is not adequate as a *certificate* of unreachability. Indeed, checking a certificate for $\boldsymbol{m}_{\text{src}} \not\to^* \boldsymbol{m}_{\text{tgt}}$ should have smaller complexity than deciding whether $\boldsymbol{m}_{\text{src}} \to^* \boldsymbol{m}_{\text{tgt}}$. This is not the case for existential linear formulas, because $\boldsymbol{m}_{\text{src}} \to^* \boldsymbol{m}_{\text{tgt}}$ can be decided in polynomial time, but checking that an existential linear formula is a separator is coNP-hard.

Proposition 6. *The problem of determining whether an existential linear formula φ is a separator for $(\boldsymbol{m}_{src}, \boldsymbol{m}_{tgt})$ is coNP-hard, even if φ is a quantifier-free formula in DNF and homogeneous.*

In the rest of the section, we introduce locally closed bi-separators, and then, in Sections 5 and 6, we respectively prove that they satisfy the following:

- If $\boldsymbol{m}_{\mathrm{src}} \not\rightarrow^* \boldsymbol{m}_{\mathrm{tgt}}$, then some locally closed bi-separator for $(\boldsymbol{m}_{\mathrm{src}}, \boldsymbol{m}_{\mathrm{tgt}})$ can be computed in polynomial time;
- Deciding whether a formula is a locally closed bi-separator is in NC.

4.1   Locally closed bi-separators

The most difficult part of checking that a formula φ is a bi-separator consists of checking that it is forward and backward invariant. Let us focus on forward invariance, backward invariance being symmetric.

Recall the definition: for all markings $\boldsymbol{m}, \boldsymbol{m}', \boldsymbol{m}''$ and every transition t: if $(\boldsymbol{m}, \boldsymbol{m}') \in [\![\varphi]\!]$ and $\boldsymbol{m}' \xrightarrow{\alpha t} \boldsymbol{m}''$ then $(\boldsymbol{m}, \boldsymbol{m}'') \in [\![\varphi]\!]$. Assume now that φ is in DNF, i.e., a disjunction of clauses $\varphi = \varphi_1 \vee \cdots \vee \varphi_n$. The forward invariance check can be decomposed into n smaller checks, one for each $i \in [1..n]$, of the form: if $(\boldsymbol{m}, \boldsymbol{m}') \in [\![\varphi_i]\!]$, then $(\boldsymbol{m}, \boldsymbol{m}'') \in [\![\varphi]\!]$. However, in general the check *cannot* be decomposed into *local* checks of the form: there exists $j \in [1..m]$ such that $(\boldsymbol{m}, \boldsymbol{m}') \in [\![\varphi_i]\!]$ implies $(\boldsymbol{m}, \boldsymbol{m}'') \in [\![\varphi_j]\!]$. Indeed, while this property is sufficient for forward invariance, it is not necessary. Intuitively, locally closed bi-separators are separators where invariance can be established by local checks.

For the formal definition, we need to introduce some notations. Given a transition t and atomic propositions ψ, ψ', we say that ψ *t-implies* ψ', written $\psi \leadsto_t \psi'$, if $(\boldsymbol{m}, \boldsymbol{m}') \in [\![\psi]\!]$ and $\boldsymbol{m}' \xrightarrow{\alpha t} \boldsymbol{m}''$ implies $(\boldsymbol{m}, \boldsymbol{m}'') \in [\![\psi']\!]$. We further say that a clause $\psi = \psi_1 \wedge \cdots \wedge \psi_m$ *t-implies* a clause $\psi' = \psi'_1 \wedge \cdots \wedge \psi'_n$, written $\psi \leadsto_t \psi'$, if for every $j \in [1..n]$, there exists $i \in [1..m]$ such that $\psi_i \leadsto_t \psi'_j$.

Definition 4. *A linear formula φ is* locally closed *w.r.t. $\mathcal{N} = (P, T, F)$ if:*

- *$\varphi = \varphi_1 \vee \cdots \vee \varphi_n$ is quantifier-free, in DNF and homogeneous,*
- *for every $t \in T$ and every $i \in [1..n]$, there exists $j \in [1..n]$ s.t. $\varphi_i \leadsto_t \varphi_j$,*
- *for every $t \in T^{\mathsf{T}}$ and every $i \in [1..n]$, there exists $j \in [1..n]$ s.t. $\varphi_i^{\mathsf{T}} \leadsto_t \varphi_j^{\mathsf{T}}$.*

Note that the definition is semantic. We make the straightforward but crucial observation that:

Proposition 7. *Locally closed formulas are bi-invariant.*

Proof. Let $\varphi = \varphi_1 \vee \cdots \vee \varphi_n$ be a locally closed formula. We only consider the forward case; the other case is symmetric. Let $(\boldsymbol{m}, \boldsymbol{m}') \in [\![\varphi]\!]$ and $\boldsymbol{m}' \xrightarrow{\alpha t} \boldsymbol{m}''$. Let $i \in [1..n]$ be such that $(\boldsymbol{m}, \boldsymbol{m}') \in [\![\varphi_i]\!]$. Since φ is locally closed, there exists $j \in [1..n]$ such that $\varphi_i \leadsto_t \varphi_j$. For every atomic proposition ψ' of φ_j, there exists an atomic proposition ψ of φ_i such that $\psi \leadsto_t \psi'$. Since each atomic proposition of φ_i is satisfied by $(\boldsymbol{m}, \boldsymbol{m}')$, we obtain $(\boldsymbol{m}, \boldsymbol{m}'') \in [\![\varphi_j]\!]$. $\square$

Proposition 7 justifies the following definition:

Definition 5. *A locally closed bi-separator for $(\boldsymbol{m}_{src}, \boldsymbol{m}_{tgt})$ is a locally closed formula φ s.t. $(\boldsymbol{m}_{src}, \boldsymbol{m}_{src}) \in [\![\varphi]\!]$, $(\boldsymbol{m}_{tgt}, \boldsymbol{m}_{tgt}) \in [\![\varphi]\!]$ and $(\boldsymbol{m}_{src}, \boldsymbol{m}_{tgt}) \notin [\![\varphi]\!]$.*

Indeed, by Proposition 7, a locally closed bi-separator is a bi-separator, as the bi-invariance condition of Definition 2 follows from local closedness.

5   Constructing locally closed bi-separators

In this section, we prove that unreachability can always be witnessed by locally closed bi-separators of polynomial size and computable in polynomial time. The proof uses the results of Section 3.

Theorem 2. *If $\boldsymbol{m}_{src} \not\rightarrow^{U^*} \boldsymbol{m}_{tgt}$, then there is a locally closed bi-separator φ for $(\boldsymbol{m}_{src}, \boldsymbol{m}_{tgt})$ w.r.t. $\mathcal{N}_U$. Further, $\varphi = \bigvee_{1 \leq i \leq n} \varphi_i$, where $n \leq 2|U|+1$ and each φ_i contains at most $2|U|+1$ atomic propositions. Moreover, φ is computable in polynomial time.*

Proof. We proceed by induction on $|U|$. First consider $U = \emptyset$. Let $p \in P$ be such that $\boldsymbol{m}_{\mathrm{src}}(p) \neq \boldsymbol{m}_{\mathrm{tgt}}(p)$. Take $\varphi(\boldsymbol{m}, \boldsymbol{m}') := \boldsymbol{e}_p \boldsymbol{m} \leq \boldsymbol{e}_p \boldsymbol{m}'$ or $-\boldsymbol{e}_p \boldsymbol{m} \leq -\boldsymbol{e}_p \boldsymbol{m}'$.

Now, assume that $U \neq \emptyset$. Consider the system $\exists \boldsymbol{x} \in \mathbb{R}_+^T : \boldsymbol{m}_{\mathrm{src}} + \mathbf{F}\boldsymbol{x} = \boldsymbol{m}_{\mathrm{tgt}} \land \mathrm{supp}(\boldsymbol{x}) \subseteq U$. Suppose first that the system has no solution. By Proposition 4, taking $S = \emptyset$ and $S' = U$, there is a linear exclusion function for $(\emptyset, U)$, i.e. a linear function f satisfying:

1. $f(\boldsymbol{m}_{\mathrm{src}}) > f(\boldsymbol{m}_{\mathrm{tgt}})$,
2. $\boldsymbol{m} \xrightarrow{u} \boldsymbol{m}'$ implies $f(\boldsymbol{m}) \leq f(\boldsymbol{m}')$ for all $u \in U$.

(The first item holds due to Item 2 of Definition 3 and $S = \emptyset$.) So we can take $\varphi(\boldsymbol{m}, \boldsymbol{m}') := (f(\boldsymbol{m}) \leq f(\boldsymbol{m}'))$.

Suppose now that the system has a solution $\boldsymbol{x} \in \mathbb{R}_+^U$. By convexity, we can suppose that $\mathrm{supp}(\boldsymbol{x}) \subseteq U$ is maximal. Indeed, if $\boldsymbol{x}'$ and $\boldsymbol{x}''$ are solutions, then $(1/2)\boldsymbol{x}' + (1/2)\boldsymbol{x}''$ is a solution with support $\mathrm{supp}(\boldsymbol{x}') \cup \mathrm{supp}(\boldsymbol{x}'')$. Let $U' := \mathrm{supp}(\boldsymbol{x})$. For every $t \in U \setminus U'$, consider the system of Proposition 4 with $S = \{t\}$ and $S' = U$. By maximality of $U' \subseteq U$, none of these systems has a solution. Consequently, for each $t \in U \setminus U'$, Proposition 4 yields a linear exclusion function for $(\{t\}, U)$, i.e. a linear function f_t that satisfies:

3. $f_t(\boldsymbol{m}_{\mathrm{src}}) \geq f_t(\boldsymbol{m}_{\mathrm{tgt}})$,
4. $\boldsymbol{m} \xrightarrow{u} \boldsymbol{m}'$ implies $f_t(\boldsymbol{m}) \leq f_t(\boldsymbol{m}')$ for all $u \in U$,
5. either $f_t(\boldsymbol{m}_{\mathrm{src}}) > f_t(\boldsymbol{m}_{\mathrm{tgt}})$, or $\boldsymbol{m} \xrightarrow{t} \boldsymbol{m}'$ implies $f_t(\boldsymbol{m}) < f_t(\boldsymbol{m}')$.

If $f_t(\boldsymbol{m}_{\mathrm{src}}) > f_t(\boldsymbol{m}_{\mathrm{tgt}})$ holds for some $t \in U \setminus U'$, then we are done by taking $\varphi(\boldsymbol{m}, \boldsymbol{m}') := (f_t(\boldsymbol{m}) \leq f_t(\boldsymbol{m}'))$ as Item 4 ensures that $\varphi \rightsquigarrow_u \varphi$ for every $u \in U$. So assume it does not hold for any $t \in U \setminus U'$, i.e. assume that $f_t(\boldsymbol{m}_{\mathrm{src}}) = f_t(\boldsymbol{m}_{\mathrm{tgt}})$ holds, and the second disjunct of Item 5 holds for all $t \in U \setminus U'$. This is the most involved case. Let

$$\varphi_{\mathrm{inv}}(\boldsymbol{m}, \boldsymbol{m}') := \bigwedge_{t \in U \setminus U'} (f_t(\boldsymbol{m}) \leq f_t(\boldsymbol{m}')) \ \text{ and } \ \varphi_t(\boldsymbol{m}, \boldsymbol{m}') := (f_t(\boldsymbol{m}) < f_t(\boldsymbol{m}')).$$

Let $Q, R \subseteq P$ be respectively the maximal siphon and trap of $\mathcal{N}_{U'}$ such that $\boldsymbol{m}_{\mathrm{src}}(Q) = 0$ and $\boldsymbol{m}_{\mathrm{tgt}}(R) = 0$ (well-defined by closure under union). Let $U'' := U' \setminus (Q^\bullet \cup {}^\bullet R)$. By Theorem 1 and Proposition 2, $Q^\bullet \cup {}^\bullet R \neq \emptyset$. Thus, U'' is a strict

subset of U', and, by induction hypothesis, there is a locally closed bi-separator w.r.t. $\mathcal{N}_{U''}$ of the form $\psi = \bigvee_{1 \leq i \leq m} \psi_i$ that satisfies the claim for set U''. Let

$$\varphi(\boldsymbol{m}, \boldsymbol{m}') \coloneqq \bigvee_{t \in U \setminus U'} \varphi_t(\boldsymbol{m}, \boldsymbol{m}') \vee [\varphi_{\mathrm{inv}}(\boldsymbol{m}, \boldsymbol{m}') \wedge \boldsymbol{m}(Q) + \boldsymbol{m}'(R) > 0] \vee$$
$$\bigvee_{1 \leq i \leq m} [\varphi_{\mathrm{inv}}(\boldsymbol{m}, \boldsymbol{m}') \wedge \boldsymbol{m}(R) + \boldsymbol{m}'(Q) \leq 0 \wedge \psi_i(\boldsymbol{m}, \boldsymbol{m}')].$$

As $(\boldsymbol{m}_{\mathrm{src}}, \boldsymbol{m}_{\mathrm{src}}) \in [\![\varphi_{\mathrm{inv}}]\!]$ and $(\boldsymbol{m}_{\mathrm{src}}, \boldsymbol{m}_{\mathrm{src}}) \in [\![\psi]\!]$, we have $(\boldsymbol{m}_{\mathrm{src}}, \boldsymbol{m}_{\mathrm{src}}) \in [\![\varphi]\!]$. Similarly, $(\boldsymbol{m}_{\mathrm{tgt}}, \boldsymbol{m}_{\mathrm{tgt}}) \in [\![\varphi]\!]$. By Item 3, $(\boldsymbol{m}_{\mathrm{src}}, \boldsymbol{m}_{\mathrm{tgt}}) \notin [\![\bigvee_{t \in U \setminus U'} \varphi_t(\boldsymbol{m}, \boldsymbol{m}')]\!]$. Further, $\boldsymbol{m}_{\mathrm{src}}(Q) + \boldsymbol{m}_{\mathrm{tgt}}(R) = 0$ and $(\boldsymbol{m}_{\mathrm{src}}, \boldsymbol{m}_{\mathrm{tgt}}) \notin [\![\psi]\!]$. So, $(\boldsymbol{m}_{\mathrm{src}}, \boldsymbol{m}_{\mathrm{tgt}}) \notin [\![\varphi]\!]$.

The number of disjuncts of φ is $|U \setminus U'| + 1 + m$ and hence at most

$$|U \setminus U'| + 1 + 2|U''| + 1 \leq |U| - |U''| + 1 + 2|U''| + 1 =$$
$$|U| + |U''| + 2 \leq |U| + (|U| - 1) + 2 = 2|U| + 1.$$

The same bounds holds for the number of atomic propositions per disjunct.

It remains to show that $\varphi(\boldsymbol{m}, \boldsymbol{m}')$ is locally closed w.r.t. $\mathcal{N}_U$. We only consider the forward case, as the backward case is symmetric. Let $(\boldsymbol{m}, \boldsymbol{m}') \in [\![\varphi]\!]$ and $\boldsymbol{m}' \xrightarrow{u} \boldsymbol{m}''$ for some $u \in U$. By Item 4, $\varphi_t \rightsquigarrow_u \varphi_t$ holds for each φ_t. Indeed, $f_t(\boldsymbol{m}) < f_t(\boldsymbol{m}')$ and $\boldsymbol{m}' \xrightarrow{u} \boldsymbol{m}''$ imply $f_t(\boldsymbol{m}) < f_t(\boldsymbol{m}') \leq f_t(\boldsymbol{m}'')$, and hence $f_t(\boldsymbol{m}) < f_t(\boldsymbol{m}'')$. To handle the other clauses, we make a case distinction on u.

- *Case $u \in U \setminus U'$.* Atomic proposition $\theta = (f_u(\boldsymbol{m}) \leq f_u(\boldsymbol{m}'))$ of φ_{inv} satisfies $\theta \rightsquigarrow_u \varphi_u$. Indeed, if $f_u(\boldsymbol{m}) \leq f_u(\boldsymbol{m}')$ and $\boldsymbol{m}' \xrightarrow{u} \boldsymbol{m}''$, then we have $f_u(\boldsymbol{m}) < f_u(\boldsymbol{m}')$ by Item 5.
- *Case $u \in U'$.* By Item 4, each atomic proposition θ of φ_{inv} satisfies $\theta \rightsquigarrow_u \theta$.
  - *Case $u \in {}^{\bullet}R$.* We have $\theta' \rightsquigarrow_u (\boldsymbol{m}(Q) + \boldsymbol{m}'(R) > 0)$ for any atomic proposition θ', since $\boldsymbol{m}' \xrightarrow{u} \boldsymbol{m}''$ implies $\boldsymbol{m}''(R) > 0$ (regardless of θ').
  - *Case $u \in Q^{\bullet}$.* If $\boldsymbol{m}'(Q) \leq 0$, then u is disabled in $\boldsymbol{m}'$. Thus, it only remains to handle $\theta_{>0} \coloneqq (\boldsymbol{m}(Q) + \boldsymbol{m}'(R) > 0)$. Since R is a trap of $\mathcal{N}_{U'}$, firing u from $\boldsymbol{m}'$ does not empty R, and hence $\theta_{>0} \rightsquigarrow_u \theta_{>0}$.
  - *Case $u \in U''$.* Let $\theta_{\leq 0} \coloneqq (\boldsymbol{m}(R) + \boldsymbol{m}'(Q) \leq 0)$ and $\theta_{>0} \coloneqq (\boldsymbol{m}(Q) + \boldsymbol{m}'(R) > 0)$. Since Q and R are respectively a siphon and trap of $\mathcal{N}_{U'}$, we have $\theta_{\leq 0} \rightsquigarrow_u \theta_{\leq 0}$ and $\theta_{>0} \rightsquigarrow_u \theta_{>0}$. Moreover, by induction hypothesis, for every $i \in [1..m]$, there exists $j \in [1..m]$ such that $\psi_i \rightsquigarrow_u \psi_j$.

We conclude the proof by observing that it is constructive and can be turned into Algorithm 1. The procedure works in polynomial time. Indeed, there are at most $|U|$ recursive calls. Moreover, each set can be obtained in polynomial time via either linear programming or maximal siphons/traps computations [9]. □

Example 2. Let us apply the construction of Theorem 2 to the Petri net and the markings of Example 1: $\boldsymbol{m}_{\mathrm{src}} = \{p_1 \mapsto 2, p_2 \mapsto 0, p_3 \mapsto 0, p_4 \mapsto 0\}$ and $\boldsymbol{m}_{\mathrm{tgt}} \coloneqq \{p_1 \mapsto 0, p_2 \mapsto 0, p_3 \mapsto 1, p_4 \mapsto 0\}$. The locally closed bi-separator is the formula φ below, where the colored arrows represent the relations $\rightsquigarrow_{t_1}, \ldots, \rightsquigarrow_{t_4}$:

t_1, t_2, t_3, t_4 ⟿ $[\boldsymbol{m}(p_4) < \boldsymbol{m}'(p_4)] \vee$

t_4 $[\boldsymbol{m}(p_4) \le \boldsymbol{m}'(p_4) \wedge \boldsymbol{m}(p_4) + \boldsymbol{m}'(p_4) > 0] \vee$ ⟿ t_1, t_2, t_3

t_4 $[\boldsymbol{m}(p_4) \le \boldsymbol{m}'(p_4) \wedge \boldsymbol{m}'(p_1) + \boldsymbol{m}'(p_2) > 0] \vee$ ⟿ t_2 ⟿ t_1, t_3 t_2

t_4 $[\boldsymbol{m}(p_4) \le \boldsymbol{m}'(p_4) \wedge \boldsymbol{m}(p_1) + \boldsymbol{m}(p_2) \le 0 \wedge -\boldsymbol{m}(p_3) \le -\boldsymbol{m}'(p_3)]$ ⟿ t_1, t_3
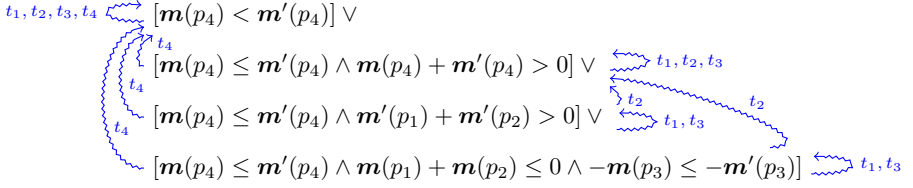
The forward separator $\psi(\boldsymbol{m}) \coloneqq \varphi(\boldsymbol{m}_{\mathrm{src}}, \boldsymbol{m})$ is, after simplifications, given by

$$\psi(\boldsymbol{m}) \equiv \boldsymbol{m}(p_1) + \boldsymbol{m}(p_2) > 0 \vee \boldsymbol{m}(p_4) > 0.$$

Similarly, we obtain this backward separator $\psi'(\boldsymbol{m}) \coloneqq \varphi(\boldsymbol{m}, \boldsymbol{m}_{\mathrm{tgt}})$:

$$\psi'(\boldsymbol{m}) \equiv \boldsymbol{m}(p_1) + \boldsymbol{m}(p_2) = 0 \wedge \boldsymbol{m}(p_3) \ge 1 \wedge \boldsymbol{m}(p_4) = 0.$$

The backward separator ψ' provides a much simpler proof of $\boldsymbol{m}_{\mathrm{src}} \overset{*}{\not\to} \boldsymbol{m}_{\mathrm{tgt}}$ than the one of Example 1. The proof goes as follows: ψ' is trivially backward invariant, because markings that only mark p_3 do not backward-enable any transition. In particular, since $\boldsymbol{m}_{\mathrm{tgt}}$ only marks p_3, it can only be reached from $\boldsymbol{m}_{\mathrm{tgt}}$.   □

Algorithm 1: Construction of a locally closed bi-sep. for $(\boldsymbol{m}_{\mathrm{src}}, \boldsymbol{m}_{\mathrm{tgt}})$.

Input: $\mathcal{N} = (P, T, F)$, $U \subseteq T$ and $\boldsymbol{m}_{\mathrm{src}}, \boldsymbol{m}_{\mathrm{tgt}} \in \mathbb{Q}_+^P$ s.t. $\boldsymbol{m}_{\mathrm{src}} \not\to^{U^*} \boldsymbol{m}_{\mathrm{tgt}}$
Output: A locally closed bi-separator w.r.t. $\mathcal{N}_U$

bi-separator(U)
  if $U = \emptyset$ **then**
    pick $p \in P$ such that $\boldsymbol{m}_{\mathrm{src}}(p) \neq \boldsymbol{m}_{\mathrm{tgt}}(p)$
    return $(\boldsymbol{a}\boldsymbol{m} \le \boldsymbol{a}\boldsymbol{m}')$ *where* $\boldsymbol{a} \coloneqq \mathrm{sign}(\boldsymbol{m}_{src}(p) - \boldsymbol{m}_{tgt}(p)) \cdot \boldsymbol{e}_p$
  else
    $\boldsymbol{b}\ \coloneqq \boldsymbol{m}_{\mathrm{tgt}} - \boldsymbol{m}_{\mathrm{src}}$
    $X\ \coloneqq \{\boldsymbol{x} \in \mathbb{R}_+^T : \mathbf{F}\boldsymbol{x} = \boldsymbol{b}, \mathrm{supp}(\boldsymbol{x}) \subseteq U\}$
    $Y_S \coloneqq \{\boldsymbol{y} \in \mathbb{R}^P : \mathbf{F}^\mathsf{T}\boldsymbol{y} \ge_U \boldsymbol{0}, \boldsymbol{b}^\mathsf{T}\boldsymbol{y} \le 0, \boldsymbol{b}^\mathsf{T}\boldsymbol{y} < \sum_{s \in S}(\mathbf{F}^\mathsf{T}\boldsymbol{y})_s\}$
    if $X = \emptyset$ **then**
      pick $\boldsymbol{y} \in Y_\emptyset$ **and return** $(\boldsymbol{y}^\mathsf{T}\boldsymbol{m} \le \boldsymbol{y}^\mathsf{T}\boldsymbol{m}')$
    else
      $U' \coloneqq \{u \in U : \boldsymbol{x}(u) > 0 \text{ for some } \boldsymbol{x} \in X\}$
      for $t \in U \setminus U'$ **do**
        pick $\boldsymbol{y}_t \in Y_{\{t\}}$; $f_t(\boldsymbol{m}) \coloneqq \boldsymbol{y}_t^\mathsf{T}\boldsymbol{m}$
        if $f_t(\boldsymbol{m}_{src}) > f_t(\boldsymbol{m}_{tgt})$ **then return** $(f_t(\boldsymbol{m}) < f_t(\boldsymbol{m}'))$
      $Q \coloneqq$ largest siphon of $\mathcal{N}_{U'}$ such that $\boldsymbol{m}_{\mathrm{src}}(Q) = 0$
      $R \coloneqq$ largest trap   of $\mathcal{N}_{U'}$ such that $\boldsymbol{m}_{\mathrm{tgt}}(R) = 0$
      $\varphi_{\mathrm{inv}} \coloneqq \bigwedge_{t \in U \setminus U'}(f_t(\boldsymbol{m}) \le f_t(\boldsymbol{m}'))$
      $\psi_1 \vee \cdots \vee \psi_m \coloneqq$ bi-separator($U' \setminus (Q^\bullet \cup {}^\bullet R)$)
      return $\bigvee_{t \in U \setminus U'} \varphi_t(\boldsymbol{m}, \boldsymbol{m}') \vee [\varphi_{\mathrm{inv}}(\boldsymbol{m}, \boldsymbol{m}') \wedge \boldsymbol{m}(Q) + \boldsymbol{m}'(R) > 0] \vee$
           $\bigvee_{1 \le i \le m}[\varphi_{\mathrm{inv}}(\boldsymbol{m}, \boldsymbol{m}') \wedge \boldsymbol{m}(R) + \boldsymbol{m}'(Q) \le 0 \wedge \psi_i(\boldsymbol{m}, \boldsymbol{m}')]$

6 Checking locally closed bi-separators is in NC

We show that the problem of deciding whether a given linear formula is a locally closed bi-separator is in NC. To do so, we provide a characterization of $\psi \leadsto_t \psi$ for homogeneous atomic propositions ψ and ψ'. We only focus on forward firability, as backward firability can be expressed as forward firability in the transpose Petri net. Recall that $\psi \leadsto_t \psi'$ holds iff the following holds:

$$(\boldsymbol{m}, \boldsymbol{m}') \in \llbracket \psi \rrbracket \text{ and } \boldsymbol{m}' \xrightarrow{\alpha t} \boldsymbol{m}'' \text{ imply } (\boldsymbol{m}, \boldsymbol{m}'') \in \llbracket \psi' \rrbracket. \qquad (*)$$

Property (*) can be rephrased as:

$$(\boldsymbol{m}, \boldsymbol{m}') \in \llbracket \psi \rrbracket \text{ and } \boldsymbol{m}' \geq \alpha \cdot \Delta_t^- \text{ imply } (\boldsymbol{m}, \boldsymbol{m}' + \alpha \cdot \Delta_t) \in \llbracket \psi' \rrbracket.$$

As we will see towards the end of the section, due to homogeneity, it actually suffices to consider the case where $\alpha = 1$, which yields this reformulation:

$$\underbrace{\{(\boldsymbol{m}, \boldsymbol{m}') \in \llbracket \psi \rrbracket : \boldsymbol{m}' \geq \Delta_t^- \}}_{X} \subseteq \underbrace{\{(\boldsymbol{m}, \boldsymbol{m}') : (\boldsymbol{m}, \boldsymbol{m}' + \Delta_t) \in \llbracket \psi' \rrbracket \}}_{Y}.$$

Therefore, testing $\psi \leadsto_t \psi'$ amounts to the inclusion check $X \subseteq Y$. Of course, if $X = \emptyset$, then this is trivial. Hence, we will suppose that $X \neq \emptyset$, assuming for now that it can somehow be tested efficiently. In the forthcoming Propositions 8 and 9, we will provide necessary and sufficient conditions for $X \subseteq Y$ to hold. In Proposition 10, we will show that these conditions are testable in NC. Then, in Proposition 11, we will explain how to check whether $X \neq \emptyset$ actually holds.

For $X \subseteq Y$, we can characterize the case of atomic propositions ψ that use "$\leq$" (rather than "$<$") with a generalization of Farkas' lemma:

Proposition 8. *Let $\boldsymbol{a}, \boldsymbol{a}', \boldsymbol{l} \in \mathbb{R}^n$ and $b' \in \mathbb{R}$. Let $X := \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{a}\boldsymbol{x} \leq 0 \wedge \boldsymbol{x} \geq \boldsymbol{l}\}$ and $Y := \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{a}'\boldsymbol{x} \leq b'\}$ be such that $X \neq \emptyset$. It is the case that $X \subseteq Y$ iff there exists $\lambda \geq 0$ such that $\lambda \boldsymbol{a} \geq \boldsymbol{a}'$ and $-b' \leq (\lambda \boldsymbol{a} - \boldsymbol{a}')\boldsymbol{l}$.*

We now give the conditions for all four combinations of "$\leq$" and "$<$":

Proposition 9. *Let $\boldsymbol{a}, \boldsymbol{a}' \in \mathbb{R}^n$, $b' \in \mathbb{R}$, $\boldsymbol{l} \geq \boldsymbol{0}$ and $\sim, \sim' \in \{\leq, <\}$. Let $X_\sim := \{\boldsymbol{x} \geq \boldsymbol{l} : \boldsymbol{a}\boldsymbol{x} \sim \boldsymbol{0}\}$ and $Y_{\sim'} := \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{a}'\boldsymbol{x} \sim' b'\}$ be such that $X_\sim \neq \emptyset$. It holds that $X_\sim \subseteq Y_{\sim'}$ iff there exists $\lambda \geq 0$ s.t. $\lambda \boldsymbol{a} \geq \boldsymbol{a}'$ and one of the following holds:*

1. $\sim' = \leq$ and $-b' \leq (\lambda \boldsymbol{a} - \boldsymbol{a}')\boldsymbol{l}$;
2. $\sim = \leq$, $\sim' = <$, and $-b' < (\lambda \boldsymbol{a} - \boldsymbol{a}')\boldsymbol{l}$;
3. $\sim = <$, $\sim' = <$, and either $-b' < (\lambda \boldsymbol{a} - \boldsymbol{a}')\boldsymbol{l}$ or $-b' = (\lambda \boldsymbol{a} - \boldsymbol{a}')\boldsymbol{l} \wedge \lambda > 0$.

Proof.

1. If $\sim = \leq$, then it follows immediately from Proposition 8. Thus, assume $\sim = <$. We claim that $X_< \subseteq Y_\leq$ iff $X_\leq \subseteq Y_\leq$. The validity of this claim concludes the proof of this case as we have handled $\sim = \leq$ and as $X_\leq \supseteq X_< \neq \emptyset$.

Let us show the claim. It is clear that $X_< \subseteq Y_\le$ is implied by $X_\le \subseteq Y_\le$. So, we only have to show direction from left to right. For the sake of contradiction, suppose that $X_< \subseteq Y_\le$ and $X_\le \not\subseteq Y_\le$. Let $X_= := X_\le \setminus X_<$. Note that $X_= \ne \emptyset$. Let $\boldsymbol{x} \in X_<$ and $\boldsymbol{x}' \in X_= \setminus Y_\le$. We have $\boldsymbol{x}, \boldsymbol{x}' \ge \boldsymbol{l}$, $\boldsymbol{a}\boldsymbol{x} < 0$, $\boldsymbol{a}\boldsymbol{x}' = 0$, $\boldsymbol{a}'\boldsymbol{x} = c \le b'$ and $\boldsymbol{a}'\boldsymbol{x}' = c' > b'$ for some $c, c' \in \mathbb{R}$. In particular, $b' \in [c, c')$. Let $\epsilon \in (0, 1]$ be such that $b' < \epsilon c + (1 - \epsilon)c'$. Let $\boldsymbol{x}'' := \epsilon \boldsymbol{x} + (1 - \epsilon)\boldsymbol{x}'$. Observe that $\boldsymbol{x}'' \ge \boldsymbol{l}$. Moreover, we have:

$$\boldsymbol{a}\boldsymbol{x}'' = \epsilon \boldsymbol{a}\boldsymbol{x} + (1 - \epsilon)\boldsymbol{a}\boldsymbol{x}' \quad = \epsilon \boldsymbol{a}\boldsymbol{x} \qquad\qquad < 0,$$
$$\boldsymbol{a}'\boldsymbol{x}'' = \epsilon \boldsymbol{a}'\boldsymbol{x} + (1 - \epsilon)\boldsymbol{a}'\boldsymbol{x}' = \epsilon c + (1 - \epsilon)c' > b'.$$

Therefore, we have $\boldsymbol{x}'' \in X_<$ and $\boldsymbol{x}'' \notin Y_\le$, which is a contradiction.

2. $\Rightarrow$) Since $X_\le \subseteq Y_<$, the system $\exists \boldsymbol{x} : \boldsymbol{x} \ge \boldsymbol{l} \wedge \boldsymbol{a}\boldsymbol{x} \le 0 \wedge \boldsymbol{a}'\boldsymbol{x} \ge b'$ has no solution. In matrix notation, the system corresponds to $\exists \boldsymbol{x} : \mathbf{A}\boldsymbol{x} \le \boldsymbol{c}$ where

$$\mathbf{A} := \begin{bmatrix} -\mathbf{I} \\ \boldsymbol{a} \\ -\boldsymbol{a}' \end{bmatrix} \text{ and } \boldsymbol{c} := \begin{pmatrix} -\boldsymbol{l} \\ 0 \\ -b' \end{pmatrix}.$$

By Farkas' lemma (Lemma 1), $\mathbf{A}^\mathsf{T}\boldsymbol{y} = \boldsymbol{0}$ and $\boldsymbol{c}^\mathsf{T}\boldsymbol{y} < 0$ for some $\boldsymbol{y} \ge \boldsymbol{0}$. In other words,

$$\exists \boldsymbol{z} \ge \boldsymbol{0}, \lambda, \lambda' \ge 0 : \lambda \boldsymbol{a} - \lambda' \boldsymbol{a}' = \boldsymbol{z} \wedge -\lambda' b' < \boldsymbol{z}\boldsymbol{l}.$$

Since $\boldsymbol{z} \ge \boldsymbol{0}$, we have $\lambda \boldsymbol{a} \ge \lambda' \boldsymbol{a}' \wedge -\lambda' b' < (\lambda \boldsymbol{a} - \lambda' \boldsymbol{a}')\boldsymbol{l}$. If $\lambda' > 0$, then we are done by dividing all terms by λ'. For the sake of contradiction, suppose that $\lambda' = 0$. This means that $\lambda \boldsymbol{a} \ge \boldsymbol{0}$ and $0 < \lambda \boldsymbol{a}\boldsymbol{l}$. We necessarily have $\lambda > 0$ and $\boldsymbol{a}\boldsymbol{l} > 0$. Let $\boldsymbol{x} \in X_\le$. We have $0 \ge \boldsymbol{a}\boldsymbol{x} \ge \boldsymbol{a}\boldsymbol{l} > 0$, which is a contradiction.

$\Leftarrow$) Let $\boldsymbol{x} \in X_\le$. We have $\boldsymbol{a}'\boldsymbol{x} < b'$ and hence $\boldsymbol{x} \in Y_<$ as desired, since:

$$
\begin{aligned}
-b' &< (\lambda \boldsymbol{a} - \boldsymbol{a}')\boldsymbol{l} \\
&\le (\lambda \boldsymbol{a} - \boldsymbol{a}')\boldsymbol{x} && \text{(by } (\lambda \boldsymbol{a} - \boldsymbol{a}') \ge \boldsymbol{0} \text{ and } \boldsymbol{x} \ge \boldsymbol{l} \ge \boldsymbol{0}) \\
&= \lambda \boldsymbol{a}\boldsymbol{x} - \boldsymbol{a}'\boldsymbol{x} \\
&\le -\boldsymbol{a}'\boldsymbol{x} && \text{(by } \lambda \ge 0 \text{ and } \boldsymbol{a}\boldsymbol{x} \le 0).
\end{aligned}
$$

3. The proof is similar albeit slightly more complicated. $\qquad\square$

The conditions arising from Proposition 9 involve solving linear programs with *one* variable λ. It is easy to see that this problem is in NC:

Proposition 10. *Given $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{Q}^n$ and $\sim \in \{\le, <\}^n$, testing $\exists \lambda \ge 0 : \boldsymbol{a}\lambda \sim \boldsymbol{b}$ is in NC.*

Recall that at the beginning of the section we made the assumption that some pair $(\boldsymbol{m}, \boldsymbol{m}') \in \llbracket \psi \rrbracket$ is such that $\boldsymbol{m}'$ enables a transition t. Checking whether this is actually true has a cost. Fortunately, we provide a simple characterization of enabledness which can checked in NC. Formally, we say that φ *enables* t if there exists $(\boldsymbol{m}, \boldsymbol{m}') \in \llbracket \varphi \rrbracket$ such that $\boldsymbol{m}'$ α-enables t for some $\alpha > 0$. We have:

Proposition 11. *Let $\varphi_\sim(\boldsymbol{m}, \boldsymbol{m}') := \boldsymbol{a}\boldsymbol{m} \sim \boldsymbol{b}\boldsymbol{m}'$ where $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^P$. This holds:*

1. $\varphi_<$ *enables u iff $\boldsymbol{a} \not\geq \boldsymbol{0}$ or $\boldsymbol{b} \not\leq \boldsymbol{0}$, and*
2. $\varphi_\leq$ *enables u iff $\boldsymbol{b}\Delta_u^- \geq 0$ or $(\boldsymbol{b}\Delta_u^- < 0 \wedge (\boldsymbol{a}, -\boldsymbol{b}) \not\geq (\boldsymbol{0}, \boldsymbol{0}))$.*

Proof.

1. $\Rightarrow$) Since $\varphi_<$ enables u, we have $[\![\varphi_<]\!] \neq \emptyset$. Let $(\boldsymbol{m}, \boldsymbol{m}') \in [\![\varphi_<]\!]$. We have $\boldsymbol{a}\boldsymbol{m} < \boldsymbol{b}\boldsymbol{m}'$. It cannot be that $\boldsymbol{a} \geq \boldsymbol{0}$ and $\boldsymbol{b} \leq \boldsymbol{0}$, as otherwise $\boldsymbol{a}\boldsymbol{m} \geq 0 \geq \boldsymbol{b}\boldsymbol{m}'$.

   $\Leftarrow$) It suffices to give a pair $(\boldsymbol{m}, \boldsymbol{m}') \in [\![\varphi_<]\!]$ such that $\boldsymbol{m}' \geq \Delta_u^-$. Informally, if $\boldsymbol{a}$ has a negative value (resp. $\boldsymbol{b}$ has a positive value), then we can consider the pair $(\boldsymbol{0}, \Delta_u^-)$ and "fix" the value on the left-hand-side (resp. right-hand side) so that $\varphi_<$ is satisfied. More formally, if $\boldsymbol{a}(p) < 0$, then $(k\boldsymbol{e}_p, \Delta_u^-) \in [\![\varphi_<]\!]$ with $k := (|\boldsymbol{b}\Delta_u^-| + 1)/|\boldsymbol{a}(p)|$; if $\boldsymbol{b}(p) > 0$, then $(\boldsymbol{0}, \Delta_u^- + k\boldsymbol{e}_p) \in [\![\varphi_<]\!]$ with $k := (|\boldsymbol{b}\Delta_u^-| + 1)/\boldsymbol{b}(p)$.

2. The proof is similar albeit slightly more complicated. $\qquad\square$

We can finally show that testing $\psi \rightsquigarrow_t \psi'$ can be done in NC, for atomic propositions ψ and ψ'. In turn, this allows us to show that we can test in NC whether a linear formula is a locally closed bi-separator.

Proposition 12. *Given a Petri net $\mathcal{N}$, a transition t and homogeneous atomic propositions ψ and ψ', testing whether $\psi \rightsquigarrow_t \psi'$ can be done in NC.*

Proof. Recall that addition, subtraction, multiplication, division and comparison can be done in NC. Note that, by Proposition 11, we can check whether ψ enables t in NC. If it does, then we must test whether $(\boldsymbol{m}, \boldsymbol{m}') \in [\![\psi]\!]$ and $\boldsymbol{m}' \xrightarrow{\alpha t} \boldsymbol{m}''$ implies $(\boldsymbol{m}, \boldsymbol{m}'') \in [\![\psi']\!]$. We claim that this amounts to testing $X \subseteq Y$, where:

$$X := \{(\boldsymbol{m}, \boldsymbol{m}') \in \mathbb{R}_+^P \times \mathbb{R}_+^P : (\boldsymbol{m}, \boldsymbol{m}') \in [\![\psi]\!] \text{ and } (\boldsymbol{m}, \boldsymbol{m}') \geq (\boldsymbol{0}, \Delta_t^-)\},$$
$$Y := \{(\boldsymbol{m}, \boldsymbol{m}') \in \mathbb{R}_+^P \times \mathbb{R}_+^P : (\boldsymbol{m}, \boldsymbol{m}' + \Delta_t) \in [\![\psi']\!]\}.$$

Let us prove this claim.

$\Rightarrow$) Let $(\boldsymbol{m}, \boldsymbol{m}') \in X$. We have $(\boldsymbol{m}, \boldsymbol{m}') \in [\![\psi]\!]$ and $(\boldsymbol{m}, \boldsymbol{m}') \geq (\boldsymbol{0}, \Delta_t^-)$. Thus $\boldsymbol{m}' \xrightarrow{t} \boldsymbol{m}' + \Delta_t$. By assumption, $(\boldsymbol{m}, \boldsymbol{m}' + \Delta_t) \in [\![\psi']\!]$, and hence $(\boldsymbol{m}, \boldsymbol{m}') \in Y$.

$\Leftarrow$) Let $(\boldsymbol{m}, \boldsymbol{m}') \in [\![\psi]\!]$ and $\boldsymbol{m}' \xrightarrow{\alpha t} \boldsymbol{m}''$. We have $\boldsymbol{m}' \geq \alpha \Delta_t^-$ and $\boldsymbol{m}'' = \boldsymbol{m}' + \alpha \Delta_t$. Let $\boldsymbol{k} := \boldsymbol{m}/\alpha$, $\boldsymbol{k}' := \boldsymbol{m}'/\alpha$ and $\boldsymbol{k}'' := \boldsymbol{m}''/\alpha$. As $\alpha > 0$ and ψ is homogeneous, we have $(\boldsymbol{k}, \boldsymbol{k}') \in [\![\psi]\!]$, $(\boldsymbol{k}, \boldsymbol{k}') \geq (\boldsymbol{0}, \Delta_t^-)$ and $\boldsymbol{k}'' = \boldsymbol{k}' + \Delta_t$. Thus, $(\boldsymbol{k}, \boldsymbol{k}') \in X \subseteq Y$. By definition of Y, this means that $(\boldsymbol{k}, \boldsymbol{k}'') \in [\![\psi']\!]$. By homogeneity, we conclude that $(\boldsymbol{m}, \boldsymbol{m}'') \in [\![\psi']\!]$.

Now that we have shown the claim, let us explain how to check whether $X \subseteq Y$ in NC. Note that $X \neq \emptyset$ since ψ enables t. Thus, by Proposition 9, testing $X \subseteq Y$ amounts to solving a linear program in one variable. For example, if $\psi = (\boldsymbol{a} \cdot (\boldsymbol{m}, \boldsymbol{m}') \leq 0)$ and $\psi' = (\boldsymbol{a}' \cdot (\boldsymbol{m}, \boldsymbol{m}') < 0)$, then we must check whether this system has a solution:

$$\exists \lambda \geq 0 : \lambda \boldsymbol{a} \geq \boldsymbol{a}' \wedge \boldsymbol{a} \cdot (\boldsymbol{0}, \Delta_t) < (\lambda \boldsymbol{a} - \boldsymbol{a}') \cdot (\boldsymbol{0}, \Delta_t^-).$$

Thus, by Proposition 10, testing $X \subseteq Y$ can be done in NC. $\qquad\square$

Theorem 3. *Given $\mathcal{N} = (P, T, F)$, $\boldsymbol{m}_{src}, \boldsymbol{m}_{tgt} \in \mathbb{Q}_+^P$ and a formula φ, testing whether φ is a locally closed bi-separator for $(\boldsymbol{m}_{src}, \boldsymbol{m}_{tgt})$ can be done in NC.*

Proof. Recall that $\varphi = \varphi_1 \vee \cdots \vee \varphi_n$ must be in DNF with homogeneous atomic propositions. As arithmetic belongs in NC and φ is in DNF, we can test whether $(\boldsymbol{m}_{\mathrm{src}}, \boldsymbol{m}_{\mathrm{src}}) \in [\![\varphi]\!]$, $(\boldsymbol{m}_{\mathrm{tgt}}, \boldsymbol{m}_{\mathrm{tgt}}) \in [\![\varphi]\!]$ and $(\boldsymbol{m}_{\mathrm{src}}, \boldsymbol{m}_{\mathrm{tgt}}) \notin [\![\varphi]\!]$ in NC by evaluating φ in parallel. We can further test whether φ is locally closed by checking the following (which is simply the definition of "locally closed"):

$$\left[\bigwedge_{\substack{t \in T \\ i \in [1..n]}} \bigvee_{j \in [1..n]} \bigwedge_{\psi \in \varphi_i} \bigvee_{\psi' \in \varphi_j} \psi \rightsquigarrow_t \psi' \right] \wedge \left[\bigwedge_{\substack{t \in T^\mathsf{T} \\ i \in [1..n]}} \bigvee_{j \in [1..n]} \bigwedge_{\psi \in \varphi_i} \bigvee_{\psi' \in \varphi_j} \psi^\mathsf{T} \rightsquigarrow_t \psi'^\mathsf{T} \right].$$

By Proposition 12, each test $\psi \rightsquigarrow_t \psi'$ can be carried in NC. Therefore, we can perform all of them in parallel. Note that we do not have to explicitly compute the transpose of transitions and formulas; we can simply swap arguments.   □

Remark 1. Testing whether φ is locally closed is even simpler if the tester is also given annotations indicating for every clause φ_i and transition t which clause φ_j is supposed to satisfy $\varphi_i \rightsquigarrow_t \varphi_j$. This mapping is a byproduct of the procedure to compute a locally closed bi-separator, and so comes at no cost.   □

7   Bi-separators for set-to-set unreachability

In most applications, one does not have to prove unreachability of one marking, but rather of a *set* of markings, usually defined by means of some simple linear constraints. We show that our approach can be extended to "set-to-set reachability", i.e. queries of the form $\exists \boldsymbol{m}_{\mathrm{src}} \in A, \boldsymbol{m}_{\mathrm{tgt}} \in B : \boldsymbol{m}_{\mathrm{src}} \rightarrow^* \boldsymbol{m}_{\mathrm{tgt}}$, which we denote by $A \rightarrow^* B$. We focus on the case where sets A and B are described by conjunctions of atomic propositions; in other words, A and B are convex polytopes defined as intersections of half-spaces. In particular, this includes "coverability" queries which are important in practice, i.e. where A is a singleton and B is of the form $\{\boldsymbol{m} : \boldsymbol{m} \geq \boldsymbol{b}\}$. More generally, our approach can directly be adapted to convex linear Horn constraints, which is a fragment of linear arithmetic that extends linear programs and that captures the expressiveness of continuous Petri nets [6].

As shown in [6, Lem. 3.7], given an atomic proposition $\psi = (\boldsymbol{a}\boldsymbol{x} \sim b)$, one can construct (in logarithmic space) a Petri net $\mathcal{N}_\psi$ and some $\boldsymbol{y} \in \{0,1\}^5$ such that $\psi(\boldsymbol{x})$ holds iff $(\boldsymbol{x}, \boldsymbol{y}) \rightarrow^* (\boldsymbol{0}, \boldsymbol{0})$ in $\mathcal{N}_\psi$. The idea—depicted in Figure 2, which is adapted from [6, Fig. 1])—is simply to cancel out positive and negative coefficients of ψ. It is straightforward to adapt this construction to a conjunction $\bigwedge_{1 \leq i \leq k} \psi_k(\boldsymbol{x})$ of atomic propositions. Indeed, it suffices to make k copies of the gadget, but where places $\{p_1, \ldots, p_n\}$ and transitions $\{t_1, \ldots, t_n\}$ are shared. In this more general setting, t_i consumes from p_i and simultaneously spawns the respective coefficient to each copy. In summary, the following holds:
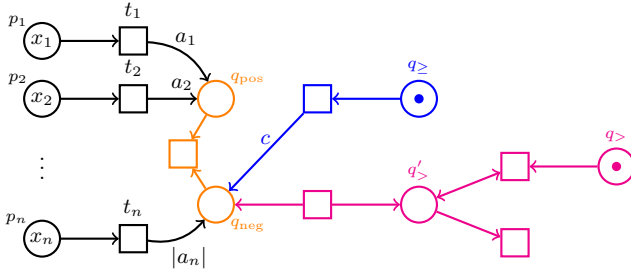
Fig. 2. Petri net for $\psi(\boldsymbol{x}) = (a_1 \cdot x_1 + \cdots + a_n \cdot x_n > c)$ where $a_1, a_2, c > 0$ and $a_n < 0$.

Proposition 13. *Given a conjunction of atomic propositions φ, it is possible to construct, in logarithmic space, a Petri net $\mathcal{N}_\varphi$ and $\boldsymbol{y} \in \{0,1\}^{5k}$ such that $\varphi(\boldsymbol{x})$ holds iff $(\boldsymbol{x}, \boldsymbol{y}) \to^* (\boldsymbol{0}, \boldsymbol{0})$ in $\mathcal{N}_\varphi$.*

With the previous construction in mind, we can reformulate any set-to-set reachability query into a standard ("marking-to-marking") reachability query.

Proposition 14. *Given a Petri net $\mathcal{N}$ and convex polytopes A and B described as conjunctions of atomic propositions, one can construct, in log. space, a Petri net $\mathcal{N}'$ and markings $\boldsymbol{m}_{src}$ and $\boldsymbol{m}_{tgt}$ s.t. $A \to^* B$ in $\mathcal{N}$ iff $\boldsymbol{m}_{src} \to^* \boldsymbol{m}_{tgt}$ in $\mathcal{N}'$.*

Proof. Let $\mathcal{N} = (P, T, \mathbf{F}_-, \mathbf{F}_+)$ where $P = \{p_1, \ldots, p_n\}$. Let us describe $\mathcal{N}' = (P', T', \mathbf{F}'_-, \mathbf{F}'_+)$ with the help of Figure 3. The Petri net $\mathcal{N}'$ extends $\mathcal{N}$ as follows:

- we add transitions $\{t_1, \ldots, t_n\}$ whose purpose is to nondeterministically guess an initial marking of $\mathcal{N}$ in P, and make a copy in $P' \coloneqq \{p'_1, \ldots, p'_n\}$;
- we add a gadget, obtained from Proposition 13, to test whether the marking in P' belongs to A; and we add a gadget, obtained from Proposition 13, to test whether the marking in P belongs to B.
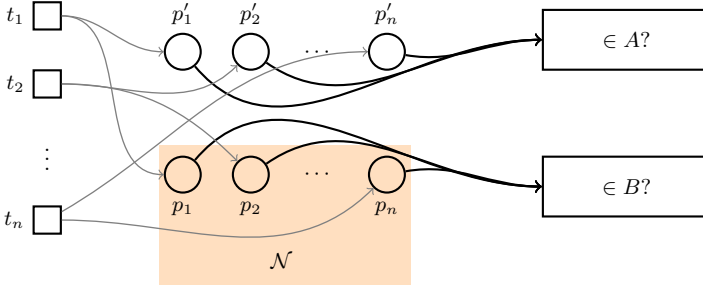

Fig. 3. Reduction from set-to-set reachability to (marking-to-marking) reachability.

The Petri net $\mathcal{N}'$ is *intended* to work sequentially as follows: (1) guess the initial marking $\boldsymbol{m}$ of $\mathcal{N}$; (2) execute $\mathcal{N}$ on $\boldsymbol{m}$ and reach a marking $\boldsymbol{m}'$; and (3) test whether $\boldsymbol{m} \in A$ and $\boldsymbol{m}' \in B$. If $\mathcal{N}'$ follows this order, then it is straightforward to see that $A \to^* B$ in $\mathcal{N}$ iff $(\boldsymbol{0}, \boldsymbol{0}, \boldsymbol{y}, \boldsymbol{y}') \to^* (\boldsymbol{0}, \boldsymbol{0}, \boldsymbol{0}, \boldsymbol{0})$ in $\mathcal{N}'$, where $\boldsymbol{y}$ and $\boldsymbol{y}'$ are obtained from Proposition 13. However, $\mathcal{N}'$ may interleave the different phases.[4] Nonetheless, this is not problematic, as any run of $\mathcal{N}'$ can be reordered in such a way that all three phases are consecutive. Indeed, phase (1) only produces tokens in $P \cup P'$, and phase (3) only consumes tokens from $P \cup P'$. □

As a consequence of Proposition 14, combined with Theorems 2 and 3, we obtain the following corollary:

Corollary 1. *A negative answer to a convex polytope query $A \to^* B$ is witnessed by a locally closed bi-separator computable in polynomial time and checkable in NC.*

8   Conclusion

We have shown that continuous Petri nets admit locally closed bi-separators that can be efficiently computed. These separators are succinct and very efficiently checkable certificates of unreachability. In particular, checking that a linear formula is a locally closed bi-separator is in NC, and only requires to solve linear inequations in one variable over the nonnegative reals.

Verification tools that have not been formally verified, or rely (as is usually the case) on external packages for linear arithmetic, can apply our results to provide certificates for their output. Further, our separators can be used as explanations of why a certain marking is unreachable. Obtaining minimal explanations is an interesting research avenue.

From a logical point of view, separators are very closely related to interpolants for linear arithmetic, which are widely used in formal verification to refine abstractions in the CEGAR approach [3,17,18,1]. We intend to explore whether they can constitute the basis of a CEGAR approach for the verification of continuous Petri nets.

References

1. Althaus, E., Beber, B., Kupilas, J., Scholl, C.: Improving interpolants for linear arithmetic. In: Proc. 13th International on Automated Technology for Verification and Analysis (ATVA). pp. 48–63 (2015). https://doi.org/10.1007/978-3-319-24953-7_5

---

[4] It is tempting to implement a lock, but this only works under discrete semantics.

2. Baumann, P., Majumdar, R., Thinniyam, R.S., Zetzsche, G.: Context-bounded verification of liveness properties for multithreaded shared-memory programs. Proceedings of the ACM on Programming Languages (PACMPL) **5**, 1–31 (2021). https://doi.org/10.1145/3434325

3. Beyer, D., Zufferey, D., Majumdar, R.: Csisat: Interpolation for LA+EUF. In: Proc. 20th International Conference on Computer Aided Verification (CAV). pp. 304–308 (2008). https://doi.org/10.1007/978-3-540-70545-1_29

4. Blondin, M., Esparza, J., Helfrich, M., Kucera, A., Meyer, P.J.: Checking qualitative liveness properties of replicated systems with stochastic scheduling. In: Proc. 32nd International Conference on Computer Aided Verification (CAV). vol. 12225, pp. 372–397 (2020). https://doi.org/10.1007/978-3-030-53291-8_20

5. Blondin, M., Finkel, A., Haase, C., Haddad, S.: The logical view on continuous Petri nets. ACM Transactions on Computational Logic (TOCL) **18**(3), 24:1–24:28 (2017). https://doi.org/10.1145/3105908

6. Blondin, M., Haase, C.: Logics for continuous reachability in Petri nets and vector addition systems with states. In: Proc. 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS). pp. 1–12 (2017). https://doi.org/10.1109/LICS.2017.8005068

7. Czerwinski, W., Orlikowski, L.: Reachability in vector addition systems is Ackermann-complete. In: Proc. 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS) (2021), to appear

8. David, R., Alla, H.: Discrete, Continuous, and Hybrid Petri nets. Springer, 2 edn. (2010)

9. Desel, J., Esparza, J.: Free choice Petri nets. No. 40, Cambridge University Press (1995)

10. Esparza, J., Helfrich, M., Jaax, S., Meyer, P.J.: Peregrine 2.0: Explaining correctness of population protocols through stage graphs. In: Proc. 18th International Symposium on Automated Technology for Verification and Analysis (ATVA). vol. 12302, pp. 550–556 (2020). https://doi.org/10.1007/978-3-030-59152-6_32

11. Feng, Y., Martins, R., Wang, Y., Dillig, I., Reps, T.W.: Component-based synthesis for complex APIs. In: Proc. 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL). pp. 599–612. ACM (2017). https://doi.org/10.1145/3009837.3009851

12. Fraca, E., Haddad, S.: Complexity analysis of continuous Petri nets. Fundamenta Informaticae **137**(1), 1–28 (2015). https://doi.org/10.3233/FI-2015-1168

13. German, S.M., Sistla, A.P.: Reasoning about systems with many processes. Journal of the ACM **39**(3), 675–735 (1992). https://doi.org/10.1145/146637.146681

14. Leroux, J.: Vector addition systems reachability problem (A simpler solution). In: Turing-100 – The Alan Turing Centenary. vol. 10, pp. 214–228 (2012). https://doi.org/10.29007/bnx2

15. Leroux, J.: The reachability problem for Petri nets is not primitive recursive. In: Proc. 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS) (2021), to appear

16. Leroux, J., Schmitz, S.: Reachability in vector addition systems is primitive-recursive in fixed dimension. In: Proc. 34th Symposium on Logic in Computer Science (LICS). pp. 1–13 (2019). https://doi.org/10.1109/LICS.2019.8785796

17. Rybalchenko, A., Sofronie-Stokkermans, V.: Constraint solving for interpolation. Journal of Symbolic Computation **45**(11), 1212–1233 (2010). https://doi.org/10.1016/j.jsc.2010.06.005

18. Scholl, C., Pigorsch, F., Disch, S., Althaus, E.: Simple interpolants for linear arithmetic. In: Proc. Conference & Exhibition on Design, Automation & Test in Europe (DATE). pp. 1–6 (2014). https://doi.org/10.7873/DATE.2014.128

Graphical Piecewise-Linear Algebra

Guillaume Boisseau[1] and Robin Piedeleu[2]

[1] University of Oxford, Oxford, UK `guillaume.boisseau@cs.ox.ac.uk`
[2] University College London, London, UK `r.piedeleu@ucl.ac.uk`

Abstract. Graphical (Linear) Algebra is a family of diagrammatic languages allowing to reason about different kinds of subsets of vector spaces compositionally. It has been used to model various application domains, from signal-flow graphs to Petri nets and electrical circuits. In this paper, we introduce to the family its most expressive member to date: Graphical Piecewise-Linear Algebra, a new language to specify piecewise-linear subsets of vector spaces.

Like the previous members of the family, it comes with a complete axiomatisation, which means it can be used to reason about the corresponding semantic domain purely equationally, forgetting the set-theoretic interpretation. We show completeness using a single axiom on top of Graphical Polyhedral Algebra, and show that this extension is the smallest that can capture a variety of relevant constructs.

Finally, we showcase its use by modelling the behaviour of stateless electronic circuits of ideal elements, a domain that had remained outside the remit of previous diagrammatic languages.

Keywords: string diagrams · piecewise-linear · prop · axiomatisation

1 Introduction

Functional thinking underpins most scientific models. Nature, however, does not distinguish inputs and outputs—physical systems are governed by laws that merely express *relations* between their observable variables. While influential scientists, like the famous control theorist J. Willems, have pointed out the blind spots of functional thinking [11], it has remained the dominant paradigm in science and engineering. Arguably, our mathematical practice, especially the foundational emphasis on sets and functions, and the limitations of standard algebraic syntax, are partially to blame for the persistence of this status quo. But there are also alternative approaches, that take relations seriously as the primitive building blocks of our mathematical models. Category theory in particular is agnostic about what constitutes a morphism and can accommodate relations as easily as functions.

Relations, with their usual composition and the cartesian product of sets, form a monoidal category—a category in which morphisms can be composed in two different ways. As a result, they admit a natural two-dimensional syntax of *string diagrams*. This notation has several advantages when it comes to

reasoning about open and interconnected systems [1]: string diagrams naturally keep track of structural properties, such as interconnectivity; they factor out irrelevant topological information that standard algebraic syntax needs to keep explicit; variable-sharing—the relational form of composition for systems—is depicted simply by wiring different components together.

As a result, a wealth of recent developments in computer science and beyond have adopted relations and their diagrammatic notation as a unifying language to reason about a broad range of systems, from electrical circuits to Petri nets [2,6,5]. Many of these follow the same methodology. 1) Given a class of systems, find a set of diagrammatic generators from which any system can be specified, using the two available forms of composition. 2) Interpret each of them as a relation between the observable variables of the system that they describe. This defines a structure-preserving mapping—a monoidal functor—from the diagrammatic syntax to the semantics, from the two-dimensional representation of a system to its behaviour. 3) Finally, identify a convenient set of equations between diagrams, from which any semantic equality between the behaviour of the corresponding systems may be derived.

Graphical linear algebra (GLA) is a paradigmatic example of this approach. It provides a diagrammatic syntax to reason compositionally about different types of linear dynamical systems (including for instance traditional *signal flow graphs*) and prove their behavioural equivalence purely diagrammatically. The syntax of GLA is generated by the following primitive components:

$$-\!\!\subset \mid -\!\bullet \mid \rangle\!\!- \mid \bullet\!- \mid -\!\!\subset\!\circ \mid -\!\circ \mid \rangle\!\!- \mid \circ\!- \mid -\!\boxed{x}\!- \quad (x \in \mathbb{K})$$

As relations, the black nodes force all of their ports to share the same value; the white nodes constrain their left ports and the right ports to sum to the same value (or to zero when there are no left/right ports); the final generator, parameterised by an element of the chosen field $\mathbb{K}$, behaves as an amplifier: its right value is x times the left value. Following point 3) of the methodology sketched above, GLA enjoys a sound and complete equational theory for the specified semantics, called the theory of *Interacting Hopf Algebras* (IH). In summary, string diagrams with n ports on the left and m ports on the right, quotiented by the axioms of IH, are precisely linear relations, i.e., linear subspaces of $\mathbb{K}^n \times \mathbb{K}^m$.

GLA was the starting point of different extensions, two of which play a prominent role in this paper. First, Graphical Affine Algebra, which adds to the syntax a generator $\mapsto$ for the constant 1. This allows it to express *affine* relations, i.e. affine subspaces of $\mathbb{K}^n \times \mathbb{K}^m$. A corresponding complete equational theory was presented in [6]. Then, Graphical Polyhedral Algebra (GPA), which assumes that $\mathbb{K}$ is an ordered field and adds a generator $-\!\geq\!-$ for this order. The resulting graphical calculus can express all polyhedral relations, i.e., polyhedra[3] in $\mathbb{K}^n \times \mathbb{K}^m$, and also comes with its own complete axiomatisation.

In this paper, we define the most expressive member of the GLA family tree to date: Graphical Piecewise-Linear Algebra (GPLA) is a hybrid of symbolic and

---

[3] For the case of $\mathbb{R}$, these include the usual polytopes, which are bounded subsets of $\mathbb{R}^n \times \mathbb{R}^m$, as well as proper polyhedra, which may have unbounded faces.

diagrammatic syntax for piecewise-linear (pl) relations—finite unions of polyhedra in $\mathbb{K}^n \times \mathbb{K}^m$—and a corresponding complete equational theory. We argue below that the proposed language strikes a convincing balance between structure and expressiveness. It is a simple extension of GPA [4], yet for $\mathbb{K} = \mathbb{R}$, it is sufficiently powerful to approximate any submanifold of $\mathbb{R}^n$ arbitrarily closely.

Furthermore, this extension completes a research program initiated in parallel with the birth of GLA [2,6,3]: its chief purpose was to give the informal graphical notation for electrical circuits a formal, compositional interpretation, with a corresponding equational theory.

Until now however, the category-theoretic setting could only accommodate components with a linear (more precisely, affine) behaviour, such as resistors, inductors, capacitors, voltage and current sources. GPLA finally makes it possible to reason equationally about electronic components, such as ideal diodes and transistors. Even when the idealised physical behaviour of these components is not necessarily piecewise-linear, GPLA is theoretically expressive enough to approximate it as closely as necessary. Indeed, piecewise-linear approximations of transistor behaviour have been proposed to bypass the unavoidable abstraction leaks of purely digital circuits [9]. In this context, GPLA can serve as a form of abstract interpretation for electronic circuits, with adjustable precision to allow for the intended semantics to be as physically realistic as desired. Of course, in practice, working with large diagrams can be prohibitive. But this is a limitation shared by all members of the Graphical Algebra family, and developing convenient tools and techniques for diagrammatic reasoning is an active research area. Our main thrust is that piecewise-linearity provides the appropriate level of structure, where general relations are too flexible to come with a useful equational theory, and linear relations are too rigid to accommodate diodes and other electronic components.

Finally, a remark about syntax. While it is possible to make the language purely diagrammatic, we found that what one gains in purity one loses in complexity. Ultimately, the hybrid syntax of union and diagrams is more convenient to manipulate and intuitive to read. In fact, this is not the first time that sums of diagrams appear in the literature [8]. Nevertheless, one of our central technical contributions is the rigorous definition of a syntax blending diagrams and binary joins, and the corresponding notion of equational theory.

Outline. In Section 2 we recall the necessary mathematical background, the fundamentals of diagrammatic syntax, and the language of Graphical Polyhedral Algebra (GPA). In Section 3, we extend the diagrammatic syntax with unions and define the notion of symmetric monoidal semi-lattice theory. From there, in Section 4, we extend GPA with unions, to capture piecewise-linear relations, and give this new language a theory that we prove is complete (Theorem 2). This is our main technical contribution. In Section 5, we explore alternative languages for piecewise-linear relations, and show that they are all equally expressive. Finally, in Section 6, we extend the compositional re-interpretation of electrical circuits from [3] to include electronic components, namely diodes and transistors.

2   Preliminaries

Informally, our starting point is a simple diagrammatic language of circuits built from the following generators:

$$\multimap\hspace{-2pt}\subset \mid \multimap\hspace{-2pt}\bullet \mid \succ\hspace{-4pt}\multimap \mid \bullet\hspace{-2pt}\multimap \mid \multimap\hspace{-2pt}\subset \mid \multimap\hspace{-2pt}\circ \mid \succ\hspace{-4pt}\multimap \mid \circ\hspace{-2pt}\multimap \mid \vdash \mid \multimap\hspace{-4pt}\boxed{\geq}\hspace{-4pt}\multimap \mid \multimap\hspace{-2pt}\boxed{x}\hspace{-2pt}\multimap \quad (x \in \mathbb{K}) \quad (1)$$

We will explain how these basic components can be wired together and give them a formal interpretation.

2.1   Props and Symmetric Monoidal Theories

The mathematical backbone of our approach is the notion of product and permutations category (prop), a structure which generalises standard algebraic theories [7]. Formally, a *prop* is a strict symmetric monoidal category (SMC) whose objects are the natural numbers and where the monoidal product $\oplus$ on objects is given by addition. Equivalently, it is a strict SMC whose objects are all monoidal products of a single generating object. *Prop morphisms* are strict symmetric monoidal functors that act as the identity on objects.

Following an established methodology, we will define two props: **Syn** and **Sem**, for the syntax and semantics respectively. To guarantee a compositional interpretation, we require $[\![\cdot]\!] : \mathbf{Syn} \to \mathbf{Sem}$, the mapping of terms to their intended semantics, to be a prop morphism.

Typically, the syntactic prop **Syn** is freely generated from a *monoidal signature* Σ, i.e. a set of arrows $g : m \to n$. In this case, we use the notation $\mathsf{P}\Sigma$ and **Syn** interchangeably. Morphisms of $\mathsf{P}\Sigma$ are terms of an $(\mathbb{N}, \mathbb{N})$-sorted syntax, whose constants are elements of Σ and whose operations are the usual composition $(-);(-) : \mathbf{Syn}(n, m) \times \mathbf{Syn}(m, l) \to \mathbf{Syn}(n, l)$ and the monoidal product $(-) \oplus (-) : \mathbf{Syn}(n_1, m_1) \times \mathbf{Syn}(n_2, m_2) \to \mathbf{Syn}(n_1 + n_2, m_1 + m_2)$, quotiented by the laws of SMCs. But this quotient is cumbersome and unintuitive to work with.

This is why we will prefer a different representation. With their two forms of composition, monoidal categories admit a natural two-dimensional graphical notation of *string diagrams*. The idea is that an arrow $c : n \to m$ of $\mathsf{P}\Sigma$ is better represented as a box with n ordered wires on the left and m on the left. We can compose these diagrams in two different ways: horizontally, by connecting the right wires of one diagram to the left wires of another, and vertically by juxtaposing two diagrams:

$$c \,;\, d = \quad \overset{n}{-}\boxed{c}\overset{m}{-}\boxed{d}\overset{l}{-} \qquad\qquad d_1 \oplus d_2 = \quad \begin{array}{c} \overset{n_1}{-}\boxed{d_1}\overset{m_1}{-} \\ \overset{n_2}{-}\boxed{d_2}\overset{m_2}{-} \end{array}$$

where the labelled wire $\overset{n}{-\!\!-}$ is syntactic sugar for a stack of n wires. The identity $id_1 : 1 \to 1$ is denoted as a plain wire —, the unit for $\oplus$, $id_0 : 0 \to 0$, as the empty diagram $\vdots\,\vdots$, and when the category is symmetric, the symmetry $\sigma_{1,1} : 2 \to 2$ is

denoted as a wire crossing $\times$. With this representation the laws of SMCs become diagrammatic tautologies.

Once we have defined $[\![\cdot]\!] :$ **Syn** $\to$ **Sem**, it is natural to look for equations to reason about semantic equality directly on the diagrams themselves. Given a set of equations E, i.e., a set containing pairs of arrows of the same type, we write $\overset{\text{E}}{=}$ for the smallest congruence wrt the two composition operations ; and $\oplus$. We say that $\overset{\text{E}}{=}$ is *sound* if $c \overset{\text{E}}{=} d$ implies $[\![c]\!] = [\![d]\!]$. It is moreover *complete* when the converse implication holds. We call a pair (Σ, E) a *symmetric monoidal theory* (SMT) and we can form the prop $\mathsf{P}\Sigma_{/E}$ obtained by quotienting $\mathsf{P}\Sigma$ by $\overset{\text{E}}{=}$. There is then a prop morphism $q : \mathsf{P}\Sigma \to \mathsf{P}\Sigma_{/E}$ witnessing this quotient.

We may also wonder what the expressive power of our diagrammatic language is. In terms of props we look to characterise precisely the image $\mathsf{Im}([\![\cdot]\!])$ of the syntax via $[\![\cdot]\!]$.

The situation for a sound and complete SMT is summarised in the commutative diagram below right.

Soundness simply means that $[\![\cdot]\!]$ factors as $s \circ q$ through $\mathsf{P}\Sigma_{/E}$ and completeness means that s is a faithful prop morphism.

$$
\begin{array}{ccc}
\mathsf{P}\Sigma_{/E} & \xrightarrow{\ \cong\ } & \mathsf{Im}([\![\cdot]\!]) \\
{\scriptstyle q}\big\uparrow & {\scriptstyle s}\searrow & \big\downarrow{\scriptstyle i} \\
\mathbf{Syn} = \mathsf{P}\Sigma & \xrightarrow[\ [\![\cdot]\!]\ ]{} & \mathbf{Sem}
\end{array}
$$

Typically, our semantic prop **Sem** will be (a subcategory of) the category of sets and relations.

Definition 1. *Let $\mathbb{K}$ be a field. $\mathsf{Rel}_{\mathbb{K}}$ is the prop*

- *whose arrows $n \to m$ are relations $R \subseteq \mathbb{K}^n \times \mathbb{K}^m$,*
- *with composition given by $R; S = \{(x, z) \mid \exists y.\ (x, y) \in R \wedge (y, z) \in S\}$, for $R : n \to m$, $S : m \to l$, and identity $n \to n$ the diagonal $\{(x, x) \mid x \in \mathbb{K}^n\}$,*
- *monoidal product given by*

$$
R_1 \oplus R_2 = \left\{ \left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \right) \mid (x_1, y_1) \in R_1 \wedge (x_2, y_2) \in R_2 \right\}
$$

for $R_1 : n_1 \to m_1$ and $R_2 : n_2 \to m_2$,
- *symmetry $n{+}m \to m{+}n$, the relation $\left\{ \left(\begin{pmatrix} x \\ y \end{pmatrix}, \begin{pmatrix} y \\ x \end{pmatrix} \right) \mid (x, y) \in \mathbb{K}^n \times \mathbb{K}^m \right\}.$*

2.2   Ordered Props and Symmetric Monoidal Inequality Theories

Our semantic prop—$\mathsf{Rel}_{\mathbb{K}}$—carries additional structure that we wish to lift to the syntax: as subsets of $\mathbb{K}^n \times \mathbb{K}^m$, relations $n \to m$ can be ordered by inclusion. The corresponding structure is that of an *ordered prop*, a prop enriched over the category of posets, whose composition and monoidal product are monotone maps.

If props can be presented by SMTs, ordered props can be presented by *symmetric monoidal inequality theories* (SMIT). Formally, the data of a SMIT is

the same as that of a SMT: a signature Σ and a set I of pairs $c, d : n \to m$ of PΣ-arrows of the same type, that we now read as *inequalities* $c \leq d$.

As for plain props, we can construct an ordered prop from a SMIT by building the free prop PΣ and passing to a quotient P$\Sigma_{/I}$. First, we build a preorder on each homset by closing I under $\oplus$ and taking the reflexive and transitive closure of the resulting relation. Then, we obtain the free ordered prop P$\Sigma_{/I}$ by quotienting the resulting preorder by imposing anti-symmetry.

SMITs subsume SMTs, since every SMT can be presented as a SMIT, by splitting each equation into two inequalities. We will refer to both simply as *theories* and their defining inequalities as *axioms*. When referring to a sound and complete theory, we will also use the term *axiomatisation*, as is standard in the literature.

2.3   Graphical Polyhedral Algebra

We now assume that $\mathbb{K}$ is an ordered field, that is, a field equipped with a total order $\geq$ compatible with the field operations in the following sense: for all $x, y, z \in \mathbb{K}$, *i)* if $x \geq y$ then $x + z \geq y + z$, and *ii)* if $x \geq 0$ and $y \geq 0$ then $xy \geq 0$.

Following [4], from the generators in (1), we define a prop, give it a semantics in $\mathsf{Rel}_{\mathbb{K}}$, characterise the image of the semantic functor, and describe an axiomatisation for the specified semantics.

– For $\Sigma_{\geq}^{+} = \{ -\!\!\!\blacktriangleleft\, , -\!\bullet\, , \blacktriangleright\!\!-\, , \bullet\!-\, , -\!\!\!\triangleleft\, , -\!\circ\, , \supset\!\!-\, , \circ\!-\, , \vdash\, , -\!\boxed{\geq}\!-\, , -\!\boxed{r}\!-\, (r \in \mathbb{K}) \}$
  define $[\![\, \cdot \,]\!] : \mathsf{P}\Sigma_{\geq}^{+} \to \mathsf{Rel}_{\mathbb{K}}$ to be the prop morphism given by

$$\left[\!\!\left[-\!\!\!\blacktriangleleft \right]\!\!\right] := \left\{ \left(x, \begin{pmatrix} x \\ x \end{pmatrix} \right) \mid x \in \mathbb{K} \right\} \qquad \left[\!\!\left[-\!\bullet \right]\!\!\right] := \{ (x, \bullet) \mid x \in \mathbb{K} \}$$

$$\left[\!\!\left[\blacktriangleright\!\!- \right]\!\!\right] := \left\{ \left(\begin{pmatrix} x \\ x \end{pmatrix}, x \right) \mid x \in \mathbb{K} \right\} \qquad \left[\!\!\left[\bullet\!- \right]\!\!\right] := \{ (\bullet, x) \mid x \in \mathbb{K} \}$$

$$\left[\!\!\left[-\!\!\!\triangleleft \right]\!\!\right] := \left\{ \left(x + y, \begin{pmatrix} x \\ y \end{pmatrix} \right) \mid x, y \in \mathbb{K} \right\} \qquad \left[\!\!\left[-\!\circ \right]\!\!\right] := \{ (0, \bullet) \}$$

$$\left[\!\!\left[\supset\!\!- \right]\!\!\right] := \left\{ \left(\begin{pmatrix} x \\ y \end{pmatrix}, x + y \right) \mid x, y \in \mathbb{K} \right\} \qquad \left[\!\!\left[\circ\!- \right]\!\!\right] := \{ (\bullet, 0) \}$$

$$\left[\!\!\left[-\!\boxed{k}\!- \right]\!\!\right] := \{ (x, k \cdot x) \mid x \in \mathbb{K} \} \ \text{ for } k \in \mathbb{K}$$

$$\left[\!\!\left[-\!\boxed{\geq}\!- \right]\!\!\right] := \{ (x, y) \in \mathbb{K} \times \mathbb{K} \mid x \geq y \} \qquad \left[\!\!\left[\vdash \right]\!\!\right] := \{ (\bullet, 1) \}$$

– The image of P$\Sigma_{\geq}^{+}$ by $[\![\, \cdot \,]\!]$ is the prop whose arrows $n \to m$ are finitely generated *polyhedra* of $\mathbb{K}^n \times \mathbb{K}^m$, i.e., subsets of the form

$$\left\{ (x, y) \in \mathbb{K}^n \times \mathbb{K}^m \mid A \begin{pmatrix} x \\ y \end{pmatrix} + b \geq 0 \right\}$$

for some matrix A and some vector b (see [4] for more details, in particular the appendix for the proof that these form a prop).

- $\mathsf{IH}_{\geq}^{+}$ provides an axiomatisation of polyhedral relations [4, Corollary 25]; it can be found in the first four blocks of Fig. 1.

Example 1 (Duality). Two diagrams play a special role in this paper: the half turns ●●⌣ and ⌣●●, called cup and cap, respectively. Using these and $\times$, we can build cups and caps for any number n of wires: ●●⌣^{n} and ⌣^{n}●●.

They allow us to associate a dual $d^{op} : n \to m$ to any diagram $d : m \to n$ by turning its left ports into right ports and vice-versa:

$$n - \boxed{d^{op}} - m \quad = \quad \overset{n}{\underset{m}{●●\!\!\boxed{d}●●}} \tag{2}$$

Correspondingly, $[\![d^{op}]\!]$ is the *opposite* relation, i.e. $[\![d^{op}]\!] = \{(y,x) \mid (x,y) \in [\![d]\!]\}$. We will use of a suggestive mirror notation to denote the dual of a given generator: $-\boxed{r}\!- := (-\boxed{r}\!-)^{op}$, $-\!\!\dashv := (\vdash\!\!-)^{op}$ and $-\!\boxed{\leq}\!- := (-\boxed{\geq}\!-)^{op}$.

3 Symmetric Monoidal Semi-Lattice Theories

There are several routes to describe piecewise-linear subsets of $\mathbb{K}^{n}$. In this paper we choose to equip our syntax with a primitive operation of join, in order to describe piecewise-linear sets as (finite) unions of polyhedra. In the same way that we moved from simple props to ordered props in Section 2.2, we now move to the setting of semi-lattice-enriched props.

A $\cup$-*prop* is a prop enriched over the monoidal category of semi-lattices – partially-ordered sets with least upper bounds for any finite subset – and join-preserving maps, with the Cartesian product as monoidal product. In other words a $\cup$-prop is a prop whose homsets are semi-lattices, with composition and monoidal product themselves join-preserving. The paradigmatic example is $\mathsf{Rel}_{\mathbb{K}}$ which is a $\cup$-prop with the union of relations as join.

As we would like to incorporate binary joins into our syntax, we need a new description of the free $\cup$-prop $\mathsf{P}_{\cup}\Sigma$ over a given signature Σ.

- The arrows $n \to m$ of $\mathsf{P}_{\cup}\Sigma$ are finite sets of arrows $n \to m$ of $\mathsf{P}\Sigma$. We use capital letters $C, D \ldots$ to range over them. We will also abuse notation slightly, using $c, d \ldots$ to refer to singletons $\{c\}, \{d\} \ldots$ and writing $d_1 \cup \cdots \cup d_k$ for the set $\{d_1, \ldots, d_k\}$. The set can be empty, yielding the bottom of the semi-lattice.
- The composition of $C : n \to m$ and $D : m \to l$ is given by $C \,;\, D = \{c \,;\, d \mid c \in C, d \in D\}$ where $c \,;\, d$ denotes composition in P_{Σ}. The identity over n is the singleton $\{id_n\}$.
- The monoidal product of $D_1 : n_1 \to m_1$ and $D_2 : n_2 \to m_2$ is given by $D_1 \oplus D_2 = \{d_1 \oplus d_2 \mid d_1 \in D_1, d_2 \in D_2\}$ where $d_1 \oplus d_2$ is the monoidal product in $\mathsf{P}_{\cup}\Sigma$.
- For the enrichment, each homset $\mathsf{P}_{\cup}\Sigma(n,m)$ is a semi-lattice with union as join. By definition, composition and monoidal product distribute over union and define join-preserving maps $(-) \,;\, (-) : \mathsf{P}_{\cup}\Sigma(n,m) \times \mathsf{P}_{\cup}\Sigma(m,l) \to \mathsf{P}_{\cup}\Sigma(n,l)$ and $(-) \oplus (-) : \mathsf{P}_{\cup}\Sigma(n_1,m_1) \times \mathsf{P}_{\cup}\Sigma(n_2,m_2) \to \mathsf{P}_{\cup}\Sigma(n_1+n_2, m_1+m_2)$

We now define a corresponding notion of theory for $\cup$-props. A *symmetric monoidal (semi-)lattice theory* (SMLT) is the data of a signature Σ and a set E of equations: formally the latter is a set of pairs (C, D) of arrows $C, D : n \to m$ from $\mathsf{P}_\cup \Sigma$. We will write the elements of E as equations of the form $\bigcup_{c \in C} c = \bigcup_{d \in D} d$. We now explain how to define a $\cup$-prop $\mathsf{P}_\cup \Sigma_{/E}$ from the data of an SMLT (Σ, E). As for SMTs, we can build the smallest congruence $\overset{\mathrm{E}}{=}$ wrt to $;$ and $\oplus$, which equates the pairs in E. Then define $\mathsf{P}_\cup \Sigma_{/E}$ to be the quotient of $\mathsf{P}_\cup \Sigma$ by $\overset{\mathrm{E}}{=}$. That this is a well-defined $\cup$-prop follows again from the distributivity of the composition and monoidal product over unions.

Note that the semi-lattice structure allows us to define an order over the homsets of any $\cup$-prop, making it into an ordered prop: we write $C \subseteq D$ as a shorthand for $C \cup D = D$. We will also use $C \overset{\mathrm{E}}{\subseteq} D$ for $C \cup D \overset{\mathrm{E}}{=} D$ in $\mathsf{P}_\cup \Sigma_{/E}$. (We prefer this notation to avoid the confusion with the order $\geq$ on $\mathbb{K}$ itself.)

Remark 1 (Reasoning in $\cup$-props). The reader familiar with string diagrams and equational reasoning might be surprised by certain features of derivations that combine diagrammatic and traditional syntax (joins, in this case). We would like to clarify one particular point. When we want to use an equality of the form $d = d_1 \cup d_2$ inside a term of the form $c_1 \cup c_2 \cup c$, we need to identify a linear context $C[-]$ (i.e. the hole occurs exactly once in C) common to c_1 and c_2 such that $c_1 = C[d_1]$ and $c_2 = C[d_2]$. Then we are allowed to use the fact that $C[d] = C[d_1] \cup C[d_2]$ to conclude that $c_1 \cup c_2 \cup c = C[d] \cup c$. An example of this form of reasoning can be found in the proof of Lemma 2, which we reproduce here: we apply the equality $\bullet\!\!-\overset{\text{(total)}}{=} \circ\!\!-\!\boxed{\leq}\!\!- \cup\ \circ\!\!-\!\boxed{\geq}\!\!-$ in

Note that, to clarify the common context to the reader, we will often use the intermediate notation $C[d_1 \cup d_2]$, as we did in the first step above.

4   The Theory of Piecewise-Linear Relations

4.1   Syntax and Semantics

For piecewise-linear relations we retain the same signature $\Sigma_\geq^+$ and consider $\mathsf{P}_\cup(\Sigma_\geq^+)$, the free $\cup$-prop over it. As we saw, its morphisms are nonempty finite sets of diagrams of $\mathsf{P}\Sigma_\geq^+$. This is our syntax.

On the semantic side, we now need to extend the functor $[\![\cdot]\!]$ to have $\mathsf{P}\Sigma_\geq^+$ as domain, retaining $\mathsf{Rel}_\mathbb{K}$ as codomain. Concretely, since we already know how to assign a relation to each diagram of $\mathsf{P}\Sigma_\geq^+$, we only need to specify how to interpret finite sets of such diagrams: unsurprisingly, we set

$$[\![\{d_1, \ldots, d_n\}]\!] := [\![d_1]\!] \cup \cdots \cup [\![d_n]\!]$$

This is join-preserving by construction, and remains monoidal and functorial.

By definition, we call piecewise-linear (pl) any relation in the image of this functor, i.e., any relation that is a finite union of polyhedral relations. As far as we know, this is the first time that this notion appears in print. However, it does capture our intuitive notion of piecewise-linearity as submanifolds of $\mathbb{K}^n$ that can be subdivided into linear subspaces.

4.2 Equational Theory

IH_{PL}, the SMLT of pl relations, is presented in Fig. 1. The first block is the theory of matrices/linear maps; the second block, IH, axiomatises all linear relations; the third block axiomatises the behaviour of the order $-\boxed{\geq}-$; the fourth, deals with the affine fragment of the theory, axiomatising the behaviour of the constant $\vdash$. Taken together, those four blocks constitute $\mathsf{IH}_{\geq}^{+}$, an axiomatisation of polyhedral relations—we refer the reader to [4] for more details on this fragment.

The key addition of IH_{PL} is the last block, containing the axiom of *totality*, which states that any real number belongs to the non-negative *or* to the non-positive fragment of $\mathbb{K}$. Remarkably, this simple axiom is the only one we need to add to $\mathsf{IH}_{\geq}^{+}$ to obtain a complete theory for pl relations. Its soundness is simply a consequence of the definition of an ordered field: the order is assumed to be total in the sense that, for any $x, y \in \mathbb{K}$ we have $x \leq y$ or $y \leq x$. Take $y = 0$ to recover the last axiom of IH_{PL}.

Remark 2. As a consequence of the Frobenius laws ($\bullet$-fr) and of (co)unitality ($\bullet$-un)-($\bullet$-coun), the diagrams $\bullet\!\!\overset{n}{\bullet\!\!\frown}$ and $\overset{n}{\frown\!\!\bullet}\bullet$ satisfy

$$\overset{n}{\underset{n}{\rightrightarrows}}\bullet\bullet \overset{\mathsf{IH}_{PL}}{=} \underline{} n \overset{\mathsf{IH}_{PL}}{=} \bullet\bullet\overset{n}{\underset{n}{\rightrightarrows}} \bullet\bullet \tag{3}$$

for any n, the defining equations of a compact closed category. Intuitively, these allow us to forget the direction of wires. In addition, compactness implies the following proposition.

Proposition 1. $C \overset{\mathsf{IH}_{PL}}{\subseteq} D$ iff $C^{op} \overset{\mathsf{IH}_{PL}}{\subseteq} D^{op}$.

Another important property of compact closed category which we will exploit to simplify the completeness proof is stated in the following proposition. It is an immediate consequence of (3).

Proposition 2. *Given* $C, D : m \to n$, $C \overset{\mathsf{IH}_{PL}}{\subseteq} D$ iff $\overset{m}{\underset{n}{\rightarrow}}\!\boxed{C}\!\rightarrow\!\bullet \overset{\mathsf{IH}_{PL}}{\subseteq} \overset{m}{\underset{n}{\rightarrow}}\!\boxed{D}\!\rightarrow\!\bullet$

4.3 Completeness Theorem

As we stated above, the axioms in Fig. 1 form a complete theory for pl relations. We will prove that claim in this section. Without loss of generality, using Proposition 2, we restrict to $n \to 0$ diagrams.

We start by defining appropriate normal forms for polyhedral and pl relations, and then show that every diagram can be reduced to normal form.
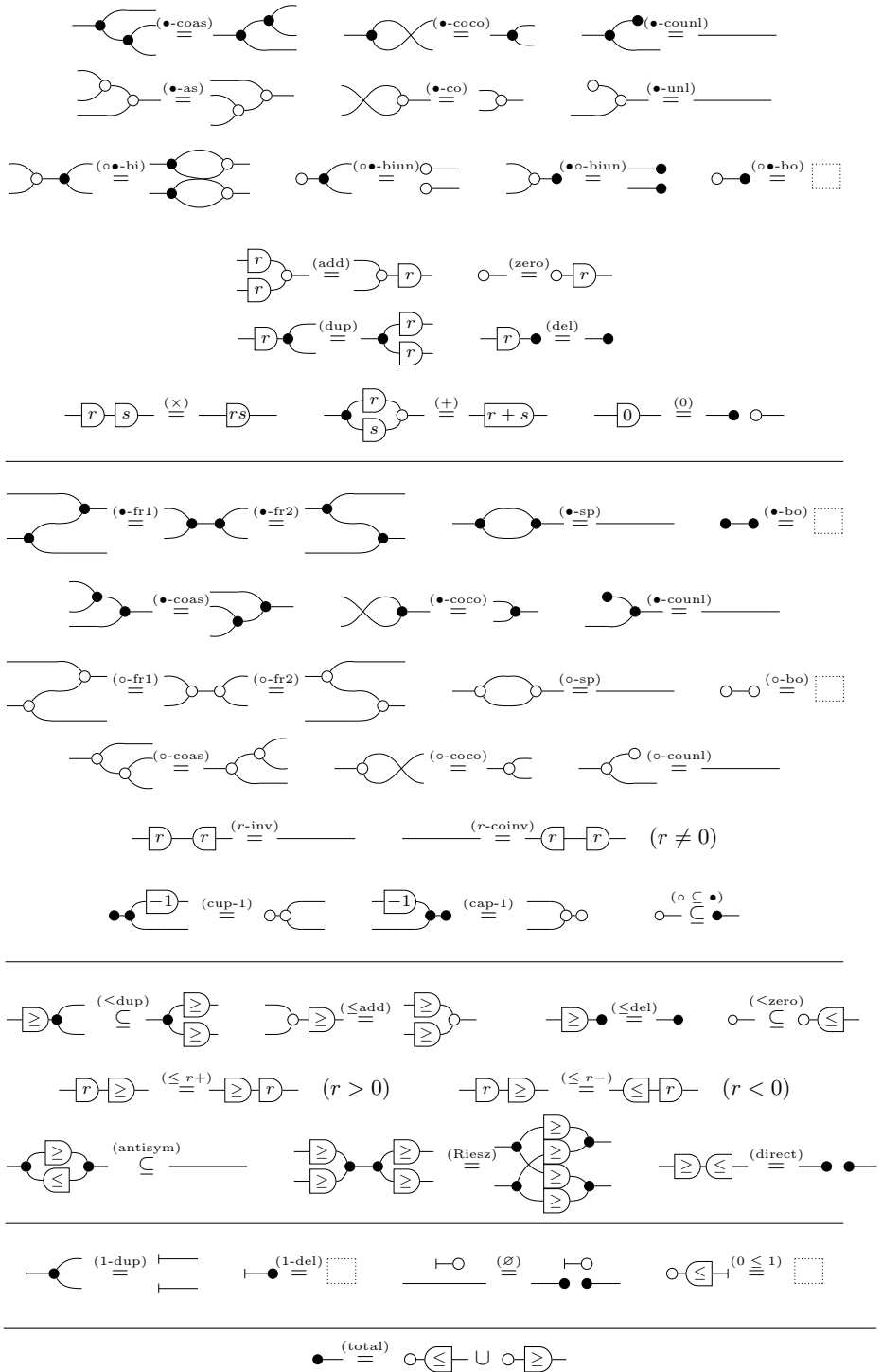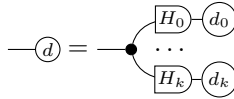
Fig. 1. Axioms of GPLA.

Definition 2. *We call hyperplane a nonzero affine map $H : n \to 1$ which we write $-\boxed{H}-$. A given hyperplane H defines two half-spaces $-\boxed{H}\boxed{\geq}\!\!\circ$ and $-\boxed{H}\boxed{\leq}\!\!\circ$, as well as an affine subspace $-\boxed{H}\!\!\circ$. Since inequality is not strict, the half-spaces include the affine subspace.*

In [4, Theorem 14], polyhedral relations have a normal form given by a set of inequations of the form $A_i x + b_i \geq 0$. In other words, the normal form is given by an intersection of half-spaces. For our purposes we define a related but slightly different normal form.

Definition 3. *A $\mathsf{P}\Sigma_{\geq}^{+}$-diagram $d : n \to 0$ is in polyhedral normal form if there are hyperplanes H_i and diagrams $-\!\boxed{d_i}\! \in \{-\!\circ, -\boxed{\geq}\!\!\circ, -\boxed{\leq}\!\!\circ\}$ such that:*

$$-\boxed{d} = -\!\!\!\left\langle\!\!\!\begin{array}{c} \boxed{H_0}\!-\!\boxed{d_0} \\ \cdots \\ \boxed{H_k}\!-\!\boxed{d_k} \end{array}\right.$$

Where the d_i are minimal in the following sense: fixing the set of hyperplanes H_i, we consider all choices of d_i that give d when composed as above. We then require the d_i in the normal form to be minimal (wrt the order of $\mathsf{IH}_{\geq}^{+}$) among those. We call the set of the d_i a valuation for d relative to the hyperplanes H_i.

Definition 4. *We say that a morphism D of $\mathsf{P}_{\cup}\Sigma_{\geq}^{+}$ is in pl normal form if it is written as a non-empty union of diagrams d_i each in the language of $\mathsf{P}\Sigma_{\geq}^{+}$ (i.e. without unions), the d_i are in the normal form defined in Definition 3, and all the normal forms use the same set of hyperplanes.*

Lemma 1. *Every $d : n \to 0$ in $\mathsf{P}\Sigma_{\geq}^{+}$ has a polyhedral normal form.*

Proof. The normal form from [4, Theorem 14] already has the right shape. We only need to find a minimal valuation. Observe that the intersection of two valuations for d is again a valuation for d: let v and v' be two valuations for d relative to the hyperplanes H_i. If we write $-\boxed{A}- := -\!\!\left\langle\!\!\begin{array}{c}\boxed{H_0}-\\\cdots\\\boxed{H_k}-\end{array}\right.$ then

$$-\boxed{A}\!\!-\!\!\left\langle\!\!\begin{array}{c}\textcircled{v}\\\textcircled{v'}\end{array}\right. = -\!\!\left\langle\!\!\begin{array}{c}\boxed{A}\!-\!\textcircled{v}\\\boxed{A}\!-\!\textcircled{v'}\end{array}\right. = -\!\!\left\langle\!\!\begin{array}{c}\textcircled{d}\\\textcircled{d}\end{array}\right. = -\!\textcircled{d}$$
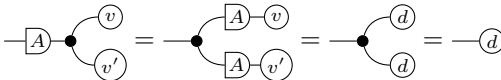
Therefore $v \cap v'$ is again a valuation for d. Since there are finitely many valuations, we construct the minimal one by intersecting them all. $\qquad\square$

Lemma 2. *If a morphism D of $\mathsf{P}_{\cup}\Sigma_{\geq}^{+}$ is in pl normal form and H is a hyperplane, there exists C in pl normal form such that $D \stackrel{\mathsf{IH}_{PL}}{=} C$ and $\mathsf{Hyperplanes}(C) = \mathsf{Hyperplanes}(D) \cup \{H\}$.*

Proof. We write the normal form of D as $D = \bigcup_i d_i$. Define C to be the following morphism:

$$-\textcircled{C} = \bigcup_i -\!\!\left\langle\!\!\begin{array}{c}\textcircled{d_i}\\\boxed{H}\!-\!\boxed{\geq}\!\!\circ\end{array}\right. \cup \bigcup_i -\!\!\left\langle\!\!\begin{array}{c}\textcircled{d_i}\\\boxed{H}\!-\!\boxed{\leq}\!\!\circ\end{array}\right.$$
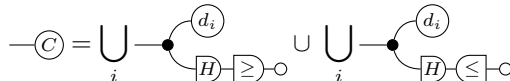
We transform C into C' by reducing all the terms in the union to polyhedral normal form. This makes C' be in pl normal form. Since we add the same hyperplane H to all d_i, $\mathsf{Hyperplanes}(C') = \mathsf{Hyperplanes}(D) \cup \{H\}$.

Moreover:

$$-\!\!\!\;\boxed{C'}\; = -\!\!\!\;\boxed{C}\; = \cdots \overset{(\text{total})}{=} \cdots = -\!\!\!\;\boxed{D}$$

$\square$

Theorem 1. *Every morphism of $\mathsf{P}_\cup \Sigma_\geq^+$ has a pl normal form.*

Proof. Let D be a $n \to 0$ morphism of $\mathsf{P}_\cup \Sigma_\geq^+$. First using distributivity of the union over sequential and parallel composition, we move all the uses of the union to the top-level.

Thus D is written $\bigcup_i d_i$ where each d_i doesn't use the union, i.e. is in the language of $\mathsf{P}\Sigma_\geq^+$. We then rewrite each d_i into polyhedral normal form using Lemma 1.

Each d_i is thus also individually in *pl normal form*, so we can use Lemma 2 to add to each d_i all the hyperplanes of the other d_j. For each i we get a new diagram $d_i' \overset{\mathsf{IH}_{PL}}{=} d_i$ in pl normal form, and all the d_i' use the same set of hyperplanes. So $\bigcup_i d_i'$ is a pl normal form for D. $\square$

Before we can prove completeness, we need a final notion: the interior of a polyhedral relation, which is the set of its points that don't touch any of its faces.

Definition 5. *Let d be morphism in polyhedral normal form. We define $\mathsf{Int}(d)$ to be the set of points $x \in [\![d]\!]$ for which $H_i(x) \neq 0$ when $-\!\boxed{d_i} \neq -\!\circ$. In other words, $H_i(x)$ is nonzero for all hyperplanes where it can be nonzero without x leaving $[\![d]\!]$.*

Note that we define Int only on polyhedral normal form diagrams. Int appears to be representation-independent at least when $\mathbb{K} = \mathbb{R}$, but we won't try to prove it in the general case as we don't need this here.

Remark 3. This is not the usual topological notion of interior. In particular, this notion is independent from the dimension of the surrounding space: a polyhedron of dimension $0 < k < n$ within $\mathbb{R}^n$ has an empty topological interior but a nonempty Int, as we'll see in the next theorem. $\mathsf{Int}(d)$ instead coincides with the interior of d with the topology of the smallest containing affine space.

Lemma 3. *Let d be a diagram in polyhedral normal form. If $[\![d]\!]$ is nonempty, then $\mathsf{Int}(d)$ is nonempty.*

Proof. First, write d in polyhedral normal form:

$$-\!\!\!\;\boxed{d}\; = -\!\!\!\;\bullet\!\!\begin{array}{c} \boxed{H_0}\!\!-\!\!\boxed{d_0} \\ \cdots \\ \boxed{H_k}\!\!-\!\!\boxed{d_k} \end{array}$$

Up to negating some of the H_i, we can assume that none of the $-\!\!\bigcirc\!\!d_i$ are $-\!\!\boxed{\leq}\!\!\circ$. If $\forall i.-\!\!\bigcirc\!\!d_i = -\!\!\circ$, then by definition $\mathsf{Int}(d) = [\![\,d\,]\!]$ which is nonempty so we're done. Assume then that $-\!\!\bigcirc\!\!d_i = -\!\!\boxed{\geq}\!\!\circ$ for at least some i. For each such i, by minimality of the d_i in the normal form there must be a $x_i \in [\![\,d\,]\!]$ such that $H_i(x_i) > 0$. We pick such an x_i for each i, and define $x := \frac{1}{p}\sum_i x_i$ to be their average. By convexity, $x \in [\![\,d\,]\!]$. H_i is an affine map, hence is concave, thus if we had picked an x_i then $H_i(x) \geq \frac{1}{p}\sum_j H_i(x_j) \geq \frac{1}{p}H_i(x_i) > 0$. Then for each i either $-\!\!\bigcirc\!\!d_i = -\!\!\circ$ or $H_i(x) > 0$, hence $x \in \mathsf{Int}(d)$. $\square$

Theorem 2 (Completeness). $[\![\,D\,]\!] \subseteq [\![\,C\,]\!] \implies D \overset{\mathsf{IH}_{PL}}{\subseteq} C$

Proof. Using Proposition 2 we can without loss of generality assume that D and C have n inputs and 0 outputs. Using Theorem 1, we reduce D and C into pl normal form. Using Lemma 2, we add each others' hyperplanes to D and C so that they both use the exact same set. So $D = \bigcup_i d_i$ and $C = \bigcup_i c_i$, where the d_i and c_i are in polyhedral normal form and use a same set of hyperplanes $\{H_i\}_i$. Pick one of the d_i in D.

If d_i is the empty polyhedron, we have $[\![\,d_i\,]\!] = \emptyset \subseteq [\![\,c_0\,]\!]$, so by completeness of $\mathsf{IH}^+_\geq$ we get $d_i \overset{\mathsf{IH}^+_\geq}{\subseteq} c_0$. Thus $d_i \overset{\mathsf{IH}_{PL}}{\subseteq} c_0 \overset{\mathsf{IH}_{PL}}{\subseteq} C$.

Otherwise d_i is nonempty, and using Lemma 3 we pick $x \in \mathsf{Int}(d_i)$. Then:

$$x \in \mathsf{Int}(d_i) \subseteq [\![\,d_i\,]\!] \subseteq [\![\,D\,]\!] \subseteq [\![\,C\,]\!] = \left[\!\!\left[\,\bigcup_j c_j\,\right]\!\!\right] = \bigcup_j [\![\,c_j\,]\!]$$

Thus there is a j such that $x \in [\![\,c_j\,]\!]$. Now pick a k. If $-\!\!\bigcirc\!\!d_{ik} = -\!\!\circ$, then $-\!\!\bigcirc\!\!d_{ik} \overset{\mathsf{IH}_{PL}}{\subseteq} -\!\!\bigcirc\!\!c_{jk}$ regardless of $-\!\!\bigcirc\!\!c_{jk}$. If $-\!\!\bigcirc\!\!d_{ik} = -\!\!\boxed{\geq}\!\!\circ$, then by definition of $\mathsf{Int}(d_i)$, we have $H_k(x) > 0$. Since moreover $x \in [\![\,c_j\,]\!]$, $-\!\!\bigcirc\!\!c_{jk}$ must be $-\!\!\boxed{\geq}\!\!\circ$. If $-\!\!\bigcirc\!\!d_{ik} = -\!\!\boxed{\leq}\!\!\circ$, similarly $-\!\!\bigcirc\!\!c_{jk}$ must be $-\!\!\boxed{\leq}\!\!\circ$. In all three cases, $-\!\!\bigcirc\!\!d_{ik} \overset{\mathsf{IH}_{PL}}{\subseteq} -\!\!\bigcirc\!\!c_{jk}$. This is the case for every k, so:

$$-\!\!\bigcirc\!\!d_i = -\!\!\bullet\!\!\langle\overset{\boxed{H_0}-\bigcirc d_{i0}}{\underset{\boxed{H_m}-\bigcirc d_{im}}{\cdots}} \subseteq -\!\!\bullet\!\!\langle\overset{\boxed{H_0}-\bigcirc c_{j0}}{\underset{\boxed{H_m}-\bigcirc c_{jm}}{\cdots}} = -\!\!\bigcirc\!\!c_j \subseteq -\!\!\bigcirc\!\!C$$

Finally, since we have $d_i \overset{\mathsf{IH}_{PL}}{\subseteq} C$ for all i, we derive $D = \bigcup_i d_i \overset{\mathsf{IH}_{PL}}{\subseteq} C$. $\square$

5   Generating Piecewise-Linear Relations

Piecewise-linear subsets of vector spaces give us a rather wide semantic space to explore. One might suspect that there exist useful structured relations that live strictly between the linear and piecewise-linear worlds.

Formally, we're interested in finding sub-props of $\mathsf{Rel}_\mathbb{K}$ that contain not only linear or polyhedral relations, but some selected non-convex relations that would be useful for particular applications. It turns out that for many sensible choices, the resulting image will coincide with pl relations—a somewhat surprising fact. Note that we are interested in generating sub-props of $\mathsf{Rel}_\mathbb{K}$ here, not $\cup$-props, since the $\cup$-prop generated by the image of $\mathsf{P}_\cup\Sigma_\geq^+$ under $[\![\,\cdot\,]\!]$ already contains all pl relations.

We will go through a few natural choices, each time defining them as a term of $\mathsf{P}_\cup\Sigma_\geq^+$, a shortcut which makes reasoning about them much easier than with their set-theoretic semantics. Of course, their semantics in $\mathsf{Rel}_\mathbb{K}$ can be recovered via $[\![\,\cdot\,]\!]$.

5.1   The n-Fold Union Generators

We first show that the main difference between polyhedral and pl relations —the unions—can be bridged. Indeed, it is not obvious that we can build arbitrary unions of diagrams without having access to the syntax of a SMLT. For this we introduce a family of diagrams we call the n-fold union generators, defined for a given n as:

$$\overset{n}{-}\!\!\!\cup\!\!\overset{n}{\underset{n}{\big\langle}} \quad := \quad \overset{n}{-}\!\!\nearrow\!\!\overset{n}{\underset{\bullet-\,n}{}} \quad \cup \quad \overset{\bullet-\overset{n}{}}{\underset{n}{-}\!\!\searrow\!\!\underset{n}{}}$$

These generators suffice to reproduce the behaviour of the syntactic union:

Theorem 3. *The image of the free prop generated by $\Sigma_\geq^+$ and the n-fold union generators for all n is the prop of pl relations.*

Proof. If $-\!\!\boxed{C}$ and $-\!\!\boxed{D}$ are non-empty $n \to 0$ diagrams,

$$\overset{n}{-}\!\!\cup\!\!\underset{\boxed{D}}{\overset{\boxed{C}}{\big\langle}} \quad = \quad \overset{\bullet-\boxed{C}}{\underset{n}{-}\!\!\nearrow\!\!\underset{\boxed{D}}{}} \quad \cup \quad \overset{n}{-}\!\!\searrow\!\!\underset{\bullet-\boxed{D}}{\overset{\boxed{C}}{}} \quad = \quad -\!\!\boxed{C} \;\cup\; -\!\!\boxed{D}$$

Since every pl relation can be written as a finite union of diagrams in $\mathsf{P}\Sigma_\geq^+$, and we can easily avoid diagrams denoting the empty relation, this generates all of pl relations. □

This means that we didn't formally need to introduce the notion of a SMLT after all: we could have defined an equivalent SMIT by adding these generators. However, this is for most purposes a much less convenient syntax, and the corresponding equational theory would be more difficult to calculate with. This is also the case for the examples that follow.

5.2   The Simplest Non-Convex Diagram

The following is one of the simplest diagrams that captures a non-convex relation:

$$-\!\!\boxed{+}\!\!- \quad := \quad -\!\!\circ\;\bullet\!\!-\; \cup \;-\!\!\bullet\;\circ\!\!-$$

It is named after its semantics: the union of the x and y axes in the plane, corresponding to the simple equation $x = 0 \vee y = 0$. Despite its simplicity, it suffices to generate all of pl relations.

Theorem 4. *The image of the free prop generated by $\Sigma_{\geq}^+$ and $-\boxed{+}-$ is the prop of pl relations.*

Proof. Define $dup : 1 \to 2$: $-\boxed{dup}\!\!\prec\ :=$
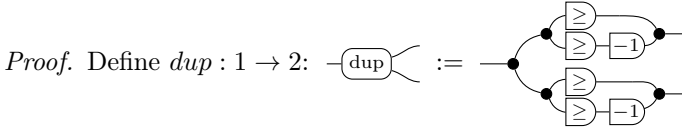
This diagram has the interesting property of duplicating black and white units:

$$\circ\!-\!\boxed{dup}\!\!\prec\ =\ \circ\!=\qquad\qquad \bullet\!-\!\boxed{dup}\!\!\prec\ =\ \bullet\!=$$

We can chain it to build $-\boxed{dup}^{\,n+1}\!- := -\boxed{dup}\,\boxed{dup}^{\,n}\!-$ for any n.

Then, let $^n\!\boxed{+}^n := {}^n\!\boxed{dup^{op}}^1\!\boxed{+}^1\!\boxed{dup}^n = {}^n\!\!-\!\circ\ \bullet\!-^n\ \cup\ {}^n\!\!-\!\bullet\ \circ\!-^n$

This allows us to build:

$$= {}^n\!\!-\!\!\nearrow\!-^n\ \cup\ {}^n\!\!-\!\!\searrow\!-^n = {}^n\!-\!\boxed{\cup}\!\prec\!-^n$$

□
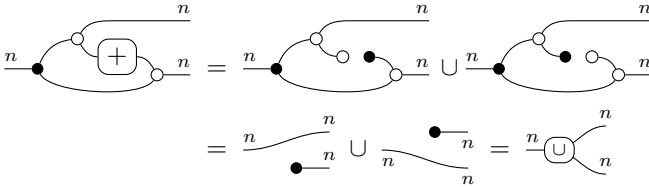
5.3   The Semantics of a Diode

Most basic electrical circuit components can be modelled with an affine semantics. The first exception is the (ideal) diode: the idealised current-voltage semantics across a diode is that the current can be negative and the voltage difference positive but not both at the same time.
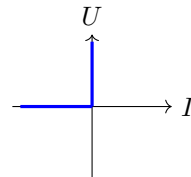
On a graph, the allowed (current, voltage difference) pairs are depicted above. Not only is this not affine, it is not even convex. The corresponding diagram, $-\boxed{\leq}\!-\!\circ\ \circ\!-\ \cup\ -\!\circ\ \circ\!-\!\boxed{\leq}\!-$, is outside of both affine and polyhedral algebra.

We will see how to model electrical circuits with diodes in more detail in the next section. We will focus here on the following fact: adding a generator with this semantics is once again enough to recover all pl relations. In fact we can even build the $\geq$ relation from the diode, so we can start from affine algebra (without requiring the generality of polyhedral algebra).

For convenience, we define a new generator whose semantics is the mirror image of the diode's graph:

$$-\boxed{\mathsf{L}}\!- := -\boxed{\geq}\!-\!\circ\ \circ\!-\ \cup\ -\!\circ\ \circ\!-\!\boxed{\leq}\!-$$
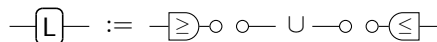
Theorem 5. *Recall that Σ^+ is $\Sigma_{\geq}^+$ without —$\geq$—. The image of the free prop generated by Σ^+ and L is the prop of pl relations.*

Proof. First, we can construct the $\geq$ generator from L:

So we generate all polyhedral relations. Then we can also recover the $+$ generator from the previous section, which is enough to generate all of pl relations:

$$\square$$

5.4 Alternative generators: *max*, *ReLu* and *abs*

Three of the most basic piecewise-linear functions one might come across are *abs*, *max* and *ReLu*. We define them diagrammatically as follows:

While the reader will certainly be familiar with the first two, *ReLu* has acquired significant fame as one of the basic building blocks of neural networks. In fact, all neural networks whose activation function is *ReLu* can be represented in GPLA. This opens up the exciting possibility of applying equational reasoning to neural networks, a possibility that we leave for future work.

Once again, adding either of them to the syntax for affine algebra suffices to construct any pl relation.

Theorem 6. *The image of the free prop generated by Σ^+ and any of max, abs or ReLu is the prop of pl relations.*

Proof. First, we notice that the three functions are inter-definable. *abs* and *ReLu* were already defined in terms of *max*, and we can complete the cycle:

$$max(x, y) = x + max(0, y - x) = x + ReLu(y - x)$$

$$ReLu(x) = max(0, x) = (x + abs(x))/2$$

So we only need to show the result for one of them. Let's pick *max*. We recover L, which we know suffices by Theorem 5. First $\;\begin{array}{c}\text{max}\end{array}\;\; = \;\; \begin{array}{c}\leq\end{array} \; \cup \; \begin{array}{c}\leq\end{array}.$

Thus $\quad \begin{array}{c} -1 \\ \text{max} \\ -1 \end{array} \;\; = \;\; -\!\geq\!\!- \;\; \cup \;\; -\!\!- \; -\!\leq\!- \;\; = \;\; -\boxed{\text{L}}- \qquad\qquad\qquad \square$
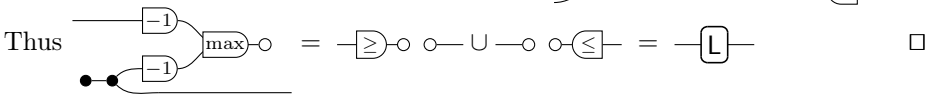
Remark 4. It is standard that *max* together with linear maps generates all continuous pl *functions*. Our result can be seen as a generalization of this fact to the relational setting.

5.5 Conclusion

These examples justify the generality of pl relations: they constitute the minimal extension of polyhedral algebra (and in some cases affine algebra) that can express any of the very useful relations above. This is interesting because pl relations form a nearly universal domain: they can approximate any smooth manifold over a bounded domain.

Despite our compelling examples, there could still be interesting props between polyhedral and pl relations. In particular, determining the prop generated by $\Sigma_{\geq}^+$ together with $\circ\!\!-\cup\vdash$ is currently an open problem.

6 Case Study: Electronic Circuits

To illustrate how one would use this theory in a concrete case, we turn to the study of electronic circuits. We build on the work done in [3]. The syntax mimics the usual circuits drawn by electrical engineers, by generating a free two-colored prop from basic elements and wires. The blue wires are electrical wires, and the black wires carry information; for details see [3].
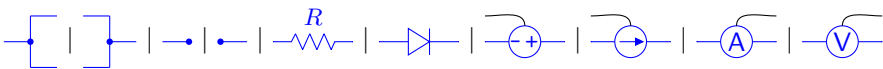
The corresponding physical model imposes constraints between two quantities: current and voltage. To express this, we map an electrical wire into two GPLA wires, the top one for voltage and the bottom one for current. We then give to each generator a semantics in GPLA that expresses the relevant physical equations. For example:
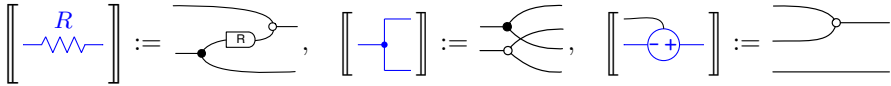
The core of this approach is the fact that composition of constraints in GPLA gives the behaviour of the corresponding composite electrical circuit. We can thus define the semantics of a whole circuit compositionally, and get the physically expected result.

So far this follows exactly [3]. Our contribution is the ability to express the behaviour of diodes:
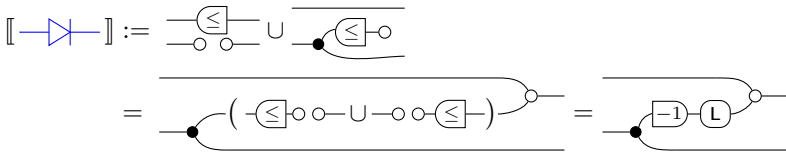
Remark 5. We cannot include capacitors and inductors, because they require semantics in $\mathsf{IH}^{+}_{\mathbb{R}(x)}$, and $\mathbb{R}(x)$ cannot be ordered in a way that would be consistent with the physics. Finding diagrammatic semantics that can accommodate both capacitors and diodes is an important open problem.

This extension allows us to model electronic circuits! As hinted in the previous section, diodes by themselves can be used to build many things. For example, we can model a simple idealized transistor as follows: [10, Fig. 59.1]
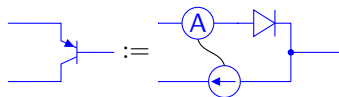
That said, it is impractical to prove the equality of two non-trivial electronic circuits explicitly as the number of alternatives grows exponentially in the number of diodes. Like in standard mathematical practice, making this practical will require finding appropriate techniques and approximations, which we leave for future work.

References

1. Baez, J.C., Coya, B., Rebro, F.: Props in network theory. Theory and Applications of Categories **33**(25), 727–783 (2018)

2. Baez, J.C., Fong, B.: A compositional framework for passive linear networks. Theory and Applications of Categories **33**(38), 1158–1222 (2018)
3. Boisseau, G., Sobociński, P.: String Diagrammatic Electrical Circuit Theory. arXiv:2106.07763 [cs] (2021), http://arxiv.org/abs/2106.07763
4. Bonchi, F., Di Giorgio, A., Sobocinski, P.: Diagrammatic Polyhedral Algebra. arXiv:2105.10946 [cs, math] (2021), http://arxiv.org/abs/2105.10946
5. Bonchi, F., Holland, J., Piedeleu, R., Sobociński, P., Zanasi, F.: Diagrammatic algebra: from linear to concurrent systems. In: Proceedings of the 46th Annual ACM SIGPLAN Symposium on Principles of Programming Languages (POPL) (2019)
6. Bonchi, F., Piedeleu, R., Sobociński, P., Zanasi, F.: Graphical Affine Algebra. In: 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS). pp. 1–12. IEEE, Vancouver, BC, Canada (2019). https://doi.org/10.1109/LICS.2019.8785877
7. Bonchi, F., Sobociński, P., Zanasi, F.: Deconstructing lawvere with distributive laws. Journal of logical and algebraic methods in programming **95**, 128–146 (2018)
8. Cvitanovic, P., Cvitanović, P.: Group theory. Princeton University Press (2008)
9. Stephan, P.R., Brayton, R.K.: Physically realizable gate models. In: Proceedings of 1993 IEEE International Conference on Computer Design ICCD'93. pp. 442–445. IEEE (1993)
10. Theraja, B., Theraja, A.: A textbook of electrical technology : in S.I. system of units. Publication division of Nirja Construction and Development Co., New Delhi (1994)
11. Willems, J.C.: The behavioral approach to open and interconnected systems. IEEE Control Systems Magazine **27**(6), 46–99 (2007)

Token Games and History-Deterministic Quantitative Automata

Udi Boker[1] and Karoliina Lehtinen[✉2]

[1] Reichman University, Herzliya, Israel `udiboker@idc.ac.il`
[2] CNRS, Marseille-Aix Université, Université de Toulon, LIS, Marseille, France `lehtinen@lis-lab.fr`

Abstract. A nondeterministic automaton is history-deterministic if its nondeterminism can be resolved by only considering the prefix of the word read so far. Due to their good compositional properties, history-deterministic automata are useful in solving games and synthesis problems. Deciding whether a given nondeterministic automaton is history-deterministic (the HDness problem) is generally a difficult task, which might involve an exponential procedure, or even be undecidable, for example for pushdown automata. *Token games* provide a PTime solution to the HDness problem of Büchi and coBüchi automata, and it is conjectured that 2-token games characterise HDness for all ω-regular automata. We extend token games to the quantitative setting and analyze their potential to help deciding HDness for quantitative automata. In particular, we show that 1-token games characterise HDness for all quantitative (and Boolean) automata on finite words, as well as discounted-sum (DSum) automata on infinite words, and that 2-token games characterise HDness of LimInf and LimSup automata. Using these characterisations, we provide solutions to the HDness problem of Inf and Sup automata on finite words in PTime, for DSum automata on finite and infinite words in NP∩co-NP, for LimSup automata in quasipolynomial time, and for LimInf automata in exponential time, where the latter two are only polynomial for automata with a logarithmic number of weights.

Keywords: Automata, History-determinism, Token games, Synthesis

1 Introduction

History-determinism. A nondeterministic [quantitative] automaton is history-deterministic (HD) [11,8] if its nondeterministic choices can be resolved by only considering the word read so far, uniformly across possible suffixes (see Fig. 2 for examples of HD and non-HD automata). More precisely, there should be a function (strategy), sometimes called a resolver, that maps the finite prefixes of a word to the transition to be taken at the last letter. The run built in this way must, in the Boolean setting, be accepting whenever the word is in the language of the automaton, and in the more general, quantitative, setting, attain the value of the automaton on the word (*i.e.*, the supremum of all its runs' values).

History-determinism lies in between determinism and nondeterminism, enjoying in some aspects the best of both worlds: HD automata are, like deterministic ones, useful for solving games and reactive synthesis [16,11,17,18,12,15,8], yet can sometimes be more expressive and/or succinct. For example, HD coBüchi and LimInf automata can be exponentially more succinct than deterministic ones [19], and HD pushdown automata are both more expressive and at least exponentially more succinct than deterministic ones [20,15]. In the (ω-)regular setting, history-determinism coincides with good-for-gameness [7], while in the quantitative setting it is stronger [8]. The problem of deciding whether a nondeterministic automaton is HD is interreducible with deciding the *best-value synthesis* problem of a deterministic automaton [14,8]. In this quantitative version of the reactive synthesis problem, the system must guarantee a behaviour that matches the value of any global behaviour compatible with the environment's actions. The witness of HDness corresponds exactly to the solution system of this synthesis problem, providing another motivation for this line of research.

Deciding history-determinism – a difficult task. History-determinism is formally defined by a *letter game* played on the automaton $\mathcal{A}$ between Adam and Eve, where Adam produces an input word w, letter by letter, and Eve tries to resolve the nondeterminism in $\mathcal{A}$ so that the resulting run attains $\mathcal{A}$'s value on w. Then $\mathcal{A}$ is HD if Eve has a winning strategy in the letter game on it. The difficulty of deciding who wins the letter game stems from its complicated winning condition – Eve wins if her run has the value of the supremum over all runs of $\mathcal{A}$ on w.

The naive solution is to determinise $\mathcal{A}$ into an automaton $\mathcal{D}$, and consider a game equivalent to the letter game that has a simple winning condition and whose arena is the product of $\mathcal{A}$ and $\mathcal{D}$ [16]. The downside with this approach, however, is that it requires the determinisation of $\mathcal{A}$, which often involves a procedure exponential in the size of $\mathcal{A}$ and sometimes is even impossible due to an expressiveness gap. Note that deciding whether an automaton is good-for-games, which is closely related to whether it is HD [7,8], is also difficult, as it requires reasoning about composition with all possible games.

Token games – a possible aid. In [3], Bagnol and Kuperberg introduced *token games* on ω-regular automata, which are closely related to the letter game, but easier to decide. In a k-token game on an automaton $\mathcal{A}$, denoted by $G_k(\mathcal{A})$, like in the letter game, Adam generates a word w letter by letter, and Eve builds a run on w by resolving the nondeterminism. In addition, Adam also has to resolve the nondeterminism of $\mathcal{A}$ to build k runs letter-by-letter over w. The winning condition for Eve in these games is that either all runs built by Adam are rejecting, or Eve's run is accepting. Such games, as they compare concrete runs, are easier to solve than the letter game.

Then, to decide HDness for a class of automata, one can attempt to show that the letter game always has the same winner as a k-token game, for some k, and solve the k-token game. (If Eve wins the letter game then she wins the k-token game, for every k, by using the same strategy, ignoring Adam's runs. However,

it might be that she wins a k-token game, taking advantage of her knowledge of how Adam resolves the nondeterminism, but loses the letter game.)

Bagnol and Kuperberg showed in [3] that on Büchi automata, the letter game and the 2-token game always have the same winner, and in [6], Boker, Kuperberg, Lehtinen and Skrzypczak extended this result to coBüchi automata. In both cases, this allows for a PTime procedure for deciding HDness. Furthermore, Bagnol and Kuperberg suggested in [3, Conclusion] that 2-token games might characterise HDness also for parity automata (and therefore for all ω-regular automata); a conjecture (termed later the $G2$ conjecture) that is still open.

Our contribution. We extend token games to the quantitative setting, and use them to decide HDness of some quantitative automata. We define a k-token game on a quantitative automaton exactly as on a Boolean one, except that Eve wins if her run has a value at least as high as all of Adam's runs.

We show first, in Section 4, that the 1-token game, in which Adam just has one run to build, characterises history-determinism for all quantitative (and Boolean) automata on finite words, and for discounted-sum (DSum) automata on infinite words. This results in a PTime decision procedure for checking HDness of Inf and Sup automata on finite words, and an NP∩coNP procedure for DSum automata on finite and infinite words. Note that the complexity for DSum automata on finite words was already known [14], but on infinite words it was erroneously believed to be NP-hard [17, Theorem 6].

Towards getting the above results, we analyse key properties of value functions of quantitative automata, and show that the 1-token game characterises HDness for every Val automaton, such that Val is present-focused (Definition 3), which is in particular the case for all Val automata on finite words [8, Lemma 16], as well as DSum automata on infinite words [8, Lemma 22].

We then show, in Section 5, that the 2-token game, in which Adam builds two runs, characterises history-determinism for both LimSup and LimInf automata. The approach here is more involved: it decomposes the quantitative automaton into a collection of Büchi or coBüchi automata such that if Eve wins the 2-token game on the original automaton, she also wins in the component automata. Since the 2-token game characterises HD for Büchi and coBüchi automata, the component automata are then HD and the witness strategies can be combined with the 2-token strategy of the original automaton to build a letter-game strategy for Eve. The general flow of our approach is illustrated in Fig. 1.

We further present, in Section 5.1, algorithms to decide the winner of the two-token games on LimInf and LimSup automata via reductions to solving parity games. The complexity of the procedure for a LimSup automaton $\mathcal{A}$ is the same as that of solving a parity game of size polynomial in the size of $\mathcal{A}$ with twice as many priorities as there are weights in $\mathcal{A}$, which is in quasipolynomial time. For LimInf automata the procedure is in exponential time. In both cases, it is only in polynomial time if the number of weights is logarithmic in the automaton size.

For some variants of the synthesis problem, the complexity of the witness of history-determinism is also of particular interest (while for other variants it is not), as it corresponds to the complexity of the implementation of the solution

system [8, Section 5]. We give an exponential upper bound to the complexity of the witness for LimSup and LimInf automata, which, for LimInf, is tight. As a corollary, we obtain that HD LimSup automata are as expressive as deterministic LimSup automata and at most exponentially more succinct.

Related work. In the ω-regular setting (where HDness coincides with good-for-gameness), [16, Section 4] provides an exponential scheme for checking HDness of all ω-regular automata, based on determinisation and checking fair simulation. HDness of Büchi automata is resolved, as mentioned above, in PTime, using 2-token games [3]. The coBüchi case is also resolved in PTime, originally via an indirect usage of "joker games" [19], and later by using 2-token games [6].

In the quantitative setting, deciding HDness coincides with best-value partial domain synthesis [14], 0-regret synthesis [18] and, for some value functions, 0-regret determinisation [13,8]. There are procedures to decide HDness (which is sometimes called good-for-gameness due to erroneously assuming them equivalent) of Sum, Avg, and DSum automata on finite words, as follows.

For Sum and Avg automata on finite words, a PTime solution combines [1, Theorem 4.1], which provides a PTime algorithm for checking whether such an automaton is "determinisable by pruning", and [8, Theorem 21], which shows that such an automaton is HD if and only if it is determinisable by pruning.

Proposition 1. *Deciding whether a* Sum *or* Avg *automaton on finite words is history-deterministic is in* PTime.

For DSum automata on finite words, [14, Theorem 23] provides an NP∩co-NP solution, using a game that is quite similar to the one-token game, differing from it in a few aspects—for example, Adam is asked to either copy Eve with his token or move into a second phase where he plays transitions first—and uses a characterisation of HD strategies resembling our notion of cautious strategies (Definition 2) specialised to DSum automata.

2   Preliminaries

Words. An *alphabet* Σ is a finite nonempty set of letters. A finite (resp. infinite) *word* $u = \sigma_0 \ldots \sigma_k \in \Sigma^*$ (resp. $w = \sigma_0 \sigma_1 \ldots \in \Sigma^\omega$) is a finite (resp. infinite) sequence of letters from Σ; ε is the empty word. We write Σ^∞ for $\Sigma^* \cup \Sigma^\omega$. We use $[i..j]$ to denote a set $\{i, \ldots, j\}$ of integers, $[i]$ for $[i..i]$, $[..j]$ for $[0..j]$, and $[i..]$ for integers equal to or larger than i. We write $w[i..j], w[..j]$, and $w[i..]$ for the infix $\sigma_i \ldots \sigma_j$, prefix $\sigma_0 \ldots \sigma_j$, and suffix $\sigma_i \ldots$ of w. A *language* is a set of words.

Games. We consider a variety of turn-based zero-sum games between Adam (A) and Eve (E). Formally, a game is played on an arena of which the positions are partitioned between the two players. A play is a maximal (finite or infinite) path. The winning condition partitions plays into those that are winning for each player. In some of the technical developments we use *parity games*, in which

moves are coloured with integer priorities and a play is winning for Eve if the maximal priority that occurs infinitely often along the play is even.

A strategy for a player $P \in \{A, E\}$ maps partial plays ending in a position belonging to P to a successor position. A (partial) play π agrees with a strategy s_P of P, written $\pi \in s_P$, if whenever its prefix p ends in a position of P, the next move is $s_P(p)$. A strategy of P is winning from a position v if all plays starting at v that agree with it are winning for P. A strategy is positional if it maps all plays that end in the same position to the same successor. A game is determined if for every position, one of the players has a winning strategy.

Quantitative Automata. A *nondeterministic quantitative*[3] *automaton* (or just automaton from here on) on words is a tuple $\mathcal{A} = (\Sigma, Q, \iota, \delta)$, where Σ is an alphabet; Q is a finite nonempty set of states; $\iota \in Q$ is an initial state; and $\delta \colon Q \times \Sigma \to 2^{(\mathbb{Q} \times Q)}$ is a transition function over weight-state pairs.

A *transition* is a tuple $(q, \sigma, x, q') \in Q \times \Sigma \times \mathbb{Q} \times Q$, also written $q \xrightarrow{\sigma:x} q'$. (There might be several transitions with different weights over the same letter between the same states.) We write $\gamma(t) = x$ for the weight of a transition $t = (q, \sigma, x, q')$. $\mathcal{A}$ is deterministic if for all $q \in Q$ and $a \in \Sigma$, $\delta(q, a)$ is a singleton. We require that the automaton $\mathcal{A}$ is *total*, namely that for every state $q \in Q$ and letter $\sigma \in \Sigma$, there is at least one state q' and a transition $q \xrightarrow{\sigma:x} q'$.

A run of $\mathcal{A}$ on a word w is a sequence $\rho = q_0 \xrightarrow{w[0]:x_0} q_1 \xrightarrow{w[1]:x_1} q_2 \dots$ of transitions where $q_0 = \iota$ and $(x_i, q_{i+1}) \in \delta(q_i, w[i])$. As each transition t_i carries a weight $\gamma(t_i) \in \mathbb{Q}$, the sequence ρ provides a weight sequence $\gamma(\rho) = \gamma(t_0)\gamma(t_1)\dots$. A Val (e.g., Sum) automaton is one equipped with a *value function* $\mathsf{Val} \colon \mathbb{Q}^* \to \mathbb{R}$ or $\mathsf{Val} \colon \mathbb{Q}^\omega \to \mathbb{R}$, which assigns real values to runs of $\mathcal{A}$. The value of a run ρ is $\mathsf{Val}(\gamma(\rho))$. The value of $\mathcal{A}$ on a word w is the supremum of $\mathsf{Val}(\rho)$ over all runs ρ of $\mathcal{A}$ on w. Two automata $\mathcal{A}$ and $\mathcal{A}'$ are *equivalent*, if they realise the same function. The size of an automaton consists of the maximum among the size of its alphabet, state-space, and transition-space.

Value functions.
For finite sequences $v_0 v_1 \dots v_{n-1}$ of rational weights:

- $\mathsf{Sum}(v) = \displaystyle\sum_{i=0}^{n-1} v_i$ — $\mathsf{Avg}(v) = \displaystyle\frac{1}{n} \sum_{i=0}^{n-1} v_i$

For finite and infinite sequences $v_0 v_1 \dots$ of rational weights:

- $\mathsf{Inf}(v) = \inf\{v_n \mid n \geq 0\}$ — $\mathsf{Sup}(v) = \sup\{v_n \mid n \geq 0\}$

- For a discount factor $\lambda \in \mathbb{Q} \cap (0, 1)$, $\lambda\text{-}\mathsf{DSum}(v) = \displaystyle\sum_{i \geq 0} \lambda^i v_i$

For infinite sequences $v_0 v_1 \dots$ of rational weights:

---

[3] We speak of "quantitative" rather than "weighted" automata, following the distinction made in [5] between the two.

- $\mathsf{LimInf}(v) = \lim\limits_{n\to\infty} \inf\{v_i \mid i \geq n\}$ - $\mathsf{LimSup}(v) = \lim\limits_{n\to\infty} \sup\{v_i \mid i \geq n\}$

ω-regular automata (with acceptance on transitions) can be viewed as special cases of quantitative automata. In particular, a Büchi (resp. coBüchi) automaton can be seen as a quantitative one, in which a rejecting transition has weight 0, an accepting transition has weight 1, and whose value function is 1 if the sequence of weighs has infinitely many 1's and 0 otherwise (resp. 1 if the sequence of weights has finitely many 0). See more on ω-regular automata, e.g., in [4].

History-determinism. Intuitively, an automaton is history-deterministic if there is a strategy to resolve its nondeterminism according to the word read so far such that for every word, the value of the resulting run is the value of the word.

Definition 1 (History-determinism [11,8]). *A* Val *automaton* $\mathcal{A}$ *is* history-deterministic *(HD) if Eve wins the following win-lose* letter game*, in which Adam chooses the next letter and Eve resolves the nondeterminism, aiming to construct a run whose value is equivalent to the generated word's value.*

Letter game: *A play begins in* $q_0 = \iota$ *(the initial state of* $\mathcal{A}$*) and at the* i^{th} *turn, from state* q_i*, it progresses to a next state as follows:*
  - *Adam picks a letter* σ_i *from* Σ *and*
  - *Eve chooses a transition* $t_i = q_i \xrightarrow{\sigma_i : x_i} q_{i+1}$*.*
  In the limit, a play consists of an infinite word w *that is derived from the concatenation of* $\sigma_0, \sigma_1, \ldots$*, as well as an infinite sequence* $\pi = t_0, t_1, \ldots$ *of transitions. For* $\mathcal{A}$ *over infinite words, Eve wins a play in the letter-game if* $\mathsf{Val}(\pi) \geq \mathcal{A}(w)$*. For* $\mathcal{A}$ *over finite words, Eve wins if for all* $i \in \mathbb{N}$*,* $\mathsf{Val}(\pi[0..i]) \geq \mathcal{A}(w[0..i])$*.*

Consider for example the LimSup automaton $\mathcal{A}$ in Fig. 2. Eve loses the letter game on $\mathcal{A}$: Adam can start with the letter a; then if Eve goes from s_0 to s_1, Adam continues to choose a forever, generating the word a^ω, where $\mathcal{A}(a^\omega) = 3$, while Eve's run has the value 2. If, on the other hand, Eve chooses on her first move to go from s_0 to s_2, Adam continues with choosing b forever, generating the word ab^ω, where $\mathcal{A}(ab^\omega) = 2$, while Eve's run has the value 1.

Families of value functions. We will provide some of our results with respect to a family of Val automata based on properties of the value function Val.

We first define *cautious strategies* for Eve in both the letter game and token games (Section 3), which we use to define *present-focused* value functions. Intuitively, a strategy is cautious if it avoids mistakes: it only builds run prefixes that can achieve the maximal value of any continuation of the current word prefix.

Definition 2 (Cautious strategies [8]). *Consider the letter game on a* Val *automaton* $\mathcal{A}$*, in which Eve builds a run of* $\mathcal{A}$ *transition by transition. A move (transition)* $t = q \xrightarrow{\sigma:x} q'$ *of Eve, played after some run* ρ *ending in a state* q*, is* non-cautious *if for some word* w*, there is a run* π' *from* q *over* σw *such that* $\mathsf{Val}(\rho\pi')$ *is strictly greater than the value of* $\mathsf{Val}(\rho\pi)$ *for any* π *starting with* t*.*
  A strategy is cautious *if it makes no non-cautious moves.*

A winning strategy for Eve in the letter game must of course be cautious; Whether all cautious strategies are winning depends on the value function. We call a value function *present-focused* if, morally, it depends on the prefixes of the value sequence, formalised by winning the letter game via cautious strategies.

Definition 3 (Present-focused value functions [8]). *A value function* Val*, on finite or infinite sequences, is* present-focused *if for all automata $\mathcal{A}$ with value function* Val*, every cautious strategy for Eve in the letter game on $\mathcal{A}$ is also a winning strategy in that game.*

Value functions on finite sequences are present-focused, as they can only depend on prefixes, while value functions on infinite sequences are not necessarily present-focused [8, Remark 17], for example LimInf and LimSup.

Proposition 2 ([8, Lemma 16]). *Every value function* Val *on finite sequences of rational values is present focused.*

Proposition 3 ([8, Lemma 22]). *For every $\lambda \in \mathbb{Q} \cap (0,1)$, λ-DSum on infinite sequences of rational values is a present-focused value function.*

3   Token Games

Token games were introduced by Bagnol and Kuperberg [3] in the scope of resolving the HDness problem of Büchi automata. In the *k-token game*, known as G_k, the players proceed as in the letter game, except that now Adam has k tokens that he must move after Eve has made her move, thus building k runs. For Adam to win, at least one of these must be better than Eve's run. In the Boolean setting, this run must be accepting, thus witnessing that the word is in the language of the automaton. Intuitively, the more tokens Adam has, the less information he is giving Eve about the future of the word he is building.

We generalise token games to the quantitative setting, defining that the maximal value produced by Adam's runs witnesses a lower bound on the value of the word, and Eve's task is to match or surpass this value on her run.

In the Boolean setting, G_2 has the same winner as the letter game for Büchi [3, Corollary 21] and coBüchi [6, Theorem 28] automata (the case of parity and more powerful automata is open). Since G_2 is solvable in polynomial time for Büchi and coBüchi acceptance conditions, this gives a PTime algorithm for deciding HDness, which avoids the determinisation used to solve the letter game directly. In the following sections we study how different token games can be used to decide HDness for different quantitative automata.

Definition 4 (k-token games). *Consider a* Val *automaton $\mathcal{A} = (\Sigma, Q, \iota, \delta)$. A configuration of the game $G_k(\mathcal{A})$ for $k \geq 1$ is a tuple $(q, p_1, \ldots p_k) \in Q^{k+1}$ of states. A play consists of an infinite sequence of configurations $(\iota, \iota, \ldots, \iota) = (q_0, p_{1,0}, \ldots, p_{k,0}), (q_1, p_{1,1}, \ldots, p_{k,1}), \ldots$. In a configuration $(q_i, p_{1,i}, \ldots, p_{k,i})$, the game proceeds to the next configuration as follows.*

- *Adam picks a letter σ_i from Σ,*
- *Eve picks a transition $q_i \xrightarrow{\sigma_i : x_{0,i}} q_{i+1}$, and*
- *Adam picks transitions, $p_{1,i} \xrightarrow{\sigma_i : x_{1,i}} p_{1,i+1}, \ldots, p_{k,i} \xrightarrow{\sigma_i : x_{k,i}} p_{k,i+1}$.*

In the limit, a play consists of an infinite word w that is derived from the concatenation of $\sigma_0, \sigma_1, \ldots$, as well as $k + 1$ infinite sequences π (picked by Eve) and $\pi_1 \ldots \pi_k$ (picked by Adam) of transitions over w. Eve wins the play if $\mathsf{Val}(\pi) \geq \max(\mathsf{Val}(\pi_1), \ldots, \mathsf{Val}(\pi_k))$.

On finite words, G_k is defined as above, except that the winning condition is a safety condition for Eve: for all finite prefixes of a play, it must be the case that the value of Eve's run is at least the value of each of Adam's runs.

Cautious strategies (Definition 2) immediately extend to Eve's strategies in $G_k(\mathcal{A})$. Note that unlike in the letter game, a winning strategy in $G_k(\mathcal{A})$ must not necessarily be cautious, since Adam's run prefixes might not allow him to build an optimal run over the word witnessing that Eve's move was non-cautious.

4   Deciding History-Determinism via One-Token Games

Bagnol and Kuperberg showed that the one-token game G_1 does not suffice to characterise HDness for Büchi automata [3, Lemma 8]. However, it turns out that G_1 does characterise HDness for all quantitative (and Boolean) automata on finite words and some quantitative automata on infinite words.

We can then use G_1 to decide history-determinism of some of these automata, over which the G_1 game is simple to decide. In particular, Inf and Sup automata on finite words and DSum automata on finite and infinite words.

Theorem 1. *Given a nondeterministic automaton $\mathcal{A}$ with a present-focused value function Val over finite or infinite words, Eve wins $G_1(\mathcal{A})$ if and only if $\mathcal{A}$ is HD. Furthermore, a winning strategy for Eve in $G_1(\mathcal{A})$ induces a HD strategy with the same memory.*

Proof. One direction is easy: if $\mathcal{A}$ is HD, Eve can use her HD strategy to win G_1 by ignoring Adam's token. For the other direction, assume that Eve wins G_1.

We consider the following family of *copycat strategies* for Adam in G_1: a copycat strategy is one where Adam moves his token in the same way as Eve until she makes a non-cautious move $t = q \xrightarrow{\sigma : x} q'$ after building a run ρ; that is, there is some word w and run π' from q on σw, such that for every run π on σw starting with t, we have $\mathsf{Val}(\rho \pi') > \mathsf{Val}(\rho \pi)$. Then the copycat strategy stops copying and directs Adam's token along the run π' and plays the word w. If Eve plays a non-cautious move in G_1 against a copycat strategy, she loses. Then, if Eve wins G_1 with a strategy s, she wins in particular against all copycat strategies and therefore s never makes a non-cautious move against such a strategy.

Eve can then play in the letter game over $\mathcal{A}$ with a strategy s' that moves her token as s would in $G_1(\mathcal{A})$ assuming Adam uses a copycat strategy. Then, s' never makes a non-cautious move and is therefore a cautious strategy. Since

Val is present-focused, any cautious strategy, and in particular s', is winning in the letter game, so $\mathcal{A}$ is HD. Note that s' requires no more memory than s.   □

Corollary 1. *Given a nondeterministic automaton $\mathcal{A}$ over finite words, Eve wins $G_1(\mathcal{A})$ if and only if $\mathcal{A}$ is HD, and winning strategies in $G_1(\mathcal{A})$ induce HD strategies for $\mathcal{A}$ of the same complexity.*

Proof. A direct consequence of Proposition 2 and Theorem 1.   □

Solving token games. For resolving the HDness problem of Val automata where Val is present-focused, it then remains to study for which of them the corresponding G_1 game is simple to decide.

Theorem 2. *Deciding whether an Inf or Sup automaton on finite words is HD is in PTIME, namely in $O(|\Sigma|n^2k)$ for Sup and $O(|\Sigma|n^2k^2)$ for Inf, where Σ is the automaton's alphabet, k the number of weights and n the number of states.*

Proof. Given a Sup automaton $\mathcal{A} = (\Sigma, Q, \iota, \delta)$ with weights W, $G_1(\mathcal{A})$ reduces to solving a safety game, whose positions $(\sigma, q, q', x_E, t) \in \Sigma \cup \{\varepsilon\} \times Q^2 \times W \times \{L, E, A\}$ consist of a possibly empty letter σ representing the last letter played, a pair of states (q, q'), one for Eve and one for Adam, which keep track of the end of the current run built by each player, a weight x_E from W, which keeps track of the maximal weight seen on Eve's run so far, and a turn variable $t \in \{L, E, A\}$ indicating whether it is Adam's turn to give a letter (L), Eve's turn to choose a transition (E), or Adam's turn to choose a transition (A). The initial position is $(\varepsilon, \iota, \iota, m, L)$ where m is the minimal weight of $\mathcal{A}$. The moves and position ownership encode the permitted moves in $G_1(\mathcal{A})$ and update x_E to reflect the maximal value of Eve's run. The winning condition for Eve is a safety condition: Adam wins if he picks a move with a weight higher than x_E, the maximal weight on Eve's run. Then plays in this game are in bijection with plays of $G_1(\mathcal{A})$, and Eve wins if and only if she can avoid Adam choosing a transition with a larger weight than x_E, that is, if she can win $G_1(\mathcal{A})$.

Then, solving $G_1(\mathcal{A})$ reduces to solving this safety game, which can be done in time linear in the number of positions of the arena, which is $3|\Sigma|n^2k$.

The case of Inf automata is similar, except that instead of keeping Eve's maximal value along her run, we need to keep the minimal value along Adam's run in some variable x_A, and the safety condition for Eve is that her current value must always be at least as big as x_A and Adam's next move. Since Adam plays after Eve in each round of the game, we also need to keep Eve's last value, thus having $3|\Sigma|n^2k^2$ positions.   □

Next, we show that solving G_1 is in NP∩co-NP for DSum automata.

Theorem 3. *For every $\lambda \in \mathbb{Q} \cap (0,1)$, deciding whether a λ-DSum automaton $\mathcal{A}$, on finite or infinite words, is HD is in NP∩co-NP[4].*

---

[4] It was already known for finite words [14]. It is perhaps surprising for infinite words, given the NP-hardness result in [17, Theorem 6]. In consultation with the authors, we have confirmed that there is an error in the hardness proof.

Proof. Consider a λ-DSum automaton $\mathcal{A} = (\Sigma, Q, \iota, \delta)$, where the weight of a transition t is denoted by $\gamma(t)$. From Propositions 2 and 3 and Theorem 1, Eve wins $G_1(\mathcal{A})$ if and only if $\mathcal{A}$ is HD. It therefore suffices to show that solving $G_1(\mathcal{A})$ is NP∩co-NP. We achieve this by reducing solving $G_1(\mathcal{A})$ to solving a discounted-sum threshold game, which Eve wins if the DSum of a play is non-negative. It is enough to consider infinite games, as they also encode finite games, by allowing Adam to move to a forever-zero-position in each of his turns.

The reduction follows the same pattern as that in the proof of Theorem 2: we represent the arena of the game $G_1(\mathcal{A})$ as a finite arena, and encode its winning condition, which requires the difference between the DSum of two runs to be non-negative, as a threshold DSum winning condition. Note first that the difference between the λ-DSum of the two sequences $x_0 x_1...$ and $x'_0 x'_1...$ of weights is equal to the λ-DSum of the sequence of differences $d_0 = (x_0 - x'_0), d_1 = (x_1 - x'_1), \ldots$, as follows: $(\sum_{i=0}^{\infty} \lambda^i x_i) - \sum_{i=0}^{\infty} \lambda^i x'_i = \sum_{i=0}^{\infty} \lambda^i (x_i - x'_i)$.

We now describe the DSum arena G in which Eve wins with a non-strict 0-threshold objective if and only if she wins $G_1(\mathcal{A})$. The arena has positions in $(\sigma, q, q', t, m) \in \Sigma \cup \{\varepsilon\} \times Q^2 \times \delta \cup \{\varepsilon\} \times \{L, E, A\}$ where σ is the potentially empty last played letter, starting with ε, the states q, q' represent the positions of Eve and Adam's tokens, t is the transition just played by Eve if $m = A$ and ε otherwise, and m denotes the move type, having L for Adam choosing a letter, E for Eve choosing a transition and A for Adam choosing a transition.

A move of Adam that chooses a transition $t' = q' \xrightarrow{\sigma:x} q''$, namely a move $(\sigma, q, q', t, A) \rightarrow (\sigma, q, q'', \varepsilon, L)$, is given weight $\gamma(t) - \gamma(t')$, that is, the difference between the weights of the transitions chosen by both players. Other transitions are given weight 0. Observe that we need to compensate for the fact that only one edge in three is weighted. One option to do it is to take a discount factor $\lambda' = \lambda^{\frac{1}{3}}$ for the DSum game G. Yet, λ' can then be irrational, which somewhat complicates things. Another option is to consider discounted-sum games with multiple discount factors [2] and choose three rational discount factors $\lambda', \lambda'', \lambda''' \in \mathbb{Q} \cap (0, 1)$, such that $\lambda' \cdot \lambda'' \cdot \lambda''' = \lambda$. Since the first two weights in every triple are 0, only the multiplication of the three discount factors toward the third weight is what matters. For $\lambda = \frac{p}{q}$, where $p < q$ are positive integers, one can choose $\lambda' = \frac{4p}{4p+1}, \lambda'' = \frac{4p+1}{4p+2}$, and $\lambda''' = \frac{2p+1}{2q}$.

Plays in $G_1(\mathcal{A})$ and in G are in bijection. It now suffices to argue that the winning condition of G, namely that the $(\lambda', \lambda'', \lambda''')$-DSum of the play is non-negative, correctly encodes the winning condition of $G_1(\mathcal{A})$, meaning that the difference between the λ-DSum of Eve's run and of Adam's run is non-negative.

Let $d_0 d_1 \ldots$ be the sequence of weight differences between the transitions played by both players in $G_1(\mathcal{A})$, and let $\lambda_0, \lambda_1, \ldots$ and $w_0, w_1, \ldots$ be the corresponding sequences of discount factors and weights in the $(\lambda', \lambda'', \lambda''')$-DSum game, respectively, where for every $i = (0 \mod 3)$, we have $w_i = 0$ and $\lambda_i = \lambda'$, for every $i = (1 \mod 3)$, we have $w_i = 0$ and $\lambda_i = \lambda''$, and for every $i = (2 \mod 3)$, we have $w_i = d_i$ and $\lambda_i = \lambda'''$. Then the value of the $(\lambda', \lambda'', \lambda''')$-DSum sequence is equal to the required DSum sequence multiplied by $\lambda' \cdot \lambda''$:

$$(\lambda', \lambda'', \lambda''')\text{-DSum} = \sum_{i=0}^{\infty}(0\cdot\prod_{j=0}^{3i-1}\lambda_j \ +0\cdot\prod_{j=0}^{3i}\lambda_j \ +w_{3i+2}\cdot\prod_{j=0}^{3i+1}\lambda_j) = \lambda'\cdot\lambda''\cdot\sum_{i=0}^{\infty}\lambda^i d_i$$

Hence Eve wins the game $G_1(\mathcal{A})$ if and only if she wins the 0-threshold $(\lambda', \lambda'', \lambda''')$-DSum game over G. As G has a state-space polynomial in the state-space of $\mathcal{A}$ and solving DSum-games is in NP∩coNP [2], solving $G_1(\mathcal{A})$, and therefore deciding whether $\mathcal{A}$ is HD, is also in NP∩coNP.   □

DSum games are positionally determined [22,23,2] so this algorithm also computes a finite-memory witness of HDness for $\mathcal{A}$ that is of polynomial size in the state-space of $\mathcal{A}$. However, a positional witness also exists [17, Section 5].

5   Deciding History-Determinism via Two Token Games

In this section we solve the HDness problem of LimSup and LimInf automata via two-token games. As is the case with Büchi and coBüchi automata, one-token games do not characterise HDness of LimSup and LimInf automata. For LimInf, a possible alternative approach is to try to solve the letter game directly: we can use an equivalent deterministic LimInf automaton to track the value of a word, and the winning condition of the letter game corresponds to comparing Eve's run to the one of the deterministic automaton. Unfortunately, determinising LimInf automata is exponential in the number of its states [10, Theorem 13], so the new game is large, and, in addition, its winning condition, which compares the LimInf value of two runs, is non-standard and needs additional work to be encoded into a parity game. For LimSup automata the situation is even worse, as they are not necessarily equivalent to deterministic LimSup automata, so it is not obvious whether the winner of the letter game is decidable at all.

Here we show that the 2-token-game approach, used to resolve HDness of Büchi and coBüchi automata, can be generalised to LimSup and LimInf automata. While the proof that G_2 has the same winner as the letter game is quite different for the Büchi and coBüchi cases, our proofs for the LimSup and LimInf cases follow the same structure, while relying on the Büchi and coBüchi results respectively. However, the argument that $G_2(\mathcal{A})$ is solvable differs according to whether $\mathcal{A}$ is a LimSup or LimInf automaton. In particular, perhaps surprisingly (since the naive approach to solving the letter game seems harder for LimSup), we show that G_2 is solvable in quasipolynomial time for LimSup while for LimInf our algorithm is exponential in the number of weights (but not in the number of states).

Without loss of generality, we assume the weights to be $\{1, 2, \ldots\}$.

We start, in Section 5.1, with analysing the 2-token game on LimSup and LimInf automata, and show, in Section 5.2, that it characterises their HDness.

5.1 G_2 on LimSup and LimInf Automata

We first observe that $G_2(\mathcal{A})$, for both a LimSup and a LimInf automaton $\mathcal{A}$, can be solved via a reduction to a parity game. The G_2 winning condition for LimSup

automata can be encoded by adding carefully chosen priorities to the arena of $G_2(\mathcal{A})$, while for LimInf the encoding requires additional positions.

Lemma 1. *Given a nondeterministic* LimSup *automaton $\mathcal{A}$ of size n with k weights, the game $G_2(\mathcal{A})$ can be solved in time quasipolynomial in n, and if k is in $O(\log n)$, in time polynomial in n.*

Proof. We encode the game $G_2(\mathcal{A})$, for a LimSup automaton $\mathcal{A} = (\Sigma, Q, \iota, \delta)$, into a parity game as follows. The arena is simply the arena of $G_2(\mathcal{A})$, seen as a product of the alphabet and three copies of $\mathcal{A}$, to reflect the current letter and the current position of each of the three runs (one for Eve, two for Adam).

Adam's letter-picking moves are labelled with priority 0, Eve's choices of transition $q \xrightarrow{\sigma:x} q'$ are labelled with priority $2x$ and Adam's choices of transition $q \xrightarrow{\sigma:x} q'$ are labelled with priority $2x - 1$.

We claim that Eve wins this parity game if and only if she wins $G_2(\mathcal{A})$, that is, the priorities correctly encode the winner of $G_2(\mathcal{A})$. Observe that the even priorities seen infinitely often in a play of the parity game are exactly priorities $2x$, where x is a weight seen infinitely often in Eve's run in the corresponding play in $G_2(\mathcal{A})$. The odd priorities seen infinitely often on the other hand are $2x - 1$, where $x > 0$ occurs infinitely often on one of Adam's runs in the corresponding play of $G_2(\mathcal{A})$. Hence, Eve can match the maximal value of Adam's runs in $G_2(\mathcal{A})$ if and only if she can win the parity game that encodes $G_2(\mathcal{A})$.

The number of positions in this game is polynomial in the size n of $\mathcal{A}$; the maximal priority is linear in the number of weights. It can be solved in quasipolynomial time, or in polynomial time if the number of weights is in $O(\log n)$, using the reader's favourite state-of-the-art parity game algorithm, for instance [9]. □

Lemma 2. *Given a nondeterministic* LimInf *automaton $\mathcal{A}$ of size n with k weights, the game $G_2(\mathcal{A})$ can be solved in time exponential in n, and if k is in $O(\log n)$, in time polynomial in n.*

Proof. As in the proof of Lemma 1, we can represent $G_2(\mathcal{A})$ as a game on an arena that is the product of three copies of $\mathcal{A}$, one for Eve and two for Adam. The winning condition for Eve is that the smallest weight seen infinitely often on the run built on her copy of $\mathcal{A}$ should be at least as large as both of the minimal weights seen infinitely often on the runs built on Adam's copies. We will encode this winning condition as a parity condition, but, unlike in the LimSup case, we will need to use an additional memory structure, which we describe now.

Intuitively, the weights on Eve's run will be encoded by *odd* priorities, with smaller weights corresponding to higher priorities, as in LimInf the lowest weight seen infinitely often is the one that matters, while weights on Adam's runs will be encoded by *even* priorities, but only once both of Adam's runs have seen the corresponding weight or a lower one. This is the role of the memory structure, which encodes which of Adam's runs has seen which weight recently.

More precisely, let k be the number of weights in $\mathcal{A}$. Moves corresponding to Eve choosing a transition of weight i have priority $2(k - i + 1) - 1$, that is,

an odd priority that is larger the smaller i is. Further, for each weight, we use a three-valued variable $x_i \in \{0, 1, 2\}$, initiated to 0, which gets updated as follows: if $x_i = 0$ and the game takes a transition with a weight $w \leq i$ on one of Adam's runs, x_i is updated to 1 or 2 according to which of Adam's run saw this weight; if $x_i = 1$ (resp. 2) and Adam's second (resp. first) run takes a transition with weight $w \leq i$, then x_i is reset to 0. Transitions that reset variables to 0 have priority $2(k-i+1)$ for the minimal i such that the transition resets x_i to 0; other transitions have priority 1. Other moves do not affect x_i, and have priority 1.

We now argue that the highest priority seen infinitely often along a play is even if and only if the LimInf value of Eve's run is at least as high as that of both of Adam's runs. Indeed, the maximal odd priority seen infinitely often on a play is $2(k - i + 1) - 1$ such that i is the minimal priority seen on Eve's run infinitely often, and the maximal even priority seen infinitely often is $2(k-j+1)$ where j is the minimal weight such that both of Adam's runs see j or a smaller priority infinitely often. In particular, $2(k - i + 1) - 1 < 2(k - j + 1)$ if and only if $i \geq j$, that is, if Eve wins $G_2(\mathcal{A})$.

This parity game is of size exponential in k due to the memory structure ($\{0, 1, 2\}^k$) and has $2k$ priorities. As the number of priorities is logarithmic in the size of the game, it can be solved in polynomial time [9]. If the number of weights is in $O(\log n)$, then the algorithm is polynomial in the size n of $\mathcal{A}$. $\square$

5.2 G_2 Characterises HDness for LimSup and LimInf Automata

The rest of the section is dedicated to proving that a LimSup or LimInf automaton is HD if and only if Eve wins the 2-token game on it. In both cases, the structure of the argument is similar. One direction is immediate: if an automaton $\mathcal{A}$ is HD, then Eve can use the letter-game strategy to win in $G_2(\mathcal{A})$, ignoring Adam's tokens. The other direction requires more work. We use an additional notion, that of k-HDness, also known as the width of an automaton [21], which generalises HDness, in the sense that Eve maintains k runs, rather than only one, and needs at least one of them to be optimal. We will then show that if Eve wins $G_2(\mathcal{A})$, then $\mathcal{A}$ is k-HD for a finite k (namely, the number of weights in $\mathcal{A}$ minus one). Finally, we will show that for automata that are k-HD, for any finite k, a strategy for Eve in $G_2(\mathcal{A})$ can be combined with the k-HD strategy to obtain a strategy for her in the letter game.

Many of the tools used in this proof are familiar from the ω-regular setting [3,6]. The main novelty in the argument is the decomposition of the LimSup (LimInf) automaton $\mathcal{A}$ with k weights into $k - 1$ Büchi (coBüchi) automata $\mathcal{A}_2, \ldots, \mathcal{A}_k$ that are HD whenever Eve wins $G_2(\mathcal{A})$. (The converse does not hold, namely $\mathcal{A}_2, \ldots, \mathcal{A}_k$ can be HD even if Eve loses $G_2(\mathcal{A})$ – see Fig. 2.) The HD strategies for $\mathcal{A}_2, \ldots, \mathcal{A}_k$ can then be combined to prove the k-HDness of $\mathcal{A}$.

Fig. 1 illustrates the flow of our arguments.

We first generalise to quantitative automata Bagnol and Kuperberg's key insight that if Eve wins G_2, then she also wins G_k for all k [3, Thm 14].
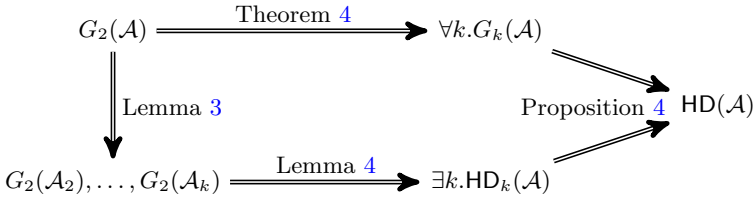
Fig. 1. The flow of arguments for showing that $G_2(\mathcal{A}) \implies \mathsf{HD}(\mathcal{A})$ for a LimInf or LimSup automaton $\mathcal{A}$.
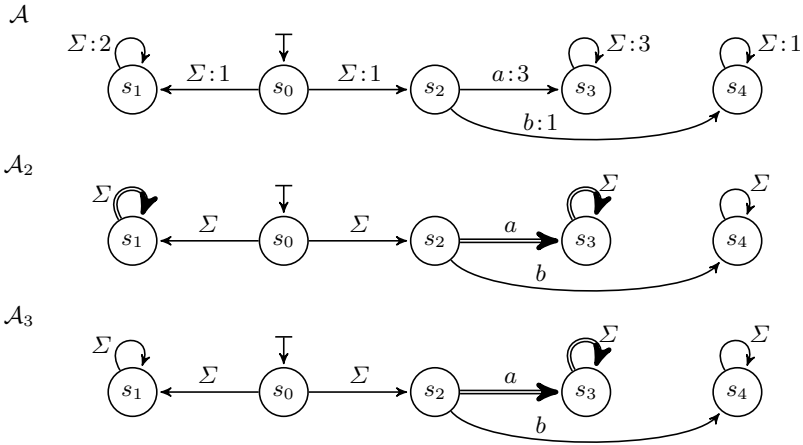

Fig. 2. A LimSup automaton $\mathcal{A}$ and corresponding Büchi automata $\mathcal{A}_2$ and $\mathcal{A}_3$, as per Lemma 3. (Accepting transitions in $\mathcal{A}_2$ and $\mathcal{A}_3$ are marked with double lines.) Observe that $\mathcal{A}$ is not HD and Eve loses the two-token game on $\mathcal{A}$, while both $\mathcal{A}_2$ and $\mathcal{A}_3$ are HD. (In $\mathcal{A}$, if Eve goes from s_0 to s_1, Adam goes from s_0 to s_2 and continues with an a, and if she goes from s_0 to s_2, Adam goes from s_0 to s_1 and continues with a b. In $\mathcal{A}_2$ Eve goes from s_0 to s_1 and in $\mathcal{A}_3$ from s_0 to s_2.)

Theorem 4. *Given a quantitative automaton $\mathcal{A}$, if Eve wins $G_2(\mathcal{A})$ then she also wins $G_k(\mathcal{A})$ for any $k \in \mathbb{N} \setminus \{0\}$. Furthermore, if her winning strategy in $G_2(\mathcal{A})$ has memory of size m and $\mathcal{A}$ has n states, then she has a winning strategy in $G_k(\mathcal{A})$ with memory of size $n^{k-1} \cdot m^k$.*

Proof. This is the generalisation of [3, Thm 14]. The proof is similar to Bagnol and Kuperberg's original proof, but without assuming positional strategies for Eve in $G_k(\mathcal{A})$. If Eve wins $G_2(\mathcal{A})$ then she obviously wins $G_1(\mathcal{A})$, using her G_2 strategy with respect to two copies of Adma's single token in G_1. We thus consider below $G_k(\mathcal{A})$ for every $k \in \mathbb{N} \setminus \{0, 1, 2\}$.

Let s_2 be a winning strategy for Eve in $G_2(\mathcal{A})$. We inductively show that Eve has a winning strategy s_i in $G_i(\mathcal{A})$ for each finite i. To do so, we assume a winning strategy s_{i-1} in $G_{i-1}(\mathcal{A})$. The strategy s_i maintains some additional

(not necessarily finite) memory that maintains the position of one virtual token in $\mathcal{A}$, a position in the (not necessarily finite) memory structure of s_{i-1}, and a position in the (not necessarily finite) memory structure of s_2. The virtual token is initially at the initial state of $\mathcal{A}$. The strategy s_i then plays as follows: at each turn, after Adam has moved his i tokens and played a letter (or, at the first turn, just played a letter), it first updates the s_{i-1} memory structure, by ignoring the last of Adam's tokens, and, treating the position of the virtual token as Eve's token in $G_{i-1}(\mathcal{A})$, it updates the position of the virtual token according to the strategy s_{i-1}; it then updates the s_2 memory structure by treating Adam's last token and the virtual token as Adam's 2 tokens in $G_2(\mathcal{A})$, and finally outputs the transition to be played according to s_2.

We now argue that this strategy is indeed winning in $G_i(\mathcal{A})$. Since s_{i-1} is a winning strategy in $G_{i-1}(\mathcal{A})$, the virtual token traces a run of which the value is at least as large as the value of any of the runs built by the first $i-1$ tokens of Adam. Since s_2 is also winning, the value of the run built by Eve's token is at least as large as the values of the runs built by the virtual token and by Adam's last token. Hence, Eve is guaranteed to achieve at least the supremum value of Adam's i runs, making this a winning strategy in $G_i(\mathcal{A})$.

As for the memory size of a winning strategy for Eve in $G_k(\mathcal{A})$, let m be the memory size of her winning strategy in $G_2(\mathcal{A})$ and n the number of states in $\mathcal{A}$. Then, by the above construction of her strategy in $G_k(\mathcal{A})$, the memory of her strategy in $G_3(\mathcal{A})$ is n for the virtual token times m for the copy of her memory in $G_2(\mathcal{A})$ times m for the copy of her memory in $G_{i-1}(\mathcal{A}) = G_2(\mathcal{A})$, namely $n \cdot m \cdot m = n \cdot m^2$. Then for $G_4(\mathcal{A})$ it is $n \cdot m \cdot (n \cdot m^2) = n^2 \cdot m^3$; for $G_5(\mathcal{A})$ it is $n \cdot m \cdot (n^2 \cdot m^3) = n^3 \cdot m^4$, and for $G_k(\mathcal{A})$ it is $n^{k-1} \cdot m^k$. $\square$

We proceed with the definition of k-HDness, also known as width [21], based on the k-runs letter game (not to be confused with G_k, the k-token game), which generalises the letter game.

Definition 5 (k-HD and k-runs letter game). *A configuration of the game on a LimSup (LimInf) automaton $\mathcal{A} = (\Sigma, Q, \iota, \delta)$ is a tuple $q^k \in Q^k$ of states of $\mathcal{A}$, initialised to ι^k.*

In a configuration $(q_{i,1}, \ldots, q_{i,k})$, the game proceeds to the next configuration $(q_{i+1,1}, \ldots, q_{i+1,k})$ as follows.

- *Adam picks a letter $\sigma_i \in \Sigma$, then*
- *Eve chooses for each $q_{i,j}$, a transition $q_{i,j} \xrightarrow{\sigma_i : x_{i,j}} q_{i+1,j}$*

In the limit, a play consists of an infinite word w that is derived from the concatenation of $\sigma_0, \sigma_1, \ldots$, as well as of k infinite sequences $\rho_0, \rho_1, \ldots$ of transitions. Eve wins the play if $\max_{j \in \{1 \ldots k\}} \mathsf{Val}(\rho_j) = \mathcal{A}(w)$.

If Eve has a winning strategy, we say that $\mathcal{A}$ is k-HD, or that $\mathsf{HD}_k(\mathcal{A})$ holds.

Notice that the standard letter game (Definition 1) is a 1-run letter game and standard HD (Definition 1) is 1-HD.

Next, we use $\mathsf{HD}_k(\mathcal{A})$ to show that G_2 characterises HDness.

Proposition 4 ([3]). *Given a quantitative automaton $\mathcal{A}$, if $\mathsf{HD}_k(\mathcal{A})$ for some $k \in \mathbb{N}$, and Eve wins G_k, then $\mathcal{A}$ is HD.*

Proof. The argument is identical to the one used in [3], which we summarise here. The strategy τ for Eve in $\mathsf{HD}_k(\mathcal{A})$ provides a way of playing k tokens that guarantees that one of the k runs formed achieves the automaton's value on the word w played by Adam. If Eve moreover wins $G_k(\mathcal{A})$ with some strategy s_k, she can, in order to win in the letter game, play s_k against Adam's letters and k virtual tokens that she moves according to τ. The winning strategy τ guarantees that one of the k runs built by the k virtual tokens achieves $\mathsf{Val}(w)$; then her strategy s_k guarantees that her run also achieves $\mathsf{Val}(w)$.   □

It remains to prove that if Eve wins $G_2(\mathcal{A})$, then $\mathsf{HD}_k(\mathcal{A})$ for some k.

Given a LimSup automaton $\mathcal{A}$, with weights $\{1, \ldots, k\}$, we define $k-1$ auxiliary Büchi automata $\mathcal{A}_2, \ldots, \mathcal{A}_k$ with acceptance on transitions, such that each $\mathcal{A}_x$ is a copy of $\mathcal{A}$, where a transition is accepting if its weight i in $\mathcal{A}$ is at least x. Each $\mathcal{A}_x$ recognises the set of words w such that $\mathcal{A}(w) \geq x$. (See Fig. 2.)

Given a LimInf automaton $\mathcal{A}$, we similarly define auxiliary *coBüchi* automata: A_x is a copy of $\mathcal{A}$ where transitions with weights smaller than x are rejecting, while those with weights x or larger are accepting. Again, $\mathcal{A}_x$ recognises the set of words w such that $\mathcal{A}(w) \geq x$.

We now use these auxiliary automata to argue that if $G_2(\mathcal{A})$ then $\mathsf{HD}_{k-1}(\mathcal{A})$.

Lemma 3. *Given a LimSup or LimInf automaton $\mathcal{A}$ with weights $\{1, \ldots, k\}$, if Eve wins $G_2(\mathcal{A})$, then for all $x \in \{2, \ldots, k\}$, Eve also wins $G_2(\mathcal{A}_x)$.*

Proof. Since $\mathcal{A}_x$ is identical to $\mathcal{A}$ except for the acceptance condition or value function, Eve can use in $G_2(\mathcal{A}_x)$ her winning strategy in $G_2(\mathcal{A})$. For the LimSup case, if one of Adam's runs sees an accepting transition infinitely often, the underlying transition of $\mathcal{A}$ visited infinitely often has weight at least x. Then, Eve's strategy guarantees that her run also sees infinitely often a value at least as large as x, corresponding to an accepting transition in $G_2(\mathcal{A}_x)$.

Similarly, for the LimInf case, if one of Adam's runs avoids seeing a rejecting transition infinitely often in $\mathcal{A}_x$, then this run's value in $\mathcal{A}$ is at least x, and Eve's strategy guarantees that her run's value in $\mathcal{A}$ is at least x, meaning that it avoids seeing a rejecting transition in $\mathcal{A}_x$ infinitely often, and accepts.   □

Lemma 4. *Given a LimSup or LimInf automaton $\mathcal{A}$ with weights $\{1, \ldots, k\}$, if Eve wins $G_2(\mathcal{A}_x)$ for all $x \in \{2, \ldots, k\}$ then $\mathsf{HD}_{k-1}(\mathcal{A})$ holds.*

Proof. From Lemma 3, if Eve wins $G_2(\mathcal{A})$, then for all $x \in \{2, \ldots, k\}$, Eve also wins $G_2(\mathcal{A}_x)$. Since each A_x is a Büchi or coBüchi automaton, this implies that for all $x \in \{2, \ldots, k\}$, the automaton A_x is HD [3,6], that is, there is a winning strategy s_x for Eve in the letter game on each $\mathcal{A}_x$. Now, in the $(k-1)$-run letter game on $\mathcal{A}$, Eve can use each s_x to move one token. Then, if Adam plays a word w with some value $\mathsf{Val}(w) = i$, this word is accepted by $\mathcal{A}_i$, and therefore the strategy s_i guarantees that the run of the ith token achieves at least the value i, corresponding to seeing accepting transitions of $\mathcal{A}_i$ infinitely often for

the LimSup case, or eventually avoiding rejecting transitions in the LimInf case.
□

Finally, we combine the G_2 and HD_{k-1} strategies in $\mathcal{A}$ to show that $\mathcal{A}$ is HD.

Theorem 5. *A nondeterministic* LimSup *or* LimInf *automaton* $\mathcal{A}$ *is HD if and only if Eve wins* $G_2(\mathcal{A})$.

Proof. If $\mathcal{A}$ is HD then Eve can use the letter-game strategy to win in $G_2(\mathcal{A})$, ignoring Adam's moves. If Eve wins $G_2(\mathcal{A})$ then by Lemma 3 and Lemma 4 she wins $HD_{k-1}(\mathcal{A})$, where k is the number of weights in $\mathcal{A}$. By Theorem 4 she also wins $G_{k-1}(\mathcal{A})$ and, finally, by Proposition 4 we get that $\mathcal{A}$ is HD.   □

Theorem 6. *Given a nondeterministic* LimSup *(resp.* LimInf*) automaton* $\mathcal{A}$ *of size n with k weights, the HDness problem of $\mathcal{A}$ can be solved in time quasipolynomial (resp. exponential) in n. In both cases, if k is in $O(\log n)$, it can be solved in time polynomial in n.*

Proof. It directly follows from Theorem 5 and Lemmas 1 and 2; the former reducing the HDness problem to solving $G_2(\mathcal{A})$, and the latter two showing that $G_2(\mathcal{A})$ can be solved in the stated complexity.   □

In contrast to the cases considered in the Section 4, where strategies in G_1 immediately induce HD strategies of the same complexity, for Büchi and coBüchi automata, a winning G_2 strategy does not necessarily induce an HD strategy (even though it implies the existence of such a strategy). We now analyse the size of the HD strategies which our proofs show exist whenever Eve wins G_2, and discuss the implications for the determinisability of HD LimSup automata.

Corollary 2. *Given an HD* LimSup *or* LimInf *automaton* $\mathcal{A}$ *of size n, there is an HD strategy for $\mathcal{A}$ with memory exponential in n. If $\mathcal{A}$ is a* LimSup *automaton with $O(\log n)$ weights then the memory is only polynomial in n.*

Proof. Let n be the size of $\mathcal{A}$ and $k+1$ the number of weights. We construct an HD strategy for $\mathcal{A}$, by combining an HD_k strategy and a G_k strategy for it.

The HD_k strategy—which, like the HD strategy, is hard to compute directly—combines the HD strategies of the k auxiliary Büchi or coBüchi automata for $\mathcal{A}$, as constructed in Lemma 3. For HD Büchi automata, which are equivalent to deterministic automata of quadratic size [19], there always exists a polynomial resolver: indeed, the letter game can be represented as a polynomial parity game, in which a positional strategy for Eve corresponds to a resolver. For HD coBüchi automata on the other hand, these auxiliary strategies might have exponential memory in the number of states of $\mathcal{A}$ [19].

The G_k strategy on the other hand is positional for LimSup, since it can be encoded as a parity game directly on the $G_k(\mathcal{A})$ arena, similarly to the reduction in Lemma 1; the size of the $G_k(\mathcal{A})$ arena is $O(n^{k+1})$. The overall HD strategy for LimSup therefore needs memory exponential in the number of weights.

For LimInf on the other hand, by Lemma 2 and Theorem 4, the G_k strategy can do with memory of size $n^{k-1} \cdot 3^{k^2}$. The overall HD strategy therefore has memory exponential in the size of $\mathcal{A}$.   □

We leave open whether this can be improved upon. Already for coBüchi automata, it is known that deciding whether an automaton is HD is polynomial despite there being automata for which the optimal HD strategy is exponential. Hence, at least for the LimInf case, we cannot expect to do much better. However, for the LimSup case, it might be that polynomial, or even positional HD strategies could suffice. However, positionality is already open for the Büchi case.

Our proof does however imply that if a LimSup automaton $\mathcal{A}$ is HD, then there is a *finite memory* HD strategy, which implies that $\mathcal{A}$ is determinisable, without increasing the number of weights, by taking a product of $\mathcal{A}$ with the finite HD strategy. (Recall that every LimInf automaton can be determinised, while not every LimSup automaton can.)

Corollary 3. *Every HD LimSup automaton is equivalent to a deterministic one with at most an exponential number of states and the same set of weights.*

6   Conclusions

We have extended the token-game approach to characterising history-determinism from the Boolean (ω-regular) to the quantitative setting. Already 1-token games turn out to be useful for characterising history-determinism for some quantitative automata. For LimSup and LimInf automata, one token is not enough, but the 2-token game does the trick. Given the correspondence between deciding history-determinism and the best-value synthesis problem, our results also directly provide algorithms both to decide whether the synthesis problem is realisable and to compute a solution strategy.

This application further motivates understanding the limits of these techniques. Whether the 2-token game G_2 characterises more general Boolean classes of automata beyond Büchi and coBüchi automata is already an open question. Similarly, we leave open whether the G_2 game also characterises history-determinism for limit-average automata and other quantitative automata. At the moment we are not aware of examples of automata of any kind (quantitative, pushdown, register, timed, ...) for which Eve could win G_2 despite the automaton not being history-deterministic, yet even for parity automata, a proof of characterisation remains elusive.

Acknowledgments

References

1. Benjamin Aminof, Orna Kupferman, and Robby Lampert. Reasoning about online algorithms with weighted automata. *ACM Trans. Algorithms*, 6(2):28:1–28:36, 2010.

2. Daniel Andersson. An improved algorithm for discounted payoff games. In *Proc. of ESSLLI Student Session*, pages 91–98, 2006.
3. Marc Bagnol and Denis Kuperberg. Büchi good-for-games automata are efficiently recognizable. In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018)*, page 16, 2018.
4. Udi Boker. Why these automata types? In *Proceedings of LPAR*, pages 143–163, 2018.
5. Udi Boker. Quantitative vs. weighted automata. In *Proc. of Reachbility Problems*, pages 1–16, 2021.
6. Udi Boker, Denis Kuperberg, Karoliina Lehtinen, and Michał Skrzypczak. On succinctness and recognisability of alternating good-for-games automata. *arXiv preprint arXiv:2002.07278*, 2020.
7. Udi Boker and Karoliina Lehtinen. Good for games automata: From nondeterminism to alternation. In *Proceedings of CONCUR*, volume 140 of *LIPIcs*, pages 19:1–19:16, 2019.
8. Udi Boker and Karoliina Lehtinen. History determinism vs. good for gameness in quantitative automata. In *Proc. of FSTTCS*, pages 35:1–35:20, 2021.
9. Cristian S Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *Proceedings of STOC*, pages 252–263, 2017.
10. Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Trans. Comput. Log.*, 11(4):23:1–23:38, 2010.
11. Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *Proceedings of ICALP*, pages 139–150, 2009.
12. Thomas Colcombet and Nathanaël Fijalkow. Universal graphs and good for games automata: New tools for infinite duration games. In *Proc. of FOSSACS*, volume 11425 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2019.
13. Emmanuel Filiot, Ismaël Jecker, Nathan Lhote, Guillermo A. Pérez, and Jean-François Raskin. On delay and regret determinization of max-plus automata. In *LICS*, pages 1–12, 2017.
14. Emmanuel Filiot, Christof Löding, and Sarah Winter. Synthesis from weighted specifications with partial domains over finite words. In Nitin Saxena and Sunil Simon, editors, *FSTTCS*, volume 182 of *LIPIcs*, pages 46:1–46:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
15. Shibashis Guha, Ismaël Jecker, Karoliina Lehtinen, and Martin Zimmermann. A bit of nondeterminism makes pushdown automata expressive and succinct. In *Proc. of MFCS*, pages 53:1–53:20, 2021.
16. Thomas Henzinger and Nir Piterman. Solving games without determinization. In *Proceedings of CSL*, pages 395–410, 2006.
17. Paul Hunter, Guillermo A. Pérez, and Jean-François Raskin. Minimizing regret in discounted-sum games. In Jean-Marc Talbot and Laurent Regnier, editors, *CSL*, volume 62 of *LIPIcs*, pages 30:1–30:17, 2016.
18. Paul Hunter, Guillermo A. Pérez, and Jean-François Raskin. Reactive synthesis without regret. *Acta Informatica*, 54(1):3–39, 2017.
19. Denis Kuperberg and Michał Skrzypczak. On determinisation of good-for-games automata. In *Proceedings of ICALP*, pages 299–310, 2015.
20. Karoliina Lehtinen and Martin Zimmermann. Good-for-games ω-pushdown automata. In *LICS20*, pages 689–702, 2020.
21. Anirban Majumdar and Denis Kuperberg. Computing the width of nondeterministic automata. *Logical Methods in Computer Science*, 15, 2019.

22. L. S. Shapley. Stochastic games. In *Proc. of Nat. Acad. Sci.*, volume 39, pages 1095–1100, 1953.
23. Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Electron. Colloquium Comput. Complex.*, 2(40), 1995.

On the Translation of Automata to Linear Temporal Logic⋆

Udi Boker[1] , Karoliina Lehtinen[2,✉] , and Salomon Sickert[3⋆⋆]

[1] Reichman University, Herzliya, Israel
`udiboker@idc.ac.il`
[2] CNRS, Aix-Marseille University and University of Toulon, LIS, Marseille, France
`lehtinen@lis-lab.fr`
[3] The Hebrew University, Jerusalem, Israel
`salomon.sickert@mail.huji.ac.il`

Abstract While the complexity of translating future linear temporal logic (LTL) into automata on infinite words is well-understood, the size increase involved in turning automata back to LTL is not. In particular, there is no known elementary bound on the complexity of translating deterministic ω-regular automata to LTL.

Our first contribution consists of tight bounds for LTL over a unary alphabet: alternating, nondeterministic and deterministic automata can be exactly exponentially, quadratically and linearly more succinct, respectively, than any equivalent LTL formula. Our main contribution consists of a translation of general counter-free deterministic ω-regular automata into LTL formulas of double exponential temporal-nesting depth and triple exponential length, using an intermediate Krohn-Rhodes cascade decomposition of the automaton. To our knowledge, this is the first elementary bound on this translation. Furthermore, our translation preserves the acceptance condition of the automaton in the sense that it turns a looping, weak, Büchi, coBüchi or Muller automaton into a formula that belongs to the matching class of the syntactic future hierarchy. In particular, it can be used to translate an LTL formula recognising a safety language to a formula belonging to the safety fragment of LTL (over both finite and infinite words).

Keywords: Linear temporal logic · Automata · Cascade decomposition

1 Introduction

Linear Temporal Logic with only future temporal operators (from here on LTL) and ω-regular automata, whether deterministic, nondeterministic or alternating, are both well-established formalisms to describe properties of infinite-word languages. LTL is popular in formal verification and synthesis due to its simple

---

syntax and semantics. Yet, while properties might be convenient to define in LTL, most verification and synthesis algorithms eventually compile LTL formulas into ω-regular automata. The expressiveness of both these key formalisms, as well as translations from LTL to automata of various types, are well understood. Here, we consider the converse translations, which, in comparison, have received less attention: up till now, no elementary upper bound on the size blow-up of going from automata to LTL was known.

Regarding expressive power, deterministic Muller automata, nondeterministic Büchi automata, and weak alternating automata recognise all ω-regular languages [21,40]. LTL-definable languages (surveyed in [13]) are a strict subset thereof, also defined by first-order logic, star-free regular expressions, aperiodic monoids, counter-free automata, and very weak alternating automata. As for succinctness, nondeterministic and alternating automata can be exponentially and double-exponentially more succinct than deterministic automata, respectively. Determinisation in particular has precise bounds [32,35,24,36,12,3].

The succinctness of various representations of LTL-definable languages is less clear: effective translations between the different models are far from straightforward, and their complexity is sometimes uncertain. In particular, to the best of our knowledge, up to now there has been no elementary bound even on the translation of deterministic counter-free automata, arguably the simplest automata model for this class of languages, into LTL formulas. (Considering LTL with both future and past temporal operators, there is a double-exponential upper bound on the length of the formula [26][4].) The complexity of obtaining a deterministic counter-free automaton from a nondeterministic one is also, to the best of our knowledge, open.

We study the complexity of translating automata to LTL (equivalently, to very weak alternating automata), considering formula length, size, and nesting depth of temporal operators.

We begin (Section 3), as a warm-up, with the unary alphabet case on finite words. We show that the size-blow up involved in translating deterministic, non-deterministic and alternating automata to LTL, when possible, is linear, quadratic and exponential, respectively, and these bounds are tight. In contrast, going from LTL to alternating, nondeterministic and deterministic automata is linear, exponential and double-exponential, respectively [33,41,19].

The case of non-unary alphabets is much more difficult. We provide a translation of counter-free deterministic ω-regular automata (with any acceptance condition) into LTL formulas with double exponential depth and triple exponential length. Our translation uses an intermediate Krohn-Rhodes *reset cascade decomposition* (wreath product) of deterministic automata, which is a deterministic automaton built from simple components.

Our main technical contribution consists of a translation of a reset cascade into an LTL formula of depth linear and length singly exponential in the number of cascade configurations. Combining this with Eilenberg's Holonomy translation of a semigroup into a cascade [14, Corollary II.7.2] and Pnueli and Maler's adapt-

---

[4] See Remark 1 on whether the upper bound in [26] is single or double exponential.

ation of it to automata [26, Theorem 3] (see Remark 1), we obtain a translation of counter-free deterministic ω-regular automata into LTL formulas of double exponential depth and triple exponential length. Our construction preserves the acceptance condition of the automaton in the sense that it turns a Büchi-looping, coBüchi-looping, weak, Büchi or coBüchi automaton into a formula that belongs to the matching class of the syntactic future hierarchy (see Definition 1 and [8]).

Related work

Finite words. While LTL is usually interpreted over infinite words, it also admits finite-word semantics that coincide with the finite word version of the other equivalent formalisms. The equivalence between FO and star-free languages on finite words is due to McNaughton and Papert [31]. Cohen, Perrin and Pin [10] used the Krohn-Rhodes decomposition to characterise the expressive power of LTL with only **X** and **F** (eventually), but do not provide bounds on the size trade-off between the different models. Wilke [42] gives a double-exponential translation from counter-free DFA to LTL. More recently, Bojańczyk provided an algebraically flavoured adaptation of Wilke's proof [2, Section 2.2.2].

Infinite words. With substantial effort over several decades, the above techniques have been extended to infinite words using intricate tools with opaque complexities. Ladner [22] and Thomas [38,39] for example extended the equivalence of star-free regular expressions and FO to infinite words, while the ω-extension of the equivalence with aperiodic languages is due to Perrin [34]. The correspondence with LTL is due to Kamp [18] and Gabbay, Pnueli, Shelah and Stavi [16]. Diekert and Gastin's survey [13] provides an algebraic translation into LTL via ω-monoids while Cohen-Chesnot gives a direct algebraic proof of the equivalence of star-free ω-regular expressions and LTL [11]. Wilke takes an automata-theoretic approach, using backward deterministic automata [43,44]. However, none of the above address the complexity of the transformations. Zuck's dissertation [46] gives a translation of star-free regular expressions into LTL, with at least non-elementary complexity. Subsequently, Chang, Mana and Pnueli [8] use Zuck's results to show that the levels of their hierarchy of future temporal properties coincide with syntactic fragments of LTL. Sickert and Esparza [37] gave an exponential translation of any LTL formula into level Δ_2 of this hierarchy.

2   Preliminaries

Languages. An alphabet Σ, of size $|\Sigma|$, is a finite set of letters. Σ^*, Σ^+, and Σ^ω denote the sets of finite, nonempty finite, and infinite words over Σ, respectively. A language of finite or infinite words is a subset of Σ^* or Σ^ω, respectively. We write $[i..j]$ and $[i..j)$, with integers $i \leq j$, for the sets $\{i, i+1, \ldots, j\}$ and $\{i, i+1, \ldots, j-1\}$, respectively. For a word $w = \sigma_0 \cdot \sigma_1 \cdots$, we write $|w|$ for its length (∞ if w is infinite), $w[i]$ for σ_i, $w_{[i..j]}$ and $w_{[i..j)}$ for its corresponding infixes ($w_{[i..i)}$ is the empty word), and $w_{[i..]}$ for its (finite or infinite) suffix $\sigma_i \cdot \sigma_{i+1} \cdots$.

Linear Temporal Logic (LTL). Let AP be a finite set of atomic propositions. LTL formulas are constructed from the constant **true**, atomic propositions $a \in AP$, the connectives $\neg$ (negation) and $\wedge$ (and), and the temporal operators $\mathbf{U}$ (until) and $\mathbf{X}$ (next). Their semantics are given by a satisfiability relation $\models$ between finite or infinite words $w \in (2^{AP})^+ \cup (2^{AP})^\omega$, and a formula φ inductively as follows:

$$
\begin{aligned}
&w \models \mathbf{true} && w \models a && \text{iff } a \in w[0] \\
&w \models \neg\varphi && \text{iff } w \not\models \varphi && w \models \varphi \wedge \psi && \text{iff } w \models \varphi \text{ and } w \models \psi \\
&w \models \mathbf{X}\varphi && \text{iff } |w| > 1 \text{ and } w_{[1..]} \models \varphi \\
&w \models \varphi\mathbf{U}\psi && \text{iff } \exists i \in [0..|w|). \ w_{[i..]} \models \psi \text{ and } \forall j \in [0..i). \ w_{[j..]} \models \varphi
\end{aligned}
$$

We also use the common shortcuts $\mathbf{false} \coloneqq \neg\mathbf{true}$, $\varphi \vee \psi \coloneqq \neg((\neg\varphi) \wedge (\neg\psi))$, $\mathbf{F}\varphi \coloneqq \mathbf{true}\mathbf{U}\varphi$, $\mathbf{G}\varphi \coloneqq \neg\mathbf{F}\neg\varphi$, and $\psi_1\mathbf{R}\psi_2 \coloneqq \neg(\neg\psi_1)\mathbf{U}(\neg\psi_2)$. The language of finite words of φ is $L^{<\omega}(\varphi) \coloneqq \{w \in (2^{AP})^+ \mid w \models \varphi\}$, and the language of infinite words is $L(\varphi) \coloneqq \{w \in (2^{AP})^\omega \mid w \models \varphi\}$. Note that we omit the "$< \omega$" superscript if it is clear from the context which set is used. The *length* $|\varphi|$ of φ is the number of nodes in its syntax tree, the *size* of φ is the number of nodes in a DAG representing this syntax tree, and its *temporal nesting depth*, denoted by $\mathsf{depth}(\varphi)$, is defined by: $\mathsf{depth}(\mathbf{true}) = 0$; $\mathsf{depth}(a) = 0$ for an atomic proposition $a \in AP$; $\mathsf{depth}(\neg\psi) = \mathsf{depth}(\psi)$; $\mathsf{depth}(\psi_1 \wedge \psi_2) = \max(\mathsf{depth}(\psi_1), \mathsf{depth}(\psi_2))$; $\mathsf{depth}(\mathbf{X}\psi) = \mathsf{depth}(\psi) + 1$; and $\mathsf{depth}(\psi_1\mathbf{U}\psi_2) = \max(\mathsf{depth}(\psi_1), \mathsf{depth}(\psi_2)) + 1$. Chang, Manna, and Pnueli define in [8] a syntactic hierarchy for LTL formulas (over infinite words):

Definition 1 (LTL Syntactic future hierarchy [8] [5]).

- $\Sigma_0 = \Pi_0 = \Delta_0$ *is the least set containing all atomic propositions and their negations, and is closed under the application of conjunction and disjunction.*
- Σ_{i+1} *is the least set containing Π_i and negated formulas of Π_{i+1} closed under the application of conjunction, disjunction, and the $\mathbf{X}$ and $\mathbf{U}$ operators.*
- Π_{i+1} *is the least set containing Σ_i and negated formulas of Σ_{i+1} closed under the application of conjunction, disjunction, and the $\mathbf{X}$ and $\mathbf{R}$ operators.*
- Δ_{i+1} *is the least set containing Σ_{i+1} and Π_{i+1} that is closed under the application of conjunction, disjunction, and negation.*

Σ_1 is referred to as *syntactic co-safety* formulas, Π_1 as *syntactic safety* formulas.

Automata. A *deterministic semiautomaton* is a tuple $\mathcal{D} = (\Sigma, Q, \delta)$, where Σ is an alphabet; Q is a finite nonempty set of states; and $\delta \colon Q \times \Sigma \to Q$ is a transition function and we extend it to finite words in the usual way. A *path* of $\mathcal{D}$ on a word $w = \sigma_0 \cdot \sigma_1 \cdots$ is a sequence of states $q_0, q_1, \ldots$, such that for every $i < |w|$, we have $\delta(q_i, \sigma_i) = q_{i+1}$.

It is a *reset* semiautomaton if for every letter $\sigma \in \Sigma$, either i) for every state $q \in Q$ we have $\delta(q, \sigma) = q$, or ii) there exists a state $q' \in Q$, such that for every state $q \in Q$ we have $\delta(q, \sigma) = q'$.

---

[5] This extends [6,37] with negation, which can be removed via negation normal form.

It is *counter free* if for every state $q \in Q$, finite word $u \in \Sigma^+$, and number $n \in \mathbb{N} \setminus \{0\}$, there is a self loop of q on u^n iff there is a self loop of q on u.

A *deterministic automaton* is a tuple $\mathcal{D} = (\Sigma, Q, \iota, \delta, \alpha)$, where (Σ, Q, δ) is a deterministic semiautomaton, $\iota \in Q$ is an initial state; and α is some acceptance condition, as detailed below. A run of $\mathcal{D}$ on a word w is a path of $\mathcal{D}$ on w that starts in ι. It is a reset or counter-free automaton if its semiautomaton is.

The *acceptance condition* of an automaton on finite words is a set $F \subseteq Q$; a run is accepting if it ends in a state $q \in F$. The *acceptance condition* of an ω-*regular automaton*, on infinite words, is defined with respect to the set $inf(r)$ of states visited infinitely often along a run r. We define below several acceptance conditions that we use in the sequel; for other conditions, see, for example, [3].

The *Muller* condition is a set $\alpha = \{M_1, \ldots, M_k\}$ of sets $M_i \subseteq Q$ of states, and a run r is accepting if there exists a set M_i, such that $M_i = inf(r)$. The *Rabin* condition is a set $\alpha = \{(G_1, B_1), \ldots, (G_k, B_k)\}$ of pairs of sets of states, and r is accepting if there exists a pair (G_i, B_i), such that $G_i \cap inf(r) \neq \emptyset$ and $B_i \cap inf(r) = \emptyset$. The *Büchi* (resp. *coBüchi*) condition is a set $\alpha \subseteq Q$ of states, and r is accepting if $\alpha \cap inf(r) \neq \emptyset$ (resp. $\alpha \cap inf(r) = \emptyset$). A *weak* automaton is a Büchi automaton, in which every strongly connected component (SCC) contains only states in α or only states out of α. A *looping* automaton is a Büchi or coBüchi automaton, where all states are in α, except for a single sink state.

Deterministic automata of the above types correspond to the hierarchy of temporal properties [28]: Looping-Büchi, looping-coBüchi, weak, Büchi, coBüchi, and Rabin/Muller deterministic automata define respectively safety, guarantee (co-safety), obligation, recurrence, persistence, and reactivity languages. If the language is also LTL-definable, then there exists an equivalent LTL formula in Π_1, Σ_1, Δ_1, Π_2, Σ_2, and Δ_2, respectively [8]. Every deterministic ω-regular automaton is equivalent to deterministic Muller and Rabin automata, where the Muller (but not always Rabin) one can be defined on the same semiautomaton.

Nondeterministic and *alternating* automata (to which we only refer in Section 3, on finite words over a unary alphabet) extend deterministic automata by having a transition function $\delta \colon Q \times \Sigma \to 2^Q$ and $\delta \colon Q \times \Sigma \to$ (positive Boolean formulas over Q), respectively. (See, for example, [7] for formal definitions.)

3   Unary Alphabet

Kupferman, Ta-Shma and Vardi [20] compared the succinctness of different automata models when *counting*, that is, recognising the singleton language $\{a^k\}$ for some k over the singleton alphabet $\{a\}$. For the succinctness gap between automata and LTL, we study the task of recognising arbitrary languages over the unary alphabet, which can be seen as sets of integers, rather than a single integer.

For a unary alphabet, since there is only one infinite word, only languages on finite words are interesting. We thus consider LTL formulas over (no) atomic propositions $AP = \emptyset$, and automata on finite unary words over the corresponding alphabet $\Sigma = 2^{AP} = \{\emptyset\}$, where we use the shorthand $a = \emptyset$. The size of a deterministic automaton is the number of its states, of a nondeterministic

automaton the number of its transitions, and of an alternating automaton the number of subformulas in its transition function.

We show that the size blow-up involved in translating deterministic, non-deterministic, and alternating automata to LTL, when possible, is linear, quadratic, and exponential, respectively.

In our analysis, we shall use the following folklore theorem, which extends Wolper's Theorem [45].

Proposition 1 (Extended Wolper's theorem, Folklore). *Consider an LTL formula φ with $\mathsf{depth}(\varphi) = n$ over the atomic propositions AP, and let $\Sigma = 2^{AP}$. Then for every words $u \in \Sigma^*$, $v \in \Sigma^+$ and $t \in \Sigma^\omega$, and numbers $i, j > n$, φ has the same truth value on the words $(uv^i t)$ and $(uv^j t)$.*

We use this to establish that unary LTL describes only finite and co-finite properties, and that there is a tight relation between the depth of LTL formulas and the length of words above which they are all in or all out of the language.

Proposition 2. *Given an LTL formula φ with $\mathsf{depth}(\varphi) = n$ on finite words over the unary alphabet $\{a\}$, $a^i \in L(\varphi)$ for all $i > n$ or $a^i \notin L(\varphi)$ for all $i > n$.*

Proposition 3. *Consider a language $L \subseteq \{a\}^+$ that agrees on all words of length over n, that is, has the same truth value on all such words. Then there is an LTL formula of size in $O(n)$ with language L.*

We now establish the trade-off between LTL and alternating automata (AFA) over unary alphabets. AFA are closed under (linear) complementation, so we use a pumping argument to bound the length after which all words have the same truth value, giving an upper bound on the LTL formula.

Lemma 1. *Every alternating automaton with n states that recognises an LTL-expressible language $L \subseteq \{a\}^+$ is equivalent to an LTL formula of size in $O(2^n)$.*

We show next that this upper bound is tight. Consider the language $\{a^{2^{n-1}}\}$, which, according to Proposition 2, is only recognised by LTL formulas of size at least 2^{n-1}. It is recognised by a weak alternating automaton with $2n$ states and size in $O(n)$, using an automaton based on Leiss's construction [23]. Intuitively, the alternating automaton represents an n-bit up-counter with two states for each bit, one for 1 and one for 0 (see Fig. 1), where the universal transitions enforce that nondeterministic transitions correctly update the counter.

Lemma 2 (Adaptation of [23, proof of Theorem 1]). *For every $n \in \mathbb{N} \setminus \{0\}$, there is a weak alternating automaton with $2n$ states and transition function of size in $O(n)$ recognising the language $\{a^{2^{n-1}}\}$.*

We continue to nondeterministic automata (NFAs), for which the arguments are more involved as they do not allow for linear complementation.
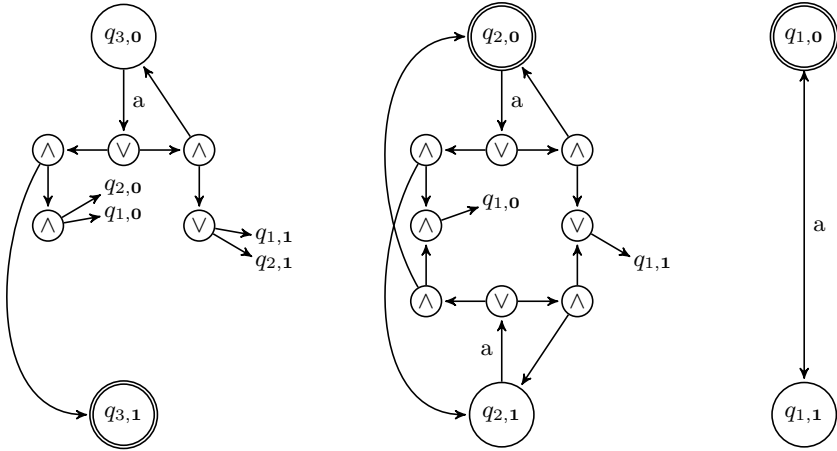
Figure 1. An alternating automaton of size in $O(n)$ recognising $\{a^{2^{n-1}}\}$; here with $n = 3$, where the initial configuration is $q_{1,\mathbf{0}} \wedge q_{2,\mathbf{0}} \wedge q_{3,\mathbf{0}}$.

Lemma 3. *Every nondeterministic automaton with n states recognising an LTL-expressible language $L \subseteq \{a\}^+$ is equivalent to an LTL formula of size in $O(n^2)$.*

Proof sketch. For finite L, by a pumping argument, $\mathcal{A}$ only accepts words up to length n, and by Proposition 3 we are done. We now consider a co-finite L.

We use 2-way deterministic automata, which are deterministic automata that process words of the form $\vdash w \dashv$, where $\vdash$ and $\dashv$ are start- and end-of-word markers respectively, and where transitions specify whether to read the letter to the right or to the left of the current position. They accept by reaching an end state, and reject by reaching a rejecting state or by failing to terminate [17], and every unary NFA $\mathcal{A}$ can be turned into a 2-way DFA $\mathcal{D}$ of size $O(n^2)$ [9].

We construct from an NFA $\mathcal{A}$ a 2-way DFA $\mathcal{D}$, and then a 2-way DFA $\mathcal{D}'$ of the same size that recognises $a^* \setminus \{a^k\}$, where a^k is the longest word not in L. We use the fact that a 2-way DFA of size m can be complemented into one of size $4m$ [17] to complement $\mathcal{D}'$ into $\mathcal{D}''$ that recognises $\{a^k\}$ and must therefore be of size at least $k + 2$ [1], so k, and by Proposition 2, an LTL formula for L, is in $O(n^2)$. $\square$

We now show that this upper bound is tight. The previous lower bound ideas do not work with nondeterminism, since we need n states to recognise $\{a^n\}$ [20]. Yet, we need not count *exactly* to n for achieving a lower bound. We can use a variant of a language used in [4, pages 10–11]: For every positive integer k, define the set of positive integers $S_k = \{m > 0 \mid \exists i, j \in \mathbb{N}. \ m = ik + j(k+1)\}$, and the language $V_k = \{a^m \mid m \in S_k\} \subseteq \{a\}^*$.

Proposition 4 (Folklore, [4, Theorem 3]). *For every $k \in \mathbb{N}$ the number $k^2 - k - 1$ is the maximal number not in S_k.*

Proposition 5 ([4, proof of Theorem 4]). *For every $n \in \mathbb{N}$, there is an NFA of size in $O(n)$ recognising a co-finite language $L \subseteq \{a\}^*$, such that a^{k^2-k-1} is not in L, while for every $t \geq k^2 - k$, we have that $a^t \in L$.*

Theorem 1. *The size blow-up involved in translating deterministic, nondeterministic, and alternating automata on finite unary words to LTL, when possible, is $\Theta(n)$, $\Theta(n^2)$, and $\Theta(2^n)$, respectively.*

4   General Alphabet

In this section we consider the more challenging task of turning counter-free ω-regular automata over arbitrary alphabets into LTL. We use the fact that these automata can be turned into reset cascade automata (Krohn-Rhodes-Holonomy decomposition), which we describe in Section 4.1. Our technical contribution is then the translation of reset cascade automata into LTL.

In brief, we build, in Section 4.2, a *parameterised LTL formula* that is satisfied by a word w iff the run of the cascade on w, starting in the parameter configuration S, reaches a parameter configuration T, such that the remaining suffix of w satisfies a parameter LTL formula τ. We then use this formula, in Section 4.4, to describe the automaton's acceptance condition.

When encoding the behavior of a cascade by an LTL formula, we need to overcome two major challenges: First, the cascade is a formalism that looks at the *past*, namely at the word read so far, to determine the next configuration, while an LTL formula obtains its value only from the future. Second, the cascade has an internal state, while an LTL formula does not. Our reachability formulas are therefore quite involved, built inductively over the number of levels in the cascade, and implicitly allowing to track the internal configuration of the cascade.

In Section 4.3 we analyse the length and depth of the resulting formulas.

4.1   Cascaded Automata

Cascades. A cascaded semiautomaton (analogous to the algebraic wreath product) over an alphabet Σ is a semiautomaton that can be described as a sequence of simple semiautomata, such that the alphabet of each of them is Σ together with the current state of each of the preceding semiautomata in the sequence. It is a reset cascade if it is a sequence of reset semiautomata. Formally, a *cascaded semiautomaton*, or just *cascade*, over alphabet Σ with n levels is a tuple $\mathcal{A} = \langle \Sigma, \mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n \rangle$, such that $\mathcal{A}_i = (\Sigma_i, Q_i, \delta_i)$ is a semiautomaton for each level i, where $\Sigma_i = \Sigma \times Q_1 \times \cdots \times Q_{i-1}$. (So $\Sigma_1 = \Sigma$, $\Sigma_2 = \Sigma \times Q_1$, etc.). It is a *reset cascade* if all $\mathcal{A}_i$'s are reset semiautomata.

An i-configuration S of $\mathcal{A}$ is a tuple $\langle q_1, q_2, \ldots, q_i \rangle \in Q_1 \times \cdots \times Q_i$. If $q_{i+1} \in Q_{i+1}$ is a state of level $i + 1$, we write $\langle S, q_{i+1} \rangle$ for the $(i + 1)$-configuration $\langle q_1, \ldots, q_i, q_{i+1} \rangle$. Note that the 0-configuration is the empty tuple $\langle \rangle$. Further, we derive the transition relation for configurations by point-wise application of the respective δ_i's. We define $\delta_{\leq i}(\langle q_1, q_2, \ldots q_i \rangle, \sigma)$ as $\langle \delta_1(q_1, \langle \sigma \rangle), \delta_2(q_2, \langle \sigma, q_1 \rangle), \ldots \rangle$.

Note that we will omit the "$\leq i$"-subscript if it is clear from context, and by just writing "configuration", we mean an n-configuration.

Notice that $\mathcal{A}$ describes a standard semiautomaton $\mathcal{D}_\mathcal{A}$ over Σ, whose states are the configurations of $\mathcal{A}$ of level n, and its transition function is $\delta_{\leq n}$. If there are up to j states in each level of $\mathcal{A}$, there are up to j^n states in $\mathcal{D}_\mathcal{A}$. Observe that when $\mathcal{A}$ is a reset cascade, it can be translated to an equivalent reset cascade with up to $n \log j$ levels, and 2 states in each level [14, Ex. I.10.2].

For a state $q \in Q_i$ of level i of a reset cascade, we denote by $\mathsf{Enter}(q)$, $\mathsf{Stay}(q)$, and $\mathsf{Leave}(q) \subseteq \Sigma \times Q_1 \times \cdots \times Q_{i-1}$ the sets of (combined) letters that enter q, stay in it, and leave it, respectively. These are sets of pairs $\langle \sigma, S \rangle$, where S is an $(i-1)$-configuration and $\sigma \in \Sigma$. Notice that $\mathsf{Enter}(q) \subseteq \mathsf{Stay}(q)$, and that $\mathsf{Leave}(q)$ is the complement of $\mathsf{Stay}(q)$ (w.r.t. the relevant (combined) letters).

A semiautomaton (Σ, Q, δ) is *homomorphic* to a cascade $\langle \Sigma, \mathcal{A}_1, \ldots, \mathcal{A}_n \rangle$ if there exists a partial surjective function $\varphi \colon Q_1 \times \cdots \times Q_n \to Q$, such that for every $\sigma \in \Sigma$ and $S \in Q_1 \times \cdots \times Q_n$, we have $\delta(\varphi(S), \sigma) = \varphi(\delta_{\leq n}(S, \sigma))$.

Proposition 6 (Part of the Krohn-Rhodes-Holonomy Decomposition [14, Corollary II.7.2], [26, Theorem 3]). *Every counter-free deterministic semiautomaton $\mathcal{D}$ with n states is homomorphic to a reset cascade $\mathcal{A}$ with up to 2^n levels and 2^n states in each level.*

Remark 1. The Krohn-Rhodes and Holonomy decomposition theorems consider also more general cascades and give results with respect to arbitrary semiautomata. The Holonomy decomposition in [14], as opposed to many other proofs of the Krohn-Rhodes decomposition, guarantees up to 2^n levels with up to 2^n states in each level. Yet, it shows that $\mathcal{A}$ *covers* $\mathcal{D}$, allowing $\mathcal{A}$ to operate over an alphabet different from that of $\mathcal{D}$. In [26,27,25], the algebraic proof of [14] is translated to an automata-theoretic one, providing the stated homomorphism. It is also stated in [26, Theorem 3.1], [27, Corollary 20], and [25, Corollary 2] that the number of configurations in $\mathcal{A}$ is singly exponential in n, but to the best of our understanding they do not provide an explicit proof for it.

Cascades with acceptance conditions. As a cascade $\mathcal{A}$ describes a standard semiautomaton (whose states are the configurations of $\mathcal{A}$), we can add to it an initial configuration and an acceptance condition to make it a standard deterministic automaton. We show below that the homomorphism between an automaton and a cascade can be extended to also transfer the same acceptance condition.

Proposition 7. *Let $\mathcal{D}$ be a deterministic Büchi, coBüchi or Rabin automaton, with a semiautomaton homomorphic to a cascade $\mathcal{A}$. There is respectively a deterministic Büchi, coBüchi or Rabin automaton $\mathcal{D}'$ equivalent to $\mathcal{D}$ with semiautomaton $\mathcal{A}$. For Rabin, $\mathcal{D}$ and $\mathcal{D}'$ have the same number of acceptance pairs.*

Proposition 8. *Consider a deterministic Muller automaton $\mathcal{D}$ with n states, whose semiautomaton is homomorphic to a reset cascade $\mathcal{A}$ with m configurations. Then there is a deterministic Muller automaton $\mathcal{D}'$ equivalent to $\mathcal{D}$, whose semiautomaton is $\mathcal{A}$ and its Muller condition has up to $2^{O(mn)}$ acceptance sets.*

4.2   Encoding Reachability within Reset Cascades by LTL Formulas

For the rest of this section, let us fix a set of atomic propositions AP, an alphabet $\Sigma = 2^{AP}$, and a reset cascade $\mathcal{A} = \langle \Sigma, \mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n \rangle$.

The main reachability formula. For every level i of $\mathcal{A}$, three configurations S, B and T of level i, and two LTL formulas β and τ, we will define the LTL formula $S \underset{B(\beta)}{\rightsquigarrow} T(\tau)$ with the intended semantics that it holds on a word $w \in \Sigma^\omega$ iff $\mathcal{A}$ goes from the 'starting' configuration S to the 'target' configuration T along some prefix u of w, such that the suffix of w after u satisfies τ and the path along u avoids the 'bad' configuration B with a suffix satisfying β.

Auxiliary reachability formulas. We will formally define the main reachability formula by induction on the level i of the involved configurations, and using four auxiliary formulas, whose intended semantics is described in Table 1. These formulas distinguish between the case that the top-level state is unchanged along the reachability path, denoted with a solid arrow $\longrightarrow$, and the case that it is changed, denoted by a dashed arrow $\dashrightarrow$. They also have dual, weak, versions.

Observe that intuitively $S \underset{B(\beta)}{\rightsquigarrow} T(\tau)$ is an extended *Until* operator, while its dual $S \overset{\text{weak}}{\underset{B(\beta)}{\rightsquigarrow}} T(\tau) = \neg(S \underset{T(\tau)}{\rightsquigarrow} B(\beta))$ is an extended *Weak until* (or *Release*) operator. We build the formulas so that for appropriate choices of β and τ, the (strong) reachability formulas 1, 3, and 5 (as numbered in Table 1) are syntactic co-safety and the weak formulas 2 and 4 are syntactic safety formulas.

Formulas 1 and 2. The main formula is simply defined as the union of two auxiliary formulas, corresponding to whether or not the top-level state changes, and its weak version is defined to be its dual.

$$S \underset{B(\beta)}{\rightsquigarrow} T(\tau) := \begin{cases} (\neg\beta)\mathbf{U}\tau & \text{if } S = \langle\rangle \\ S \xrightarrow{B(\beta)} T(\tau) \vee S \underset{B(\beta)}{\dashrightarrow} T(\tau) & \text{otherwise.} \end{cases}$$

$$S \overset{\text{weak}}{\underset{B(\beta)}{\rightsquigarrow}} T(\tau) := \neg\left(S \underset{T(\tau)}{\rightsquigarrow} B(\beta) \right)$$

Formula 3. Since the formula should ensure that the top-level state s is unchanged, we first distinguish between four cases, depending on which of the source configuration $\langle S, s\rangle$, bad configuration $\langle B, b\rangle$, and target configuration $\langle T, t\rangle$ are equal. The definitions of the four cases only differ in whether or not each of β and τ are satisfied in the first position of the word.

We define them using an intermediate common formula that is indifferent to the first position, which we mark by "> 0" on top of the arrow. We then define the "> 0" formula by using the main reachability formula with respect to a lower level, namely with respect to the configurations S and T instead of $\langle S, s\rangle$ and $\langle T, t\rangle$, and having corresponding disjunctions and conjunctions on all the combined letters of the top level that belong to $\mathsf{Stay}(s)$ and $\mathsf{Leave}(s)$.

Reachability formula φ	Intended semantics Intuitively: Reading a word w from the configuration S or $\langle S, s \rangle$ Formally: $w \models \varphi \iff$
1. $\quad S \xrightsquigarrow[\overline{B(\beta)}]{} T(\tau)$	not reaching $B(\beta)$ until reaching $T(\tau)$. $\exists i \geq 0.\ \delta(S, w_{[0..i)}) = T \wedge w_{[i..]} \models \tau$ $\quad \wedge\ (\forall j \in [0..i).\ \delta(S, w_{[0..j)}) \neq B \vee w_{[j..]} \not\models \beta)$
2. $\quad S \xrightsquigarrow[\overline{B(\beta)}]{\text{weak}} T(\tau)$	reaching $T(\tau)$ releases not reaching $B(\beta)$. $\forall i \geq 0.\ (\delta(S, w_{[0..i)}) = B \wedge w_{[i..]} \models \beta)$ $\quad \rightarrow\ (\exists j \in [0..i].\ \delta(S, w_{[0..j)}) = T \wedge w_{[j..]} \models \tau)$
3. $\langle S, s \rangle \xrightarrow[\overline{\langle B, b \rangle(\beta)}]{} \langle T, t \rangle(\tau)$	not reaching $\langle B, b \rangle(\beta)$ until reaching $\langle T, t \rangle(\tau)$, while staying in s. $\exists i \geq 0.\ \delta(\langle S, s \rangle, w_{[0..i)}) = \langle T, t \rangle \wedge w_{[i..]} \models \tau$ $\quad \wedge\ (\forall j \in [0..i).\ \delta(\langle S, s \rangle, w_{[0..j)}) \neq \langle B, b \rangle \vee w_{[j..]} \not\models \beta)$ $\quad \wedge\ (\forall j \in [0..i).\ \langle w[j], \delta(S, w_{[0..j)}) \rangle \in \mathsf{Stay}(s))$
4. $\langle S, s \rangle \xrightarrow[\overline{\langle B, b \rangle(\beta)}]{\text{weak}} \langle T, t \rangle(\tau)$	reaching $\langle T, t \rangle(\tau)$ releases not (reaching $\langle B, b \rangle(\beta)$ or leaving s). $\forall i \geq 0.\ \big((\delta(\langle S, s \rangle, w_{[0..i)}) = \langle B, b \rangle \wedge w_{[i..]} \models \beta)$ $\quad \vee\ (i > 0 \wedge \langle w[i-1], \delta(S, w_{[0..i-1)}) \rangle \in \mathsf{Leave}(s))\big)$ $\quad \rightarrow\ (\exists j \in [0..i].\ \delta(\langle S, s \rangle, w_{[0..j)}) = \langle T, t \rangle \wedge w_{[j..]} \models \tau)$
5. $\langle S, s \rangle \dashrightarrow[\overline{\langle B, b \rangle(\beta)}]{} \langle T, t \rangle(\tau)$	not reaching $\langle B, b \rangle(\beta)$ until reaching $\langle T, t \rangle(\tau)$ and leaving s. $\exists i_1, i_2 \geq 0.\ \delta(\langle S, s \rangle, w_{[0..i_1)}) = \langle T, t \rangle \wedge w_{[i_1..]} \models \tau$ $\quad \wedge\ (\exists j_1 \in [0..i_1).\ \langle w[j_1], \delta(S, w_{[0..j_1)}) \rangle \in \mathsf{Enter}(t))$ $\quad \wedge\ \langle w[i_2], \delta(S, w_{[0..i_2)}) \rangle \in \mathsf{Leave}(s)$ $\quad \wedge\ (\forall j_2 \in [0..\max(i_1-1, i_2)].\ \delta(\langle S, s \rangle, w_{[0..j_2)}) \neq \langle B, b \rangle$ $\qquad\qquad \vee\ w_{[j_2..]} \not\models \beta)$

Table 1. The intended semantics of reachability formulas. Orange subformulas show the difference between the auxiliary formulas and the first or second (main) formula.

$$\langle S, s \rangle \xrightarrow[\overline{\langle B, b \rangle(\beta)}]{} \langle T, t \rangle(\tau) :=$$

$$\begin{cases} \langle S, s \rangle \xrightarrow[\overline{\langle B, b \rangle(\beta)}]{>0} \langle T, t \rangle(\tau) & \text{if } \langle S, s \rangle \neq \langle B, b \rangle \text{ and } \langle S, s \rangle \neq \langle T, t \rangle \\[4pt] \langle S, s \rangle \xrightarrow[\overline{\langle B, b \rangle(\beta)}]{>0} \langle T, t \rangle(\tau) \vee \tau & \text{if } \langle S, s \rangle \neq \langle B, b \rangle \text{ and } \langle S, s \rangle = \langle T, t \rangle \\[4pt] \langle S, s \rangle \xrightarrow[\overline{\langle B, b \rangle(\beta)}]{>0} \langle T, t \rangle(\tau) \wedge \neg\beta & \text{if } \langle S, s \rangle = \langle B, b \rangle \text{ and } \langle S, s \rangle \neq \langle T, t \rangle \\[4pt] \left(\langle S, s \rangle \xrightarrow[\overline{\langle B, b \rangle(\beta)}]{>0} \langle T, t \rangle(\tau) \wedge \neg\beta\right) \vee \tau & \text{if } \langle S, s \rangle = \langle B, b \rangle \text{ and } \langle S, s \rangle = \langle T, t \rangle \end{cases}$$

where $\langle S, s \rangle \xrightarrow[\overline{\langle B, b \rangle(\beta)}]{>0} \langle T, t \rangle(\tau) := \bigvee_{\substack{\langle \sigma, T' \rangle \in \mathsf{Stay}(s) \\ \text{s.t. } \langle T', s \rangle \xrightarrow{\sigma} \langle T, t \rangle}} \left(S \xrightsquigarrow[\overline{S(\mathbf{false})}]{} T'(\sigma \wedge \mathbf{X}\tau)\right)$

$$\wedge \bigwedge_{\langle \eta, L\rangle \in \mathsf{Leave}(s)} S \overset{}{\underset{\overline{L(\eta)}}{\rightsquigarrow}} T' \left(\sigma \wedge \mathbf{X}\tau\right) \quad \wedge \bigwedge_{\substack{\langle \rho, B'\rangle \in \mathsf{Stay}(s) \\ \text{s.t. } \langle B', s\rangle \overset{\rho}{\rightarrow} \langle B, b\rangle}} S \overset{}{\underset{\overline{B'(\rho \wedge \mathbf{X}\beta)}}{\rightsquigarrow}} T' \left(\sigma \wedge \mathbf{X}\tau\right) \Bigg)$$

Formula 4. Its intended semantics is also that the top-level state s is unchanged, but we weaken Formula 3 by not enforcing that the target configuration $\langle T, t\rangle$ is reached and τ is satisfied. Thus as long as the top-level state s stays unchanged and the bad configuration $\langle B, b\rangle$ is not reached while satisfying β, Formula 4 is also satisfied. Note that since both Formula 3 and Formula 4 need to ensure that the top-level state s is unchanged they cannot simply be defined as the dual of each other. However, they share the same construction principle:

$$\langle S, s\rangle \xrightarrow[\overline{\langle B,b\rangle(\beta)}]{\text{weak}} \langle T, t\rangle \left(\tau\right) :=$$

$$\begin{cases} \langle S, s\rangle \xrightarrow[\overline{\langle B,b\rangle(\beta)}]{\text{weak},>0} \langle T, t\rangle \left(\tau\right) & \text{if } \langle S, s\rangle \neq \langle B, b\rangle \text{ and } \langle S, s\rangle \neq \langle T, t\rangle \\ \langle S, s\rangle \xrightarrow[\overline{\langle B,b\rangle(\beta)}]{\text{weak},>0} \langle T, t\rangle \left(\tau\right) \vee \tau & \text{if } \langle S, s\rangle \neq \langle B, b\rangle \text{ and } \langle S, s\rangle = \langle T, t\rangle \\ \langle S, s\rangle \xrightarrow[\overline{\langle B,b\rangle(\beta)}]{\text{weak},>0} \langle T, t\rangle \left(\tau\right) \wedge \neg\beta & \text{if } \langle S, s\rangle = \langle B, b\rangle \text{ and } \langle S, s\rangle \neq \langle T, t\rangle \\ \left(\langle S, s\rangle \xrightarrow[\overline{\langle B,b\rangle(\beta)}]{\text{weak},>0} \langle T, t\rangle \left(\tau\right) \vee \tau\right) \wedge \neg\beta & \text{if } \langle S, s\rangle = \langle B, b\rangle \text{ and } \langle S, s\rangle = \langle T, t\rangle \end{cases}$$

where

$$\langle S, s\rangle \xrightarrow[\overline{\langle B,b\rangle(\beta)}]{\text{weak},>0} \langle T, t\rangle \left(\tau\right) :=$$

$$\bigvee_{\substack{\langle \sigma, T'\rangle \in \mathsf{Stay}(s) \\ \text{s.t. } \langle T', s\rangle \overset{\sigma}{\rightarrow} \langle T, t\rangle}} \left(\bigwedge_{\langle \eta, L\rangle \in \mathsf{Leave}(s)} S \overset{\text{weak}}{\underset{\overline{L(\eta)}}{\rightsquigarrow}} T' \left(\sigma \wedge \mathbf{X}\tau\right) \wedge \bigwedge_{\substack{\langle \rho, B'\rangle \in \mathsf{Stay}(s) \\ \text{s.t. } \langle B', s\rangle \overset{\rho}{\rightarrow} \langle B, b\rangle}} S \overset{\text{weak}}{\underset{\overline{B'(\rho \wedge \mathbf{X}\beta)}}{\rightsquigarrow}} T' \left(\sigma \wedge \mathbf{X}\tau\right) \right) \quad (1)$$

$$\bigvee \left(\bigwedge_{\langle \eta, L\rangle \in \mathsf{Leave}(s)} S \overset{\text{weak}}{\underset{\overline{L(\eta)}}{\rightsquigarrow}} S \left(\mathbf{false}\right) \quad \wedge \bigwedge_{\substack{\langle \rho, B'\rangle \in \mathsf{Stay}(s) \\ \text{s.t. } \langle B', s\rangle \overset{\rho}{\rightarrow} \langle B, b\rangle}} S \overset{\text{weak}}{\underset{\overline{B'(\rho \wedge \mathbf{X}\beta)}}{\rightsquigarrow}} S \left(\mathbf{false}\right) \right) \quad (2)$$

Formula 5. The definition of the last reachability formula is the most challenging, since the top-level state changes ($s \neq t$), which prevents the direct usage of lower level configurations.

Intuitively, before reaching the target configuration $\langle T, t\rangle$, the run must see a combined letter $\langle \sigma, T'\rangle \in \mathsf{Enter}(t)$, after which the top-level state t is preserved and the bad situation $\langle B, b\rangle(\beta)$ is avoided. This is line (1) of the definition.

The run must also not see $\langle B, b\rangle(\beta)$ before reaching T', which is handled in line (2), whose difference from line (1) is the additional constraint on the path from S to T'. (Line (1) is required for the case that $\mathsf{Enter}(b)$ is empty.) We use Formula 4 for that constraint, rather than Formula 3 which could also be used, in order to ensure that Formula 5 can be a syntactic co-safety formula.

Lastly, line (3) ensures that the top-level state is indeed changed.

$$\langle S, s \rangle \dashrightarrow_{\overline{\langle B,b \rangle (\beta)}} \langle T, t \rangle (\tau) \coloneqq$$

$$\bigvee_{\substack{\langle \sigma, T' \rangle \in \\ \mathsf{Enter}(t)}} \left(S \rightsquigarrow_{\overline{S(\mathbf{false})}} T' \left(\sigma \wedge \mathbf{X} \Big(\delta(\langle T', \cdot \rangle, \sigma) \xrightarrow{\overline{\langle B,b \rangle (\beta)}} \langle T, t \rangle (\tau) \Big) \right) \right) \wedge \tag{1}$$

$$\bigwedge_{\substack{\langle \eta, R \rangle \in \\ \mathsf{Enter}(b)}} S \xrightarrow[\overline{R(\eta \wedge \mathbf{X}(\delta(\langle R, \cdot \rangle, \eta) \xrightarrow[\overline{\langle T,t \rangle (\tau)}]{weak} \langle B,b \rangle (\beta)))}]{} T' \left(\sigma \wedge \mathbf{X} \Big(\delta(\langle T', \cdot \rangle, \sigma) \xrightarrow{\overline{\langle B,b \rangle (\beta)}} \langle T, t \rangle (\tau) \Big) \right) \tag{2}$$

$$\wedge \bigvee_{\substack{\langle \sigma, L \rangle \in \\ \mathsf{Leave}(s)}} \langle S, s \rangle \xrightarrow{\overline{\langle B,b \rangle (\beta)}} \langle L, s \rangle \left(\sigma \wedge \begin{cases} \neg \beta & \text{if } \langle L, s \rangle = \langle B, b \rangle \\ \mathbf{true} & \text{otherwise.} \end{cases} \right) \tag{3}$$

We prove the correctness of the above definitions with respect to the intended meaning of Table 1 by induction on the level of the involved configurations.

Lemma 4. *The intended semantics of Table 1 hold for all infinite words $w \in \Sigma^\omega = (2^{AP})^\omega$, configurations S, B, T of level $m \leq n$, states s, b, t in level $m + 1$ (when $m < n$), and LTL formulas β and τ over AP.*

Using the same induction principle we prove that the reachability formulas stay within certain classes of the syntactic future hierarchy (Definition 1). We use $S \rightsquigarrow_{\overline{\mathcal{B}(X)}} T(Y) \in Z$ as a shorthand for saying that for every formulas $\beta \in X$ and $\tau \in Y$, the formula $S \rightsquigarrow_{\overline{\mathcal{B}(\beta)}} T(\tau)$ is in Z.

Lemma 5. *Let S, B, T be configurations of level $m \leq n$, and let s, b, t be states in level $m + 1$ (when $m < n$). Then for $i \geq 1$ it holds that:*

- $S \rightsquigarrow_{\overline{\mathcal{B}(\Pi_i)}} T(\Sigma_i)$, $\langle S, s \rangle \xrightarrow{\overline{\langle B,b \rangle (\Pi_i)}} \langle T, t \rangle (\Sigma_i)$, $\langle S, s \rangle \dashrightarrow_{\overline{\langle B,b \rangle (\Pi_i)}} \langle T, t \rangle (\Sigma_i) \in \Sigma_i$
- $S \rightsquigarrow_{\overline{\mathcal{B}(\Sigma_i)}}^{weak} T(\Pi_i)$, $\langle S, s \rangle \xrightarrow[\overline{\langle B,b \rangle (\Sigma_i)}]{weak} \langle T, t \rangle (\Pi_i) \in \Pi_i$

4.3   Depth and Length Analysis

We analyze the length and temporal-nesting depth of the LTL reachability formulas defined in Section 4.2. Notice that both measures are of independent interest, as there might be a non-elementary gap between the depth and length of LTL formulas [15, Theorem 6]. Since we provide upper bounds, the bound on the length of formulas obviously gives also a bound on their size.

We consider a reset cascade $\mathcal{A}$ with n levels, as in Section 4.2, and further assume for the length and depth analysis that it has up to n states in each level. (This assumption holds in the reset cascades that result from the Krohn-Rohdes decomposition as per Proposition 6.)

We define for each of the five reachability formulas a *depth function* $\mathsf{D}_x(i, d)$ and a *length function* $\mathsf{L}_x(i, l)$, where x refers to the number of the reachability

formula, to bound the depth and length of the formulas. These depend on the level i of its input configurations S, B and T, and the maximal depth d and length l of its input formulas β and τ. For the main (first) reachability formula, we also use D and L, standing for D_1 and L_1. For example, the length of the first formula $S \underset{\mathcal{B}(\beta)}{\rightsquiggle} T(\tau)$ over configurations S, B and T of level 7 and formulas β and τ of length up to 77 is bounded by the value of $\mathsf{L}_1(7, 77)$.

For simplicity, we consider the LTL representation of an alphabet letter $\sigma \in \Sigma$ to be of length 1, while its actual length is $3 \log_2 |\Sigma|$. This increase is due to the need to encode an alphabet letter $\sigma \in \Sigma = 2^{AP}$ as a conjunction of atomic propositions in AP. The representation length can be multiplied by the total length of the final relevant formula (e.g., a formula equivalent to the entire reset cascade), since it remains constant along all steps of our inductive computation.

We provide in Table 2 upper bounds on the depth and length functions, relative to values of other depth and length functions with respect to configurations of the same or lower-by-one level. The table is constructed by following the syntactic definitions of the reachability formulas, and applying basic simplifications to the resulting expressions. For example, $\mathsf{L}_1(0, l) = 2 + 2l$ standing for the length of $(\neg \beta)\mathbf{U}\tau$. In Lemma 6 we will use Table 2 to bound the absolute depth and length of the main reachability formula.

Depth Analysis. The temporal nesting depth of the main reachability formula $S \underset{\mathcal{B}(\beta)}{\rightsquiggle} T(\tau)$ is intuitively exponential in the number n of levels of the reset cascade (linear in the number of configurations), since it is defined inductively along these levels, and the depth of a level-$(i + 1)$ formula is about twice the depth of a level-i formula. The parameters of the reachability formula are both the configurations S, B and T of level i, and the formulas β and τ; yet, the depth of the reachability formula only linearly depends on the depth of β and τ.

Length Analysis. Intuitively, the overall length of the main reachability formula $S \underset{\mathcal{B}(\beta)}{\rightsquiggle} T(\tau)$ with respect to configurations of the top level is doubly exponential in the number n of levels of the reset cascade (and thus singly exponential in the number of configurations), since the formula is defined inductively along these levels, and the length $\mathsf{L}(i, l)$ is roughly $\mathsf{L}(i-1, l) \cdot \mathsf{L}(i-1, l)$. More precisely, $\mathsf{L}(i, l) = l \cdot f(i)$ for some doubly exponential function $f(i)$.

Now, why is $\mathsf{L}(i, l)$ roughly equal to $\mathsf{L}(i-1, l) \cdot \mathsf{L}(i-1, l)$? The dominant component of the level-i reachability formula is line (2) in the definition of $\langle S, s \rangle \underset{\langle B, b \rangle (\beta)}{\dashrightarrow} \langle T, t \rangle (\tau)$. It is a level-$(i-1)$ reachability formula whose formula-parameters are themselves auxiliary reachability formulas of level i with formula parameters of length l. The length of an auxiliary reachability formula of level i is roughly as of the main reachability formula of level $i-1$, implying that the length of $L_i(l)$ is roughly $L_{i-1}(L_{i-1}(l))$. By the inductive proof that $L_{i-1}(l) = l \cdot f(i-1)$, we get that $L_i(l) = L_{i-1}(L_{i-1}(l)) = L_{i-1}(l) \cdot f(i-1) = l \cdot f(i-1) \cdot f(i-1)$.

As for the many disjunctions and conjunctions that appear in the formulas, observe that the number of disjuncts and conjuncts does not depend on the

Reachability formula φ	Bounds on $\mathsf{depth}(\varphi)$ and length $|\varphi|$
1. $\quad S \xrightsquigarrow[\mathcal{B}(\beta)] T\,(\tau)$	$D_1(i,d) \leq \begin{cases} d+1 & \text{if } i=0 \\ \max(D_3(i,d),\,D_5(i,d)) & \text{otherwise.} \end{cases}$ $L_1(i,l) \leq \begin{cases} 2+2l & \text{if } i=0 \\ 1+L_3(i,l)+L_5(i,l) & \text{otherwise.} \end{cases}$
2. $\quad S \xrightsquigarrow[\mathcal{B}(\beta)]{\text{weak}} T\,(\tau)$	$D_2(i,d) = D_1(i,d)$ $L_2(i,l) = 1+L_1(i,l)$
3. $\langle S,s \rangle \xrightarrow[\langle B,b \rangle(\beta)] \langle T,t \rangle\,(\tau)$	$D_3(i,d) \leq D_1(i-1,d+1)$ $L_3(i,l) \leq 3+2l+|\Sigma|n^{i-1}\big(1+L_1(i-1,3+l)+$ $\quad 1+|\Sigma|n^{i-1}(L_1(i-1,3+l)+1)+$ $\quad 1+|\Sigma|n^{i-1}(L_1(i-1,3+l)+1)\big)$ $\quad \leq 3+2l+4|\Sigma|^2 n^{2(i-1)} L_1(i-1,l+3)$
4. $\langle S,s \rangle \xrightarrow[\langle B,b \rangle(\beta)]{\text{weak}} \langle T,t \rangle\,(\tau)$	$D_4(i,d) \leq D_2(i-1,d+1) = D_1(i-1,d+1)$ $L_4(i,l) \leq 3+2l+(1+|\Sigma|n^{i-1})\big(1+|\Sigma|n^{i-1}(1+L_2(i-1,l+3))\big)$ $\quad \leq 3+2l+4|\Sigma|^2 n^{2(i-1)} L_1(i-1,l+3)$
5. $\langle S,s \rangle \dashrightarrow[\langle B,b \rangle(\beta)] \langle T,t \rangle\,(\tau)$	$D_5(i,d) \leq D_1(i-1,\max(1+D_3(i,d),\,1+D_4(i,d)))$ $L_5(i,l) \leq |\Sigma|n^{i-1} \cdot \big(L_1(i-1,3+L_3(i,l))+2+$ $\quad |\Sigma|n^{i-1} \cdot \big(L_1(i-1,\max(3+L_3(i,l),3+L_4(i,l)))+1\big)\big)$ $\quad +1+|\Sigma|n^{i-1} \cdot (1+L_3(i,3+l))$

Table 2. The relative depths and lengths of the reachability formulas over configurations of level i, and LTL formulas β and τ of depth at most d and length at most l. For the first two reachability formulas, we consider $i \geq 0$ and for the other formulas $i \geq 1$.

formula-parameters β and τ, but only the level i of the configurations S, B, and T. Hence, they do not dominate the growth rate of the overall formula length.

Lemma 6. *Consider a reset cascade $\mathcal{A}$ with n levels and up to n states in each level, and a formula $\zeta = S \xrightsquigarrow[\mathcal{B}(\beta)] T\,(\tau)$ with configurations S, B and T of $\mathcal{A}$ of level $i \leq n$. Let $d = \max(\mathsf{depth}(\beta), \mathsf{depth}(\tau))$ and let $l = \max(|\beta|,|\tau|)$. Then:*

$$(a)\ \mathsf{depth}(\zeta) \leq d + 3^i \quad and \quad (b)\ |\zeta| \leq l \cdot (10|\Sigma|^2 n)^{4^i}$$

Lemma 6 is proven by induction on i and the details of this proof can be found in the full version [5].

4.4   Translating Deterministic Counter-Free Automata to LTL

We use the reachability formulas of Section 4.2 to translate a reset cascade $\mathcal{A}$ to an equivalent LTL formula. Our LTL formulation of $\mathcal{A}$'s acceptance condition

is based on an LTL formulation of "C is visited finitely/infinitely often along a run of $\mathcal{A}$ on a word w", for a given configuration C of $\mathcal{A}$. It thus applies to every ω-regular acceptance condition and by Propositions 6 and 8 to every deterministic counter-free ω-regular automaton. We introduce two shorthands to the main reachability formula: the first is satisfied if we reach T from S without any side constraints (which is always satisfied in the case that $S = T$), and the second requires that we reach it along a nonempty prefix.

$$S \rightsquigarrow T := S \underset{T(\mathbf{false})}{\rightsquigarrow} T\,(\mathbf{true}) \qquad S \overset{>0}{\rightsquigarrow} T := \bigvee_{\sigma \in \Sigma} \Big(\sigma \wedge \mathbf{X}(\delta(S, \sigma) \rightsquigarrow T)\Big)$$

With Lemmas 4 and 5 we then obtain (the proof can be found in [5]):

Lemma 7. *Consider a reset cascade $\mathcal{A} = \langle 2^{AP}, \mathcal{A}_1, \ldots, \mathcal{A}_n \rangle$ together with an initial configuration ι and some configuration C. Then for a word $w \in (2^{AP})^{\omega}$, the run of $\mathcal{A}$ on w starting in ι visits C finitely often iff w satisfies the formula $Fin(C) := \neg(\iota \rightsquigarrow C) \vee \iota \rightsquigarrow C(\neg(C \overset{>0}{\rightsquigarrow} C))$. Furthermore, $Fin(C) \in \Sigma_2$.*

We are now in position to give our main result.

Theorem 2. *Every counter-free deterministic ω-regular automaton $\mathcal{D}$ over alphabet 2^{AP} with n states (and any acceptance condition) is equivalent to an LTL formula φ over atomic propositions AP of double-exponential temporal-nesting depth (in $O(2^{2^n})$) and triple-exponential length (in $2^{2^{O(2^n)}}$). If $\mathcal{D}$ is a looping-Büchi, looping-coBüchi, weak, Büchi, coBüchi, or Muller automaton then φ is respectively in the $\Pi_1, \Sigma_1, \Delta_1, \Pi_2, \Sigma_2$, or Δ_2 syntactic fragment of LTL.*

Proof. We first prove the general result, w.r.t. an arbitrary counter-free deterministic automaton $\mathcal{D}$, and then take into account $\mathcal{D}$'s acceptance condition, to establish the last part of the theorem.

Consider a counter-free deterministic ω-regular automaton $\mathcal{D}$ with some acceptance condition and n states. Recall that there is a Muller automaton $\mathcal{D}'$ equivalent to $\mathcal{D}$ over the semiautomaton of $\mathcal{D}$. By Propositions 6 and 8, $\mathcal{D}'$ is equivalent to a deterministic Muller automaton $\mathcal{D}''$ that is described by a reset cascade $\mathcal{A}$ with up to $m = 2^n$ levels and m states in each level (and thus up to m^m configurations), and whose acceptance condition has up to $k \in 2^{O(m^m n)} = 2^{O(m^m)}$ acceptance sets. An LTL formula φ equivalent to $\mathcal{D}$ can be defined by formulating the acceptance condition of $\mathcal{D}'$ along Lemma 7.

Recall that the Muller condition is a k-elements disjunction, where each disjunct M is a conjunction of requirements to visit infinitely often every configuration from some set G and finitely often every configuration not in G. Observe that M can be formulated as a disjunction over all the configurations in $\mathcal{D}''$ (at most m^m), having for each configuration C the LTL formula $Fin(C)$ or $\neg Fin(C)$, as defined in Lemma 7, depending on whether or not $C \in G$. Hence, the overall formula φ is a combination of disjunctions and conjunctions of up to $k \cdot m^m$ subformulas of the form $Fin(C)$ or $\neg Fin(C)$. Therefore, the depth of φ is the same as of $Fin(C)$, while $|\varphi| \in O(km^m|Fin(C)|) \leq 2^{O(m^m)}|Fin(C)|$. For calculating $\mathsf{depth}(Fin(C))$ and $|Fin(C)|$, we use Lemma 6 bottom up over the subformulas of $Fin(C)$.

Depth.

$\mathsf{depth}(\iota \rightsquigarrow C) \leq 3^m$; $\mathsf{depth}(C \overset{>0}{\rightsquigarrow} C) \leq 3^m + 1$

$\mathsf{depth}(\iota \rightsquigarrow C(\neg(C \overset{>0}{\rightsquigarrow} C))) \leq 2 \cdot 3^m + 1$

$\mathsf{depth}(Fin(C)) = \max(3^m, 2 \cdot 3^m + 1) \in O(3^m) = O(2^{2^n})$,
implying $\mathsf{depth}(\varphi) \in O(2^{2^n})$.

Length.

$|\iota \rightsquigarrow C| \leq (10|\Sigma|^2 m)^{4^m}$; $|C \overset{>0}{\rightsquigarrow} C| \leq (4|\Sigma|) \cdot (10|\Sigma|^2 m)^{4^m}$

$|\iota \rightsquigarrow C(\neg(C \overset{>0}{\rightsquigarrow} C))| \leq (4|\Sigma|(10|\Sigma|^2 m)^{4^m}+1)(10|\Sigma|^2 m)^{4^m} \in (|\Sigma|m)^{2^{O(m)}}$

$|Fin(C)| \in 2 + (10|\Sigma|^2 m)^{4^m} + (|\Sigma|m)^{2^{O(m)}} \in (|\Sigma|m)^{2^{O(m)}}$.

Therefore, $|\varphi| \in 2^{O(m^m)} \cdot (m^m) \cdot ((|\Sigma|m)^{2^{O(m)}}) = |\Sigma|^{2^{O(m)}}$.

Expressing the length of φ with respect to the number n of states in the automaton $\mathcal{D}$, and taking into account the fact that the alphabet Σ has at most n^n different letters (any additional letter must have the same behavior as another letter), we have: $|\varphi| \in |\Sigma|^{2^{O(2^n)}} \leq (2^n)^{2^{O(2^n)}} = 2^{2^{O(2^n)}}$.

We now sketch the second part of the theorem connecting the syntactic hierarchy and the different acceptance conditions of $\mathcal{D}$. We only consider the cases in which $\mathcal{D}$ is either a Muller or a coBüchi automaton. The complete analysis is given in the full version [5]. If $\mathcal{D}$ is a Muller automaton, then the overall formula φ is in Δ_2, since it is a Boolean combination of $Fin(C)$ formulas, which by Lemma 7 belong to Σ_2. If $\mathcal{D}$ is a coBüchi automaton, then we construct the formula φ directly from the coBüchi condition α: φ is a conjunction of $Fin(C)$ formulas over all configurations C that are mapped to states in α. As $Fin(C)$ belongs to Σ_2, so does φ.   □

Observe that by Theorem 2, we get the following result, extending the result of [39, Theorem 3.2] that only considers Rabin automata.

Corollary 1. *Every counter-free deterministic ω-regular automaton (with any acceptance condition) recognises an LTL-definable language.*

Proof. Recall that every deterministic ω-regular automaton is equivalent to a deterministic Muller automaton over the same semiautomaton (see, e.g., [3]). The claim is then a direct consequence of Theorem 2.   □

Remark 2. Theorem 2 can be adapted to the finite-word setting. While on infinite words, the ne**X**t operator is self-dual, i.e., $\neg\mathbf{X}\psi$ is equivalent to $\mathbf{X}\neg\psi$, over finite words, this equivalence does not hold on words of length 1. Thus **X** gains a dual *weak next*, defined as $\tilde{\mathbf{X}}\psi := \neg\mathbf{X}\neg\psi$. In the finite word case, syntactic cosafety (safety) formulas are constructed from **true**, **false**, a, $\neg a$, $\vee$, $\wedge$, and the temporal operators **U** and **X** (**R** and $\tilde{\mathbf{X}}$). Observe that **X** and $\tilde{\mathbf{X}}$ differ only on words of length 1, and thus the only required change in our translation scheme is to replace some **X**s with $\tilde{\mathbf{X}}$s in the reachability formula 4. For finite words a

translation of a counter-free DFA to an LTL formula with only a double exponential size blow-up is known [42]; however, unlike our translation, it does not guarantee syntactic safety (cosafety) formulas for safety (cosafety) languages.

Lastly, we provide a corollary on looping automata, using Theorem 2 and the following known result.

Proposition 9 (Rephrased Theorem 13 from [29]). *Let $\mathcal{D}$ be a deterministic looping-Büchi automaton with n states that recognises an LTL-definable language. Then there exists an equivalent counter-free deterministic looping-Büchi automaton $\mathcal{D}'$ with at most n states.*

Corollary 2. *Every deterministic looping-Büchi (looping-coBüchi) automaton with n states that recognises an LTL-definable language is equivalent to an LTL formula $\varphi \in \Pi_1$ (Σ_1) of temporal nesting depth in $O(2^{2^n})$ and length in $2^{2^{O(2^n)}}$.*

This is an elementary upper bound for two constructions for which either the upper bound was unknown or non-elementary: the liveness-safety decomposition of LTL [29] and the translation of semantic safety LTL to syntactic safety LTL.

5   Conclusions

We have studied the size trade-offs between LTL and automata. Over a unary alphabet, the situation is straightforward and we provided tight complexity bounds. The general case of infinite words over an arbitrary alphabet is more complex. We gave to our knowledge the first elementary complexity bound on the translation of counter-free deterministic ω-regular automata into LTL formulas.

Every ω-regular automaton recognising an LTL-definable language can be translated to a counter-free deterministic automaton [39, Theorem 3.2]. Yet, we are unaware of a bound on the size blow-up involved in such a translation. Once established, it can be combined with our translation to get a general bound on the translation of automata to LTL. It will also provide a (currently unknown[6]) elementary upper bound on the translation of LTL with both future and past operators to LTL with only future operators (which is the version of LTL that we have considered), as (both version of) LTL can be translated to nondeterministic Büchi automata with a single exponential size blow-up [41, Theorem 2.1].

While going from non-elementary to double-exponential depth and triple-exponential length is an improvement, these upper bounds might not be tight—there is currently no known non-linear lower bound! Closing this gap is a challenging open problem, which might require new lower bound techniques for alternating automata, as LTL formulas are an inherently alternating model.

---

[6] In consultation with the author of [30], we have confirmed that while the lower bound provided in that paper holds, the stated upper bound is erroneous.

References

1. Birget, J.C.: Two-way automata and length-preserving homomorphisms. Mathematical Systems Theory **29**(3), 191–226 (1996)
2. Bojańczyk, M.: Languages recognised by finite semigroups, and their generalisations to objects such as trees and graphs, with an emphasis on definability in monadic second-order logic (2020)
3. Boker, U.: Why these automata types? In: Proc. of LPAR. pp. 143–163 (2018)
4. Boker, U., Kupferman, O.: The quest for a tight translation of Büchi to co-Büchi automata. In: Fields of Logic and Computation, pp. 147–164. Springer (2010)
5. Boker, U., Lehtinen, K., Sickert, S.: On the translation of automata to linear temporal logic (2022), https://arxiv.org/abs/2201.10267, full version
6. Cerná, I., Pelánek, R.: Relating hierarchy of temporal properties to model checking. In: MFCS. Lecture Notes in Computer Science, vol. 2747, pp. 318–327. Springer (2003)
7. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. J. ACM **28**(1), 114–133 (Jan 1981)
8. Chang, E.Y., Manna, Z., Pnueli, A.: Characterization of temporal property classes. In: Kuich, W. (ed.) Automata, Languages and Programming, 19th International Colloquium, ICALP92, Vienna, Austria, July 13-17, 1992, Proceedings. Lecture Notes in Computer Science, vol. 623, pp. 474–486. Springer (1992)
9. Chrobak, M.: Finite automata and unary languages. Theoretical Computer Science **47**, 149–158 (1986)
10. Cohen, J., Perrin, D., Pin, J.E.: On the expressive power of temporal logic. Journal of computer and System Sciences **46**(3), 271–294 (1993)
11. Cohen-Chesnot, J.: On the expressive power of temporal logic for infinite words. Theoretical Computer Science **83**(2), 301–312 (1991)
12. Colcombet, T., Zdanowski, K.: A tight lower bound for determinization of transition labeled Büchi automata. In: International Colloquium on Automata, Languages, and Programming. pp. 151–162. Springer (2009)
13. Diekert, V., Gastin, P.: First-order definable languages. In: Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]. Texts in Logic and Games, vol. 2, pp. 261–306 (2008)
14. Eilenberg, S.: Automata, Languages, and Machines Volume B. Academic Press, Inc., USA (1976)
15. Etessami, K., Vardi, M.Y., Wilke, T.: First-order logic with two variables and unary temporal logic. Inf. Comput. **179**(2), 279–295 (2002)
16. Gabbay, D., Pnueli, A., Shelah, S., Stavi, J.: On the temporal analysis of fairness. In: Proc. of POPL. p. 163–173. New York, NY, USA (1980)
17. Geffert, V., Mereghetti, C., Pighizzini, G.: Complementing two-way finite automata. Information and Computation **205**(8), 1173–1187 (2007)
18. Kamp, J.A.W.: Tense logic and the theory of linear order. University of California, Los Angeles (1968)
19. Kupferman, O., Rosenberg, A.: The blowup in translating LTL to deterministic automata. In: Proc. of Model Checking and Artificial Intelligence. pp. 85–94 (2010)
20. Kupferman, O., Ta-Shma, A., Vardi, M.Y.: Counting with automata. In: Proc. of LICS (1999)
21. Kupferman, O., Vardi, M.Y.: Weak alternating automata are not that weak. ACM Transactions on Computational Logic (TOCL) **2**(3), 408–429 (2001)

22. Ladner, R.E.: Application of model theoretic games to discrete linear orders and finite automata. Information and Control **33**(4), 281–303 (1977)
23. Leiss, E.: Succinct representation of regular languages by boolean automata. Theoretical computer science **13**(3), 323–330 (1981)
24. Löding, C.: Optimal bounds for transformations of ω-automata. In: Rangan, C.P., Raman, V., Ramanujam, R. (eds.) Foundations of Software Technology and Theoretical Computer Science. pp. 97–109. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
25. Maler, O.: On the Krohn-Rhodes cascaded decomposition theorem. In: Time for Verification, Essays in Memory of Amir Pnueli. Lecture Notes in Computer Science, vol. 6200, pp. 260–278. Springer (2010)
26. Maler, O., Pnueli, A.: Tight bounds on the complexity of cascaded decomposition of automata. In: Proc. of FOCS. pp. 672–682 (1990)
27. Maler, O., Pnueli, A.: On the cascaded decomposition of automata, its complexity and its application to logic. Unpublished. Available at: http://www-verimag.imag.fr/~maler/Papers/decomp.pdf (1994)
28. Manna, Z., Pnueli, A.: A hierarchy of temporal properties. In: PODC. pp. 377–410. ACM (1990)
29. Maretic, G.P., Dashti, M.T., Basin, D.A.: LTL is closed under topological closure. Inf. Process. Lett. **114**(8), 408–413 (2014)
30. Markey, N.: Temporal logic with past is exponentially more succinct. Bull. EATCS **79**, 122–128 (2003)
31. McNaughton, R., Papert, S.A.: Counter-Free Automata (MIT research monograph no. 65). The MIT Press (1971)
32. Michel, M.: Complementation is more difficult with automata on infinite words. CNET, Paris **15** (1988)
33. Muller, D.E., Saoudi, A., Schupp, P.E.: Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In: Proceedings Third Annual Symposium on Logic in Computer Science. pp. 422–423. IEEE Computer Society (1988)
34. Perrin, D.: Recent results on automata and infinite words. In: International Symposium on Mathematical Foundations of Computer Science. pp. 134–148. Springer (1984)
35. Safra, S.: Complexity of automata on infinite objects. Ph.D. thesis, Weizmann Institute, Rehovot, Israel (1989)
36. Schewe, S.: Büchi Complementation Made Tight. In: Albers, S., Marion, J.Y. (eds.) Proc. of 26th International STACS. Leibniz International Proceedings in Informatics (LIPIcs), vol. 3, pp. 661–672 (2009)
37. Sickert, S., Esparza, J.: An efficient normalisation procedure for linear temporal logic and very weak alternating automata. In: LICS. pp. 831–844. ACM (2020)
38. Thomas, W.: Star-free regular sets of ω-sequences. Information and Control **42**(2), 148–156 (1979)
39. Thomas, W.: A combinatorial approach to the theory of ω-automata. Information and Control **48**(3), 261–283 (1981)
40. Thomas, W.: Automata on infinite objects. In: Formal Models and Semantics, pp. 133–191. Elsevier (1990)
41. Vardi, M., Wolper, P.: An automata-theoretic approach to automatic program verification. In: Proc. of LICS. pp. 332–344 (1986)
42. Wilke, T.: Classifying discrete temporal properties. In: Annual symposium on theoretical aspects of computer science. pp. 32–46. Springer (1999)

43. Wilke, T.: Past, present, and infinite future. In: 43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2016)
44. Wilke, T.: Backward deterministic Büchi automata on infinite words. In: 37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2018)
45. Wolper, P.: Temporal logic can be more expressive **56**(1–2), 72–99 (1983)
46. Zuck, L.D.: Past Temporal Logic. Ph.D. thesis, The Weizmann Institute of Science, Israel (Aug 1986)

Categorical composable cryptography⋆

Anne Broadbent and Martti Karvonen(✉)

Department of Mathematics and Statistics, University of Ottawa, Ottawa, Canada
{abroadbe,martti.karvonen}@uottawa.ca

Abstract. We formalize the simulation paradigm of cryptography in terms of category theory and show that protocols secure against abstract attacks form a symmetric monoidal category, thus giving an abstract model of composable security definitions in cryptography. Our model is able to incorporate computational security, set-up assumptions and various attack models such as colluding or independently acting subsets of adversaries in a modular, flexible fashion. We conclude by using string diagrams to rederive the security of the one-time pad and no-go results concerning the limits of bipartite and tripartite cryptography, ruling out e.g., composable commitments and broadcasting.

Keywords: Cryptography · composable security · quantum cryptography · category theory

1 Introduction

Modern cryptographic protocols are complicated algorithmic entities, and their security analyses are often no simpler than the protocols themselves. Given this complexity, it would be highly desirable to be able to design protocols and reason about them compositionally, i.e., by breaking them down into smaller constituent parts. In particular, one would hope that combining protocols proven secure results in a secure protocol without need for further security proofs. However, this is not the case for stand-alone security notions that are common in cryptography. To illustrate such failures of composability, let us consider the history of quantum key distribution (QKD), as recounted in [60]: QKD was originally proposed in the 80s [7]. The first security proofs against unbounded adversaries followed a decade later [8, 49, 50, 64]. However, since composability was originally not a concern, it was later realized that the original security definitions did not provide a good enough level of security [42]—they didn't guarantee security if the keys were to be actually used, since even a partial leak of the key would compromise the rest. The story ends on a positive note, as eventually a new security criterion was proposed, together with stronger proofs [5, 62].

In this work we initiate a categorical study of composable security definitions in cryptography. In the viewpoint developed here one thinks of cryptography

---

as a resource theory: cryptographic functionalities (e.g. secure communication channels) are viewed as resources and cryptographic protocols let one transform some starting resources to others. For instance, one can view the one-time-pad as a protocol that transforms an authenticated channel and a shared secret key into a secure channel. For a given protocol, one can then study whether it is secure against some (set of) attack model(s), and protocols secure against a fixed set of models can always be composed sequentially and in parallel.

This is in fact the viewpoint taken in constructive cryptography [47], which also develops the one-time-pad example above in more detail. However [47] does not make a formal connection to resource theories as usually understood, whether as in quantum physics [16,39], or more generally as defined in order theoretic [32] or categorical [20] terms. Instead, constructive cryptography is usually combined with abstract cryptography [48] which is formalized in terms of a novel algebraic theory of systems [46].

Our work can be seen as a particular formalization of the ideas behind constructive cryptography, or alternatively as giving a categorical account of the real-world-ideal-world paradigm (also known as the simulation paradigm [34]), which underlies more concrete frameworks for composable security, such as universally composable cryptography [13] and others [2,3,38,43,44,51,58]. We will discuss these approaches and abstract and constructive cryptography in more detail in Section 1.1

Our long-term goal is to enable cryptographers to reason about composable security at the same level of formality as stand-alone security, *without having to fix all the details of a machine model nor having to master category theory.* Indeed, our current results already let one define multipartite protocols and security against arbitrary subsets of malicious adversaries *in any symmetric monoidal category* **C**. Thus, as long as one's model of interactive computation results in a symmetric monoidal category, or more informally, one is willing to use pictures such as fig. 1d to depict connections between computational processes without further specifying the order in which the picture was drawn, one can use the simulation paradigm to reason about multipartite security against malicious participants composably—and specifying finer details of the computational model is only needed to the extent that it affects the validity of one's argument. Moreover, as our attack models and composition theorems are fairly general, we hope that more refined models of adversaries can be incorporated.

We now highlight our contributions to cryptography: We show how to adapt resource theories as categorically formulated [20] in order to reason abstractly about *secure* transformations between resources. This is done in Section 3 by formalizing the simulation paradigm in terms of an abstract attack model (Definition 1), designed to be general enough to capture standard attack models of interest (and more) while still structured enough to guarantee composability. This section culminates in Corollary 1, which shows that for any fixed set of attack models, the class of protocols secure against each of them results in a symmetric monoidal category. In Theorem 3 we observe that under suitable conditions, images of secure protocols under monoidal functors remain secure, which

gives an abstract variant of the lifting theorem [68, Theorem 15] that states that perfectly UC-secure protocols are quantum UC-secure. We adapt this framework to model *computational security* in two ways: either by replacing equations with an equivalence relation, abstracting the idea of computational indistinguishability, as is done in section 4, or by working with a notion of distance, deferred to a full version. In the case of a distance, one can then either explicitly bound the distance between desired and actually achieved behavior, or work with sequences of protocols that converge to the target in the limit: the former models working in the finite-key regime [67] and the latter models the kinds of asymptotic security and complexity statements that are common in cryptography.Finally, we apply the framework developed to study bipartite and tripartite cryptography. We first prove pictorially the security of the one-time pad. We then reprove the no-go-theorems of [46, 48, 61] concerning two-party commitments (resp. three-party broadcasting) in this setting, and reinterpret them as limits on what can be achieved securely in any compact closed category (resp. symmetric monoidal category). The key steps of the proof are done graphically, thus opening the door for cryptographers to use such pictorial representations as rigorous tools rather than merely as illustrations.

Moreover, we discuss some categorical constructions capturing aspects of resource theories appearing in the physics literature. These contributions may be of independent interest for further categorical studies on resource theories. In [20] it is observed that many resource theories arise from an inclusion $\mathbf{C}_F \hookrightarrow \mathbf{C}$ of free transformations into a larger monoidal category, by taking the resource theory of states. We observe that this amounts to applying the monoidal Grothendieck construction [53] to the functor $\mathbf{C}_F \to \mathbf{C} \xrightarrow{\hom(I,-)} \mathbf{Set}$. This suggests applying this construction more generally to the composite of monoidal functors $F \colon \mathbf{D} \to \mathbf{C}$ and $R \colon \mathbf{C} \to \mathbf{Set}$. In Example 1 we note that choosing F to be the n-fold monoidal product $\mathbf{C}^n \to \mathbf{C}$ captures resources shared by n parties and n-partite transformations between them. In the extended version, we model categorically situations where there is a notion of distance between resources, and instead of exact resource conversions one either studies approximate transformations or sequences of transformations that succeed in the limit. In the extended version, we discuss a variant of a construction on monoidal categories, used in special cases in [31] and discussed in more detail in [23, 33], that allows one to declare some resources free and thus enlarge the set of possible resource conversions.

1.1 Related work

We have already mentioned that cryptographers have developed a plethora of frameworks for composable security, such as universally composable cryptography [13], reactive simulatability [2, 3, 58] and others [38, 43, 44, 51]. Moreover, some of these frameworks have been adapted to the quantum setting [6, 54, 68]. One might hence be tempted to think that the problem of composability in cryptography has been solved. However, it is fair to say that most mainstream cryptography is not formulated composably and that composable cryptography

has yet to realize its full potential. Moreover, this proliferation of frameworks should be taken as evidence of the continued importance of the issue, and is in fact reflected by the existence of a recent Dagstuhl seminar on this matter [12]. Indeed, the aforementioned frameworks mostly consist of setting up fairly detailed models of interacting machines, which as an approach suffers from two drawbacks: Firstly, in order to be more realistic, the detailed models are often complicated, both to reason in terms of and to define, thus making practicing cryptographers less willing to use them. Perhaps more importantly it is not always clear whether the results proven in a particular model apply more generally for other kinds of machines, whether those of a competing framework or those in the real world. It is true that the choice of a concrete machine model does affect what can be securely achieved—for instance, quantum cryptography differs from classical cryptography and similarly classical cryptography behaves differently in synchronous and asynchronous settings [4, 40]. Nevertheless, one might hope that composable cryptography could be done at a similar level of formality as complexity theory, where one rarely worries about the number of tapes in a Turing machine or of other low-level details of machine models. Second, changing the model slightly (to e.g., model different kinds of adversaries or to incorporate a different notion of efficiency) often requires reproving "composition theorems" of the framework or at least checking that the existing proof is not broken by the modification.

In contrast to frameworks based on detailed machine models, there are two closely related top-down approaches to cryptography: constructive cryptography [47] and its cousin abstract cryptography [48]. We are indebted to both of these approaches, and indeed our framework could be seen as formalizing the key idea of constructive cryptography—namely, cryptography as a resource theory—and thus occupying a similar space as abstract cryptography. A key difference is that constructive cryptography is usually instantiated in terms of abstract cryptography [48], which in turn is based on a novel algebraic theory of systems [46]. However, our work is not merely a translation from this theory to categorical language, as there are important differences and benefits that stem from formalizing cryptography in terms of a well-established and well-studied algebraic theory of systems—that of (symmetric) monoidal categories:

The fact that cryptographers wish to compose their protocols *sequentially and in parallel* strongly suggests using *monoidal categories*, that have these composition operations as primitives. In our framework, protocols secure against a fixed set of attack models results in a symmetric monoidal category. In contrast, the algebraic theory of systems [46] on which abstract cryptography is based takes parallel composition and internal wiring as its primitives. This design choice results in some technical kinks and tangles that are natural with any novel theory but have already been smoothed out in the case of category theory. For instance, in the algebraic theory of systems of [46] the parallel composition is a partial operation and in particular the parallel composite of a system with itself is never

defined[1] and the set of wires coming out of a system is fixed once and for all[2]. In contrast, in a monoidal category parallel composition is a total operation and whether one draws a box with n output wires of types $A_1, \ldots A_n$ or single output wire of type $\bigotimes_{i=1}^{n} A_i$ is a matter of convenience. Technical differences such as these make a direct formal comparison or translation between the frameworks difficult, even if informally and superficially there are similarities.

We do not abstract away from an attacker model, but rather make it an explicit part of the formalism that can be modified without worrying about composability. This makes it possible to consider and combine very easily different security properties, and in particular paves the way to model attackers with limited powers such as honest-but-curious adversaries. In our framework, one can first fix a protocol transforming some resource to another one, and then discuss whether this transformation is secure against different attack models. In contrast, in abstract cryptography a cryptographic resource is a tuple of functionalities, one for each set of dishonest parties, and thus has no prior existence before fixing the attack model. This makes the question "what attack models is this protocol secure against?" difficult to formalize.

As category theory is de facto the lingua franca between several subfields of mathematics and computer science, elucidating the categorical structures present in cryptography opens up the door to further connections between cryptography and other fields. For instance, game semantics readily gives models of interactive, asynchronous and probabilistic (or quantum) computation [18, 19, 69] in which our theory can be instantiated, and thus further paves the way for programming language theory to inform cryptographic models of concurrency.

Category theory comes with existing theory, results and tools that can readily be applied to questions of cryptographic interest. In particular, the graphical calculi of symmetric monoidal and compact closed categories [63] enables one to rederive impossibility results shown in [46, 48, 61] purely pictorially. In fact, such pictures were already often used as heuristic devices that illuminate the official proofs, and viewing these pictures categorically lets us promote them from mere illustrations to rigorous yet intuitive proofs. Indeed, in [48, Footnote 27] the authors suggest moving from a 1-dimensional symbolic presentation to a 2-dimensional one, and this is exactly what the graphical calculus achieves.

The approaches above result in a framework where security is defined so as to guarantee composability. In contrast, approaches based on various protocol logics [25–30] aim to characterize situations where composition can be done securely, even if one does not use composable security definitions throughout. As these approaches are based on process calculi, they are categorical under the hood [52, 55] even if not overtly so. There is also earlier work explicitly discussing

---

[1] While the suggested fix is to assume that one has "copies" of the same system with disjoint wire labels, it is unclear how one recognizes or even defines *in terms of the system algebra* that two distinct systems are copies of each other.

[2] Indeed, while [59] manages to bundle and unbundle ports along isomorphism when convenient, it seems like the chosen technical foundation makes this more of a struggle than it should be.

category theory in the context of cryptography [9, 10, 21, 22, 35–37, 41, 56, 57, 65, 66], but they concern stand-alone security of particular cryptographic protocols, rather than categorical aspects of composable security definitions.

2   Resource theories

We briefly review the categorical viewpoint on resource theories of [20]. Roughly speaking, a resource theory can be seen as a SMC but the change in terminology corresponds to a change in viewpoint: usually in category theory one studies global properties of a category, such as the existence of (co)limits, relationships to other categories, etc. In contrast, when one views a particular SMC **C** as resource theory, one is interested in local questions. One thinks of objects of **C** as resources, and morphisms as processes that transform a resource to another. From this point of view, one mostly wishes to understand whether $\hom_{\mathbf{C}}(X, Y)$ is empty or not for resources X and Y of interest. Thus from the resource-theoretic point of view, most of the interesting information in **C** is already present in its preorder collapse. As concrete examples of resource-theoretic questions, one might wonder if (i) some noisy channels can simulate a (almost) noiseless channel [20, Example 3.13.], (ii) there is a protocol that uses only local quantum operations and classical communication and transforms a particular quantum state to another one [17], (iii) some non-classical statistical behavior can be used to simulate other such behavior [1]. In [20] the authors show how many familiar resource theories arise in a uniform fashion: starting from an SMC **C** of processes equipped with a wide sub-SMC $\mathbf{C}_F$, the morphisms of which correspond to "free" processes, they build several resource theories (=SMCs). Perhaps the most important of these constructions is the resource theory of states: given $\mathbf{C}_F \hookrightarrow \mathbf{C}$, the corresponding resource theory of states can be explicitly constructed by taking the objects of this resource theory to be states of **C**, i.e., maps $r \colon I \to A$ for some A, and maps $r \to s$ are maps $f \colon A \to B$ in $\mathbf{C}_F$ that transform r to s as in fig. 1a.

We now turn our attention towards cryptography. As contemporary cryptography is both broad and complex in scope, any faithful model of it is likely to be complicated as well. A benefit of the categorical idiom is that we can build up to more complicated models in stages, which is what we will do in the sequel. We phrase our constructions in terms of an arbitrary SMC **C**, but in order to model actual cryptographic protocols, the morphisms of **C** should represent interactive computational machines with open "ports", with composition then amounting to connecting such machines together. Different choices of **C** set the background for different kinds of cryptography, so that quantum cryptographers want **C** to include quantum systems whereas in classical cryptography it is sufficient that these computational machines are probabilistic. Constructing such categories **C** in detail is not trivial but is outside our scope—we will discuss this in more detail in section 6.

Our first observation is that there is no reason to restrict to inclusions $\mathbf{C}_F \hookrightarrow \mathbf{C}$ in order to construct a resource theory of states. Indeed, while it

is straightforward to verify explicitly that the resource theory of states is a symmetric monoidal category, it is instructive to understand more abstractly why this is so: in effect, the constructed category is the category of elements of the composite functor $\mathbf{C}_F \to \mathbf{C} \xrightarrow{\hom(I,-)} \mathbf{Set}$. As this composite is a (lax) symmetric monoidal functor, the resulting category is automatically symmetric monoidal as observed in [53]. Thus this construction goes through for any symmetric (lax) monoidal functors $\mathbf{D} \xrightarrow{F} \mathbf{C} \xrightarrow{R} \mathbf{Set}$. Here we may think of F as interpreting free processes into an ambient category of all processes, and $R\colon \mathbf{C} \to \mathbf{Set}$ as an operation that gives for each object A of $\mathbf{C}$ the set $R(A)$ of resources of type A.

Explicitly, given symmetric monoidal functors $\mathbf{D} \xrightarrow{F} \mathbf{C} \xrightarrow{R} \mathbf{Set}$, the category of elements $\int RF$ has as its objects pairs (r, A) where A is an object of $\mathbf{D}$ and $r \in RF(A)$, the intuition being that r is a resource of type $F(A)$. A morphism $(r, A) \to (s, B)$ is given by a morphism $f\colon A \to B$ in $\mathbf{D}$ that takes r to s, i.e., satisfies $RF(f)(r) = s$. The symmetric monoidal structure comes from the symmetric monoidal structures of $\mathbf{D}, \mathbf{Set}$ and RF. Somewhat more explicitly, $(r, A) \otimes (s, B)$ is defined by $(r \otimes s, A \otimes B)$ where $r \otimes s$ is the image of (r, s) under the function $RF(A) \times RF(B) \to RF(A \otimes B)$ that is part of the monoidal structure on RF, and on morphisms of $\int RF$ the monoidal product is defined from that of $\mathbf{D}$.

From now on we will assume that F is strong monoidal, and while $R = \hom(I, -)$ captures our main examples of interest, we will phrase our results for an arbitrary lax monoidal R. This relaxation allows us to capture the n-partite structure often used when studying cryptography, as shown next.

Example 1. Consider the resource theory induced by $\mathbf{C}^n \xrightarrow{\otimes} \mathbf{C} \xrightarrow{\hom(I,-)} \mathbf{Set}$, where we write $\otimes$ for the n-fold monoidal product[3]. The resulting resource theory has a natural interpretation in terms of n agents trying to transform resources to others: an object of this resource theory corresponds to a pair $((A_i)_{i=1}^n, r\colon I \to \bigotimes A_i)$, and can be thought of as an n-partite state, depicted in fig. 1b, where the ith agent has access to a port of type A_i. A morphism $\bar{f} = (f_1, \ldots f_n)\colon ((A_i)_{i=1}^n, r) \to ((B_i)_{i=1}^n, s)$ between such resources then amounts to a protocol that prescribes, for each agent i a process f_i that they should perform so that r gets transformed to s as in fig. 1c.

In this resource theory, all of the agents are equally powerful and can perform all processes allowed by $\mathbf{C}$, and this might be unrealistic: first of all, $\mathbf{C}$ might include computational processes that are too powerful/expensive for us to use in our cryptographic protocols. Moreover, having agents with different computational powers is important to model e.g., blind quantum computing [11] where a client with access only to limited, if any, quantum computation tries to securely delegate computations to a server with a powerful quantum computer. This limitation is easily remedied: we could take the ith agent to be able to implement computations in some sub-SMC $\mathbf{C}_i$ of $\mathbf{C}$, and then consider $\prod_{i=1}^n \mathbf{C}_i \to \mathbf{C}$.

---

[3] As $\mathbf{C}$ is symmetric, the functor $\otimes$ is strong monoidal.

(a) A map f in the resource theory of states

(b) An n-partite state

(c) An n-partite transformation

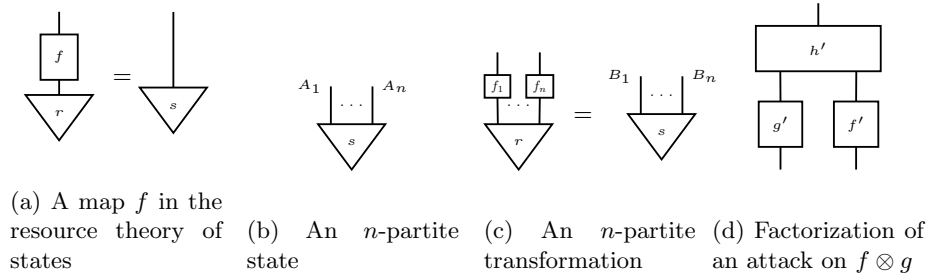(d) Factorization of an attack on $f \otimes g$

Fig. 1: Some resource transformations

A more serious limitation is that such transformations have no security guarantees—they only work if each agent performs f_i as prescribed by the protocol. We fix this next.

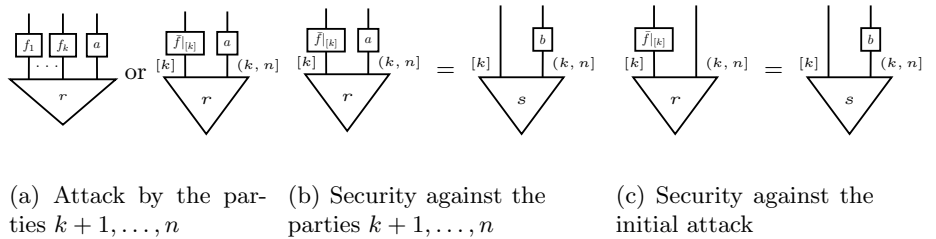## 3   Cryptography as a resource theory

(a) Attack by the parties $k + 1, \ldots, n$

(b) Security against the parties $k + 1, \ldots, n$

(c) Security against the initial attack

Fig. 2: Attacks and security constraints

In order for a protocol $\bar{f} = (f_1, \ldots, f_n) \colon ((A_i)_{i=1}^n, r) \to ((B_i)_{i=1}^n, s)$ to be secure, we should have some guarantees about what happens if, as a result of *an attack* on the protocol, something else than $(f_1, \ldots, f_n)$ happens. For instance, some subset of the parties might deviate from the protocol and do something else instead. In the simulation paradigm [34], security is then defined by saying that, anything that could happen when running the real protocol, i.e., $\bar{f}$ with r, could also happen in the ideal world, i.e., with s. A given protocol might be secure against some kinds of attacks and insecure against others, so we define security against an abstract attack model. This abstract notion of an attack model is one of the main definitions of our paper. It isolates conditions needed for the composition theorem (Theorem 1). It also captures our key examples that we use to illustrate the definition after giving it. Note that most proofs are deferred to an extended version.

Definition 1. *An* attack model $\mathcal{A}$ *on an SMC* **C** *consists of giving for each morphism f of* **C** *a class $\mathcal{A}(f)$ of morphisms of* **C** *such that*

(i) $f \in \mathcal{A}(f)$ for every f.

(ii) For any $f\colon A \to B$ and $g\colon B \to C$ and composable $g' \in \mathcal{A}(g)$, $f' \in \mathcal{A}(f)$ we have $g' \circ f' \in \mathcal{A}(g \circ f)$. Moreover, any $h \in \mathcal{A}(g \circ f)$ factorizes as $g' \circ f'$ with $g' \in \mathcal{A}(g)$ and $f' \in \mathcal{A}(f)$.

(iii) For any $f\colon A \to B$, $g\colon C \to D$ in $\mathbf{C}$ and $f' \in \mathcal{A}(f)$, $g' \in \mathcal{A}(g)$ we have $f' \otimes g' \in \mathcal{A}(f \otimes g)$. Moreover, any $h \in \mathcal{A}(f \otimes g)$ factorizes as $h' \circ (f' \otimes g')$ with $f' \in \mathcal{A}(f)$, $g' \in \mathcal{A}(g)$ and $h' \in \mathcal{A}(\mathrm{id}_{B \otimes D})$.

Let $f\colon (A, r) \to (B, s)$ define a morphism in the resource theory $\int RF$ induced by $F\colon \mathbf{D} \to \mathbf{C}$ and $R\colon \mathbf{C} \to \mathbf{Set}$. We say that f is secure against an attack model $\mathcal{A}$ on $\mathbf{C}$ (or $\mathcal{A}$-secure) if for any $f' \in \mathcal{A}(F(f))$ with $\mathrm{dom}(f') = F(A)$ there is $b \in \mathcal{A}(\mathrm{id}_{F(B)})$ with $\mathrm{dom}(b) = F(B)$ such that $R(f')r = R(b)s$.

The above definition of security asks for perfect equality and corresponds to information-theoretic security in cryptography. This is often too much to hope for, and we will replace this by an equivalence relation in section 4 and by a notion of distance in an extended version.

The intuition is that $\mathcal{A}$ gives, for each process in $\mathbf{C}$, the set of behaviors that the attackers could force to happen instead of honest behavior. In particular, $\mathcal{A}(\mathrm{id}_B)$ give the set of behaviors that is available to attackers given access to a system of type B. Then property (i) amounts to the assumption that the adversaries could behave honestly. The first halves of properties (ii) and (iii) say that, given an attack on g and one on f, both attacks could happen when composing g and f sequentially or in parallel. The second parts of these say that attacks on composite processes can be understood as composites of attacks. However, note that (iii) does not say that an attack on a product has to be a product of attacks: the factorization says that any $h \in \mathcal{A}(g \otimes f)$ factorizes as in fig. 1d with $g' \in \mathcal{A}(g)$, $f' \in \mathcal{A}(f)$ and $h' \in \mathcal{A}(\mathrm{id}_{B \otimes D})$. The intuition is that an attacker does not have to attack two parallel protocols independently of each other, but might play the protocols against each other in complicated ways. This intuition also explains why we do not require that all morphisms in $\mathcal{A}(f)$ have $F(A)$ as their domain, despite the definition of $\mathcal{A}$-security quantifying only against those: when factoring $h \in \mathcal{A}(g \circ f)$ as $g' \circ f'$ with $g' \in \mathcal{A}(g)$ and $f' \in \mathcal{A}(f)$, we can no longer guarantee that $F(B)$ is the domain of g'—perhaps the attackers take us elsewhere when they perform f'.

If one thinks of $F\colon \mathbf{D} \to \mathbf{C}$ as representing the inclusion of free processes into general processes, one also gets an explanation why we do not insist that free processes and attacks live in the same category, i.e., that $F = \mathrm{id}_{\mathbf{C}}$. This is simply because we might wish to prove that some protocols are secure against attackers that can use more resources than we wish or can use in the protocols.

Example 2. For any SMC $\mathbf{C}$ there are two trivial attack models: the minimal one defined by $\mathcal{A}(f) = \{f\}$ and the maximal one sending f to the class of all morphisms of $\mathbf{C}$. We interpret the minimal attack model as representing honest behavior, and the maximal one as representing arbitrary malicious behavior.

Proposition 1. *If $\mathcal{A}_1, \ldots, \mathcal{A}_n$ are attack models on SMCs $\mathbf{C}_1, \ldots, \mathbf{C}_n$ respectively, then there is a product $\prod_{i=1}^{n} \mathcal{A}_i$ attack model on $\prod_{i=1}^{n} \mathbf{C}_i$ defined by $(\prod_{i=1}^{n} \mathcal{A}_i)(f_1, \ldots, f_n) = \prod_{i=1}^{n} \mathcal{A}_i(f_i)$.*

This proposition, together with the minimal and maximal attack models, is already expressive enough to model multi-party computation where some subset of the parties might do arbitrary malicious behavior. Indeed, consider the n-partite resource theory induced by $\mathbf{C}^n \xrightarrow{\otimes} \mathbf{C} \xrightarrow{\mathrm{hom}(I,-)} \mathbf{Set}$. Let us first model a situation where the first $n-1$ participants are honest and the last participant is dishonest. In this case we can set $\mathcal{A} = \prod_{i=1}^{n} \mathcal{A}_i$ where each of $\mathcal{A}_1, \ldots, \mathcal{A}_{n-1}$ is the minimal attack model on $\mathbf{C}$ and $\mathcal{A}_n$ is the maximal attack model. Then, an attack on $\bar{f} = (f_1, \ldots f_n) \colon ((A_i)_{i=1}^{n}, r) \to ((B_i)_{i=1}^{n}, s)$ can be represented by the first $n-1$ parties obeying the protocol and the n-th party doing an arbitrary computation a, as depicted in the two pictures of fig. 2a, where $[n] := \{1, \ldots, n\}$, $(k, n] := \{k+1, \ldots n\}$, $\bar{f}|_{[k]} := \bigotimes_{i=1}^{k} f_i$, and here $k = n-1$. The latter representation will be used when we do not need to emphasize pictorially the fact that the honest parties are each performing their own individual computations.

   If instead of just one attacker, there are several *independently* acting adversaries, we can take $\mathcal{A} = \prod_{i=1}^{n} \mathcal{A}_i$ where $\mathcal{A}_i$ is the minimal or maximal attack structure depending on whether the ith participant is honest or not. If the set of dishonest parties can collude and communicate arbitrarily during the process, we need the flexibility given in Definition 1 and have the attack structure live in a different category than where our protocols live. For simplicity of notation, assume that the first k agents are honest but the remaining parties are malicious and might do arbitrary (joint) processes in $\mathbf{C}$. In particular, the action done by the dishonest parties $k+1, \ldots, n$ need not be describable as a product $\bigotimes_{i=k+1}^{n}(a_i)$ of individual actions. In that case we define $\mathcal{A}$ as follows: we first consider our resource theory as arising from $\mathbf{C}^n \xrightarrow{\mathrm{id}^k \times \otimes} \mathbf{C}^k \times \mathbf{C} \xrightarrow{\otimes} \mathbf{C} \xrightarrow{\mathrm{hom}(I,-)} \mathbf{Set}$, and define $\mathcal{A}$ on $\mathbf{C}^k \times \mathbf{C}$ as the product of the minimal attack model on $\mathbf{C}^k$ and the maximal one on $\mathbf{C}$. Concretely, this means that the first k agents always obey the protocol, but the remaining agents can choose to perform arbitrary joint behaviors in $\mathbf{C}$. Then a generic attack on a protocol $\bar{f}$ can be represented exactly as before in fig. 2a, except we no longer insist that $k = n-1$. Now a protocol $\bar{f}$ is $\mathcal{A}$-secure if for any a with $\mathrm{dom}(a) = (A_i)_{i=k+1}^{n}$ there is a b with $\mathrm{dom}(b) = (B_i)_{i=k+1}^{n}$ satisfying the equation of fig. 2b.

   If one is willing to draw more wire crossings, one can easily depict and define security against an arbitrary subset of the parties behaving maliciously, and henceforward this is the attack model we have in mind when we say that some n-partite protocol is secure against some subset of the parties. Moreover, for any subset J of dishonest agents, one could consider more limited kinds of attacks: for instance, the agents might have limited computational power or limited abilities to perform joint computations—as long as the attack model satisfies the conditions of Definition 1 one automatically gets a composable notion of secure protocols by Theorem 1 below.

Theorem 1. *Given symmetric monoidal functors $F\colon \mathbf{D} \to \mathbf{C}$, $R\colon \mathbf{C} \to \mathbf{Set}$ with F strong monoidal and R lax monoidal, and an attack model $\mathcal{A}$ on $\mathbf{C}$, the class of $\mathcal{A}$-secure maps forms a wide sub-SMC of the resource theory $\int RF$ induced by RF.*

So far we have discussed security only against a single, fixed subset of dishonest parties, while in multi-party computation it is common to consider security against any subset containing e.g., at most $n/3$ or $n/2$ of the parties. However, as monoidal subcategories are closed under intersection, we immediately obtain composability against multiple attack models.

Corollary 1. *Given a non-empty family of functors $(\mathbf{D} \xrightarrow{F_i} \mathbf{C_i} \xrightarrow{R_i} \mathbf{Set})_{i \in I}$ with $R_i F_i = R_j F_j =: R$ for all $i, j \in I$ and attack models $\mathcal{A}_i$ on $\mathbf{C}_i$ for each i, the class of maps in $\int R$ that is secure against each $\mathcal{A}_i$ is a sub-SMC of $\int R$.*

Using Corollary 1 one readily obtains composability of protocols that are simultaneously secure against different attack models $\mathcal{A}_i$. Thus one could, in principle, consider composable cryptography in an n-party setting where some subsets are honest-but-curious, some might be outright malicious but have limited computational power, and some subsets might be outright malicious but not willing or able to coordinate with each other, without reproving any composition theorems.

While the security definition of f quantifies over $\mathcal{A}(f)$, which may be infinite, under suitable conditions it is sufficient to check security only on a subset of $\mathcal{A}(f)$, so that whether f is $\mathcal{A}$-secure often reduces to finitely many equations.

Definition 2. *Given $f\colon A \to B$, a subset X of $\mathcal{A}(f)$ is said to be* initial *if any $f' \in \mathcal{A}(f)$ with $\mathrm{dom}(f') = A$ can be factorized as $b \circ a$ with $a \in X$ and $b \in \mathcal{A}(\mathrm{id}_B)$.*

Theorem 2. *Let $f\colon (A, r) \to (B, s)$ define a morphism in the resource theory induced by $F\colon \mathbf{D} \to \mathbf{C}$ and $R\colon \mathbf{C} \to \mathbf{Set}$ and let $\mathcal{A}$ be an attack model on $\mathbf{C}$. If $X \subset \mathcal{A}(F(f))$ is initial, then f is $\mathcal{A}$-secure if, and only if, the security condition holds against attacks in X, i.e., if for any $f' \in X$ with $\mathrm{dom}(f') = F(A)$ there is $b \in \mathcal{A}(\mathrm{id}_{F(B)})$ such that $R(f')r = R(b)s$.*

Let us return to the example of $\mathbf{C}^n \to \mathbf{C}$ with the first k agents being honest and the final $n - k$ dishonest and collaborating. Then we can take a singleton as our initial subset of attacks on $\bar{f}$, and this is given by $\bar{f}|_{[k]} \otimes (\bigotimes_{i=k+1}^n \mathrm{id})$. Intuitively, this represents a situation where the dishonest parties $k + 1, \ldots, n$ merely stand by and forward messages between the environment and the functionality, so that initiality can be seen as explaining "completeness of the dummy adversary" [13, Claim 11] in UC-security. In this case the security condition can be equivalently phrased by saying that there exists $b \in \mathcal{A}([\mathrm{id}_b])$ satisfying the equation of fig. 2c, which reproduces the pictures of [51]. Similarly, for classical honest-but-curious adversaries one usually only considers the initial such adversary, who follows the protocol otherwise except that they keep track of the protocol transcript.

Theorem 3. *In the resource theory of n-partite states, if $(f_1, \ldots f_n)$ is secure against some subset J of $[n]$ and F is a strong monoidal, then $(Ff_1, \ldots, Ff_n)$ is secure against J as well.*

For instance, if the inclusion of classical interactive computations into quantum ones is strong monoidal, i.e., respects sequential and parallel composition (up to isomorphism), then unconditionally secure classical protocols are also secure in the quantum setting, as shown in the context of UC-security in [68, Theorem 15]. More generally, this result implies that the construction of the category of n-partite transformations secure against any fixed subset of $[n]$ is functorial in **C**, and this is in fact also true for any family of subsets of $[n]$ by Corollary 1.

4   Computational security

The discussion above has been focused on perfect security, so that the equations defining security hold exactly. This is often too high a standard for security to hope for, and consequently cryptographers routinely work with computational or approximate security. We model this in two ways. The first approach replaces equations with an equivalence relation abstracting from the idea that the end results are "computationally indistinguishable" rather than strictly equal. The latter approach amounts to working in terms of a (pseudo)metric quantifying how close we are to the ideal resource and is needed to model statements in finite-key cryptography [67]. The typical metric is given by "distinguisher advantage for polynomial-time environments", enabling one to use computational complexity theory. In a nutshell, this amounts to working with sequences of protocols and defining security by saying "for any $\epsilon > 0$, for sufficiently large n, for any attack on the nth protocol there is an attack on the target resource such that the end results are within ϵ". The first approach is mathematically straightforward and we discuss it next, while the second approach is relegated to an extended version.

Replacing strict equations with equivalence relations is easy to describe on an abstract level as an instance of the theory so far: one just assumes that **C** has a monoidal congruence $\approx$ and then works with the resource theory induced by $\mathbf{C}^n \to \mathbf{C}/\approx \xrightarrow{\hom(I,-)} \mathbf{Set}$ with similar attack models as above. More explicitly, as long as each hom-set of **C** is equipped with an equivalence relation $\approx$ that respects $\otimes$ and $\circ$ in that $f \approx f'$ and $g \approx g'$ imply $gf \approx g'f'$ (whenever defined) and $g \otimes f \approx g' \otimes f'$, then working with $\mathbf{C}^n \to \mathbf{C}/\approx \xrightarrow{\hom(I,-)} \mathbf{Set}$ results in security conditions that replace $=$ in **C** with $\approx$ throughout. If **C** describes (interactive) computational processes and $\approx$ represents computational indistinguishability (inability for any "efficient" process to distinguish between the two), one might need to replace **C** (and consequently functionalities, protocols and attacks on them) with the subcategory of **C** of efficient processes so that $\approx$ indeed results in a congruence.

5   Applications

We will now explore how the one-time pad (OTP) fits into our framework, paralleling the discussion of OTP in [47]. We will start from the category **FinStoch** of finite sets and stochastic maps between them, with $\otimes$ given by cartesian product

of sets. This is sufficient for OTP, even if more complicated and interactive cryptographic protocols will need a different starting category. However, the actual category $\mathbf{C}$ we work in is built from **FinStoch**, essentially by a tripartite variant of the "resource theory of universally-combinable processes" of [20, Section 3.4]. We will defer the detailed construction of $\mathbf{C}$ to an extended version and work in it more heuristically, allowing us to focus on the OTP.

Roughly speaking, a "basic object" of $\mathbf{C}$ consists of finite sets A_i, B_i, E_i for $i = 1, 2$, and of a map $f \colon A_1 \otimes B_1 \otimes E_1 \to A_2 \otimes B_2 \otimes E_2$ in **FinStoch**, depicted in fig. 3a. The intuition is that $\langle (A_i, B_i, E_i)_{i \in \{1,2\}}, f \rangle$ represents a box shared

(a) Box shared by Alice, Bob and Eve (b) The OTP protocol (c) A secure PRNG (d) Secure channel
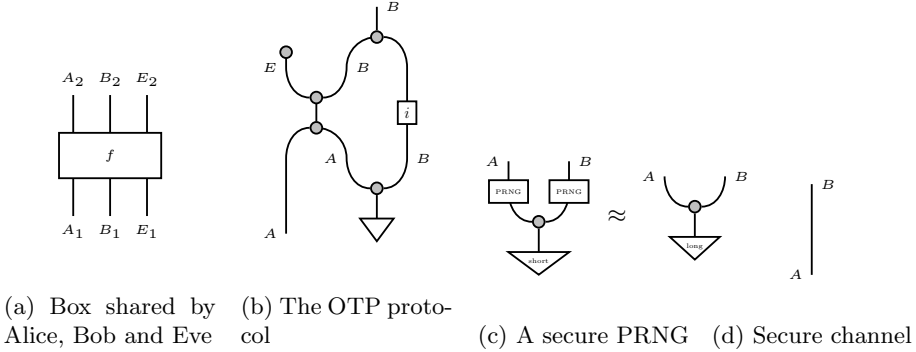
Fig. 3: Some resources and protocols

by Alice, Bob and Eve, with Alice's inputs and outputs ranging over A_1 and A_2 respectively, and similarly for Bob and Eve. We will often label the ports just by the party who controls it, and omit labeling trivial ports. For example, if fig. 4a depicts the copy map $X \to X \otimes X$ for some set X in **FinStoch**, then

(b) Alice's copy map (c) Alice broadcasting to Bob and Eve (d) Random shared key
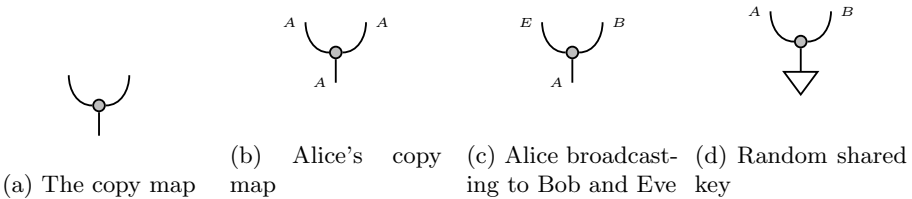
(a) The copy map

Fig. 4: Variants of the copy map

fig. 4b denotes an object of $\mathbf{C}$ representing Alice copying data privately, whereas fig. 4c denotes an object $\mathbf{C}$ that sends Alice's input unchanged to Bob and to Eve—which we view as an insecure (but authenticated) channel from Alice to Bob.

A general object of $\mathbf{C}$ then consists of a list of such basic objects, representing a list of such resources shared between Alice, Bob and Eve. A morphism of $\mathbf{C}$ is roughly speaking a way of using the starting resources and local computation by the three parties to produce the target resources: a more formal description will

be given in an extended version. In our attack model Alice and Bob are honest but Eve is dishonest, so she might do arbitrary local computation instead of whatever our protocols might prescribe.

In the version of the OTP we discuss, our starting resources consist of an insecure but authenticated channel[4] from Alice to Bob as in fig. 4c and (i.e., $\otimes$) of a random key over the same message space, shared by Alice and Bob (fig. 4d). The goal is to build a secure channel from Alice to Bob (fig. 3d) from these.

The local ingredients of OTP and the axioms they obey are depicted in fig. 5 and correspond to a Hopf algebra with an integral in a SMC. Any finite group gives rise to such a structure in **FinStoch**, with the integral given by the uniform distribution. Concretely, this means that Alice and Bob must agree on a group structure on the message space, and the fact that this multiplication forms a group and that the key is random can be captured by the equations of fig. 5.
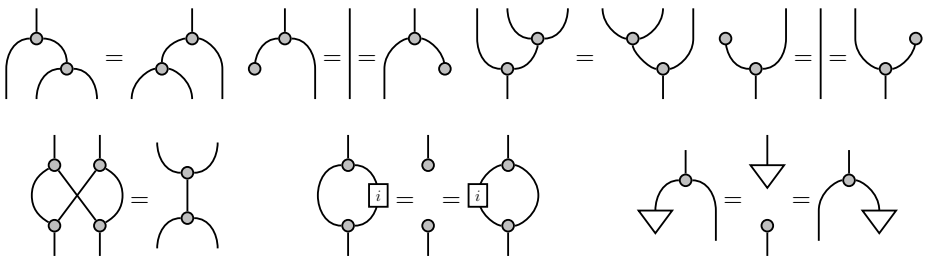

Fig. 5: Local ingredients of OTP and the axioms they obey

The OTP protocol is then depicted in fig. 3b, i.e., Alice adds the key to her message, broadcasts it to Eve and Bob. Eve deletes her part and Bob adds the inverse of the key to the ciphertext to recover the message.

To show that the protocol is secure, note that Eve has an initial attack given by just reading the ciphertext. The pictorial security proof is depicted in fig. 6. The first equation is the interaction between multiplication and copying, the second uses (co)associativity, the third one properties of inverses, the fourth and last one use unitality, and the fifth one follows from the key being random. Taken together, these show that Eve's initial attack is equal to her just producing a random message herself with Alice and Bob sharing the target resource. The correctness of the protocol can be proven similarly. Thus OTP gives a map shared key $\otimes$ authenticated channel $\rightarrow$ secure channel that is secure against Eve.

We now use this example to illustrate the use of the composition theorems. A major drawback of OTP, despite its perfect security, is the fact that one needs a key that is as long as the message. In practice, Alice and Bob might only share a short key and wish to promote it a long key. If they agree on a pseudo-random number generator (PRNG) with their key as the seed, they can map the short key to a longer key. If the PRNG is computationally secure, then the end-result is (computationally) indistinguishable from a long key, depicted in

---

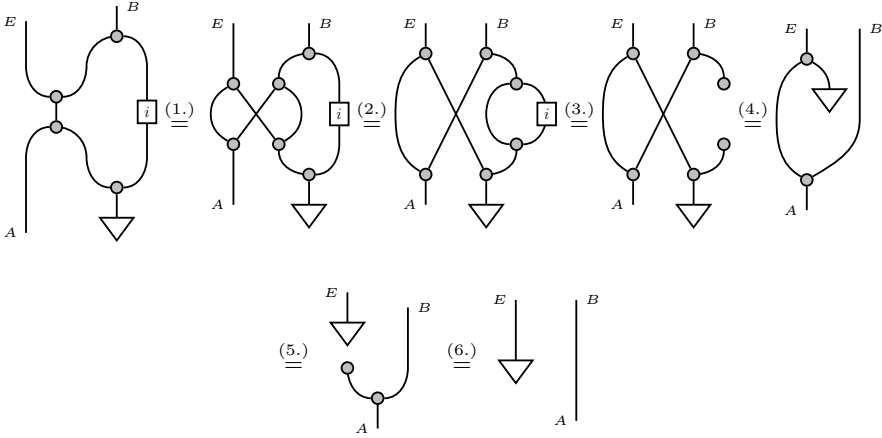[4] If the insecure channel allows Eve to tamper with the message, the analysis changes.

Fig. 6: Security proof of OTP

fig. 3c, where $\approx$ stands for computational indistinguishability. We envision the computational security of the chosen PRNG to be proven "the usual way" and not graphically—after all, we believe that our framework is there to supplement ordinary cryptographic reasoning and not to replace it. The PRNG then results in a (computationally) secure way of promoting a short shared key into a long shared key, and then the composition theorems guarantee that these protocols can be composed, resulting in the security of the stream cipher.

Composable security is a stronger constraint than stand-alone security, and indeed many cryptographic functionalities are known to be impossible to achieve "in the plain model", i.e., without set-up assumptions. A case in point is bit commitment, which was shown to be impossible in the UC-framework in [14]. This result was later generalized in [61] to show that any two-party functionality that can be realized in the plain UC-framework is "splittable". While the authors of [61] remark that their result applies more generally than just to the UC-framework, this wasn't made precise until [48][5]. We present a categorical proof of this result in our framework, which promotes the pictures "illustrating the proof" in [61] into a full proof—the main difference is that in [61] the pictures explicitly keep track of an environment trying to distinguish between different functionalities, whereas we prove our result in the case of perfect security and then deduce the asymptotic claim.

We now assume that **C**, our ambient category of interactive computations is compact closed[6]. As we are in the 2-party setting, we take our free computations

---

[5] Except that in their framework the 2-party case seems to require security constraints also when both parties cheat.

[6] We do not view this as overtly restrictive, as many theoretical models of concurrent interactive (probabilistic/quantum) computation are compact closed [18, 19, 69].

to be given by $\mathbf{C}^2$, and we consider two attack models: one where Alice cheats and Bob is honest, and one where Bob cheats and Alice is honest. We think of $\smile$ as representing a two-way communication channel, but this interpretation is not needed for the formal result.

Theorem 4. *For Alice and Bob (one of whom might cheat), if a bipartite functionality r can be securely realized from a communication channel between them, i.e., from $\smile$, then there is a g such that*

$$
\underset{r}{\overset{A \quad B}{\bigtriangledown}} = \underset{r}{\bigtriangledown}\, \boxed{g}\, \underset{r}{\bigtriangledown}. \tag{$*$}
$$

Proof. If a protocol (f_A, f_B) achieves this, security constraints give us s_A, s_B

such that $\boxed{f_A}\,\smile = \underset{r}{\overset{s_B}{\bigtriangledown}}$ and $\smile\,\boxed{f_B} = \underset{r}{\overset{s_A}{\bigtriangledown}}$

so that $\underset{r}{\bigtriangledown} = \boxed{f_A}\,\boxed{f_B} = \boxed{f_A}\,\frown\,\boxed{f_B} = \underset{r}{\overset{s_B}{\bigtriangledown}}\,\underset{r}{\overset{s_A}{\bigtriangledown}}$

Corollary 2. *Given a compact closed $\mathbf{C}$ modeling computation in which wires model communication channels, (composable) bit commitment and oblivious transfer are impossible in that model without setup, even asymptotically in terms of distinguisher advantage.*

Proof. If r represents bit commitment from Alice to Bob, it does not satisfy the equation required by Theorem 4 for any g, and the two sides of $(*)$ can be distinguished efficiently with at least probability $1/2$. Indeed, take any f and let us compare the two sides of $(*)$: if the distinguisher commits to a random bit b, then Bob gets a notification of this on the left hand-side, so that f has to commit to a bit on the right side of $(*)$ to avoid being distinguished from the left side. But this bit coincides with b with probability at most $1/2$, so that the difference becomes apparent at the reveal stage. The case of OT is similar.

We now discuss a similar result in the tripartite case, which rules out building a broadcasting channel from pairwise channels securely against any single party cheating. In [46] comparable pictures are used to illustrate the official, symbolically rather involved, proof, whereas in our framework the pictures are the proof. Another key difference is that [46] rules out broadcasting directly, whereas we show that any tripartite functionality realizable from pairwise channels satisfies some equations, and then use these equations to rule out broadcasting.

Formally, we are working with the resource theory given by $\mathbf{C}^3 \xrightarrow{\otimes} \mathbf{C} \xrightarrow{\mathrm{hom}(I,-)}$ **Set** where $\mathbf{C}$ is an SMC, and reason about protocols that are secure against three

kinds of attacks: one for each party behaving dishonestly while the rest obey the protocol. Note that we do not need to assume compact closure for this result, and the result goes through for any state on $A \otimes A$ shared between each pair of parties: we will denote such a state by $\smile$ by convention.

Theorem 5. *If a tripartite functionality r can be realized from each pair of parties sharing a state $\smile$, securely against any single party, then there are simulators s_A, s_B, s_C such that*

Proof. Any tripartite protocol building on top of each pair of parties sharing $\smile$ can be drawn as in the left side of
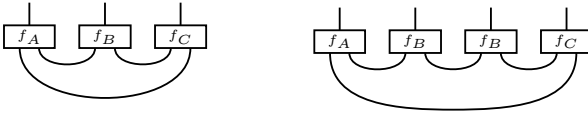
Consider now the morphism in **C** depicted on the right: it can be seen as the result of three different attacks on the protocol (f_A, f_B, f_C) in $\mathbf{C}^3$: one where Alice cheats and performs f_A and f_B (and the wire connecting them), one where Bob performs f_B twice, and one where Charlie performs f_B and f_C. The security of (f_A, f_B, f_C) against each of these gives the required simulators.

Corollary 3. *Given a SMC **C** modeling interactive computation, and a state $\smile$ on $A \otimes A$ modeling pairwise communication, it is impossible to build broadcasting channels securely (even asymptotically in terms of distinguisher advantage) from pairwise channels.*

Proof. We show that a channel r that enables Bob to broadcast an input bit to Alice and Charlie never satisfies the required equations for any s_A, s_B, s_C. Indeed, assume otherwise and let the environment plug "broadcast 0" and "broadcast 1" to the two wires in the middle. The leftmost picture then says that Charlie receives 1, the rightmost picture implies that Alice gets 0 and the middle picture that Alice and Bob get the same output (if anything at all)—a contradiction. Indeed, one cannot satisfy all of these simultaneously with high probability, which rules out an asymptotic transformation.

6   Outlook

We have presented a categorical framework providing a general, flexible and mathematically robust way of reasoning about composability in cryptography. Besides contributing a further approach to composable cryptography and potentially helping with cross-talk and comparisons between existing approaches [12], we believe that the current work opens the door for several further questions.

First, due to the generality of our approach we hope that one can, besides honest and malicious participants, reason about more refined kinds of adversaries composably. Indeed, we expect that Definition 1 is general enough to capture e.g., honest-but-curious adversaries[7]. It would also be interesting to see if this captures even more general attacks, e.g., situations where the sets of participants and dishonest parties can change during the protocol. This might require understanding our axiomatization of attack models more structurally and perhaps generalizing it. Does this structure (or a variant thereof) already arise in category theory? While we define an attack model on a category, perhaps one could define an attack model on a (strong) monoidal functor F, the current definition being recovered when $F = \mathrm{id}$.

Second, we expect that rephrasing cryptographic questions categorically would enable more cross-talk between cryptography and other fields already using category theory as an organizing principle. For instance, many existing approaches to composable cryptography develop their own models of concurrent, asynchronous, probabilistic and interactive computations. As categorical models of such computation exist in the context of game semantics [18,19,69], one is left wondering whether the models of the semanticists' could be used to study and answer cryptographic questions, or conversely if the models developed by cryptographers contain valuable insights for programming language semantics.

Besides working inside concrete models—which ultimately blends into "just doing composable cryptography"—one could study axiomatically how properties of a category relate to cryptographic properties in it. As a specific conjecture in this direction, one might hope to talk about honest-but-curious adversaries at an abstract level using environment structures [21], that axiomatize the idea of deleting a system. Similarly, having agents purify their actions is an important tool in quantum cryptography [45]—can categorical accounts of purification [15, 21,24] elucidate this?

Finally, we hope to get more mileage out of the tools brought in with the categorical viewpoint. For instance, can one prove further no-go results pictorially? More specifically, given the impossibility results for two and three parties, one wonders if the "only topology matters" approach of string diagrams can be used to derive general impossibility results for n parties sharing pairwise channels. Similarly, while diagrammatic languages have been used to reason about positive cryptographic results in the stand-alone setting [9,10,41], can one push such approaches further now that composable security definitions have a clear categorical meaning? Besides the graphical methods, thinking of cryptography as a resource theory suggests using resource-theoretic tools such as monotones. While monotones have already been applied in cryptography [70], a full understanding of cryptographically relevant monotones is still lacking.

---

[7] Heuristically speaking this is the case: an honest-but-curious attack on $g \circ f$ should be factorizable as one on g and one on f, and similarly an honest-but-curious attack on $g \otimes f$ should be factorisable into ones on g and f that then forward their transcripts to an attack on $\mathrm{id} \otimes \mathrm{id}$.

References

1. Abramsky, S., Barbosa, R.S., Karvonen, M., Mansfield, S.: A comonadic view of simulation and quantum resources. In: 2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS). IEEE (2019). https://doi.org/10.1109/LICS.2019.8785677

2. Backes, M., Pfitzmann, B., Waidner, M.: A general composition theorem for secure reactive systems. In: 1st Theory of Cryptography Conference—TCC 2004. pp. 336–354 (2004). https://doi.org/10.1007/978-3-540-24638-1_19

3. Backes, M., Pfitzmann, B., Waidner, M.: The reactive simulatability (RSIM) framework for asynchronous systems. Information and Computation **205**(12), 1685–1720 (2007). https://doi.org/10.1016/j.ic.2007.05.002

4. Ben-Or, M., Canetti, R., Goldreich, O.: Asynchronous secure computation. In: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing. pp. 52–61 (1993). https://doi.org/10.1145/167088.167109

5. Ben-Or, M., Horodecki, M., Leung, D.W., Mayers, D., Oppenheim, J.: The universal composable security of quantum key distribution. In: 2nd Theory of Cryptography Conference—TCC 2005. pp. 386–406 (2005). https://doi.org/10.1007/978-3-540-30576-7_21

6. Ben-Or, M., Mayers, D.: General security definition and composability for quantum & classical protocols (2004), https://arxiv.org/abs/quant-ph/0409062

7. Bennett, C.H., Brassard, G.: Quantum cryptography: Public key distribution and coin tossing. In: International Conference on Computers, Systems and Signal Processing. pp. 175–179 (1984)

8. Biham, E., Boyer, M., Boykin, P.O., Mor, T., Roychowdhury, V.: A proof of the security of quantum key distribution (extended abstract). In: 32nd Annual ACM Symposium on Theory of Computing—STOC 2000. pp. 715 – 724 (2000). https://doi.org/10.1145/335305.335406

9. Breiner, S., Kalev, A., Miller, C.A.: Parallel self-testing of the GHZ state with a proof by diagrams. In: Proceedings of QPL 2018. Electronic Proceedings in Theoretical Computer Science, vol. 287, pp. 43–66 (2018). https://doi.org/10.4204/eptcs.287.3

10. Breiner, S., Miller, C.A., Ross, N.J.: Graphical methods in device-independent quantum cryptography. Quantum **3**, 146 (2019). https://doi.org/10.22331/q-2019-05-27-146

11. Broadbent, A., Fitzsimons, J., Kashefi, E.: Universal blind quantum computation. In: 50th Annual Symposium on Foundations of Computer Science—FOCS 2009. pp. 517–526 (2009). https://doi.org/10.1109/FOCS.2009.36

12. Camenisch, J., Küsters, R., Lysyanskaya, A., Scafuro, A.: Practical Yet Composably Secure Cryptographic Protocols (Dagstuhl Seminar 19042). Dagstuhl Reports **9**(1), 88–103 (2019). https://doi.org/10.4230/DagRep.9.1.88

13. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd Annual Symposium on Foundations of Computer Science—FOCS 2001. pp. 136–145 (2001). https://doi.org/10.1109/SFCS.2001.959888

14. Canetti, R., Fischlin, M.: Universally composable commitments. In: Advances in cryptology—CRYPTO 2001. pp. 19–40. Springer (2001). https://doi.org/10.1007/3-540-44647-8_2

15. Chiribella, G., D'Ariano, G.M., Perinotti, P.: Probabilistic theories with purification. Physical Review A **81**(6) (Jun 2010). https://doi.org/10.1103/physreva.81.062348

16. Chitambar, E., Gour, G.: Quantum resource theories. Reviews of Modern Physics **91**(2), 025001 (2019). https://doi.org/10.1103/revmodphys.91.025001
17. Chitambar, E., Leung, D., Mančinska, L., Ozols, M., Winter, A.: Everything you always wanted to know about LOCC (but were afraid to ask). Communications in Mathematical Physics **328**(1), 303–326 (2014). https://doi.org/10.1007/s00220-014-1953-9
18. Clairambault, P., De Visme, M., Winskel, G.: Game semantics for quantum programming. Proceedings of the ACM on Programming Languages **3**(POPL), 1–29 (2019). https://doi.org/10.1145/3290345
19. Clairambault, P., de Visme, M., Winskel, G.: Concurrent quantum strategies. In: International Conference on Reversible Computation. pp. 3–19. Springer (2019). https://doi.org/10.1007/978-3-030-21500-2_1
20. Coecke, B., Fritz, T., Spekkens, R.W.: A mathematical theory of resources. Information and Computation **250**, 59–86 (2016). https://doi.org/10.1016/j.ic.2016.02.008
21. Coecke, B., Perdrix, S.: Environment and classical channels in categorical quantum mechanics. Logical Methods in Computer Science **Volume 8, Issue 4** (2012). https://doi.org/10.2168/LMCS-8(4:14)2012
22. Coecke, B., Wang, Q., Wang, B., Wang, Y., Zhang, Q.: Graphical calculus for quantum key distribution (extended abstract). Electronic Notes in Theoretical Computer Science **270**(2), 231–249 (2011). https://doi.org/10.1016/j.entcs.2011.01.034
23. Cruttwell, G., Gavranović, B., Ghani, N., Wilson, P., Zanasi, F.: Categorical foundations of gradient-based learning (2021), https://arxiv.org/abs/2103.01931
24. Cunningham, O., Heunen, C.: Purity through factorisation. In: Proceedings of QPL 2017. Electronic Proceedings in Theoretical Computer Science, vol. 266, pp. 315–328 (2017). https://doi.org/10.4204/EPTCS.266.20
25. Datta, A., Derek, A., Mitchell, J.C., Pavlovic, D.: A derivation system for security protocols and its logical formalization. In: 16th IEEE Computer Security Foundations Workshop, 2003. Proceedings. pp. 109–125. IEEE (2003). https://doi.org/10.1109/csfw.2003.1212708
26. Datta, A., Derek, A., Mitchell, J.C., Pavlovic, D.: Secure protocol composition. Electronic Notes in Theoretical Computer Science **83**, 201–226 (2003). https://doi.org/10.1016/s1571-0661(03)50011-1
27. Datta, A., Derek, A., Mitchell, J.C., Pavlovic, D.: A derivation system and compositional logic for security protocols. Journal of Computer Security **13**(3), 423–482 (Aug 2005). https://doi.org/10.3233/JCS-2005-13304
28. Datta, A., Derek, A., Mitchell, J.C., Roy, A.: Protocol composition logic (PCL). Electronic Notes in Theoretical Computer Science **172**, 311–358 (Apr 2007). https://doi.org/10.1016/j.entcs.2007.02.012
29. Durgin, N., Mitchell, J., Pavlovic, D.: A compositional logic for protocol correctness. In: Proceedings. 14th IEEE Computer Security Foundations Workshop, 2001. IEEE (2001). https://doi.org/10.1109/csfw.2001.930150
30. Durgin, N., Mitchell, J., Pavlovic, D.: A compositional logic for proving security properties of protocols. Journal of Computer Security **11**(4), 677–721 (Oct 2003). https://doi.org/10.3233/JCS-2003-11407
31. Fong, B., Spivak, D., Tuyeras, R.: Backprop as functor: A compositional perspective on supervised learning. In: 2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS) (2019). https://doi.org/10.1109/lics.2019.8785665
32. Fritz, T.: Resource convertibility and ordered commutative monoids. Mathematical Structures in Computer Science **27**(6), 850–938 (2015). https://doi.org/10.1017/s0960129515000444

33. Gavranović, B.: Compositional deep learning (2019), https://arxiv.org/abs/1907.08292
34. Goldwasser, S., Micali, S.: Probabilistic encryption. Journal of Computer and System Sciences **28**(2), 270–299 (1984). https://doi.org/10.1016/0022-0000(84)90070-9
35. Heunen, C.: Compactly accessible categories and quantum key distribution. Logical Methods in Computer Science **4**(4) (2008). https://doi.org/10.2168/lmcs-4(4:9)2008
36. Hillebrand, A.: Superdense coding with GHZ and quantum key distribution with W in the ZX-calculus. In: Proceedings of QPL 2011. Electronic Proceedings in Theoretical Computer Science, vol. 95, pp. 103–121 (2011). https://doi.org/10.4204/EPTCS.95.10
37. Hines, P.M.: A diagrammatic approach to information flow in encrypted communication (2020). https://doi.org/10.1007/978-3-030-62230-5_9
38. Hofheinz, D., Shoup, V.: GNUC: A new universal composability framework. Journal of Cryptology **28**(3), 423–508 (2015). https://doi.org/10.1007/s00145-013-9160-y
39. Horodecki, M., Oppenheim, J.: (Quantumness in the context of) Resource Theories. International Journal of Modern Physics B **27**(01n03), 1345019 (2013). https://doi.org/10.1142/s0217979213450197
40. Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally composable synchronous computation. In: Theory of Cryptography, pp. 477–498. Springer (2013). https://doi.org/10.1007/978-3-642-36594-2_27
41. Kissinger, A., Tull, S., Westerbaan, B.: Picture-perfect quantum key distribution (2017), https://arxiv.org/abs/1704.08668
42. König, R., Renner, R., Bariska, A., Maurer, U.: Small accessible quantum information does not imply security. Physical Review Letters **98**(14), 140502 (2007). https://doi.org/10.1103/PhysRevLett.98.140502
43. Küsters, R., Tuengerthal, M., Rausch, D.: The IITM model: a simple and expressive model for universal composability. Journal of Cryptology **33**(4), 1461–1584 (2020). https://doi.org/10.1007/s00145-020-09352-1
44. Liao, K., Hammer, M.A., Miller, A.: ILC: a calculus for composable, computational cryptography. In: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 640–654. ACM (Jun 2019). https://doi.org/10.1145/3314221.3314607
45. Lo, H.K., Chau, H.F.: Is quantum bit commitment really possible? Physical Review Letters **78**(17), 3410–3413 (1997). https://doi.org/10.1103/PhysRevLett.78.3410
46. Matt, C., Maurer, U., Portmann, C., Renner, R., Tackmann, B.: Toward an algebraic theory of systems. Theoretical Computer Science **747**, 1–25 (2018). https://doi.org/10.1016/j.tcs.2018.06.001
47. Maurer, U.: Constructive cryptography–a new paradigm for security definitions and proofs. In: Joint Workshop on Theory of Security and Applications—TOSCA 2011. pp. 33–56 (2011). https://doi.org/10.1007/978-3-642-27375-9_3
48. Maurer, U., Renner, R.: Abstract cryptography. In: Innovations in Computer Science—ICS 2011 (2011)
49. Mayers, D.: The trouble with quantum bit commitment (1996), http://arxiv.org/abs/quant-ph/9603015
50. Mayers, D.: Unconditional security in quantum cryptography. Journal of the ACM **48**(3), 351–406 (2001). https://doi.org/10.1145/382780.382781

51. Micciancio, D., Tessaro, S.: An equational approach to secure multi-party computation. In: 4th Conference on Innovations in Theoretical Computer Science—ITCS 2013. pp. 355–372 (2013). https://doi.org/10.1145/2422436.2422478

52. Mifsud, A., Milner, R., Power, J.: Control structures. In: Proceedings of Tenth Annual IEEE Symposium on Logic in Computer Science. pp. 188–198. IEEE (1995). https://doi.org/10.1109/lics.1995.523256

53. Moeller, J., Vasilakopoulou, C.: Monoidal Grothendieck construction. Theory and Applications of Categories **35**(31), 1159–1207 (2020)

54. Müller-Quade, J., Renner, R.: Composability in quantum cryptography. New Journal of Physics **11**(8), 085006 (2009). https://doi.org/10.1088/1367-2630/11/8/085006

55. Pavlovic, D.: Categorical logic of names and abstraction in action calculi. Mathematical Structures in Computer Science **7**(6), 619–637 (1997). https://doi.org/10.1017/S0960129597002296

56. Pavlovic, D.: Tracing the man in the middle in monoidal categories. In: Coalgebraic Methods in Computer Science. pp. 191–217. Springer (2012). https://doi.org/10.1007/978-3-642-32784-1_11

57. Pavlovic, D.: Chasing diagrams in cryptography. In: Casadio, C., Coecke, B., Moortgat, M., Scott, P. (eds.) Categories and Types in Logic, Language, and Physics: Essays Dedicated to Jim Lambek on the Occasion of His 90th Birthday, pp. 353–367. Springer Berlin Heidelberg, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54789-8_19

58. Pfitzmann, B., Waidner, M.: A model for asynchronous reactive systems and its application to secure message transmission. In: 2001 IEEE Symposium on Security and Privacy—S&P 2001. pp. 184–200 (2000). https://doi.org/10.1109/SECPRI.2001.924298

59. Portmann, C., Matt, C., Maurer, U., Renner, R., Tackmann, B.: Causal boxes: quantum information-processing systems closed under composition. IEEE Transactions on Information Theory **63**(5), 3277–3305 (2017). https://doi.org/10.1109/TIT.2017.2676805

60. Portmann, C., Renner, R.: Cryptographic security of quantum key distribution (2014), https://arxiv.org/abs/1409.3525

61. Prabhakaran, M., Rosulek, M.: Cryptographic complexity of multi-party computation problems: Classifications and separations. In: Advances in Cryptology—CRYPTO 2008. pp. 262–279 (2008). https://doi.org/10.1007/978-3-540-85174-5_15

62. Renner, R.: Security of quantum key distribution. International Journal of Quantum Information **06**(01), 1–127 (2005). https://doi.org/10.1142/S0219749908003256

63. Selinger, P.: A survey of graphical languages for monoidal categories. In: New structures for physics, pp. 289–355. Springer (2010). https://doi.org/10.1007/978-3-642-12821-9_4

64. Shor, P.W., Preskill, J.: Simple proof of security of the BB84 quantum key distribution protocol. Physical Review Letters **85**(2), 441–444 (2000). https://doi.org/10.1103/physrevlett.85.441

65. Stay, M., Vicary, J.: Bicategorical semantics for nondeterministic computation. In: Proceedings of the Twenty-ninth Conference on the Mathematical Foundations of Programming Semantics, MFPS XXIX. Electronic Notes in Theoretical Computer Science, vol. 298, pp. 367 – 382 (2013). https://doi.org/10.1016/j.entcs.2013.09.022

66. Sun, X., He, F., Wang, Q.: Impossibility of quantum bit commitment, a categorical perspective. Axioms **9**(1), 28 (2020). https://doi.org/10.3390/axioms9010028

67. Tomamichel, M., Lim, C.C.W., Gisin, N., Renner, R.: Tight finite-key analysis for quantum cryptography. Nature Communications **3**,   634 (2012). https://doi.org/10.1038/ncomms1631
68. Unruh, D.: Universally composable quantum multi-party computation. In: Advances in Cryptology—EUROCRYPT 2010. pp. 486–505 (2010). https://doi.org/10.1007/978-3-642-13190-5_25
69. Winskel, G.: Distributed probabilistic and quantum strategies. Electronic Notes in Theoretical Computer Science **298**, 403–425 (2013). https://doi.org/10.1016/j.entcs.2013.09.024
70. Wolf, S., Wullschleger, J.: New monotones and lower bounds in unconditional two-party computation. IEEE Transactions on Information Theory **54**(6), 2792–2797 (2008). https://doi.org/10.1109/tit.2008.921674

DyNetKAT: An Algebra of Dynamic Networks *

Georgiana Caltais[1]($\boxtimes$)(ID), Hossein Hojjat[2](ID), Mohammad Reza Mousavi[3](ID), and
Hünkar Can Tunç[4](ID)

[1] University of Konstanz, Germany & University of Twente, The Netherlands
`g.g.c.caltais@utwente.nl`
[2] TeIAS, Khatam University & University of Tehran, Iran
`hojjat@ut.ac.ir`
[3] King's College London, UK
`mohammad.mousavi@kcl.ac.uk`
[4] University of Konstanz, Germany & Aarhus University, Denmark
`tunc@cs.au.dk`

Abstract. We introduce a formal language for specifying dynamic updates for Software Defined Networks. Our language builds upon Network Kleene Algebra with Tests (NetKAT) and adds constructs for synchronisations and multi-packet behaviour to capture the interaction between the control- and data-plane in dynamic updates. We provide a sound and ground-complete axiomatisation of our language. We exploit the equational theory and provide an efficient method for reasoning about safety properties. We implement our equational theory in DyNetiKAT – a tool prototype, based on the Maude Rewriting Logic and the NetKAT tool, and apply it to a case study. We show that we can analyse the case study for networks with hundreds of switches using our tool prototype.

Keywords: Software Defined Networks · Dynamic Updates · Dynamic Network Reconfiguration · NetKAT · Process Algebra · Equational Reasoning.

1   Introduction

Software-Defined Networking (SDN) is an approach to networking that enables the network to be centrally programmed. There is a spectrum of mathematically inspired network programming languages that varies between those with a small number of language constructs and those with expressive language design which allow them to support more networking features. Flowlog [16] and Kinetic [12] are points on the more expressive side of the spectrum, which provide support for formal reasoning based on SAT-solving and model checking, respectively.

NetKAT [3,10] is an example of a minimalist language based on Kleene algebra with tests that has a sound and complete equational theory. While the core of the language is very simple with a few number of operators, the language has been extended in various ways to support different aspects of networking such as congestion control [9], history-based routing [6] and higher-order functions [20].

Our starting point is NetKAT, because it provides a clean and analysable framework for specifying SDNs. The minimalist design of NetKAT does not cater for some common (failure) patterns in SDNs, particularly those arising from dynamic reconfiguration and the interaction between the data- and control-plane flows. In [13], the authors have proposed an extension to NetKAT to support stateful network updates. The extension embraces the notion of mutable state which is in contrast to the pure functional nature of the language. The purpose of this paper is to propose an extension of NetKAT to support dynamic and stateful behaviours. On the one hand, we preserve the big-step denotational semantics of NetKAT-specific constructs enabling, for instance, handling flow table updates atomically, in the spirit of [17]. On the other hand, we extend NetKAT in a modular fashion, to integrate concurrent SDN behaviours such as dynamic updates, defined via a small-step operational semantics. To this end, we pledge to keep the minimalistic design of NetKAT by adding only a few new operators. Furthermore, our extension does not contradict the nature of the language. DyNetKAT is a conservative extension [2] of NetKAT that enables reusing in a modular fashion frameworks previously developed for NetKAT. Examples include the NetKAT axiomatisation in [3], for instance.

A number of concurrent extensions of NetKAT have been introduced to date [11,18,21]. These extensions followed different design decisions than the present paper and a comparison of their approaches with ours is provided in Section 2; however, the most important difference lies in the fact that inspired by earlier abstractions in this domain [17], we were committed to create different layers for data-plane flows and dynamic updates such that every data-plane packet observes a single set of flow tables through its flight through the network. This allowed us, unlike the earlier approaches, to build a layer on top of NetKAT without modifying its semantics. Although our presentation in this paper is based on NetKAT, we envisage that our concurrency layer can be modularly (in the sense of Modular SOS [14]) used for other network programming languages in the above-mentioned spectrum. We leave a more careful investigation of the modularity on other network languages for future work.

Running Example. To illustrate our language concepts, we focus on modelling with DyNetKAT an example of a stateful firewall that involves dynamically updating the flow table. The example is overly simplified for the purpose of presentation. Towards the end of this paper and also in the extended version [7], we treat more complex and larger-scale case studies to evaluate the applicability and analysability of our language.

A firewall is supposed to protect the intranet of an organisation from unauthorised access from the Internet. However, due to certain requests from the intranet, it should be able to open up connections from the Internet to intranet.
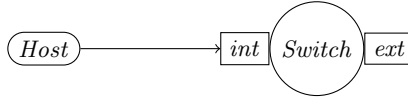
Fig. 1: Stateful Firewall

An example is when a user within the intranet requests a secure connection to a node on the Internet; in that case, the response from the node should be allowed to enter the intranet. The behaviour of updating the flow tables with respect to some events in the network such as receiving a specific packet is a challenging phenomenon for languages such as NetKAT.

Figure 1 shows a simplified version of the stateful firewall network. Note that we are not interested in the flow of packets but interested in the flow update. In this version, the *Switch* does not allow any packet from the port *ext* to *int* at the beginning. When the *Host* sends a request to the *Switch* it opens up the connection.

Our Contributions. The contributions of this paper are summarised as follows. (a) We define the syntax and operational semantics of a dynamic extension of NetKAT that allows for modelling and reasoning about control-plane updates and their interaction with data-plane flows (Sections 2.3, 2.4). (b) We give a sound and ground-complete axiomatisation of our language (Section 3). (c) We devise analysis methods for reasoning about flow properties using our axiomatisation, apply them on examples from the domain and gather and analyse evidence of applicability and efficiency for our approach (Sections 4, 5, 6).

2   Language Design

In what follows, we provide a brief overview of the NetKAT syntax and semantics [3]. Then, we motivate our language design decisions, we introduce the syntax of DyNetKAT and its underlying semantics, and provide the corresponding encoding of our running example.

2.1   Brief Overview of NetKAT

We proceed by first introducing some basic notions used throughout the paper.

Definition 1 (Network Packets.) *Let $F = \{f_1, \ldots, f_n\}$ be a set of field names f_i with $i \in \{1, \ldots n\}$. We call* network packet *a partial function in $F \to \mathbb{N}$ that maps field names in F to values in $\mathbb{N}$. We use σ, σ' to range over network packets. We write, for instance, $\sigma(f_i) = v_i$ to denote a test checking whether the value of f_i in σ is v_i. Furthermore, we write $\sigma[f_i := n_i]$ to denote the assignment of f_i to v_i in σ. A (possibly empty)* list of packets *is defined as a partial function from natural numbers to packets, where the natural number in the domain denotes the position of the packet in the list such that the domain of the function forms an interval starting from 0. The* empty list *is denoted by $\langle \rangle$ and is defined as the empty function (the function with the empty set as its domain). Let σ be a*

packet and l be a list, then $\sigma :: l$ is the list l' in which σ is at position 0 in l', i.e., $l'(0) = \sigma$, and $l'(i+1) = l(i)$, for all i in the domain of l.

NetKAT Syntax:
$$Pr ::= \mathbf{0} \mid \mathbf{1} \mid f = n \mid Pr + Pr \mid Pr \cdot Pr \mid \neg Pr$$
$$N ::= Pr \mid f \leftarrow n \mid N + N \mid N \cdot N \mid N^* \mid \mathbf{dup}$$

NetKAT Semantics:

$$[\![\mathbf{1}]\!](h) \triangleq \{h\} \qquad\qquad [\![p \cdot q]\!](h) \triangleq ([\![p]\!] \bullet [\![q]\!])(h)$$
$$[\![\mathbf{0}]\!](h) \triangleq \{\} \qquad\qquad [\![p^*]\!](h) \triangleq \bigcup_{i \in N} F^i(h)$$
$$[\![f = n]\!](\sigma::h) \triangleq \begin{cases} \{\sigma::h\} & \text{if } \sigma(f) = n \\ \{\} & \text{otherwise} \end{cases} \qquad F^0(h) \triangleq \{h\}$$
$$F^{i+1}(h) \triangleq ([\![p]\!] \bullet F^i)(h)$$
$$[\![\neg a]\!](h) \triangleq \{h\} \setminus [\![a]\!](h) \qquad\qquad (f \bullet g)(x) \triangleq \bigcup \{g(y) \mid y \in f(x)\}$$
$$[\![f \leftarrow n]\!](\sigma::h) \triangleq \{\sigma[f := n]::h\} \qquad [\![\mathbf{dup}]\!](\sigma::h) \triangleq \{\sigma::(\sigma::h)\}$$
$$[\![p + q]\!](h) \triangleq [\![p]\!](h) \cup [\![q]\!](h)$$

Fig. 2: NetKAT: Syntax and Semantics [3]

In Figure 2, we recall the NetKAT syntax and semantics [3]. The predicate for dropping a packet is denoted by $\mathbf{0}$, while passing on a packet (without any modification) is denoted by $\mathbf{1}$. The predicate checking whether the field f of a packet has value n is denoted by $(f = n)$; if the predicate fails on the current packet it results on dropping the packet, otherwise it will pass the packet on. Disjunction and conjunction between predicates are denoted by $Pr + Pr$ and $Pr \cdot Pr$, respectively. Negation is denoted by $\neg Pr$. Predicates are the basic building blocks of NetKAT policies and hence, a predicate is a policy by definition. The policy that modifies the field f of the current packet to take value n is denoted by $(f \leftarrow n)$. A multicast behaviour of policies is denoted by $N + N$, while sequencing policies (to be applied on the same packet) are denoted by $N \cdot N$. The repeated application of a policy is encoded as N^*. The construct $\mathbf{dup}$ simply makes a copy of the current network packet.

In [3], lists of packets are referred to as *histories*. Let H stand for the set of packet histories, and $\mathcal{P}(H)$ denote the powerset of H. More formally, the denotational semantics of NetKAT policies is inductively defined via the semantic map $[\![-]\!] : N \to (H \to \mathcal{P}(H))$ in Figure 2, where N stands for the set of NetKAT policies, $h \in H$ is a packet history, $a \in Pr$ denotes a NetKAT predicate and $\sigma \in F \to \mathbb{N}$ is a network packet.

For a reminder, the equational axioms of NetKAT include the Kleene Algebra axioms, Boolean Algebra axioms and the so-called Packet Algebra axioms that handle NetKAT networking specific constructs such as field assignments and $\mathbf{dup}$. In this paper, we write E_{NK} to denote the NetKAT axiomatisation [3].

2.2 Design Decisions

Our main motivation behind DyNetKAT is to have a *minimalist* language that can model *control-plane* and *data-plane* network traffic and their interaction. Our choice for a minimal language is motivated by our desire to use our language as a basis for scalable analysis. We would like to be able to compile major

practical languages into ours. Our minimal design helps us reuse much of the well-known scalable analysis techniques. Regarding its modelling capabilities, we are interested in modelling the stateful and dynamic behaviour of networks emerging from these interactions. We would like to be able to model control messages, connections between controllers and switches, data packets, links among switches, and model and analyse their interaction in a seamless manner.

Based on these motivations, we start off with NetKAT as a fundamental and minimal network programming language, which allows us to model the basic policies governing the network traffic. The choice of NetKAT, in addition to its minimalist nature, is motivated by its rigorous semantics and equational theory, and the existing techniques and tools for its analysis. This motivates our next design constraint, namely, to build upon NetKAT in a hierarchical manner and without redefining its semantics. This constraint should not be taken lightly as the challenges in the recent concurrent extensions of NetKAT demonstrated [11, 18, 21]. We will elaborate on this point, in the presentation of our syntax and semantics. We can achieve this thanks to the abstractions introduced in the domain [17] that allow for a neat layering of data-plane and control-plan flows such that every data-plane flow sees one set of flow-tables in its flight through the network.

We introduce a few extensions and modifications to cater for the phenomena we desire to model in our extension regarding control-plane and dynamic and stateful behaviour, as follows. (a) *Parallel composition and synchronisation*: we introduce a basic mechanism for parallel composition based on handshake synchronisation with the possibility of communicating a network program (a flow table). The point of adding parallel composition is to have parallel controllers and switches as separate syntactic entities: controllers trigger reconfigurations and switches accept different types of reconfiguration and change their continuation accordingly. (b) *Guarded recursion*: we introduce the concept of recursion to model the (persistent) dynamic changes that result from control messages and stateful behaviour. In other words, recursion is used to model the new state of the flow tables. An alternative modelling construct could have been using "global" variables and guards, but we prefer recursion due to its neat algebraic representation. We restrict the use of recursion to guarded recursion, that is a policy should be applied before changing state to a new recursive definition, in order to remain within a decidable and analyse-able realm. A natural extension of our framework could introduce formal parameters and parameterised recursive variables; this future extension is orthogonal to our existing extensions and in this paper, we go for a minimal extension in which the parameters are coded in variable names. (c) *Multi-packet semantics*: we introduce the semantics of treating a list of packets, which is essential for studying the interaction between control- and data plane packets. This is in contrast with NetKAT where a single-packet semantics is introduced. The introduction of multi-packet semantics also called for a new operator to denote the end of applying a flow-table to the current packet and proceeding with the next packet (possibly with the modified flow-table in place). This is our new sequential composition operator, denoted

by ";". Inspired by the abstractions in the software defined networking community [17], we assume each packet is processed either using the configuration in place prior to the update, or the configuration in place after the update, but never a mixture of the two.

2.3 DyNetKAT Syntax

As already mentioned, NetKAT provides the possibility of recording the individual "hops" that packets take as they go through the network by using the so-called **dup** construct. The latter keeps track of the state of the packet at each intermediate hop. As a brief reminder of the approach in [3]: assume a NetKAT switch policy p and a topology t, together with an ingress *in* and an egress *out*. Checking whether *out* is reachable from *in* reduces to checking: $in \cdot \mathbf{dup} \cdot (p \cdot t \cdot \mathbf{dup})^* \cdot out \not\equiv \mathbf{0}$ (see Definition 2 and Theorem 4 in [3]). Furthermore, as shown in [10], **dup** plays a crucial role in devising the NetKAT language semantics in a coalgebraic fashion, via Brzozowski-like derivatives on top of NetKAT coalgebras (or NetKAT automata) corresponding to NetKAT expressions.

We decided to depart from NetKAT in this respect, due to our important constraint not to redefine the NetKAT semantics: the **dup** expression allows for observable intermediate steps that result from incomplete application of flow-tables and in concurrency scenarios, the same data packet may become subject to more than one flow table due to the concurrent interactions with the control plane. For this semantics to be compositional, one needs to define a small step operational semantics in such a way that the small steps in predicate evaluation also become visible (see our past work on compositionality of SOS with data on such constraints [15]). This will first break our constraint in building upon NetKAT semantics and secondly, due to the huge number of possible interleavings, make the resulting state-space intractable for analysis.

In addition to the argumentation above, note that similarly to the approach in [3], we work with packet fields ranging over finite domains. Consequently, our analyses can be formulated in terms of reachability properties, further verifiable by means of **dup**-free expressions of shape: $in \cdot (p \cdot t)^* \cdot out \not\equiv \mathbf{0}$. Hence, we chose to define DyNetKAT synchronisation, guarded recursion and multi-packet semantics on top of the **dup**-free fragment of NetKAT, denoted by NetKAT$^{-\mathbf{dup}}$.

The syntax of DyNetKAT is defined on top of the **dup**-free fragment of NetKAT as:

$$
\begin{aligned}
N &::= \text{NetKAT}^{-\mathbf{dup}} \\
D &::= \bot \mid N\,;D \mid x?N\,;D \mid x!N\,;D \mid D \,||\, D \mid D \oplus D \mid X \\
&\quad\ X \triangleq D
\end{aligned}
\tag{1}
$$

We write $p \in \text{NetKAT}$, $p \in \text{NetKAT}^{-\mathbf{dup}}$ or, respectively, $p \in \text{DyNetKAT}$ in order to refer to a NetKAT, NetKAT$^{-\mathbf{dup}}$ or, respectively, DyNetKAT policy p.

The DyNetKAT-specific constructs are as follows. By $\bot$ we denote a dummy policy without behaviour. Our new sequential composition operator, denoted by $N\,;D$, specifies when the NetKAT$^{-\mathbf{dup}}$ policy N is applicable to the current

packet has come to a successful end and, thus, the packet can be transmitted further and the next packet can be fetched for processing according to the rest of the policy D.

Communication in DyNetKAT, encoded via $x!N; D$ and $x?N; D$, consists of two steps. In the first place, sending and receiving NetKAT$^{-\mathbf{dup}}$ policies through channel x are denoted by $x!N$, and $x?N$. In an expression such as $x?N; P_N$, the combination of the channel name x and the update type N, determine how the continuation process P_N, considering N as a placeholder in P_N, enables defining compositional and compact parameterised DyNetKAT specifications. Secondly, as soon as the sending or receiving messages are successfully communicated, a new packet is fetched and processed according to D. The parallel composition of two DyNetKAT policies (to enable synchronisation) is denoted by $D \,\|\, D$.

As it will become clearer in Section 2.4, communication in DyNetKAT guarantees preservation of well-defined behaviours when transitioning between network configurations. This corresponds to the so-called per-packet consistency in [17], and it guarantees that every packet traversing the network is processed according to exactly one NetKAT$^{-\mathbf{dup}}$ policy.

Non-deterministic choice of DyNetKAT policies is denoted by $D \oplus D$. For a non-determinstic choice over a finite domain P, we use the syntactic sugar $\oplus_{p \in P} P'$, where p appears as "bound variable" in P'; this is interpreted as a sum of finite summand by replacing the variable p with all its possible values in P.

Finally, one can use recursive variables X in the specification of DyNetKAT policies, where each recursive variable should have a unique defining equation $X \triangleq D$. For the simplicity of notation, we do not explicitly specify the trailing "$; \bot$" in our policy specifications, whenever clear from the context.

In Figure 3 we provide the DyNetKAT formalisation of the firewall in Example 1. In the DyNetKAT encoding, we use the message channel $secConReq$ to open up the connection and $secConEnd$ to close it. We model the behaviour of the switch using the two programs $Switch$ and $Switch'$.

$$
\begin{aligned}
Switch &\triangleq \big((port = int) \cdot (port \leftarrow ext)\big) ; Switch \oplus \\
&\quad \big((port = ext) \cdot \mathbf{0}\big) ; Switch \oplus \\
&\quad secConReq?\mathbf{1} ; Switch' \\
Switch' &\triangleq \big((port = int) \cdot (port \leftarrow ext)\big) ; Switch' \oplus \\
&\quad \big((port = ext) \cdot (port \leftarrow int)\big) ; Switch' \oplus \\
&\quad secConEnd?\mathbf{1} ; Switch
\end{aligned}
$$

$$
\begin{aligned}
Host &\triangleq secConReq!\mathbf{1} ; Host \oplus \\
&\quad secConEnd!\mathbf{1} ; Host \\[4pt]
Init &\triangleq Host \,\|\, Switch
\end{aligned}
$$

Fig. 3: Stateful Firewall in DyNetKAT

2.4   DyNetKAT Semantics

The operational semantics of DyNetKAT in Figure 4 is provided over configurations of shape (d, H, H'), where d stands for the current DyNetKAT policy, H is the list of packets to be processed by the network according to d and H' is the list of packets handled successfully by the network. The rule labels γ range over

$$(\textbf{cpol}^{\checkmark}_{;})\dfrac{\sigma' \in [\![p]\!](\sigma::\langle\rangle)}{(p;q,\sigma::H,H') \xrightarrow{(\sigma,\sigma')} (q,H,\sigma'::H')} \quad (\textbf{cpolx})\dfrac{(p,H_0,H_1) \xrightarrow{\gamma} (p',H_0',H_1')}{(X,H_0,H_1) \xrightarrow{\gamma} (p',H_0',H_1')}X \triangleq p$$

$$(\textbf{cpol}_{\oplus})\dfrac{(p,H_0,H_0') \xrightarrow{\gamma} (p',H_1,H_1')}{(p \oplus q,H_0,H_0') \xrightarrow{\gamma} (p',H_1,H_1')} \quad (\textbf{cpol}_{||})\dfrac{(p,H_0,H_0') \xrightarrow{\gamma} (p',H_1,H_1')}{(p||q,H_0,H_0') \xrightarrow{\gamma} (p'||q,H_1,H_1')}$$

$$(\textbf{cpol}_{\bullet})\dfrac{}{(x \bullet p;q,H,H') \xrightarrow{x\bullet p} (q,H,H')} \quad \bullet \in \{?,!\}$$

$$(\textbf{cpol}_{\clubsuit\spadesuit})\dfrac{(q,H,H') \xrightarrow{x\,\clubsuit\,p} (q',H,H') \quad (s,H,H') \xrightarrow{x\,\spadesuit\,p} (s',H,H')}{(q||s,H,H') \xrightarrow{\textbf{rcfg(x,p)}} (q'||s',H,H')} \quad \begin{array}{c} \clubsuit =? \quad \spadesuit =! \\ \text{or} \\ \clubsuit =! \quad \spadesuit =? \end{array}$$

$$\gamma ::= (\sigma,\sigma') \mid x!q \mid x?q \mid \textbf{rcfg(x,q)}$$

Fig. 4: DyNetKAT: Operational Semantics (relevant excerpt)

pairs of packets (σ,σ') or communication/reconfiguration-like actions of shape $x!q$, $x?q$ or $\textbf{rcfg}(\textbf{x},\textbf{q})$, depending on the context.

Note that the DyNetKAT semantics is devised in a "layered" fashion. Rule $(\textbf{cpol}^{\checkmark}_{;})$ in Figure 4 is the base rule that makes the transition between the NetKAT denotations and DyNetKAT operations. More precisely, whenever σ' is a packet resulted from the successful evaluation of a NetKAT policy p on σ, a (σ,σ')-labelled step is observed at the level of DyNetKAT. This transition applies whenever the current configuration encapsulates a DyNetKAT policy of shape $p;q$ and a list of packets to be processed starting with σ. The resulting configuration continues with evaluating q on the next packet in the list, while σ' is marked as successfully handled by the network.

The remaining rules in Figure 4 define non-deterministic choice $\oplus$, synchronisation $||$ and recursion X in the standard fashion. Note that synchronisations leave the packet lists unchanged. Moreover, we choose not to hide the channel x and the policy p being communicated (as it is usually the case in ACP), but rather keep this information visible outside the SDN being modelled, by means of the label $\textbf{rcfg}(\textbf{x},\textbf{p})$. Due to space limitation, we omitted the explicit definitions of the symmetric cases for $\oplus$ and $||$. The full semantics is provided in [7].

In Figure 5 we depict a labelled transition system (LTS) encoding a possible behaviour of the stateful firewall in Example 1. We assume the list of network packets to be processed consists of a "safe" packet σ_i travelling from *int* to *ext* (i.e., $\sigma_i(port) = int$) followed by a potentially "dangerous" packet σ_e travelling from *ext* to *int* (i.e., $\sigma_e(port) = ext$). For the simplicity of notation, in Figure 5 we write H for *Host*, S for *Switch*, S' for *Switch'*, *SCR* for *secConReq* and *SCE* for *secConEnd*. Note that σ_e can enter the network only if a secure connection request was received. More precisely, the transition

labelled (σ_e, σ_i) is preceded by a transition labelled $SCR?1$ or $\mathbf{rcfg}(SCR, \mathbf{1})$:
$$n_2 \xrightarrow{SCR?1, \; \mathbf{rcfg}(SCR,\mathbf{1})} n_3 \xrightarrow{(\sigma_e, \sigma_i)} n_4.$$
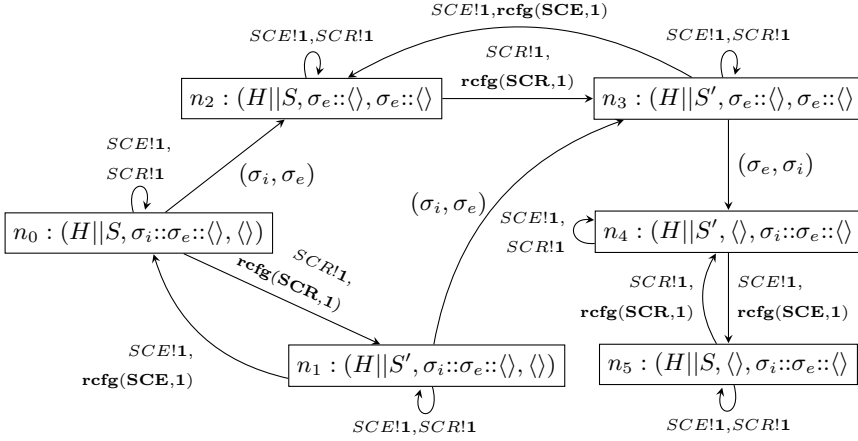
Fig. 5: Stateful Firewall LTS

3 Semantic Results

In this section we define bisimilarity of DyNetKAT policies and provide a corresponding sound and ground-complete axiomatization. We start with strong bisimilarity because it lends itself to a neat theory. Once we establish a theory for strong bisimilarity, a theory for other notions of equivalence in the linear-time and branching-time spectrum can be obtained by adding a specific set of axioms following a standard recipe for each notion. We use this approach to reason about safety properties that are about traces.

Bisimilarity of DyNetKAT terms is defined in the standard fashion:

Definition 2 (Bisimilarity ($\sim$)) *A symmetric relation R over DyNetKAT policies is a* bisimulation *whenever for $(p, q) \in R$ the following holds:*
If $(p, H_0, H_1) \xrightarrow{\gamma} (p', H_0', H_1')$ then exists q' s.t. $(q, H_0, H_1) \xrightarrow{\gamma} (q', H_0', H_1')$ and $(p', q') \in R$, with $\gamma ::= (\sigma, \sigma') \mid x?r \mid x!r \mid \mathbf{rcfg}(\mathbf{x}, \mathbf{r})$.
We call bisimilarity *the largest bisimulation relation. Two policies p and q are bisimilar $(p \sim q)$ iff there is a bisimulation relation R such that $(p, q) \in R$.*

Semantic equivalence of NetKAT$^{-\mathbf{dup}}$ policies is preserved by DyNetKAT.

Proposition 1 (Semantic Layering). *Let p and q be NetKAT$^{-\mathbf{dup}}$ policies. The following holds: $[\![p]\!] = [\![q]\!]$ iff $(p; d) \sim (q; d)$ for any DyNetKAT policy d.*

for $p, q, r \in$ DyNetKAT and $z, y \in$ NetKAT$^{-\mathbf{dup}}$

for $a ::= z \mid x?z \mid x!z \mid \mathbf{rcfg}_{x,z}$

$$\mathbf{0} \, ; p \equiv \bot \qquad (A0)$$
$$(z + y) \, ; p \equiv z \, ; p \oplus y \, ; p \qquad (A1)$$
$$p \oplus q \equiv q \oplus p \qquad (A2)$$
$$(p \oplus q) \oplus r \equiv p \oplus (q \oplus r) \qquad (A3)$$
$$p \oplus p \equiv p \qquad (A4)$$
$$p \oplus \bot \equiv p \qquad (A5)$$
$$p \, || \, q \equiv q \, || \, p \qquad (A6)$$
$$p \, || \, \bot \equiv p \qquad (A7)$$
$$p \, || \, q \equiv p \| q \oplus q \| p \oplus p \mid q \qquad (A8)$$
$$\bot \| p \equiv \bot \qquad (A9)$$
$$(a \, ; p) \| q \equiv a \, ; (p \, || \, q) \qquad (A10)$$
$$(p \oplus q) \| r \equiv (p \| r) \oplus (q \| r) \qquad (A11)$$
$$(x?z \, ; p) \mid (x!z \, ; q) \equiv \mathbf{rcfg}_{x,z} \, ; (p \, || \, q) \qquad (A12)$$
$$(p \oplus q) \mid r \equiv (p \mid r) \oplus (q \mid r) \qquad (A13)$$
$$p \mid q \equiv q \mid p \qquad (A14)$$
$$p \mid q \equiv \bot \ [owise] \qquad (A15)$$

for $at ::= \alpha \cdot \pi \mid x?z \mid x!z \mid \mathbf{rcfg}_{x,z}$:

$$\delta_{\mathcal{L}}(\bot) \equiv \bot \qquad (\delta_{\bot})$$
$$\delta_{\mathcal{L}}(at \, ; p) \equiv at \, ; \delta_{\mathcal{L}}(p) \ \text{if} \ at \notin \mathcal{L} \ (\delta_{;})$$
$$\delta_{\mathcal{L}}(at \, ; p) \equiv \bot \ \text{if} \ at \in \mathcal{L} \qquad (\delta_{;}^{\bot})$$
$$\delta_{\mathcal{L}}(p \oplus q) \equiv \delta_{\mathcal{L}}(p) \oplus \delta_{\mathcal{L}}(q) \qquad (\delta_{\oplus})$$

for $n \in \mathbb{N}$:

$$\pi_0(p) \equiv \bot \qquad (\Pi_0)$$
$$\pi_n(\bot) \equiv \bot \qquad (\Pi_{\bot})$$
$$\pi_{n+1}(at \, ; p) \equiv at \, ; \pi_n(p) \qquad (\Pi_{;})$$
$$\pi_n(p \oplus q) \equiv \pi_n(p) \oplus \pi_n(q) \qquad (\Pi_{\oplus})$$

$$p \equiv q \ \text{if} \ \forall n \in \mathbb{N} : \pi_n(p) \equiv \pi_n(q) \ (AIP)$$

$$E_{NK}$$

Fig. 6: The axiom system E_{DNK} (including E_{NK})

Proof sketch.   This follows according to $\sim$ and $(\mathbf{cpol}_{-\,;}^{\checkmark})$ in Figure 4.   ∎

We further provide some additional ingredients needed to introduce the DyNetKAT axiomatisation in Figure 6. First, note that our notion of bisimilarity identifies synchronisation steps as in $(\mathbf{cpol}_{\clubsuit \spadesuit})$ in Figure 4. At the axiomatisation level, this requires introducing corresponding constants $\mathbf{rcfg}_{x,z}$ defined as:

$$\overline{(\mathbf{rcfg}_{x,z} \, ; p, H_0, H_1) \xrightarrow{\mathbf{rcfg(x,z)}} (p, H_0, H_1)}.$$

In accordance with standard approaches to process algebra (see, e.g., [1,4]) we consider the restriction operator $\delta_{\mathcal{L}}(-)$ with $\mathcal{L}$ a set of forbidden actions ranging over $x?z$ and $x!z$ as in (1). In practice, we use the restriction operator to force synchronous communication. We also define a projection operator $\pi_n(-)$ that, intuitively, captures the first n steps of a DyNetKAT policy. $\pi_n(-)$ is crucial for defining the so-called "Approximation Induction Principle" that enables reasoning about equivalence of recursive DyNetKAT specifications. Last, but not least, in our axiomatisation we employ the left-merge operator ($\|$) and the communication-merge operator ($\mid$) utilised for axiomatising parallel composition. Intuitively, a process of shape $p\|q$ behaves like p as a first step, and then continues as the parallel composition between the remaining behaviour of p and q. A process of shape $p \mid q$ forces the synchronous communication between p and q in a first step, and then continues as the parallel composition between the remaining behaviours of p and q. The full description of these auxiliary operators is provided in [7].

From this point onward, we denote by DyNetKAT the extension with the operators $\delta_{\mathcal{L}}(-)$, $\pi_n(-)$ and $\mathbf{rcfg}_{x,z}$:

$$
\begin{aligned}
N &::= \text{NetKAT}^{-\mathbf{dup}} \\
D_e &::= \bot \mid N \mid D \mid x?N \,;D_e \mid x!N \,;D_e \mid \mathbf{rcfg}_{x,N} \,;D_e \mid \\
&\quad D_e \parallel D_e \mid D_e \oplus D_e \mid \delta_{\mathcal{L}}(D_e) \mid \pi_n(D_e) \mid D_e \| D_e \mid D_e | D_e \mid X \\
X &\triangleq D_e,\ n \in \mathbb{N},\ \mathcal{L} = \{c \mid c ::= x?N \mid x!N\}
\end{aligned}
\tag{2}
$$

Bisimilarity is defined for DyNetKAT terms as in (2) in the natural fashion.

Lemma 3 *For* DyNetKAT, *bisimilarity is a* congruence.

Proof sketch. The result follows from the fact that the semantic rules defined in this paper comply to the congruence formats proposed in [15]; the notion of bisimilarity used in our paper coincides with the notion of stateless bisimilarity in [15] and hence, the lemma follows. ∎

In Figure 6, we introduce E_{DNK} – the axiom system of DyNetKAT, including the NetKAT axiomatisation E_{NK}. Most of the axioms in Figure 6 comply to the standard axioms of parallel and communicating processes [4], where, intuitively, $\oplus$ plays the role of non-deterministic choice, ; resembles sequential composition and $\bot$ is a process that deadlocks. An interesting axiom is $(A7) : p \| \bot \equiv p$ which, intuitively, states that if one network component fails, then the whole system continues with the behaviour of the remaining components. This is a departure from the approach in [11], where recovery is not possible in case of a component's failure; i.e., $e \| 0 \equiv 0$. Additionally, $(A12)$ "pin-points" a communication step via the newly introduced constants of form $\mathbf{rcfg}_{x,z}$. Axiom $(A0)$ states that if the current packet is dropped as a result of the unsuccessful evaluation of a NetKAT policy, then the continuation is deadlocked. $(A1)$ enables mapping the non-deterministic choice at the level of NetKAT to the setting of DyNetKAT.

The axioms encoding the restriction operator $\delta_{\mathcal{L}}(-)$ and the projection operator $\pi_n(-)$ are defined in the standard fashion, on top of DyNetKAT normal forms later defined in this section. Intuitively, normal forms are defined inductively, as sums of complete tests and complete assignments $\alpha \cdot \pi$, or communication steps $x?q$, $x!q$ and $\mathbf{rcfg}_{x,q}$, followed by arbitrary DyNetKAT policies. Complete tests (typically denoted by α) and complete assignments (typically denoted by π) were originally introduced in [3]. In short: let $F = \{f_1, \ldots, f_n\}$ be a set of fields names with values in V_i, for $i \in \{1, \ldots, n\}$. We call *complete test* (resp., *complete assignment*) an expression $f_1 = v_1 \cdot \ldots \cdot f_n = v_n$ (resp., $f_1 \leftarrow v_1 \cdot \ldots \cdot f_n \leftarrow v_n$), with $v_i \in V_i$, for $i \in \{1, \ldots, n\}$. Last, but not least, axiom (AIP) corresponds to the so-called "Approximation Induction Principle", and it provides a mechanism for reasoning about the equivalence of recursive behaviours, up to a certain limit denoted by n.

In what follows, we show that the axiom system E_{DNK} is sound and ground-complete with respect to DyNetKAT bisimilarity.

Lemma 4 (NetKAT$^{-\mathbf{dup}}$ Normal Forms) *We call a* NetKAT$^{-\mathbf{dup}}$ *policy q in* normal form *(n.f.) whenever q is of shape $\Sigma_{\alpha\cdot\pi\in\mathcal{A}}\alpha\cdot\pi$ with $\mathcal{A} = \{\alpha_i\cdot\pi_i \mid i \in I\}$. E_{NK} is normalising for* NetKAT$^{-\mathbf{dup}}$.

Proof sketch.   The result follows from Lemma 4 in [3] stating that the standard semantics of every NetKAT expression is equal to the union of its minimal nonzero terms. In the context of NetKAT$^{-\mathbf{dup}}$ and packet values drawn from finite domains (as is the case in [3]), this union can be equivalently expressed as a sum of complete tests and complete assignments. I.e., $\vdash r \equiv \Sigma_{i\in I}\alpha_i \cdot \pi_i$ for every NetKAT$^{-\mathbf{dup}}$ expression r. ∎

Definition 5 (DyNetKAT Normal Forms) *We call a* DyNetKAT *policy in* normal form *(n.f.) if it is of shape*

$$\Sigma^{\oplus}_{i\in I}(\alpha_i \cdot \pi_i); d_i \oplus \Sigma^{\oplus}_{j\in J}c_j; d_j \, (\oplus\bot)$$

where d_i, d_j range over DyNetKAT policies and $c_j ::= x?q \mid x!q \mid \mathbf{rcfg}_{x,q}$ with q denoting terms in NetKAT$^{-\mathbf{dup}}$.

Definition 6 (Guardedness) *A DyNetKAT policy p is* guarded *if and only if all occurrences of all variables X in p are* guarded. *An occurrence of a variable X in a policy p is* guarded *if and only if (i) p has a subterm of shape $p';t$ such that either p' is variable-free, or all the occurrences of variables Y in p' are guarded, and X occurs in t, or (ii) if p is of shape $y?X;t$, $y!X;t$ or $\mathbf{rcfg}_{X,t}$.*

Note that guarded DyNetKAT policies are finitely branching. In what follows, we assume DyNetKAT policies are guarded.

Lemma 7 (DyNetKAT Normalisation) *E_{DNK} is normalising for* DyNetKAT.

Proof sketch.   The proof follows from Lemma 4 and $(A1)$, by structural induction. *Base cases:* $\boxed{p \triangleq \bot}$ trivially holds; $\boxed{p \triangleq q; d}$ with q a NetKAT$^{-\mathbf{dup}}$ term holds by Lemma 4 and $(A1)$; $\boxed{p \triangleq c; d}$ with $c ::= x?q \mid x!q \mid \mathbf{rcfg}_{x,q}$ trivially holds. *Induction step, cases:* $\boxed{p \triangleq X}$ - discarded, as p is not guarded; $\boxed{p \triangleq p_1 \oplus p_2}$; $\boxed{p \triangleq p_1 \| p_2}$; $\boxed{p \triangleq \pi_n(p')}$; $\boxed{p \triangleq p_1 \mid p_2}$; $\boxed{p \triangleq \delta_{\mathcal{L}}(p')}$ and, eventually, $\boxed{p \triangleq p_1 \,\|\, p_2}$. All items before follow by the axiom system E_{DNK} and the induction hypothesis, under the assumption that p_1, p_2 and p' are guarded. ∎

Lemma 8 (Soundness of $E_{\mathrm{DyNetKAT}\backslash AIP}$) *Let $E_{\mathrm{DyNetKAT}\backslash AIP}$ stand for the axiom system E_{DNK} in Figure 6, without the axiom (AIP). $E_{\mathrm{DyNetKAT}\backslash AIP}$ is sound for* DyNetKAT *bisimilarity.*

Proof sketch. This is proven in a standard fashion, by case analysis on transitions of shape $(p, H_0, H'_0) \xrightarrow{\gamma} (q, H_1, H'_1)$ with $\gamma ::= (\sigma, \sigma') \mid x?n \mid x!n \mid \mathbf{rcfg}(\mathbf{x}, \mathbf{n})$, according to the semantic rules of the DyNetKAT operators in (2). Take $(A0)$

for instance. The left hand-side $\mathbf{0}; p$ can only evolve according to $(\mathbf{cpol}^{\checkmark}_{;})$ in Fig. 4 which, in turn, has an empty premise as $[\![\mathbf{0}]\!](\sigma :: \langle \rangle) = \{\}$ for all σ. Thus, $(\mathbf{cpol}^{\checkmark}_{;})$ does not entail any step for this case. Symmetrically, there is no semantic transition for $\bot$ in Fig. 4. In other words, none of the left/right hand-sides of $(A0)$ displays any behaviour, therefore the axiom is sound.   ■

Lemma 9 (Soundness of AIP**)** *The Approx. Induction Principle* (AIP) *is sound for* DyNetKAT *bisimilarity.*

Proof sketch.   The proof is close to the one of Theorem 2.5.8 in [4] and uses the branching finiteness property of guarded DyNetKAT policies.   ■

Theorem 1 (Soundness & Completeness). E_{DNK} *is sound and ground-complete for* DyNetKAT *bisimilarity.*

Proof. Soundness: if $E_{DNK} \vdash p \equiv q$ then $p \sim q$, follows from Lemma 8 and Lemma 9. Completeness: if $p \sim q$ then $E_{DNK} \vdash p \equiv q$, is shown as follows. Without loss of generality, assume p and q are in n.f., according to Lemma 7. We want to show that $p \equiv q \oplus p$ and $q \equiv p \oplus q$ which, by ACI of $\oplus$ implies $p \equiv q$. This reduces to showing that every summand of p is a summand of q and vice-versa. We first argue that every summand of p is a summand of q. The reasoning is by structural induction.

Base case $p \triangleq \bot$ holds by the hypothesis $p \sim q$ that $q \triangleq \bot$.

Induction step. Case $p \triangleq ((\alpha \cdot \pi); p') \oplus p''$: then, $(p, \sigma_\alpha :: H, H') \xrightarrow{(\sigma_\alpha, \sigma_\pi)} (p', H, \sigma_\pi :: H')$ implies by the hypothesis $p \sim q$ that $(q, \sigma_\alpha :: H, H') \xrightarrow{(\sigma_\alpha, \sigma_\pi)} (q', H, \sigma_\pi :: H')$ and $p' \sim q'$. Recall that q is in n.f.; hence, by the shape of the semantic rules in Figure 4 it holds that $q \triangleq ((\alpha \cdot \pi); q') \oplus q''$. By the induction hypothesis, it holds that $p' \equiv q'$ hence, $(\alpha \cdot \pi); p'$ is a summand of q as well. Cases $p \triangleq (c; p') \oplus p''$ with $c ::= x?n \mid x!n \mid \mathbf{rcfg}_{x,n}$ follow in a similar fashion. Hence, $p \equiv q \oplus p$ holds. The symmetric case $q \equiv p \oplus q$ follows the same reasoning.

We refer to [7] for the complete proofs and additional details.

4   A Framework for Safety

In this section we provide a language for specifying safety properties for networks characterized by DyNetKAT, together with a procedure for reasoning about safety in an equational fashion. Intuitively, safety properties enable specifying the absence of undesired network behaviours.

Definition 10 (Safety Properties - Syntax) *Let* $\mathcal{A}$ *be an alphabet over letters of shape* $\alpha \cdot \pi$ *and* $\mathbf{rcfg}_{x,p}$*, with* α *and* π *ranging over complete tests and assignments, and* $\mathbf{rcfg}_{x,p}$ *ranging over reconfiguration actions. Safety properties are defined in the following fashion:*

$$act ::= \alpha \cdot \pi \mid \mathbf{rcfg}_{x,p} \quad (\alpha \cdot \pi, \mathbf{rcfg}_{x,p} \in \mathcal{A})$$
$$regexp ::= true \mid act \mid \neg act \mid regexp + regexp \mid regexp \cdot regexp \mid$$
$$(regexp)^n \quad (\text{with } n \geq 1)$$
$$prop ::= [regexp]false$$

A safety property specification *prop* is satisfied whenever the behaviour encoded by *regexp* should not be observed within the network. Regular expressions *regexp* are defined with respect to actions *act*: a flow of shape $\alpha \cdot \pi$ is the observable behaviour of a (NetKAT$^{-\mathbf{dup}}$) policy transforming a packet encoded by α into α_π, whereas $\mathbf{rcfg}_{x,p}$ corresponds to a reconfiguration step in a network. Recursively, a sum of regular expressions $regexp_1 + regexp_2$ encodes the union of the two behaviours, a concatenation of regular expressions $regexp_1 \cdot regexp_2$ encodes the behaviour of $regexp_1$ followed by the behaviour of $regexp_2$. A property of shape $[\neg a]false$, with $a \in \mathcal{A}$, states that the system cannot do anything apart from a as a first step. The property $[true]false$ states that no action can be observed in the network, whereas $[r^n]false$ encodes the repeated application of r for n times.

Note that *true*, negated expressions $\neg a$ and repetitions r^n are mere syntactic sugars of equivalent expressions free of these operations. Not surprisingly, "de-sugaring" $(ds(-))$ is defined as:

$$ds(true) \triangleq \Sigma_{a \in \mathcal{A}} a$$
$$ds(\neg a) \triangleq \Sigma_{\substack{a_i \in \mathcal{A} \\ a_i \neq a}} a_i \qquad ds(r^n) \triangleq ds(\underbrace{r \cdot r \cdot \ldots \cdot r}_{n \text{ times}})$$
$$ds(r_1 \cdot r_2) \triangleq ds(r_1) \cdot ds(r_2) \text{ if } r_1 \cdot r_2 \text{ not de-sugared}$$
$$ds(r_1 + r_2) \triangleq ds(r_1) + ds(r_2) \text{ if } r_1 + r_2 \text{ not de-sugared}$$
$$ds(r) \triangleq r \text{ [owise]}$$

The complete formal definition of the de-sugaring function is provided in [7].

Definition 11 (Safety Properties - Semantics) *Let $\mathcal{A}$ be an alphabet over letters of shape $\alpha \cdot \pi$ and $\mathbf{rcfg}(\mathbf{x}, \mathbf{p})$, with α and π ranging over complete tests and assignments, and $\mathbf{rcfg}(\mathbf{x}, \mathbf{p})$ ranging over reconfiguration actions. We write w, w' for (non-empty) words with letters in $\mathcal{A}$ (i.e., $w, w' \in \mathcal{A}^*$) and $\mid w \mid$ for the length of w. We write $w' \preceq w$ whenever w' is a prefix of w (including w).*

Let r be a de-sugared regular expression (regexp) as in Definition 10. We call head normal form (h.n.f.) of r, denoted by $hnf(r)$, the sum of words as above obtained by left-/right- distributing $\cdot$ over $+$ in r, in the standard fashion. Note that such a h.n.f. always exists for r. Let Prop stand for the set of all properties as in Definition 10, in h.n.f.

The semantic map $\llbracket - \rrbracket : Prop \to \text{DyNetKAT}$ associates to each safety property in Prop a DyNetKAT expression as follows. Let Θ be the DyNetKAT policy (in normal form) encoding all possible behaviours over $\mathcal{A}$: $\Theta \triangleq \Sigma_{a \in \mathcal{A}}^{\oplus}(a; \bot \oplus a; \Theta)$. Then:

$$\llbracket [\Sigma_{\substack{i \in I \\ w_i \in A^*}} w_i]false \rrbracket \triangleq \Sigma_{\substack{w \in \mathcal{A}^* \\ \mid w \mid < M \\ \forall i \in I : w_i \npreceq w}}^{\oplus} \overline{w}; \bot \quad \oplus \quad \Sigma_{\substack{w \in \mathcal{A}^* \\ \mid w \mid = M \\ \forall i \in I : w_i \npreceq w}}^{\oplus} (\overline{w}; \bot \oplus \overline{w}; \Theta) \quad (3)$$

such that M is the length of the longest word w_i, with $i \in I$, and $\overline{w}$ is a DyNetKAT-compatible term obtained from w where all letters have been separated by ; and inductively defined in the obvious way. Namely, $\overline{a} \triangleq a$ for $a \in \mathcal{A}$

and $\overline{a \cdot w} \triangleq a; \overline{w}$ for $a \in \mathcal{A}$ and $w \in \mathcal{A}^*$. The semantic map $[\![-]\!]$ is defined following the intuition provided earlier in this section. For instance, as shown in (3), if none of the sequences of steps w_i can be observed in the system, then the associated DyNetKAT term prevents the immediate execution of all w_i.

Typically, safety analysis is reduced to reachability. In our context, a safety property is violated whenever the network system under analysis displays a (finite) execution that is not in the behaviour of the property. Thus, the aforementioned semantic map is based on traces (or words in $\mathcal{A}^*$) and is not sensitive to branching. This paves the way to reasoning about safety properties in an equational fashion.

Definition 12 (Safe Network Systems) *Let E^{tr}_{DNK} stand for the equational axioms in Figure 6, including the additional axiom that enables switching from the context of bisimilarity to trace equivalence of DyNetKAT policies, namely: $p; (q \oplus r) \equiv p; q \oplus p; r$. Assume a specification given as the safety formula s and a network system implemented as the DyNetKAT policy i. We say that the network is* safe *whenever the following holds: $E^{tr}_{DNK} \vdash [\![s]\!] \oplus i \equiv [\![s]\!]$. In words: checking whether i satisfies s reduces to checking whether the trace behaviour of i is included into that of s.*

For an example, consider the firewall in Figure 1 and the corresponding encoding in Figure 3. Recall that reaching *int* from *ext* without observing a secure connection request is a faulty behaviour. This entails the safety formula s_n defined as $[(\neg \mathbf{rcfg}_{secConReq,1})^n \cdot (\alpha \cdot \pi)]false$, for $n \in \mathbb{N}$, $\alpha \triangleq (port = ext)$ and $\pi \triangleq (port \leftarrow int)$. Therefore, checking whether the network is safe reduces to checking, for all $n \in \mathbb{N}$: $E^{tr}_{DNK} \vdash [\![s_n]\!] \oplus Init \equiv [\![s_n]\!]$. Note that, for a fixed n, the verification procedure resembles bounded model checking [5].

5   Implementation

In this section, we describe our implementation for formal reasoning about dynamic networks. Our prototype tool, called DyNetiKAT (available at https://github.com/hcantunc/DyNetiKAT) is based on Maude [8], the NetKAT decision procedure [10], and Python [19] as a glue language. Our modular extension of NetKAT allows for reusing the NetKAT tools in our framework. In our prototype, we focus on checking reachability and waypointing in a dynamic setting. We build upon the methods for checking reachability and waypointing properties in NetKAT [3]. For a reminder, in NetKAT, reachability and waypointing properties are characterised as follows: for reachability properties, an egress point *out* is reachable from an ingress point *in*, in the context of a switch policy p and topology t, whenever the following NetKAT equivalence holds: $in \cdot (p \cdot t)^* \cdot out \not\equiv \mathbf{0}$. For waypointing properties, an intermediate point w between *in* and *out* is considered a waypoint from *in* to *out* if all the packets from *in* to *out* go through w. Such a property is satisfied if the following equivalence holds:

$$in \cdot (p \cdot t)^* \cdot out + in \cdot (\neg out \cdot p \cdot t)^* \cdot w \cdot (\neg in \cdot p \cdot t)^* \cdot out$$
$$\equiv in \cdot (\neg out \cdot p \cdot t)^* \cdot w \cdot (\neg in \cdot p \cdot t)^* \cdot out$$

In order to utilise the NetKAT decision procedure for property checking we represent the properties given as regular expressions (as described in Section 4). To this end, we introduced the operators $head(D)$, and $tail(D, R)$, where D is a DyNetKAT term and R is a set of terms of shape $\mathbf{rcfg}_{X,N}$. Intuitively, the operator $head(D)$ returns a NetKAT policy representing the current configuration in D, and $tail(D, R)$ returns a DyNetKAT policy which is the sum of policies in D that appear after the synchronisation events in R. We utilise these operators as follows: for a given DyNetKAT term we apply our equational reasoning framework to unfold the expression and rewrite it into the normal form. Then, we extract the desired configurations by using the head and tail operators. After this step, the resulting expression is a NetKAT term and we use the NetKAT decision procedure for checking properties. For example, consider the safety property $[(true)^n \cdot (\alpha \cdot \pi)]false$ as in Definition 10, and a network SDN. Note that for a given complete assignments, there exists a corresponding complete test with the same values, e.g., the corresponding complete test for the complete assignment $f_0 \leftarrow v_0 \ldots f_n \leftarrow v_n$ is $f_0 = v_0 \ldots f_n = v_n$. Henceforth, we write α_π to represent the corresponding complete tests of π. The property $[(true)^n \cdot (\alpha \cdot \pi)]false$ can be encoded in the style of NetKAT as follows:

$$\alpha \cdot head(\pi_n(SDN)) \cdot \alpha_\pi \equiv \mathbf{0} \tag{4}$$
$$\alpha \cdot head(tail(\pi_n(SDN), R)) \cdot \alpha_\pi \equiv \mathbf{0} \tag{5}$$

where R is the set of all synchronisation events in the network and $\pi_n(-)$ is the projection operator equationally defined in Figure 6. In our technical report [7] we provide the corresponding correctness specification of the stateful example discussed in Section 1. Note that in practice the parameter n in π_n is a fixed value specified by the user. Intuitively, (4) expresses that the initial configuration of the network is not able to transform the packets satisfying the predicate α such that they satisfy the predicate α_π and (5) expresses that this transformation is still not possible in the configurations after any sequence of synchronisation events. Formally, the operators $head$ and $tail$ are defined as follows:

$$head(\bot) = \mathbf{0} \qquad\qquad tail(\bot, R) = \bot$$
$$head(N; D) = N + head(D) \qquad\qquad tail(N; D, R) = tail(D, R)$$
$$head(D \oplus Q) = head(D) + head(Q) \qquad tail(D \oplus Q, R) = tail(D, R) \oplus tail(Q, R)$$
$$head(\mathbf{rcfg}_{X,N}; D) = \mathbf{0} \qquad\qquad tail(\mathbf{rcfg}_{X,N}; D, R) = D \oplus tail(D, R) \text{ if } \mathbf{rcfg}_{X,Z} \in R$$
$$tail(\mathbf{rcfg}_{X,N}; D, R) = \bot \text{ if } \mathbf{rcfg}_{X,N} \notin R$$

Note that we assume the DyNetKAT terms given as input to the operators $head$ and $tail$ do not contain terms of shape $x?q$ and $x!q$. This can be ensured by applying the restriction operator δ on the input terms.

Observe that the safety properties of Definition 10 are designed to capture unsafe flows. Similarly, one can also define the syntax $\langle regexp \rangle true$ to express that a certain safe flow is possible and reason about it. For an example, consider the stateful firewall example and the property $\langle (\mathbf{rcfg}_{secConReq,1})^n \cdot (\alpha \cdot \pi) \rangle true$
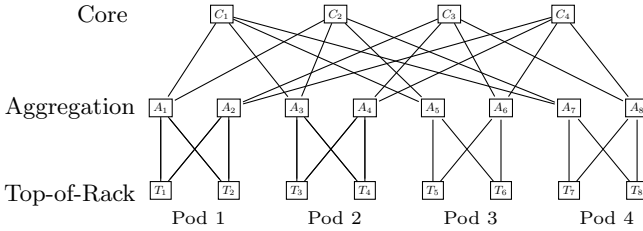
Core

Aggregation

Top-of-Rack

Pod 1   Pod 2   Pod 3   Pod 4

Fig. 7: A FatTree Topology

where $\alpha \triangleq (port = ext)$ and $\pi \triangleq (port \leftarrow int)$. This property expresses that the flow from port ext to port int is possible after the event $\mathbf{rcfg}_{secConReq,\mathbf{1}}$. This property can be encoded in the NetKAT style as $\alpha \cdot head(tail(\pi_n(Init), R)) \cdot \alpha_\pi \not\equiv \mathbf{0}$ where $R = \{\mathbf{rcfg}_{secConReq,\mathbf{1}}\}$.

6   Experimental Evaluation

In this section we evaluate the applicability of our implementation based on a FatTree [22] topology case. FatTrees are hierarchical topologies commonly used in data centers. Figure 7 illustrates a FatTree with 3 levels: core, aggregation and top-of-rack (ToR). The switches at each level contain a number of redundant links to the upper level. The groups of ToR switches and their corresponding aggregation switches are called pods. For our experiments, we generated 6 Fat-Trees that grow in size and achieve a maximum size of 1344 switches. For these networks we computed a shortest path forwarding policy between all pairs of ToR switches. The number of switches in the ToR layer is set to $k^3/4$ where k is the number of pods in the network.

We check dynamic properties on these networks and assess the time performance of our tool. We consider a scenario involving two ToR switches T_a and T_b that reside in different pods. Initially, all packets from T_a to T_b traverse through a firewall A_x in the aggregation layer which filters SSH packets. The controller then decides to shift the firewall from A_x to another switch $A_{x'}$ in the aggregation layer. For this purpose, the controller updates the corresponding aggregation and core layer switches resulting in 4 updates. The checked properties are as follows: (i) At any point while the controller is performing the updates, non-SSH packets from T_a can always reach T_b. (ii) At any point while the controller is performing the updates, SSH packets from T_a can never reach T_b. (iii) After all the updates are performed, $A_{x'}$ is a waypoint between T_a and T_b.

We conducted the experiments on an Ubuntu 20.04 LTS OS with 16 core 2.4GHz Intel i9-9980HK processor and 64 GB RAM. The results are depicted in Figure 8. We report the preprocessing time, the time taken for checking properties (i), (ii), and (iii) individually (referred to as Reachability-I, Reachability-II, and Waypointing, respectively), and also time taken to check all the properties in parallel (referred to as All Properties). The reported times are the average of 10 runs.

The results indicate that preprocessing step is a non-negligible factor that contributes to overall time. However, preprocessing is independent of the property that is being checked and this procedure only needs to be done once for a given network. After the preprocessing step, the individual properties can be checked in less than 2 seconds for networks with less than 100 switches. For larger networks with sizes up to 931 and 1344 switches, the individual properties can be checked in a maximum of 5 minutes and 11 minutes, respectively. Checking for the property (iii) takes more than twice as much time as checking for the properties (i) and (ii). In the experiments where we check all properties in parallel, we allocated one thread for each property. In this setting, checking all properties introduced 24% overhead on average. After preprocessing, on average 87% of the running times are spent in the NetKAT decision procedure and this step becomes the bottleneck in analysing larger networks.
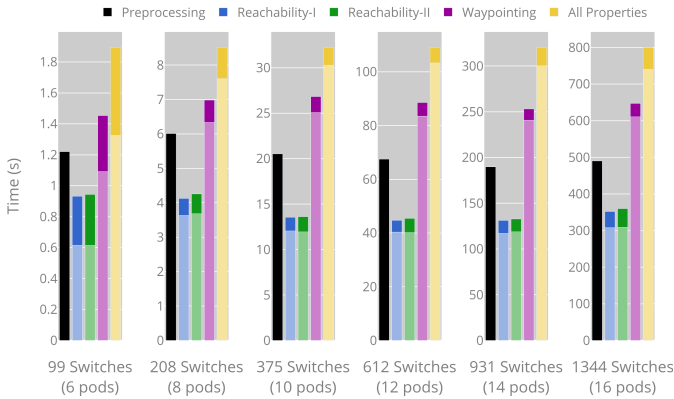
Fig. 8: Results of FatTree experiments. Light-coloured areas indicate the time spent in the NetKAT tool and solid coloured areas indicate the time spent in our tool.

7 Conclusions

We develop the language DyNetKAT for modelling and reasoning about dynamic reconfigurations in Software Defined Networks. Our language builds upon the concepts, syntax, and semantics of NetKAT and hence, provides a modular extension and makes it possible to reuse the theory and tools of NetKAT. We define a formal semantics for our language and provide a sound and ground-complete axiomatisation. We exploit our axiomatisation to analyse reachability properties of dynamic networks and show that our approach scales to networks with hundreds of switches. We assume that each data plane packet sees one set of flow tables throughout their flight in the network [17]. We plan to investigate small-step semantics in which the control plane updates can have a finer interleaving with in-flight packet as future work. Another natural direction for future work is devising compilation schemes enabling the translation of DyNetKAT programs into real running code.

References

1. Luca Aceto, Bard Bloom, and Frits W. Vaandrager. Turning SOS rules into equations. *Inf. Comput.*, 111(1):1–52, 1994. `doi:10.1006/inco.1994.1040`.
2. L. Aceto, W. J. Fokkink, and C. Verhoef. Conservative extension in structural operational semantics. *Bull. EATCS*, 69:110–132, 1999.
3. Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. NetKAT: semantic foundations for networks. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 113–126. ACM, 2014. `doi:10.1145/2535838.2535862`.
4. Jos C. M. Baeten and W. P. Weijland. *Process algebra*, volume 18 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, 1990.
5. Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.
6. Ryan Beckett, Michael Greenberg, and David Walker. Temporal netkat. In Chandra Krintz and Emery Berger, editors, *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016*, pages 386–401. ACM, 2016. `doi:10.1145/2908080.2908108`.
7. G. Caltais, H. Hojjat, M. R. Mousavi, and H. C. Tunc. DyNetKAT: An algebra of dynamic networks. *CoRR*, abs/2102.10035, 2021.
8. Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott. Full Maude: Extending Core Maude. In Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott, editors, *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*, pages 559–597. Springer, 2007. `doi:10.1007/978-3-540-71999-1\_18`.
9. Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, and Alexandra Silva. Probabilistic NetKAT. In Peter Thiemann, editor, *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9632 of *Lecture Notes in Computer Science*, pages 282–309. Springer, 2016. `doi:10.1007/978-3-662-49498-1\_12`.
10. Nate Foster, Dexter Kozen, Matthew Milano, Alexandra Silva, and Laure Thompson. A Coalgebraic Decision Procedure for NetKAT. In Sriram K. Rajamani and David Walker, editors, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 343–355. ACM, 2015. `doi:10.1145/2676726.2677011`.
11. Tobias Kappé, Paul Brunet, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. Concurrent Kleene Algebra with Observations: from Hypotheses to Completeness. *CoRR*, abs/2002.09682, 2020. URL: https://arxiv.org/abs/2002.09682, `arXiv:2002.09682`.
12. Hyojoon Kim, Joshua Reich, Arpit Gupta, Muhammad Shahbaz, Nick Feamster, and Russell J. Clark. Kinetic: Verifiable dynamic network control. In *12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15, Oakland, CA, USA, May 4-6, 2015*, pages 59–72. USENIX Association, 2015. URL: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/kim.

13. Jedidiah McClurg, Hossein Hojjat, Nate Foster, and Pavol Cerný. Event-driven network programming. In Chandra Krintz and Emery Berger, editors, *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016*, pages 369–385. ACM, 2016. `doi:10.1145/2908080.2908097`.

14. Peter D. Mosses. Modular structural operational semantics. J. Log. Algebraic Methods Program. 60-61: 195-228, 2004. `doi.org/10.1016/j.jlap.2004.03.008`

15. Mohammad Reza Mousavi, Michel A. Reniers, and Jan Friso Groote. Notions of bisimulation and congruence formats for SOS with data. *Information and Computation*, 200(1):107 – 147, 2005. `doi.org/10.1016/j.ic.2005.03.002`.

16. Tim Nelson, Andrew D. Ferguson, Michael J. G. Scheer, and Shriram Krishnamurthi. Tierless programming and reasoning for software-defined networks. In Ratul Mahajan and Ion Stoica, editors, *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2014, Seattle, WA, USA, April 2-4, 2014*, pages 519–531. USENIX Association, 2014. URL: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/nelson.

17. Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schlesinger, and David Walker. Abstractions for network update. In Lars Eggert, Jörg Ott, Venkata N. Padmanabhan, and George Varghese, editors, *ACM SIGCOMM 2012 Conference, SIGCOMM '12, Helsinki, Finland - August 13 - 17, 2012*, pages 323–334. ACM, 2012. `doi:10.1145/2342356.2342427`.

18. Alexandra Silva. Models of Concurrent Kleene Algebra. In Elvira Albert and Laura Kovács, editors, *LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Alicante, Spain, May 22-27, 2020*, volume 73 of *EPiC Series in Computing*, page 516. EasyChair, 2020. URL: https://easychair.org/publications/paper/6C8R.

19. Guido van Rossum. Python programming language. In Jeff Chase and Srinivasan Seshan, editors, *Proceedings of the 2007 USENIX Annual Technical Conference, Santa Clara, CA, USA, June 17-22, 2007*. USENIX, 2007.

20. Alexander Vandenbroucke and Tom Schrijvers. P$\lambda\omega$nk: functional probabilistic netkat. *Proc. ACM Program. Lang.*, 4(POPL):39:1–39:27, 2020. `doi:10.1145/3371107`.

21. Jana Wagemaker, Paul Brunet, Simon Docherty, Tobias Kappé, Jurriaan Rot, and Alexandra Silva. Partially Observable Concurrent Kleene Algebra. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPIcs*, pages 20:1–20:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.CONCUR.2020.20`.

22. Al-Fares, Mohammad and Loukissas, Alexander and Vahdat, Amin. A Scalable, Commodity Data Center Network Architecture. ACM SIGCOMM Comput. Commun. Rev. 38, 4, 63-74, 2008. `doi:10.1145/1402946.1402967`.

A new criterion for $\mathcal{M}, \mathcal{N}$-adhesivity, with an application to hierarchical graphs[⋆]

Davide Castelnovo[1]✉, Fabio Gadducci[2], and Marino Miculan[1]

[1] Department of Mathematics, Computer Science and Physics,
University of Udine, Udine, Italy.
davide.castelnovo@uniud.it, marino.miculan@uniud.it
[2] Department of Computer Science, University of Pisa, Pisa, Italy.
fabio.gadducci@unipi.it

Abstract. *Adhesive categories* provide an abstract framework for the algebraic approach to rewriting theory, where many general results can be recast and uniformly proved. However, checking that a model satisfies the adhesivity properties is sometimes far from immediate. In this paper we present a new criterion giving a sufficient condition for $\mathcal{M}, \mathcal{N}$-*adhesivity*, a generalisation of the original notion of adhesivity. We apply it to several existing categories, and in particular to *hierarchical graphs*, a formalism that is notoriously difficult to fit in the mould of algebraic approaches to rewriting and for which various alternative definitions float around.

1 Introduction

The introduction of *adhesive categories* marked a watershed moment for the algebraic approaches to the rewriting of graph-like structures [16,9]. Until then, key results of the approaches on e.g. parallelism and confluence had to be proven over and over again for each different formalism at hand, despite the obvious similarity of the procedure. Differently from previous solutions to such problems, as the one witnessed by the *butterfly lemma* for graph rewriting [8, Lemma 3.9.1], the introduction of adhesive categories provided such a disparate set of formalisms with a common abstract framework where many of these general results could be recast and uniformly proved once and for all.

Despite the elegance and effectiveness of the framework, proving that a given category satisfies the conditions for being adhesive can be a daunting task. For this reason, we look for simpler general criteria implying adhesivity for a class of categories. Similar criteria have been already provided for the core framework of adhesive categories; e.g., every elementary topos is adhesive [17], and a category is (quasi)adhesive if and only if can be suitably embedded in a topos [15,12]. This covers many useful categories such as sets, graphs, etc.; on the other hand, there are many categories of interest which are not (quasi)adhesive, such as directed graphs, posets, and many of their subcategories. In these cases we can try to prove the more general $\mathcal{M}, \mathcal{N}$-adhesivity for suitable $\mathcal{M}, \mathcal{N}$; however, so far this

---

has been achieved only by means of *ad hoc* arguments. To this end, one of the main contributions of this paper is a new criterion for $\mathcal{M}, \mathcal{N}$-adhesivity, based on the verification of some properties of functors connecting the category of interest to a family of suitable adhesive categories. This criterion allows us to prove in a uniform and systematic way some previous results about the adhesivity of categories built by products, exponents, and comma construction.

Moreover, it is well-known that categorical properties are often *prescriptive*, indicating abstractly the presence of some good behaviour of the modelled system. Adhesivity is one such property, as it is highly sought after when it comes to rewriting theories. Thus, our criterion for proving $\mathcal{M}, \mathcal{N}$-adhesivity can be seen also as a "litmus test" for the given category. This is useful in situations that are not completely settled, and for which different settings have been proposed. An important example is that of *hierarchical graphs*, for which we roughly can find two alternative proposals: on the one hand, algebraic formalisms where the edges have some algebraic structures, so that the nesting is a side effect of the term construction; on the other hand, combinatorial approaches where the topology of a standard graph is enriched by some partial order, either on the nodes or on the edges, where the order relation indicates the presence of nesting. By applying our criterion, we can show that the latter approach yields indeed an $\mathcal{M}, \mathcal{N}$-adhesive category, confirming and overcoming the limitations of some previous approaches to hierarchical graphs [21,23,24], which we briefly recall next.

The more straightforward proposal is by Palacz [24], using a poset of edges instead of just a set; however, the class of rules has to be restricted in order to apply the approach, which in any case predates the introduction of adhesive categories. Our work allows to rephrase in terms of adhesive properties and generalise Palacz's proposal, dropping his constraint on rules. Another attempt are Mylonakis and Orejas' *graphs with layers* [21], for which $\mathcal{M}$-adhesivity is proved for a class of monomorphisms in the category of symbolic graphs; however, nodes between edges at different layers cannot be shared. Padberg [23] goes for a coalgebraic presentation via a peculiar "superpower set" functor; this gives immediately $\mathcal{M}$-adhesivity provided that this superpower set functor is well-behaved with respect to limits. However this approach is rather *ad hoc*, not modular and not very natural for actual modelling.

Summarising, the main contributions of this work are: (a) a new general criterion for assessing $\mathcal{M}, \mathcal{N}$-adhesivity; (b) new proofs of $\mathcal{M}, \mathcal{N}$-adhesivity for some relevant categories, systematising previous known proofs; (c) the first proof that a category of hierarchical graph is $\mathcal{M}, \mathcal{N}$-adhesive.

Synopsis. After having recalled some basic notions, in Section 2 we introduce the new criterion for $\mathcal{M}, \mathcal{N}$-adhesivity; using it, we show $\mathcal{M}, \mathcal{N}$-adhesivity of several constructions, such as products and comma categories. In Section 3 we apply this theory to various example categories, such as directed (acyclic) graphs, trees and term graphs. We show also the adhesivity of several categories obtained by combining adhesive ones, and in particular of the elusive category of hierarchical graphs. Conclusions and directions for future work are in Section 4. An extended version of this paper is available at [6].

2 $\mathcal{M}, \mathcal{N}$-adhesivity via creation of (co)limits

In this section we recall some definitions and results about $\mathcal{M}, \mathcal{N}$-adhesive categories and provide a new criterion to prove this property.

2.1 $\mathcal{M}, \mathcal{N}$-adhesive categories

Intuitively, an adhesive category is one in which pushouts of monomorphisms exist and "behave more or less as they do in the category of sets" [16]. Formally, we require pushouts of monomorphisms to be Van Kampen colimits.

Definition 2.1. *A* Van Kampen square *in a category* **A** *is a pushout square*

$$
\begin{array}{ccc}
A & \xrightarrow{n} & B \\
m \downarrow & & \downarrow f \\
C & \xrightarrow{g} & D
\end{array}
$$

such that for any cube as follows, where the back faces are pullbacks,

the top face is a pushout if and only if the front faces are pullbacks.
    Pushout squares which enjoy the "if" of this condition are called stable.

Given a category **A** we will denote by $\mathsf{Mor}(\mathbf{A}), \mathsf{Mono}(\mathbf{A}), \mathsf{Reg}(\mathbf{A})$ respectively the classes of morphisms, monomorphisms and regular monomorphisms of **A**.

Definition 2.2. *Let* **A** *be a category and* $\mathcal{A} \subseteq \mathsf{Mor}(\mathbf{A})$. *Then we say that* $\mathcal{A}$ *is*

- stable under pushouts *if for every pushout square as aside,*
  if $m \in \mathcal{A}$ *then* $n \in \mathcal{A}$;
  stable under pullbacks *if for every pullback square as aside,*
  if $n \in \mathcal{A}$ *then* $m \in \mathcal{A}$;

$$
\begin{array}{ccc}
A & \xrightarrow{f} & B \\
m \downarrow & & \downarrow n \\
C & \xrightarrow{g} & D
\end{array}
$$

- closed under composition *if* $g, f \in \mathcal{A}$ *implies* $g \circ f \in \mathcal{A}$ *whenever g and f are composable;*
- closed under $\mathcal{B}$-decomposition *(where $\mathcal{B}$ is another subclass of* $\mathsf{Mor}(\mathbf{A})$*) if* $g \circ f \in \mathcal{A}$ *and* $g \in \mathcal{B}$ *implies* $f \in \mathcal{A}$;
- closed under decomposition *if it is closed under $\mathcal{A}$-decomposition.*

Remark 2.1. Clearly, "decomposition" corresponds to "left cancellation", but we prefer to stick to the name commonly used in literature (see e.g. [14]).

We are now ready to give the definition of $\mathcal{M}, \mathcal{N}$-adhesive category [14,25].

Definition 2.3. *Let* $\mathbf{A}$ *be a category and* $\mathcal{M} \subseteq \mathsf{Mono}(\mathbf{A})$, $\mathcal{N} \subseteq \mathsf{Mor}(\mathbf{A})$ *where*

(i) $\mathcal{M}$ *and* $\mathcal{N}$ *contain all isomorphisms and are closed under composition and decomposition;*
(ii) $\mathcal{N}$ *is closed under* $\mathcal{M}$*-decomposition;*
(iii) $\mathcal{M}$ *and* $\mathcal{N}$ *are stable under pullbacks and pushouts.*

Then we say that $\mathbf{A}$ *is* $\mathcal{M}, \mathcal{N}$*-adhesive if*

(a) *every cospan* $C \xrightarrow{g} D \xleftarrow{m} B$ *with* $m \in \mathcal{M}$ *can be completed to a pullback (such pullbacks will be called* $\mathcal{M}$*-pullbacks);*
(b) *every span* $C \xleftarrow{m} A \xrightarrow{n} B$ *with* $m \in \mathcal{M}$ *and* $n \in \mathcal{N}$ *can be completed to a pushout; such pushouts will be called* $\mathcal{M}, \mathcal{N}$*-pushouts;*
(c) $\mathcal{M}, \mathcal{N}$*-pushouts are Van Kampen squares.*

Remark 2.2. $\mathcal{M}$*-adhesivity* as defined in [2] coincides with $\mathcal{M}, \mathsf{Mor}(\mathbf{A})$-adhesivity, while *adhesivity* and *quasiadhesivity* [16,12] coincide with $\mathsf{Mono}(\mathbf{A})$-adhesivity and $\mathsf{Reg}(\mathbf{A})$-adhesivity, respectively. Notice that, in the $\mathcal{M}$-adhesive case, stability under pushouts of $\mathcal{M}$ derives from properties (a)–(c) of Definition 2.3, while closure under decomposition follows from stability under pullbacks in any category, so there is no need to prove it independently.

Other authors have introduced weaker notions of $\mathcal{M}$-adhesivity; see, e.g., [9,11,28], where our $\mathcal{M}$-adhesive categories are called *adhesive HLR categories*.

In general, proving that a given category is $\mathcal{M}, \mathcal{N}$-adhesive by verifying the conditions of Definition 2.3 may be long and tedious; hence, we seek criteria which are sufficient for adhesivity, and simpler to prove. A prominent example is the following result due to Lack and Sobociński.

Theorem 2.1 ([17], Thm. 26). *Any elementary topos is an adhesive category.*

In particular the category **Set** of sets and any presheaf category are adhesive. However, there are many important categories for (graph) rewriting which are not toposes, hence the need for more general criteria.

2.2   A new criterion for $\mathcal{M}, \mathcal{N}$-adhesivity

In this section we present our main result, i.e., that $\mathcal{M}, \mathcal{N}$-adhesivity is guaranteed by the existence of a family of functors with sufficiently nice properties. We will adapt some definitions from [1].

Definition 2.4. *Let* $I : \mathbf{I} \to \mathbf{C}$ *be a diagram and* J *a set. We say that a family* $F = \{F_j\}_{j \in J}$ *of functors* $F_j : \mathbf{C} \to \mathbf{D}_j$

1. jointly preserves (co)limits *of* I *if given a (co)limiting (co)cone* $(L, l_i)_{i \in \mathbf{I}}$ *for* I, *every* $(F_j(L), F_j(l_i))_{i \in \mathbf{I}}$ *is (co)limiting for* $F_j \circ I$;
2. jointly reflects (co)limits *of* I *if a (co)cone* $(L, l_i)_{i \in \mathbf{I}}$ *is (co)limiting for* I *whenever* $(F_j(L), F_j(l_i))_{i \in \mathbf{I}}$ *is (co)limiting for* $F_j \circ I$ *for every* $j \in J$;

3. jointly lifts (co)limits *of I if given a (co)limiting (co)cone* $(L_j, l_{j,i})_{i \in \mathbf{I}}$ *for every* $F_j \circ I$, *there exists a (co)limiting (co)cone* $(L, l_i)_{i \in \mathbf{I}}$ *for I such that* $(F_j(L), F_j(l_i))_{i \in \mathbf{I}} = (L_j, l_{j,i})_{i \in \mathbf{I}}$ *for every* $j \in J$;

4. jointly creates (co)limits *of I if* $F_j \circ I$ *has a (co)limit for every* $j \in J$, *I has a (co)limit and F jointly preserves and reflects it.*

Remark 2.3. Joint preservation, reflection, lifting or creation of (co)limits of $F = \{F_j : \mathbf{A} \to \mathbf{B}_j\}_{j \in J}$ is equivalent to the usual preservation, reflection, lifting or creation of (co)limits for the functor $\mathbf{A} \to \prod_{j \in J} \mathbf{B}_j$ induced by F. Notice that our notion of creation follows [22], which is more lax than, e.g., [19, Def. V.1].

Theorem 2.2. *Let* $\mathbf{A}$ *be a category,* $\mathcal{M} \subset \mathsf{Mono}(\mathbf{A})$, $\mathcal{N} \subset \mathsf{Mor}(\mathbf{A})$ *satisfying conditions (i)–(iii) of Definition 2.3, and F a non empty family of functors* $F_j : \mathbf{A} \to \mathbf{B}_j$ *such that* $\mathbf{B}_j$ *is* $\mathcal{M}_j, \mathcal{N}_j$-*adhesive.*

1. *If every* F_j *preserves pullbacks,* $F_j(\mathcal{M}) \subset \mathcal{M}_j$ *and* $F_j(\mathcal{N}) \subset \mathcal{N}_j$ *for every* $j \in J$, *F jointly preserves* $\mathcal{M},\mathcal{N}$-*pushouts, and jointly reflects pushout squares*

$$
\begin{array}{ccc}
F_j(A) & \xrightarrow{F_j(f)} & F_j(B) \\
F_j(m) \downarrow & & \downarrow F_j(n) \\
F_j(C) & \xrightarrow{F_j(g)} & F_j(D)
\end{array}
$$

   with $m, n \in \mathcal{M}$ *and* $f \in \mathcal{N}$, *then* $\mathcal{M},\mathcal{N}$-*pushouts in* $\mathbf{A}$ *are stable.*
   Moreover if in addition F jointly reflects $\mathcal{M}$-*pullbacks and* $\mathcal{N}$-*pullbacks then* $\mathcal{M},\mathcal{N}$-*pushouts are Van Kampen squares.*

2. *If F satisfies the assumptions of the previous points and jointly creates both* $\mathcal{M}$-*pullbacks and* $\mathcal{N}$-*pullbacks, then* $\mathbf{A}$ *is* $\mathcal{M},\mathcal{N}$-*adhesive.*

3. *If F jointly creates all pushouts and all pullbacks, then* $\mathbf{A}$ *is* $\mathcal{M}_F, \mathcal{N}_F$-*adhesive, where*

$$
\mathcal{M}_F := \{m \in \mathsf{Mor}(\mathbf{A}) \mid F_j(m) \in \mathcal{M}_j \text{ for every } j \in J\}
$$
$$
\mathcal{N}_F := \{n \in \mathsf{Mor}(\mathbf{A}) \mid F_j(n) \in \mathcal{N}_j \text{ for every } j \in J\}
$$

Proof. (1.) Take a cube in which the bottom face is an $\mathcal{M},\mathcal{N}$-pushout and all the vertical faces are pullbacks (below, left). Applying any $F_j \in F$ we get another cube in $\mathbf{B}_j$ (below, right) in which the bottom face is an $\mathcal{M}_j, \mathcal{N}_j$-pushout (because $F_j(m) \in \mathcal{M}_j$ and $F_j(n) \in \mathcal{N}_j$) and the vertical faces are pullbacks, thus the top face of the second cube is a pushout for every $j \in J$
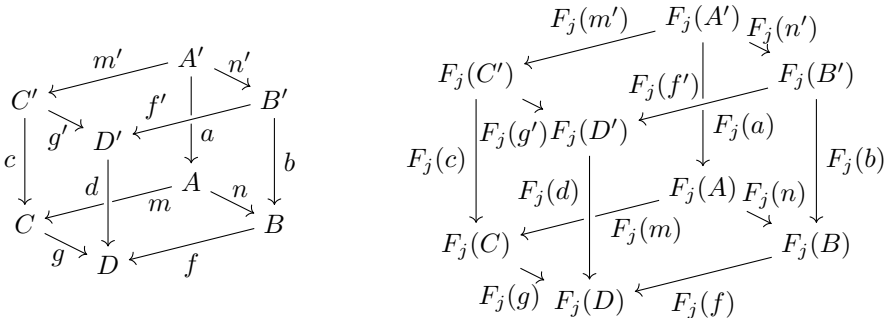
Now $m', f' \in \mathcal{M}$ and $n' \in \mathcal{N}$ since they are the pullbacks of m, f and n and thus we can conclude.

Suppose now that F jointly reflects $\mathcal{M}$-pullbacks and $\mathcal{N}$-pullbacks, we have to show that the front faces of the first cube above are pullbacks if the top one is a pushout. In the second cube, the bottom and top face are $\mathcal{M}_j, \mathcal{N}_j$-pushouts and the back faces are pullbacks, then the front faces are pullbacks too by $\mathcal{M}_j, \mathcal{N}_j$-adhesivity. Now, notice that $f \in \mathcal{M}$ and $g \in \mathcal{N}$ (since $\mathcal{M}$ and $\mathcal{N}$ are closed under pushouts) and thus we can conclude since F jointly reflects pullbacks along arrows in $\mathcal{M}$ or in $\mathcal{N}$.

(2.) Let us show properties (a), (b), (c) defining $\mathcal{M}, \mathcal{N}$-adhesivity.

(a) Given a cospan $C \xrightarrow{g} D \xleftarrow{m} B$ in $\mathbf{A}$ with $m \in \mathcal{M}$ we can apply $F_j \in F$ to it and get $F_j(C) \xrightarrow{F_j(g)} F_j(D) \xleftarrow{F_j(m)} F_j(B)$ which is a cospan in $\mathbf{B}_j$ with $F_j(g) \in \mathcal{M}_j$, thus, by hypothesis it has a limiting cone $(P_j, p_{F_j(B)}, p_{F_j(C)})$ in $\mathbf{B}_j$. Since F jointly creates $\mathcal{M}$-pullbacks there exists a limiting cone (P, p_B, p_C) for the cospan $C \xrightarrow{g} D \xleftarrow{m} B$.

(b) Analogously: for every span $C \xleftarrow{m} A \xrightarrow{n} B$ in $\mathbf{A}$ with $m \in \mathcal{M}$ and $n \in \mathcal{N}$, we have $F_j(C) \xleftarrow{F_j(m)} F_j(A) \xrightarrow{F_j(n)} F_j(B)$ in each $\mathbf{B}_j$ with $F_j(m) \in \mathcal{M}_j$ and $F_j(n) \in \mathcal{N}_j$ and thus there exists a colimiting cocone $(Q_j, q_{F_j(B)}, q_{F_j(C)})$ in $\mathbf{B}_j$. Now we can conclude because F jointly creates $\mathcal{M}, \mathcal{N}$-pushouts.

(c) This follows at once by the second half of the previous point.

(3.) By the previous point it is enough to show that $\mathcal{M}_F$ and $\mathcal{N}_F$ satisfy conditions (i)–(iii) of Definition 2.3.

(i) If $f \in \mathsf{Mor}(\mathbf{A})$ is an isomorphism then so is $F_j(f)$ for every $F_j \in F$. Thus $F_j(f)$ belongs to $\mathcal{M}_j$ and $\mathcal{N}_j$ for every $j \in J$, implying f is in $\mathcal{M}_F$ and in $\mathcal{N}_F$. The parts regarding composition and decomposition follow immediately by functoriality of each $F_j \in F$.

(ii) Suppose that $g \circ f \in \mathcal{N}_F$, with $g \in \mathcal{M}_F$ then for every $j \in F$ $F_j(g \circ f) = F_j(g) \circ F_j(f) \in \mathcal{N}_j$ and $F_j(g) \in \mathcal{M}_j$, thus $F_j(f) \in \mathcal{N}_j$ and so $f \in \mathcal{N}_F$.

(iii) Take a square

$$
\begin{array}{ccc}
A & \xrightarrow{f} & B \\
{\scriptstyle m}\downarrow & & \downarrow{\scriptstyle n} \\
C & \xrightarrow{g} & D
\end{array}
$$

and suppose that it is a pullback with $n \in \mathcal{M}_F$ ($\mathcal{N}_F$), then applying any $F_j \in F$ we get that $F_j(m)$ is the pullback of $F_j(n)$ along $F_j(g)$, since $F_j(n)$ is in $\mathcal{M}_j$ (in $\mathcal{N}_j$), which implies that $F_j(m) \in \mathcal{M}_j$ ($\mathcal{N}_j$). This is true for every $j \in J$, from which the thesis follows. Stability under pushouts is proved applying the same argument to m.   □

Applying the previous theorem to the families given by, respectively, projections, evaluations and the inclusion we get immediately the following three corollaries (cfr. also [9, Thm. 4.15]).

Corollary 2.1. *Let* $\{\mathbf{A}\}_{i \in I}$ *be a family of categories such that each* $\mathbf{A}_i$ *is* $\mathcal{M}_i, \mathcal{N}_i$-*adhesive. Then the product category* $\prod_{i \in I} \mathbf{A}_i$ *is* $\prod_{i \in I} \mathcal{M}_i, \prod_{i \in I} \mathcal{N}_i$-*adhesive, where*

$$\prod_{i \in I} \mathcal{M}_i := \{(m_i)_{i \in I} \in \mathsf{Mor}(\prod_{i \in I} \mathbf{A}_i) \mid m_i \in \mathcal{M}_i \text{ for every } i \in I\}$$

$$\prod_{i \in I} \mathcal{N}_i := \{(n_i)_{i \in I} \in \mathsf{Mor}(\prod_{i \in I} \mathbf{A}_i) \mid n_i \in \mathcal{N}_i \text{ for every } i \in I\}$$

Corollary 2.2. *Let* $\mathbf{A}$ *be an* $\mathcal{M}, \mathcal{N}$-*adhesive category. Then for every other category* $\mathbf{C}$, *the category of functors* $\mathbf{A}^{\mathbf{C}}$ *is* $\mathcal{M}^{\mathbf{C}}, \mathcal{N}^{\mathbf{C}}$-*adhesive, where*

$$\mathcal{M}^{\mathbf{C}} := \{\eta \in \mathsf{Mor}(\mathbf{A}^{\mathbf{C}}) \mid \eta_C \in \mathcal{M} \text{ for every object } C \text{ of } \mathbf{C}\}$$

$$\mathcal{N}^{\mathbf{C}} := \{\eta \in \mathsf{Mor}(\mathbf{A}^{\mathbf{C}}) \mid \eta_C \in \mathcal{N} \text{ for every object } C \text{ of } \mathbf{C}\}$$

Corollary 2.3. *Let* $\mathbf{A}$ *be a full subcategory of an* $\mathcal{M}, \mathcal{N}$-*adhesive category* $\mathbf{B}$ *and* $\mathcal{M}' \subset \mathsf{Mono}(\mathbf{A})$, $\mathcal{N}' \subset \mathsf{Mor}(\mathbf{A})$ *satisfying the first three conditions of Definition 2.3 such that* $\mathcal{M}' \subset \mathcal{M}$, $\mathcal{N}' \subset \mathcal{N}$ *and* $\mathbf{A}$ *is closed in* $\mathbf{B}$ *under pullbacks and* $\mathcal{M}', \mathcal{N}'$-*pushouts. Then* $\mathbf{A}$ *is* $\mathcal{M}', \mathcal{N}'$-*adhesive.*

2.3   Comma categories

In this section we show how to apply Theorem 2.2 to the comma construction [19] in order to guarantee some adhesivity properties under suitable hypotheses.

Definition 2.5. *For any two functors* $L : \mathbf{A} \to \mathbf{C}$, $R : \mathbf{B} \to \mathbf{C}$, *the* comma *category* $L \downarrow R$ *is the category in which*

- *objects are triples* (A, B, f) *with* $A \in \mathbf{A}$, $B \in \mathbf{B}$, *and* $f : L(A) \to R(B)$;
- *a morphism* $(A, B, f) \to (A', B', g)$ *is a pair* (h, k) *with* $h : A \to A'$, $k : B \to B'$ *such that the following diagram commutes*

$$\begin{array}{ccc} L(A) & \xrightarrow{L(h)} & L(A') \\ f \downarrow & & \downarrow g \\ R(C) & \xrightarrow{R(k)} & R(C') \end{array}$$

We have two obvious forgetful functors

$$\begin{array}{cc} U_L : L \downarrow R \to \mathbf{A} & U_R : L \downarrow R \to \mathbf{B} \\ (A, B, f) \longmapsto A & (A, B, f) \longmapsto B \\ (h, k) \downarrow \qquad \downarrow h & (h, k) \downarrow \qquad \downarrow k \\ (A', B', g) \longmapsto A' & (A', B', g) \longmapsto B' \end{array}$$

Example 2.1. **Graph** *is equivalent to the comma category made from the identity functor on* **Set** *and the product functor sending* X *to* $X \times X$.

We have a classic result relating limits and colimits in the comma category with those preserved by L or R.

Lemma 2.1. *Let $I : \mathbf{I} \to L{\downarrow}R$ be a diagram such that L preserves the colimit (if it exists) of $U_L \circ I$. Then the family $\{U_L, U_R\}$ jointly creates colimits of I.*

Corollary 2.4. *The family $\{U_L, U_R\}$ jointly creates limits along every diagram $I : \mathbf{I} \to L{\downarrow}R$ such that R preserves the limit of $U_R \circ I$.*

Proof. Apply the previous lemma to $R^{op}{\downarrow}L^{op}$ which is equivalent to $(L{\downarrow}R)^{op}$. $\qquad\square$

We are now able to deduce the following result from Theorem 2.2.

Theorem 2.3. *Let $\mathbf{A}$ and $\mathbf{B}$ be respectively $\mathcal{M}, \mathcal{N}$-adhesive and $\mathcal{M}', \mathcal{N}'$-adhesive categories, $L : \mathbf{A} \to \mathbf{C}$ a functor that preserves $\mathcal{M}, \mathcal{N}$-pushouts, and $R : \mathbf{B} \to \mathbf{C}$ a pullback preserving one. Then $L{\downarrow}R$ is $\mathcal{M}{\downarrow}\mathcal{M}', \mathcal{N}{\downarrow}\mathcal{N}'$-adhesive, where*

$$\mathcal{M}{\downarrow}\mathcal{M}' := \{(h, k) \in \mathsf{Mor}(L{\downarrow}R) \mid h \in \mathcal{M}, k \in \mathcal{M}'\}$$
$$\mathcal{N}{\downarrow}\mathcal{N} := \{(h, k) \in \mathsf{Mor}(L{\downarrow}R) \mid h \in \mathcal{N}, k \in \mathcal{N}'\}.$$

3   Some paradigmatic examples

In this section we apply the results provided in Section 2, to some important categories, such as directed (acyclic) graphs, hierarchical (hyper)graphs, directed (acyclic) hypergraphs, and term graphs. These examples have been chosen for their importance in graph rewriting, and because we can recover their $\mathcal{M}, \mathcal{N}$-adhesivity in a uniform and systematic way. In fact, in the case of hierarchical (hyper)graphs we give the first proof of $\mathcal{M}, \mathcal{N}$-adhesivity, to our knowledge.

3.1   Directed (acyclic) graphs

Among visual formalisms, directed (also known as "simple") graphs represent one of the most-used paradigms, since they adhere to the classical view of graphs as relations included in the cartesian product of vertices. It is also well-known that directed graphs are not quasiadhesive [15], not even in their acyclic variant. In this section we are going to exploit Corollary 2.3 to show that these categories of (acyclic) graphs have nevertheless adhesivity properties.

Definition 3.1. *A directed multigraph is a 4-tuple (E, V, s, t) where E and V are sets, called the set of edges and nodes respectively, and $s, t : E \to V$ are functions, called source and target. An edge e is between v and w if $s(e) = v$ and $t(e) = w$, $E(v, w)$ is the set of edges between v and w. A morphism $(E, V, s, t) \to (F, W, s', t')$ is a pair (f, g) of functions $f : E \to F$, $g : V \to W$ such that the following diagrams commute*

$$
\begin{array}{ccc}
E \xrightarrow{\ s\ } V & \quad & E \xrightarrow{\ t\ } V \\
f\downarrow \quad \downarrow g & \quad & f\downarrow \quad \downarrow g \\
F \xrightarrow[\ s'\ ] W & \quad & F \xrightarrow[\ t'\ ] W
\end{array}
$$

We will denote by **Graph** *the category so defined. A* directed graph *is a directed multigraph in which there is at most one edge between two nodes,* **DGraph** *is the full subcategory of* **Graph** *given by directed graphs.*

A path $[e_i]_{i=1}^n$ *in a directed multigraph is a finite list of edges such that* $t(e_i) = s(e_{i+1})$ *for all* $1 \le i \le n-1$. *A path is called a* cycle *if* $s(e_1) = t(e_n)$. *A* directed acyclic graph *is a directed graph without cycles, directed acyclic graphs form a full subcategory* **DAG** *of* **DGraph** *and* **Graph**.

Remark 3.1. **Graph** is equivalent to the category of presheaves on $\bullet \rightrightarrows \bullet$, the category with just two objects and only two parallel arrows between them (besides the identities), thus it is a topos and as such adhesive. Notice that this also implies that limits and colimits are computed component-wise and that an arrow in **Graph** is mono if and only if both its underlying functions are injective.

Remark 3.2. Notice that if $(f, g) : (E, V, s, t) \to (F, W, s', t')$ is an arrow in **DGraph** with f injective, then g is injective too.

We will state now two categorical properties of **DGraph** that will be useful in the following.

Proposition 3.1. *The following properties hold*

1. *the inclusion functor $I : \textbf{DGraph} \to \textbf{Graph}$ has a left adjoint $L : \textbf{Graph} \to$ **DGraph** which sends a graph (V, E, s, t) to the graph on the same vertices but in which edges with the same source and target are identified;*
2. *an arrow $(f, g) : (E, V, s, t) \to (F, W, s', t')$ of **DGraph** is a regular monomorphism if and only if f is injective and $E(v_1, v_2)$ is non empty whenever $F(f(v_1), f(v_2)) \neq \emptyset$.*

Remark 3.3. Notice that, since L does not modify the vertices part of a graph, Remark 3.2 implies that L preserves monomorphisms.

Example 3.1. In [15] it is shown that **DGraph** is not quasiadhesive. Take the cube
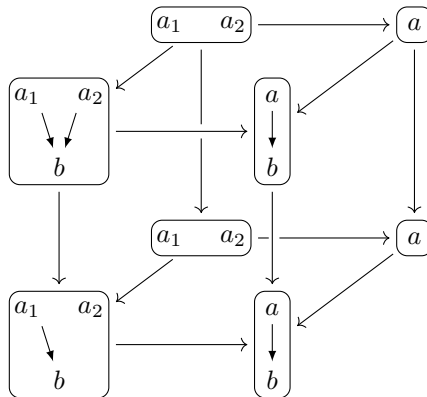
By the results of Proposition 3.1 the top and bottom faces are pushouts along regular monos and the back faces are pullbacks, but the front one is not, contradicting the Van Kampen property. The same example shows that even **DAG** is not quasiadhesive.

Definition 3.2. *A monomorphism* $(f, g) : (E, V, s, t) \to (F, W, s', t')$ *in* **Graph** *is said to be* downward closed *if, for all* $e \in F$, $e \in f(E)$ *whenever* $t'(e) \in g(V)$ *(in particular this implies that* $s'(e) \in g(V)$ *too). We denote by* dclosed, dclosed$_d$ *and* dclosed$_{da}$ *the classes of downward closed morphisms in* **Graph**, **DGraph** *and* **DAG** *respectively.*

Remark 3.4. The functor L of Proposition 3.1 sends downward closed morphisms to downward closed morphisms.

Remark 3.5. By Proposition 3.1 it is clear that any downward closed morphism is regular. The vice-versa does not hold: a counterexample is given by

$$
\boxed{b} \longrightarrow \boxed{\begin{array}{c} a \\ \downarrow \\ b \end{array}}
$$
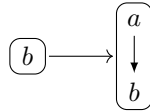
Lemma 3.1. DGraph *and* **DAG** *are closed in* **Graph** *under pullbacks. Moreover,* **DGraph** *is closed under* Reg(**DGraph**), Mono(**DGraph**)*-pushouts and* **DAG** *under* dclosed$_{da}$, Mono(**DAG**)*-pushouts.*

Theorem 3.1. *The category* **DGraph** *is* Reg(**DGraph**), Mono(**DGraph**)- *and* Mono(**DGraph**), Reg(**DGraph**)*-adhesive, while* **DAG** *is* dclosed$_{da}$, Mono(**DAG**)*-adhesive.*

3.2   Tree Orders

In this section we present *trees* as partial orders and show that the resulting category is actually a topos of presheaves, hence adhesive. This fact will be exploited in Section 3.3 to construct a category of hierarchical graphs, where the hierarchy between edges is modelled by trees.

Definition 3.3. *A tree order is a partial order* $(E, \leq)$ *such that for every* $e \in E$, $\downarrow e$ *is a finite set totally ordered by the restriction of* $\leq$. *Since* $\downarrow e$ *is a finite chain we can define the* immediate predecessor function

$$
i_E : E \to E \sqcup \{*\} \qquad e \mapsto \begin{cases} \max(\downarrow e \smallsetminus \{e\}) & \downarrow e \neq \{e\} \\ * & \downarrow e = \{e\} \end{cases}
$$

Let i_E^0 *be the inclusion* $E \to E \sqcup \{*\}$; *then, for any* $k \in \mathbb{N}_+$, *the* k^{th} *predecessor function* $i_E^k : E \to E \sqcup \{*\}$ *is defined by induction as follows:*

$$
e \mapsto \begin{cases} i_E(i_E^{k-1}(e)) & i_E^{k-1}(e) \in E \\ * & i_E^{k-1}(e) = * \end{cases}
$$

Let $f : (E, \leq) \to (F, \leq)$ be a monotone map and $f_* : E \sqcup \{*\} \to F \sqcup \{*\}$ be its extension sending $*$ to $*$. We say that f is strict *if the following diagram commutes*

$$
\begin{array}{ccc}
E & \xrightarrow{i_E} & E \sqcup \{*\} \\
f \downarrow & & \downarrow f_* \\
F & \xrightarrow[i_F]{} & F \sqcup \{*\}
\end{array}
$$

We define the category **Tree** *as the subcategory of* **Poset** *given by tree orders and strict morphisms.*

Example 3.2. A strict morphisms is simply a monotone function that preserves immediate predecessors (and thus every predecessor). For instance the function $\{0\} \to \{0, 1\}$ sending 0 to 1 and where we endow the codomain with the order $0 \leq 1$, is not a strict morphism.

Remark 3.6. Clearly $i_E^1 = i_E$ and it holds that $i_E^k(e) = *$ if and only if $|{\downarrow}e| \leq k$. In this case an easy induction shows that $\left|{\downarrow}i_E^k(e)\right| = |{\downarrow}e| - k$.

Remark 3.7. We have an obvious forgetful functor

$$
\begin{array}{c}
|-| : \textbf{Tree} \to \textbf{Set} \\
(E, \leq) \longmapsto E \\
f \downarrow \qquad \downarrow f \\
(F, \leq) \longmapsto F
\end{array}
$$

Remark 3.8. Let $(E, \leq)$ be an object of **Tree** and ω the first infinite ordinal, then we can define its *associated presheaf* $\widehat{E} : \omega^{op} \to \textbf{Set}$ sending n to the set

$$
\{e \in E \mid |{\downarrow}e \smallsetminus \{e\}| = n\}
$$

If $n \leq m$ in ω, we can define a function

$$
\iota_{n,m}^E : \widehat{E}(m) \to \widehat{E}(n) \qquad e \mapsto i_E^{m-n}(e)
$$

which is well defined since $|{\downarrow}e| > m - n$ so

$$
\left|{\downarrow}i_E^{m-n}(e)\right| = |{\downarrow}e| - m + n = m + 1 - m + n = n + 1
$$

Notice that if $m = n$, $i_E^{m-n}(e)$ is the identity, while for any $k \leq n \leq m$ we have

$$
\iota_{k,n}^E(\iota_{n,m}^E(e)) = i_E^{n-k}(i_E^{m-n}(e)) = i_E^{n-k+m-n}(e) = i_E^{m-k}(e) = \iota_{m-k}^E(e)
$$

so $\widehat{E}$ is really a presheaf on ω.

Theorem 3.2. *There exists an equivalence of categories* $\widehat{(-)} : \textbf{Tree} \to \textbf{Set}^{\omega^{op}}$ *sending* $(E, \leq)$ *to* $\widehat{E}$.

Corollary 3.1. **Tree** *is adhesive and the forgetful functor* $|-| : \textbf{Tree} \to \textbf{Set}$ *preserves all colimits.*

3.3   Various kinds of hierarchical graphs

In this section we construct several categories of hierarchical graphs combining
sufficiently adhesive categories of preorders or graphs (modelling the hierarchy
between the edges) and the wanted structure on the nodes. For each of them
we can readily prove suitable adhesivity properties, leveraging the modularity
provided by Theorem 2.2. Besides hypergraphs and interfaces, this methodology
can be applied to other settings such as Petri nets (see [10]).

Hierarchical graphs We can use trees to produce a category of hierarchical graphs
[24], which, in addition, can be equipped with an interface, modelled by a func-
tion into the set of nodes.

Definition 3.4. *The category* **HIGraph** *of hierarchical graphs with interface
has as objects 6-tuples* $((E, \leq), V, X, f, s, t)$ *where* $(E, \leq)$ *is a tree order,* f *is a
function* $X \to V$ *and* s, t *are functions* $E \to V$, *and as arrows triples* $(h, k, l) :$
$((E, \leq), V, X, f, s, t) \to ((F, \leq), W, Y, g, s', t')$ *with* $h : (E, \leq) \to (F, \leq)$ *in* **Tree**,
$k : V \to W$ *and* $l : X \to Y$ *in* **Set** *such that the following squares commute*

$$
\begin{array}{ccc}
E \xrightarrow{s} V & E \xrightarrow{t} V & X \xrightarrow{f} V \\
h\downarrow \quad \downarrow k & h\downarrow \quad \downarrow k & l\downarrow \quad \downarrow k \\
F \xrightarrow{s'} W & F \xrightarrow{t'} W & Y \xrightarrow{g} W
\end{array}
$$

We can realise **HIGRaph** as a comma category: as L we take the functor $|-| :$
Tree $\to$ **Set** of Remark 3.7, while as R we take the composition of $\mathsf{cod} : \mathbf{Set}^2 \to$
Set, sending an arrow to its codomain, with the functor **Set** $\to$ **Set** that sends
a set X to $X \times X$. Notice that cod preserves limits since it coincides with the
forgetful functor $\mathsf{id}_{\mathbf{Set}} \downarrow \mathsf{id}_{\mathbf{Set}}$, so we can apply Theorem 2.3 to get the following.

Theorem 3.3. HIGraph *is an adhesive category.*

The next step is to move to hypergraphs, using the Kleene star $(-)^\star :$ **Set** $\to$
Set (the monoid monad) instead of the product functor. This step is not trivial:
it relies on the fact that the monoid monad preserves all connected limits (such
monads are called *cartesian*), which in turn rests upon the fact that the theory
of monoids is a *strongly regular theory* (see [5, Sec. 3] and [18, Ch.4] for details).

Hierarchical hypergraphs A variation on the previous example is obtained by
allowing an edge to be mapped to an arbitrary subset of nodes. In this way, we
obtain a category of hypergraphs whose edges form a tree order, corresponding
to Milner's (pure) bigraphs [20], with possibly infinite edges[3].

Definition 3.5. *The category* **HHGraph** *of hierarchical hypergraphs with in-
terface has as objects 5-tuples* $((E, \leq), V, X, f, e)$ *where* $(E, \leq)$ *is a tree order
and* $f : X \to V$, $e : E \to V^\star$ *two functions; arrows are triples* $(h, k, l) :$
$((E, \leq), V, X, f, e) \to ((F, \leq), W, Y, g, e')$ *with* $h : (E, \leq) \to (F, \leq)$ *in* **Tree**,
$k : V \to W$ *and* $l : X \to Y$ *in* **Set** *such that the following squares commute*

---

[3] In bigraph terminology, "controls" and "edges" correspond to our edges and nodes.
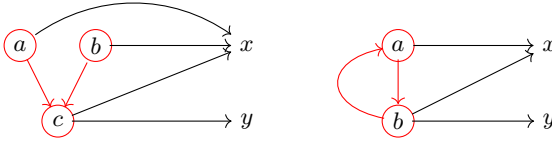
Fig. 1. A **DAG**-hypergraph (left) and a **DGraph**-hypergraph corresponding to the CCS process $P = a(x).b(xy).P$ (right). Relation between edges is depicted in red.

$$
\begin{array}{ccc}
E \xrightarrow{e} V^* & X \xrightarrow{f} V \\
h\downarrow \quad \downarrow k^* & l\downarrow \quad \downarrow k \\
F \xrightarrow[e']{} W^* & Y \xrightarrow[g]{} W
\end{array}
$$

Even in this case **HHGraph** is a comma category: on the left side we take $|-|$ as before, on the right side we take the composition of cod with the Kleene star, so even in this case we can deduce adhesivity.

Theorem 3.4. HHGRaph *is adhesive.*

DGraph *and* **DAG**-*hypergraphs*   We can consider more general relations between edges, besides tree orders. An interesting case is when edges form a directed acyclic graph, yielding the category of **DAG**-*hypergraphs*; this corresponds to (possibly infinite) *bigraphs with sharing*, where an edge can have more than one parent, as in [27] (see also Fig. 1, left). Even more generally, we can consider any relation between edges, i.e., the edges form a generic directed graph possibly with cycles, yielding the category of **DGraph**-*hypergraphs*. These can be seen as "recursive bigraphs", i.e., bigraphs which allow for cyclic dependencies between controls, like in recursive processes; an example is in Fig. 1 (right).

Definition 3.6. *We define the category of* **DGraph**-*hypergraphs (respectively* **DAG**-*hypergraphs) with interface* **DHGraph** *(***DAGHGraph***) as the one in which objects are 5-tuples* $((E, T, s, t), V, X, f, e)$ *where* (E, T, s, t) *is in* **DGraph** *(in* **DAG**), *f is a function* $X \to V$, *and* e *a function* $T \to V^\star$ *and as arrows triple* $((h_1, h_2), k, l) : ((E, T, s, t), V, X, f, e) \to ((F, T', s', t'), W, Y, g, e')$ *with* $(h_1, h_2) : (E, T, s, t) \to (F, T', s', t')$ *in* **DAG** *(in* **DGraph**), $k : V \to W$ *and* $l : X \to Y$ *in* **Set** *such that the following squares commute*

$$
\begin{array}{ccc}
T \xrightarrow{e} V^\star & X \xrightarrow{f} V \\
h_2\downarrow \quad \downarrow k^\star & l\downarrow \quad \downarrow k \\
T' \xrightarrow[e']{} W^\star & Y \xrightarrow[g]{} W
\end{array}
$$

We can realise also **DHGraph** and **DAGHGraph** as comma categories: it is enough to take respectively the forgetful functors **DGraph** $\to$ **Set** and **DAG** $\to$ **Set** on one side and again the composition of the Kleene star with cod.

Theorem 3.5. DHGraph *is adhesive with respect to the classes*

$$\{((h_1, h_2), k, l) \in \mathsf{Mor}(\mathbf{DHGraph}) \mid (h_1, h_2) \in \mathsf{Reg}(\mathbf{DGraph}), k, l \in \mathsf{Mono}(\mathbf{Set})\}$$
$$\{((h_1, h_2), k, l) \in \mathsf{Mor}(\mathbf{DHGraph}) \mid (h_1, h_2) \in \mathsf{Mono}(\mathbf{DGraph})\}$$

while **DAGHGraph** *is adhesive with respect to the classes*

$$\{((h_1, h_2), k, l) \in \mathsf{Mor}(\mathbf{DAGHGraph}) \mid (h_1, h_2) \in \mathsf{dclosed_{da}}, k, l \in \mathsf{Mono}(\mathbf{Set})\}$$
$$\{((h_1, h_2), k, l) \in \mathsf{Mor}(\mathbf{DHGraph}) \mid (h_1, h_2) \in \mathsf{Mono}(\mathbf{DAG})\}$$

3.4   Term graphs

The use of term graphs has been advocated as a tool for the optimal implementation of terms, with the intuition that the graphical counterpart of trees can allow for the sharing of sub-terms [26]. A brute force proof of quasiadhesivity of the category of terms graphs was given in [7]. In this section we recover that result by exploiting our new criterion for adhesivity.

Definition 3.7. *Let $\Sigma = (O, \mathsf{ar})$ be an algebraic signature (O is a set and* ar : $O \to \mathbb{N}$ *a function called* arity function*). A* term graph *over Σ is a triple (V, l, s) where V is a set, $l : V \rightharpoonup O$, $s : V \rightharpoonup V^\star$ are partial functions such that*

- *$\mathrm{dom}(l) = \mathrm{dom}(s)$;*
- *for each $v \in \mathrm{dom}(l)$, $\mathsf{ar}(l(v)) = \mathsf{length}(s(a))$, where $\mathsf{length} : V^\star \to \mathbb{N}$ associates to each word its length.*

Elements of V are called nodes*, a node v not in $\mathrm{dom}(l)$ is called* empty*. A morphism $(V, l, s) \to (W, t, r)$ is a function $f : V \to W$ such that*

$$t(f(v)) = l(v) \qquad r(f(v)) = f^\star(s(v))$$

for every $v \in \mathrm{dom}(l)$. We will denote by $\mathbf{TG}_\Sigma$ the category of term graphs over Σ and their morphisms. We will use $\mathcal{U}$ to denote the forgetful functor $\mathbf{TG}_\Sigma \to \mathbf{Set}$ sending a term graph to the set of its nodes and that is the identity on arrows.

Definition 3.8. *We define a functor $\Delta : \mathbf{Set} \to \mathbf{TG}_\Sigma$ putting*

$$\begin{array}{ccc}
X & \longmapsto & (X, e_1, e_2) \\
f \downarrow & & \downarrow f \\
Y & \longmapsto & (Y, e_1', e_2')
\end{array}$$

where the domains of the structural functions e_1, e_2 of $\Delta(X)$ are the empty set.

Lemma 3.2. *The following properties hold*

1. *$\Delta \dashv \mathcal{U}$;*
2. *$\mathbf{TG}_\Sigma$ has equalizers and binary products.*

Remark 3.9. Right adjoints preserves monomorphisms, so, by the first point of Lemma 3.2, if $f : (V, l, s) \to (W, t, r)$ is a monomorphism then its underlying function is injective. On the other hand $\mathcal{U}$ is faithful and thus reflects monomorphisms, i.e. also the other implication holds.

Remark 3.10. $\mathbf{TG}_\Sigma$ in general does not have terminal objects. Since $\mathcal{U}$ preserves limits, if a terminal object exists it must have the singleton as set of nodes. Now take as signature the one given by two operations $\{a, b\}$ both of arity 0, then we have three term graphs with only one node v: $\Delta(\{v\})$, $(\{v\}, l, s)$ and $(\{v\}, t, s)$ where $l(v) = a$, $t(v) = b$ and s sends v to the empty word. Clearly there are no morphisms between the last two and from the last two to the first one, and thus neither of them can be terminal.

Remark 3.11. $\mathbf{TG}_\Sigma$ is not an adhesive category. In particular it does not have pushouts along all monomorphisms. Take the signature of the previous remark, then we can use the identity $\{v\} \to \{v\}$ to form a span

$$(\{v\}, l, s) \xleftarrow{i} \Delta(\{v\}) \xrightarrow{i'} (\{v\}, t, s).$$

This span cannot be completed to commutative a square: if

$$
\begin{array}{ccc}
\Delta(\{v\}) & \xrightarrow{i} & (\{v\}, t, s) \\
{\scriptstyle i'}\downarrow & & \downarrow{\scriptstyle g} \\
(\{v\}, l, s) & \xrightarrow[f]{} & (V, p, r)
\end{array}
$$

is commutative then $f(v) = g(v)$; therefore

$$a = l(v) = p(f(v)) = p(g(v)) = t(v) = b$$

and this is absurd.

Remark 3.12. It is worth to spell out the explicit construction of equalizers in $\mathbf{TG}_\Sigma$. Given two arrows $f, g : (V, l, s) \to (W, t, r)$, let

$$E = \{v \in V \mid f(v) = g(v)\}$$

be the equalizer of $\mathcal{U}(f)$ and $\mathcal{U}(g)$ in **Set**. We have a partial function $p : E \rightharpoonup O$ given by the restriction of l to E. Moreover, if $v \in E \cap \mathrm{dom}(s)$ then

$$f^\star(s(v)) = r(f(v)) = r(g(v)) = g^\star(s(v))$$

hence $s(v) \in E^\star$ (which is the equalizer of $f^\star$ and $g^\star$, see [5]), thus we can restrict s to $q : E \rightharpoonup E^\star$. In this way we get a term graph (E, p, q) with an arrow into (V, l, s) which clearly equalize f and g.

On the other hand, if $k : (U, a, b) \to (V, l, s)$ is such that

$$g \circ k = f \circ k$$

then the induced function $\bar{k} : U \to E$ is a morphism of $\mathbf{TG}_\Sigma$.

Remark 3.13. Lemma 3.2 implies that $\mathbf{TG}_\Sigma$ has pullbacks. In the following we will need their explicit description. The pullback of a cospan

$$(V, l, s) \xrightarrow{f} (W, t, r) \xleftarrow{g} (U, a, b)$$

is given by (P, p, q) where

$$P = \{(v, u) \in V \times U \mid f(u) = g(v)\}$$

is the pullback of f along g in **Set** and

$$p : P \rightharpoonup O \qquad (v, u) \mapsto \begin{cases} l(v) & v \in \mathrm{dom}(l), w \in \mathrm{dom}(t) \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$q : P \rightharpoonup P^\star \qquad (v, u) \mapsto \begin{cases} [(s(v)_i, r(u)_i)]_{i=1}^{\mathrm{ar}(l(v))} & v \in \mathrm{dom}(l), w \in \mathrm{dom}(t) \\ \text{undefined} & \text{otherwise} \end{cases}$$

where, given $x \in X^\star$, x_i denotes its ith letter and, given $x_1, \ldots, x_n \in X$, $[x_i]_{i=1}^n$ denotes the element in $X^\star$ such that $([x_i]_{i=1}^n)_i$ is exactly x_i.

Now, notice that q is the unique partial function $P \rightharpoonup P^\star$ that makes the projections arrows of $\mathbf{TG}_\Sigma$. Moreover even p has a uniqueness property: it is the unique partial function $P \rightharpoonup O$ such that the projections are arrows of $\mathbf{TG}_\Sigma$ and $p(x)$ is undefined if and only if at least one of its image is undefined. In particular this implies the following result.

Proposition 3.2. $\mathcal{U}$ *creates pullbacks along arrows which preserves empty nodes.*

This is especially useful when paired with the following result from [7].

Proposition 3.3 ([7], Prop. 4.3). *An arrow* $f : (V, l, s) \to (W, t, r)$ *in* $\mathbf{TG}_\Sigma$ *is a regular mono if and only if* f *is injective and preserves empty nodes.*

Proof. ($\Rightarrow$) Follows by the construction of equalizers given in Remark 3.12.
($\Leftarrow$) Consider (U, a, b) where $U = W \sqcup (W \smallsetminus f(V))$. Let i_1 and i_2 be the inclusions of W and $W \smallsetminus f(V)$ into U, we can define

$$a : U \rightharpoonup O \qquad u \mapsto \begin{cases} t(w) & u = i_1(w), w \in \mathrm{dom}(t) \\ t(w) & u = i_2(w), w \in (W \smallsetminus f(V)) \cap \mathrm{dom}(t) \\ \text{undefined} & \text{otherwise} \end{cases}$$

while for $b : U \rightharpoonup U^\star$, we put $b(u) = r(w)$ if $u = i_1(w), w \in \mathrm{dom}(r)$, while if $u = i_2(w)$ with $w \in \mathrm{dom}(r)$ we define $b(u) = [u_i]_{i=1}^{\mathrm{ar}(a(u))}$ where

$$u_i = \begin{cases} i_2(r(w)_i) & r(w) \in W \smallsetminus f(V) \\ i_1(r(w)_1) & r(w) \in f(V) \end{cases}$$

We have two functions $(V, t, r) \to (U, a, b)$: one is just i_1, while the other one is given by

$$g : W \to U \qquad w \mapsto \begin{cases} i_1(w) & w \in f(V) \\ i_2(w) & w \notin f(V) \end{cases}$$

Now, $i_1 \circ f$ and $g \circ f$ both send v to $i_1(f(v))$, therefore

$$i_1 \circ f = g \circ f$$

Suppose that $h : (P, p, q) \to (W, t, r)$ equalizes i_1 and g, thus $h(x) \in f(V)$ for every $x \in P$, and we have a unique function $h' : P \to V$ such that $f \circ h' = h$. For every $x \in \mathrm{dom}(p)$, $t(h(x)) = p(x)$, thus $h(x) = f(h'(x)) \in \mathrm{dom}(t)$. Since f preserves the empty nodes, $h'(x)$ belongs to $\mathrm{dom}(l)$, so:

$$p(x) = t(h(x)) = t(f(h'(x))) = l(h'(x))$$

Preservation of successors follows at once, while uniqueness follows from the uniqueness of the function h' in **Set**. $\qquad\square$

Lemma 3.3. $\mathcal{U}$ preserves and lifts pushouts along regular monomorphisms, moreover it reflects all pushout squares

$$
\begin{array}{ccc}
\mathcal{U}(P, p, q) & \xrightarrow{\;\mathcal{U}(f)\;} & \mathcal{U}(W, t, r) \\
\mathcal{U}(m) \downarrow & & \downarrow \mathcal{U}(n) \\
\mathcal{U}(V, l, s) & \xrightarrow{\;\mathcal{U}(g)\;} & \mathcal{U}(U, a, b)
\end{array}
$$

in which n is regular. In addition $\mathsf{Reg}(\mathbf{TG}_\Sigma)$ is closed under pushouts.

We can now use the first point of Theorem 2.2 to get half of the following result.

Theorem 3.6 ([7, Thm. 4.2]). *The category* $\mathbf{TG}_\Sigma$ *is quasi-adhesive.*

Proof. We already know by Lemmas 3.2 and 3.3 and Theorem 2.2 that pushouts along regular monos are stable. So, let us take a cube
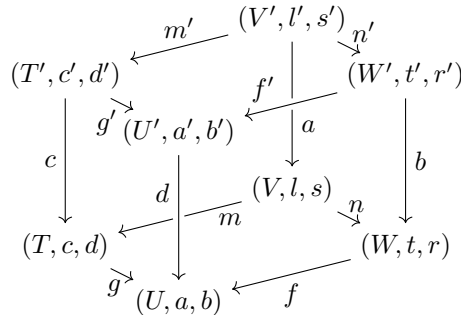
in which m is regular, the top and bottom faces are pushouts and the back faces pullbacks. Applying $\mathcal{U}$ we get another cube

$$
\begin{array}{ccc}
 & m' \quad V' \quad n' & \\
T' \quad f' \quad | \quad W' & \\
c \quad g' \quad U' \quad a \quad b & \\
 & d \quad V \quad n & \\
T \quad m \quad E & \\
g \quad W \quad f &
\end{array}
$$
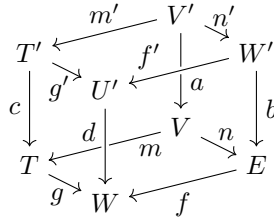
with pushouts along monos as top and bottom faces and pullbacks as vertical ones. By Proposition 3.2 $\mathcal{U}$ creates pullbacks along regular monos and $f \in \mathsf{Reg}(\mathbf{TG}_\Sigma)$, then we can conclude that the front right face of the starting cube is a pullback as well. We have to show that the front left face of the starting cube is a pullback too. Suppose it is not, then, by the explicit description of pullbacks, there must be a node $t \in T'$ which is empty in (T', c', d') and such that $g'(t)$ and $c(t)$ are non empty. By the computation of pushouts along regular monos we can deduce that $g'(t) \in \mathrm{dom}(a')$ implies the existence of $v \in V'$, necessarily empty, such that $m'(v) = t$ and $f'(n'(v)) = g'(t)$, thus $n'(v)$ is non empty since f' is regular. Moreover, $c(m'(v)) = m(a(v))$ and the left hand side is non empty, therefore even $a(v)$ is non empty by the regularity of m, but this contradicts the hypothesis that the back right face is a pullback.   □

4   Conclusions

In this paper we have introduced a new criterion for $\mathcal{M}, \mathcal{N}$-adhesivity, based on the verification of some properties of functors connecting the category of interest to a family of suitably adhesive categories. This criterion can be seen as a distilled abstraction of many *ad hoc* proofs of adhesivity found in literature. This criterion allows us to prove in a uniform and systematic way some previous results about the adhesivity of categories built by products, exponents, and comma construction. We have applied the criterion to several significant examples, such as term graphs and directed (acyclic) graphs; moreover, using the modularity of our approach, we have readily proved suitable adhesivity properties to categories constructed by combining simpler ones. In particular, we have been able to tackle the adhesivity problem for several categories of hierarchical (hyper)graphs, including Milner's bigraphs, bigraphs with sharing, and a new version of bigraphs with recursion.

As future work, we plan to analyse other categories of graph-like objects using our criterion; an interesting case is that of *directed bigraphs* [13,3,4]. Moreover, it is worth to verify whether the $\mathcal{M}, \mathcal{N}$-adhesivity that we obtain from the results of this paper is suited for modelling specific rewriting systems, e.g. based on the DPO approach. As an example, $\mathbf{TG}_\Sigma$ is quasiadhesive but this does not suffice in most applications, because the rules are often spans of monomorphisms, and not of regular monos [7].

References

1. J. Adámek, H. Herrlich, and G. E. Strecker. Abstract and concrete categories: The joy of cats. *Reprints in Theory and Applications of Categories*, 17:1–507, 2006.
2. G. G. Azzi, A. Corradini, and L. Ribeiro. On the essence and initiality of conflicts in $\mathcal{M}$-adhesive transformation systems. *Journal of Logical and Algebraic Methods in Programming*, 109:100482, 2019.
3. G. Bacci, D. Grohmann, and M. Miculan. DBtk: A toolkit for directed bigraphs. In A. Kurz, M. Lenisa, and A. Tarlecki, editors, *CALCO 2009*, volume 5728 of *LNCS*, pages 413–422. Springer, 2009.
4. F. Burco, M. Miculan, and M. Peressotti. Towards a formal model for composable container systems. In C. Hung, T. Cerný, D. Shin, and A. Bechini, editors, *SAC 2020*, pages 173–175. ACM, 2020.
5. A. Carboni and P. Johnstone. Connected limits, familial representability and Artin glueing. *Mathematical Structures in Computer Science*, 5(4):441–459, 1995.
6. D. Castelnovo, F. Gadducci, and M. Miculan. A new criterion for $\mathcal{M}, \mathcal{N}$-adhesivity, with an application to hierarchical graphs. *CoRR*, abs/2201.00233, 2022.
7. A. Corradini and F. Gadducci. On term graphs as an adhesive category. In M. Fernández, editor, *TERMGRAPH 2004*, volume 127(5) of *ENTCS*, pages 43–56. Elsevier, 2005.
8. A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic approaches to graph transformation - Part I: Basic concepts and double pushout approach. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, pages 163–246. World Scientific, 1997.
9. H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, 2006.
10. H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce. Parallelism and concurrency in high-level replacement systems. *Mathematical Structures in Computer Science*, 1(3):361–404, 1991.
11. H. Ehrig, A. Habel, J. Padberg, and U. Prange. Adhesive high-level replacement categories and systems. In H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg, editors, *ICGT 2004*, LNCS, pages 144–160. Springer, 2004.
12. R. Garner and S. Lack. On the axioms for adhesive and quasiadhesive categories. *Theory and Applications of Categories*, 27(3):27–46, 2012.
13. D. Grohmann and M. Miculan. Directed bigraphs. In M. Fiore, editor, *MFPS 2007*, volume 173 of *ENTCS*, pages 121–137. Elsevier, 2007.
14. A. Habel and D. Plump. $\mathcal{M}, \mathcal{N}$-adhesive transformation systems. In H. Ehrig, G. Engels, H. Kreowski, and G. Rozenberg, editors, *ICGT 2012*, volume 7562 of *LNCS*, pages 218–233. Springer, 2012.
15. P. T. Johnstone, S. Lack, and P. Sobocinski. Quasitoposes, quasiadhesive categories and Artin glueing. In T. Mossakowski, U. Montanari, and M. Haveraaen, editors, *CALCO 2007*, volume 4624 of *LNCS*, pages 312–326. Springer, 2007.
16. S. Lack and P. Sobociński. Adhesive and quasiadhesive categories. *RAIRO-Theoretical Informatics and Applications*, 39(3):511–545, 2005.
17. S. Lack and P. Sobocinski. Toposes are adhesive. In A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, and G. Rozenberg, editors, *ICGT 2006*, volume 4178 of *LNCS*, pages 184–198. Springer, 2006.
18. T. Leinster. *Higher operads, higher categories*. Cambridge University Press, 2004.
19. S. Mac Lane. *Categories for the working mathematician*. Springer, 2013.

20. R. Milner. *The Space and Motion of Communicating Agents*. Cambridge University Press, 2009.
21. N. Mylonakis and F. Orejas. A framework of hierarchical graphs and its application to the semantics of SRML. Technical Report LSI-12-1-R, Facultad de Informática, Universitat Politècnica da Catalunya, 2012.
22. nLab. Creation of limits, 2016. Last accessed on January 26, 2022. http://nlab-pages.s3.us-east-2.amazonaws.com/nlab/show/created+limit.
23. J. Padberg. Hierarchical graph transformation revisited - Transformations of coalgebraic graphs. In J. de Lara and D. Plump, editors, *ICGT 2017*, volume 10373 of *LNCS*, pages 20–35. Springer, 2017.
24. W. Palacz. Algebraic hierarchical graph transformation. *Journal of Computer and System Sciences*, 68(3):497–520, 2004.
25. C. Peuser and A. Habel. Composition of $\mathcal{M}, \mathcal{N}$-adhesive categories with application to attribution of graphs. In D. Plump, editor, *GCM 2015*, volume 73 of *Electronic Communications of the EASST*. EASST, 2016.
26. D. Plump. Term graph rewriting. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformations, Vol. 2: Applications, Languages, and Tools*, pages 3–61. World Scientific, 1999.
27. M. Sevegnani and M. Calder. Bigraphs with sharing. *Theoretical Computer Science*, 577:43–73, 2015.
28. P. Sobociński and N. Behr. Rule algebras for adhesive categories. *Logical Methods in Computer Science*, 16, 2020.

Quantifier elimination for counting extensions of Presburger arithmetic

Dmitry Chistikov[1] , Christoph Haase[2] , and Alessio Mansutti[2](✉)

[1] Centre for Discrete Mathematics and its Applications (DIMAP) & Department of Computer Science, University of Warwick, Coventry, UK
d.chistikov@warwick.ac.uk
[2] Department of Computer Science, University of Oxford, Oxford, UK
{christoph.haase,alessio.mansutti}@cs.ox.ac.uk

Abstract. We give a new quantifier elimination procedure for Presburger arithmetic extended with a unary counting quantifier $\exists^{=x}y\,\Phi$ that binds to the variable x the number of different y satisfying Φ. While our procedure runs in non-elementary time in general, we show that it yields nearly optimal elementary complexity results for expressive counting extensions of Presburger arithmetic, such as the *threshold counting* quantifier $\exists^{\geq c}y\,\Phi$ that requires that the number of different y satisfying Φ be at least $c \in \mathbb{N}$, where c can succinctly be defined by a Presburger formula. Our results are cast in terms of what we call the *monadically-guarded fragment* of Presburger arithmetic with unary counting quantifiers, for which we develop a 2ExpSpace decision procedure.

1 Introduction

Counting the number of solutions to an equation, or the number of elements in a set subject to constraints, is a fundamental and often computationally challenging problem studied in logic, mathematics and computer science. In discrete geometry, counting the number of integral points in a polyhedron is a canonical #P-complete problem. Barvinok's celebrated algorithm solves this problem in polynomial time when the dimension is fixed [2]. In this paper, we investigate a generalization of this problem and study algorithmic aspects of counting the number of models of formulae of *Presburger arithmetic,* the first-order theory of the integers with addition and order, and more generally, extensions of this logic with *counting quantifiers.*

Counting quantifiers such as the *Härtig quantifier*, which allows to assert equal-cardinality constraints on the sets of satisfying assignments of two given first-order formulae, have long been studied in first-order logic [6]. In first-order theories of integer arithmetic, it is compelling to consider variants of counting quantifiers that bind the number of satisfying assignments of a formula to a first-order variable. Apelt [1] and Schweikardt [10] studied the decidability of Presburger arithmetic enriched with the unary counting quantifier $\exists^{=x}y$ with the following semantics: given an assignment of integers to the first-order variables $x, z_1, \ldots, z_n$, a formula $\exists^{=x}y\,\Phi(x, y, z_1, \ldots, z_n)$ evaluates to true whenever

the number of different y satisfying $\Phi(x, y, z_1, \ldots, z_n)$ is exactly x. In both [1] and [10], decidability is shown by developing a quantifier elimination procedure for this extension of Presburger arithmetic which eliminates a counting quantifier by translating it into an equivalent quantified formula of Presburger arithmetic, i.e., one that only uses standard first-order quantifiers. This immediately gives decidability of Presburger arithmetic extended with the unary counting quantifier $\exists^{=x}y$ since Presburger arithmetic is decidable in 2ExpSpace [9,3,12]. Unfortunately, the quantifier elimination procedures in [1,10] do not yield a similar elementary upper bound for the extended theory, as the elimination of a single quantifier $\exists^{=x}y$ results in an exponential blow-up of the formula size and introduces nested first-order quantifiers. It is a widely open problem whether there is a decision procedure for Presburger arithmetic extended with the counting quantifier $\exists^{=x}y$ with elementary running time, or whether this theory admits a significantly stronger lower bound than standard Presburger arithmetic.

To shed more light on the complexity of Presburger arithmetic extended with the aforementioned unary counting quantifier, Habermehl and Kuske gave a quantifier elimination procedure for Presburger arithmetic extended with a unary *modulo counting* quantifier $\exists^{(r,q)}y$, where r and q are positive natural numbers [4]. Here, $\exists^{(r,q)}y\,\Psi(y, z_1, \ldots, z_n)$ holds whenever the number of different y satisfying $\Psi(y, z_1, \ldots, z_n)$ is congruent to r modulo q. An analysis of the growth of the constants and coefficients occurring in their procedure then enables them to derive a 2ExpSpace upper bound for the logic, matching the complexity of Presburger arithmetic on deterministic machines. This noteworthy result shows that there is still room to extend Presburger arithmetic with non-trivial counting quantifiers without increasing the computational cost of deciding the logic.

Note that in order to keep the logic decidable, the counting quantifiers considered in the literature must be *unary*. Indeed, consider a binary counting quantifier $\exists^{=x}(y_1, y_2)$ counting the number of different y_1 and y_2 satisfying a formula. Then, $\Phi(x, z) = \exists^{=x}(y_1, y_2)(0 \leq y_1, y_2 < z)$ holds for $x = z^2$, which in turn allows defining multiplication, leading to undecidability of the resulting theory.

Our contribution. Following the lines of [4] while trying to avoid the limitations of the procedures in [1,10], our goal is to study decision procedures for Presburger arithmetic enriched with variants of counting quantifiers that do not increase the complexity of the Presburger arithmetic. To begin with, we develop a new quantifier elimination procedure for Presburger arithmetic with unary counting quantifiers $\exists^{=x}y$ that, in contrast to [1,10], does not require the introduction of first-order quantifiers. While the procedure still runs in non-elementary time, avoiding first-order quantification allows us not only to derive exponentially better bounds on the size of the formula obtained after eliminating a single $\exists^{=x}y$, but also to identify the sources of non-elementary growth. We exploit those observations to extend the range of counting quantifiers that can be added to Presburger arithmetic without increasing the complexity of the resulting logic.

The first type of counting quantifiers we consider is a *threshold counting quantifier* $\exists^{\geq c}y$ for some integer c. A formula $\exists^{\geq c}y\,\Psi(y, z_1, \ldots, z_n)$ evaluates to true whenever there are at least c different values of y satisfying $\Psi(y, z_1, \ldots, z_n)$. We

show that Presburger arithmetic enriched with threshold counting quantifiers can be decided in 2ExpSpace, even when the threshold c itself is succinctly given as the unique solution of a Presburger arithmetic formula. This is surprising since in Presburger arithmetic one can define numbers that are triply exponential in the size of the formula used to encode them [7, pp. 151–152]. Furthermore, we show that if we restrict c to be at most doubly exponential in the size of its encoding then Presburger arithmetic with threshold counting quantifiers is decidable in $\mathrm{STA}(*, 2^{2^{n^{\mathcal{O}(1)}}}, \mathcal{O}(n))$, matching the complexity of Presburger arithmetic [3]. Here, $\mathrm{STA}(s(n), t(n), a(n))$ is the class of all decision problems in which inputs of length n can be decided by an alternating Turing machine in space $s(n)$ and time $t(n)$ using $a(n)$ alternations, where "$*$" stands for unbounded availability of a certain resource.

Our results on the quantifier $\exists^{\geq c} x$ arise from studying a more general extension of Presburger arithmetic that relies on the notion of monadic decomposition put forward by Veanes et al. in [11] and studied by Hague et al. [5] in the context of integer linear arithmetic. Briefly, a formula $\Phi(x, y_1, \ldots, y_n)$ is said to be monadically decomposable on the variable x whenever it is equivalent to a formula of the form $\bigvee_{i \in I} \Delta_i(x) \wedge \Psi_i(y_1, \ldots, y_n)$, i.e., a formula where the satisfaction of constraints on x does not depend on the values of $y_1, \ldots, y_n$. Based on this definition, we extend Presburger arithmetic by allowing the general unary counting quantifiers $\exists^{=x} y$ to appear with guards of the form $\exists x(\Psi \wedge \exists^{=x} y \, \Phi)$, where Ψ is monadically decomposable on the variable x. The resulting logic is very powerful, as it not only generalizes the quantifiers $\exists^{\geq c} x$ but also the modulo counting quantifiers $\exists^{(r,q)} y$ from [4]. We establish two further results for this *monadically-guarded fragment* of Presburger arithmetic with counting quantifiers. First, we develop a 3ExpTime quantifier elimination procedure for the logic, matching the complexity of the best possible quantifier elimination procedures for Presburger arithmetic. Second, we exploit this procedure to obtain a quantifier relativization argument showing that the logic is decidable in 2ExpSpace.

2   Presburger arithmetic with counting quantifiers

General notation. The symbols $\mathbb{Z}$, $\mathbb{N}$ and $\mathbb{N}_+$ denote the set of integers, natural numbers including zero, and natural numbers without zero, respectively. We usually use $a, b, c, \ldots$ for integers, which we assume being encoded in binary. Given $n \in \mathbb{N}$, we write $[n] \stackrel{\text{def}}{=} \{0, \ldots, n-1\}$, and $\#A$ for the cardinality of a set A. If A is infinite, then $\#A = \infty$, and we postulate $n \leq \infty$ for all $n \in \mathbb{Z}$.

Structure. We consider the structure $\mathcal{Z} = \langle \mathbb{Z}, (c)_{c \in \mathbb{Z}}, +, <, (\equiv_q)_{q \in \mathbb{N}_+} \rangle$ of Presburger arithmetic, where $(c)_{c \in \mathbb{Z}}$ are constant symbols that shall be interpreted as their homographic integer numbers, the binary function symbol $+$ is interpreted as addition on $\mathbb{Z}$, the binary relation $<$ is interpreted as "less than", and $\equiv_q$ is interpreted as the modulo relation, i.e., $a \equiv_q b$ if and only if q divides $a - b$.

Basic syntax. Let $X = \{x, y, z, \dots\}$ be a countable set of first-order variables. *Linear terms*, usually denoted by t, t_1, t_2, etc., are expressions of the form $a_1 x_1 + \cdots + a_d x_d + c$ where $x_1, \dots, x_d \in X$, $a_1, \dots, a_d, c \in \mathbb{Z}$. The integer a_i is the *coefficient* of the variable x_i. Variables not appearing in the linear term are tacitly assumed to have a 0 *coefficient*. A term t is said to be *x-free* if the coefficient of the variable x in t is 0. The integer c is the *constant* of the linear term. Linear terms with constant 0 are said to be *homogeneous*.

Given a term t, the lexeme $t < 0$ is understood as a *linear inequality*, and $t \equiv_q 0$ is a *modulo constraint*. Syntactically, Presburger arithmetic (PA) is the closure of linear inequalities and modulo constraints under the Boolean connectives $\wedge$ and $\neg$ (i.e., conjunction and negation, respectively) and the *first-order quantifier* $\exists y$. Presburger arithmetic with counting quantifiers (PAC) extends PA with the *(unary) counting quantifier* $\exists^{=x} y$, where x and y are two syntactically distinct variables from X. Formulae of PAC are denoted by Φ, Ψ, Γ, etc.

We write $\mathrm{vars}(\Phi)$ and $\mathrm{fv}(\Phi)$ for the set of variables and free variables of Φ, respectively, with $\mathrm{fv}(\exists^{=x} y\, \Phi) \overset{\mathrm{def}}{=} \{x\} \cup (\mathrm{fv}(\Phi) \setminus \{y\})$. A *sentence* is a formula Φ with $\mathrm{fv}(\Phi) = \emptyset$. We sometimes write $\Phi(x_1, \dots, x_k)$ or $\Phi(\boldsymbol{x})$, with $\boldsymbol{x} = (x_1, \dots, x_k)$ a tuple of variables, for a formula Φ with $\mathrm{fv}(\Phi) = \{x_1, \dots, x_k\}$. We say that Φ is *z-free* if $z \in X$ does not occur in Φ. Given terms t and t', $\Phi[t'/t]$ stands for the formula obtained from Φ by syntactically replacing every occurrence of t by t'. Given $\Phi(x_1, \dots, x_k)$ and terms $t_1, \dots, t_k$, $\Phi(t_1, \dots, t_k)$ stands for $\Phi[t_1/x_1] \dots [t_k/x_k]$.

Semantics. An *assignment* is a function $\nu \colon X \to \mathbb{Z}$ assigning an integer value to every variable. As usual, we extend ν in the standard way to a function that maps every term to an element of $\mathbb{Z}$. For instance, $\nu(x + 3x + 2) = \nu(x) + 3\nu(y) + 2$. Given a variable x and an integer n, we write $\nu[n/x]$ for the assignment obtained form ν by updating the value of x to n, i.e. $\nu[n/x](x) = n$, and for all variables y distinct from x, $\nu[n/x](y) = \nu(y)$. Given a formula Φ of PAC and an assignment ν, the satisfaction relation $\nu \models \Phi$ is defined as usual for linear inequalities, modulo constraints, Boolean connectives and the existential quantifier ranging over $\mathbb{Z}$. For the counting quantifier, we define

$$\nu \models \exists^{=x} y\, \Phi \text{ if and only if } \#\{n \in \mathbb{Z} \mid \nu[n/y] \models \Phi\} = \nu(x).$$

Informally, $\exists^{=x} y\, \Phi$ is satisfied by ν if there are exactly $\nu(x)$ distinct values for the variable y that make Φ true. A formula Φ of PAC is *satisfiable* (resp. *valid*) if $\nu \models \Phi$ holds for an assignment (resp. *every assignment*) ν. A formula Φ *entails* a formula Ψ, written $\Phi \models \Psi$, whenever every assignment satisfying Φ also satisfies Ψ. We write $\Phi \Leftrightarrow \Psi$ to denote that Φ and Ψ are *equivalent*, i.e. $\Phi \models \Psi$ and $\Psi \models \Phi$.

Syntactic abbreviations. We define $\bot \overset{\mathrm{def}}{=} 0 < 0$ and $\top \overset{\mathrm{def}}{=} \neg \bot$. The Boolean connectives $\vee$, $\rightarrow$ and $\leftrightarrow$ and the universal first-order quantifier $\forall$ are derived as usual, and so are the (in)equalities $<$, $\leq$, $=$, $\geq$, and $>$, between terms. For instance, $t_1 < t_2$ corresponds to $t_1 - t_2 < 0$, where we tacitly manipulate $t_1 - t_2$ with standard operations of linear arithmetic to obtain an equivalent term. Similarly, $t_1 \equiv_q t_2$ is short for $t_1 - t_2 \equiv_q 0$, whereas $|t_1| + t_2 < 0$ is short for

$(t_1 < 0 \rightarrow t_2 - t_1 < 0) \wedge (t_1 \geq 0 \rightarrow t_1 + t_2 < 0)$. For a variable $x \in X$ and $r \in [q]$, we call $x \equiv_q r$ a *simple* modulo constraint. All modulo constraints introduced by our quantifier elimination procedure given in Section 3 are simple.

The counting quantifier $\exists^{\geq x} y$. Historically [1,10], the quantifier $\exists^{=x} y$ has been the unary counting quantifier of choice when it comes to PAC. However, a priori one could define PAC as the extension of PA featuring counting quantifiers $\exists^{\geq x} y$, where $\nu \models \exists^{\geq x} y \, \Phi$ holds for an assignment ν whenever there are at least $\nu(x)$ values $n \in \mathbb{Z}$ for y such that $\nu[n/y] \models \Phi$. Notice that the counting quantifier $\exists^{=y}$ can be expressed using $\exists^{\geq y}$, and vice versa:

- $\exists^{=x} y \, \Phi \iff \exists^{\geq x} y \, \Phi \wedge \exists x' : x' = x + 1 \wedge \neg \exists^{\geq x'} y \, \Phi$; and
- $\exists^{\geq x} y \, \Phi \iff (\forall z \, \exists y : |z| \leq |y| \wedge \Phi) \vee \exists x' : x' \geq x \wedge \exists^{=x'} y \, \Phi$.

Two comments are in order: first, translating a PAC formula by swapping the type of counting quantifiers using the equivalences above has the unpleasant effect of increasing the size of the formula, exponentially if the nesting depth of quantifiers is unbounded. Second, the subformula $\forall z \, \exists y : |z| \leq |y| \wedge \Phi$ used in the last equivalence states that there are infinitely many values for y that make the formula Φ true. This formula highlights the main difference between $\exists^{=x} y$ and $\exists^{\geq x} y$ quantifiers: the latter is true in the presence of infinitely many values for y, whereas the former is false. Throughout the paper, we focus on the quantifier $\exists^{=x} y$, as done in [1,10], but use this observation to argue that our results can be readily adapted to the counting quantifier $\exists^{\geq x} y$. Full details of this adaptation are given in the full version of the paper.

Parameters of formulae. To analyze quantifier-elimination procedures, following [8,12], we introduce a number of parameters for formulae of PAC:

- $|\Phi|$ denotes the *length* of the formula Φ, i.e., the number of symbols to write down φ, with numbers encoded in binary. We always assume $|\Phi| \geq 2$;
- $\mathrm{qr}(\Phi)$ (resp. $\mathrm{nr}(\Phi)$) denotes the *quantifier* (resp. *negation*) *rank* of the formula Φ, i.e., the depth of nesting of the quantifiers (resp. negations) of Φ;
- $\mathrm{fd}(\Phi)$ denotes the overall *depth* of Φ, i.e., the depth of nesting of all constructors (i.e. $\wedge$, $\neg$, $\exists x$ and $\exists^{=x} y$) in the formula Φ;
- $\mathrm{lin}(\Phi)$ is the set containing the term 0 plus all the terms t that appear in linear inequalities $t < 0$ of Φ (recall that $t_1 < t_2$ is short for $t_1 - t_2 < 0$);
- $\mathrm{hom}(\Phi)$ is the set of homogeneous linear terms obtained from all terms in $\mathrm{lin}(\Phi)$ by setting their constants to 0;
- $\mathrm{const}(\Phi)$ is the set of all constants appearing in linear terms of $\mathrm{lin}(\Phi)$; and
- $\mathrm{mod}(\Phi)$ is the set of all moduli $q \in \mathbb{N}$ appearing in modulo constraints $t_1 \equiv_q t_2$ of Φ. We postulate $1 \in \mathrm{mod}(\Phi)$, even if Φ has no modulo constraints.

Given a vector $\boldsymbol{v} = (v_1, \ldots, v_d) \in \mathbb{Z}^d$, we write $\|\boldsymbol{v}\| = \max\{|v_i| : 1 \leq i \leq d\}$ for the *infinity norm* of $\boldsymbol{v}$. Similarly, for a linear term t, we write $\|t\|$ for the maximum absolute value of a coefficient or constant appearing in t. Given a finite set of vectors or a finite set of terms A, we define $\|A\| = \max\{\|a\| : a \in A\}$. Given a matrix $A \in \mathbb{Z}^{n \times d}$, its infinity norm is the maximal infinity norm of its column vectors. Notice that $\|\mathrm{lin}(\Phi)\| = \|\mathrm{hom}(\Phi) \cup \mathrm{const}(\Phi)\|$. For a formula Φ, we define $\|\Phi\| \overset{\text{def}}{=} \|\mathrm{lin}(\Phi) \cup \mathrm{mod}(\Phi)\|$.

Complexity remarks. The proposition below characterizes the complexity of PA.

Proposition 1 ([3]). *Presburger arithmetic is* $\mathrm{STA}(*, 2^{2^{n^{\mathcal{O}(1)}}}, \mathcal{O}(n))$-*complete.*

To be more precise, the number of alternations required to decide the validity or satisfiability of a formula Φ from Presburger arithmetic is linear in $\mathrm{nr}(\Phi)$. Notice that $2\mathrm{NExpTime} \subseteq \mathrm{STA}(*, 2^{2^{n^{\mathcal{O}(1)}}}, \mathcal{O}(n)) \subseteq 2\mathrm{ExpSpace}$.

3 A quantifier elimination procedure for PAC

In this section, we develop a new quantifier elimination procedure (QE procedure) for the counting quantifier $\exists^{=x}y$:

Proposition 2. *Let Φ be quantifier-free. Then $\exists^{=x}y\,\Phi$ is equivalent to a Boolean combination of linear inequalities and simple modulo constraints.*

We quantify the growth of parameters in the formula in Section 4. Upper bounds on this growth are at the core of our results. Without any bounds (as stated), Proposition 2 is known and can be obtained by chaining the quantifier elimination procedure developed by Schweikardt [10] together with the standard quantifier elimination procedure for Presburger arithmetic. An advantage of our QE procedure for the quantifier $\exists^{=x}y$ is that it avoids the introduction of additional $\exists$- and $\forall$-quantifiers when eliminating a counting quantifier on which Schweikardt's procedure relies. More precisely, given a formula $\exists^{=x}y\,\Phi$ where Φ is quantifier-free (q.f. in short), the QE procedure in [10] requires a full transformation of Φ into disjunctive normal form, and eliminates the quantifier $\exists^{=x}y$ by introducing first-order quantifiers, producing an equivalent formula Ψ of Presburger arithmetic. This strategy comes at a cost: the size of the q.f. formula obtained after removing the quantifiers from Ψ is doubly exponential in the size of $\exists^{=x}y\,\Phi$. By avoiding the introduction of first-order quantifiers, our QE procedure already exponentially improves upon Schweikardt's procedure.

Our QE procedure performs a series of formula manipulations, divided into five steps. At the end of the i-th step, the procedure produces a formula Φ_i equivalent to the original formula $\exists^{=x}y\,\Phi$. Ultimately, Φ_5 is a Boolean combination of inequalities and simple modulo constraints allowing us to establish Proposition 2. In this section, we present the procedure and briefly discuss its correctness, leaving the computational analysis of parameters $\mathrm{lin}(\Phi_5)$, $\mathrm{hom}(\Phi_5)$, $\mathrm{const}(\Phi_5)$ and $\mathrm{mod}(\Phi_5)$ to subsequent sections.

Step I: Normalise the coefficients of y. Given the input formula $\Phi_0 = \exists^{=x}y\,\Phi$, with Φ q.f., the first step of the procedure is a standard step for QE procedures for Presburger arithmetic. It produces an equivalent formula Φ_1 in which all nonzero coefficients of y appearing in a linear term are normalized to 1 or -1. For simplicity, we first translate every modulo constraint in Φ into simple modulo constraints, by relying on the lemma below.

Lemma 1. *Every constraint $t \equiv_q 0$ is equivalent to a Boolean combination Ψ of simple modulo constraints such that $\mathrm{vars}(\Psi) \subseteq \mathrm{vars}(t \equiv_q 0)$ and $\mathrm{mod}(\Psi) = \{q\}$.*

The first step of our QE procedure is as follows:

1 Translate every modulo constraint in Φ into simple modulo constraints (Lemma 1).
2 Let k be the lcm of the absolute values of all coefficients of y appearing in $\mathrm{hom}(\Phi)$.
3 Let Φ' be the formula obtained from Φ by applying the following three rewrite rules to each linear inequality and simple modulo constraint in which y appears:
   - $ay + t < 0 \;\longrightarrow\; ky + (k/a) \cdot t < 0, \qquad$ if $a > 0$,
   - $ay + t < 0 \;\longrightarrow\; -ky - (k/a) \cdot t < 0, \qquad$ if $a < 0$, and
   - $y \equiv_q r \;\longrightarrow\; ky \equiv_{kq} kr$,

   where t is a term, $q \geq 1$ and $r \in [q]$:
4 Define $\Phi_1 \stackrel{\text{def}}{=} \exists^{=x} y\, (y \equiv_k 0 \wedge \Phi'[y/ky])$.

Claim 1. $\Phi_0 \Leftrightarrow \Phi_1$, and in Φ_1, all non-zero coefficients of y are either 1 or -1.

Step II: Subdivide the formula according to term orderings and residue classes.
We define an *ordering for a set of linear terms T* to be a formula of the form

$$(t_1 \lhd_1 t_2) \wedge (t_2 \lhd_2 t_3) \wedge \cdots \wedge (t_{n-1} \lhd_{n-1} t_n), \tag{1}$$

where $\{t_1, \ldots, t_n\} = T$ and $\{\lhd_1, \ldots, \lhd_{n-1}\} \subseteq \{<, =\}$.

Lemma 2. *There is an algorithm that, given a set T of n linear terms over d variables, computes in time $n^{\mathcal{O}(d)} \log \|T\|^{\mathcal{O}(1)}$ a set $\{O_1, \ldots, O_o\}$ of orderings for T s.t. (1) $o = \mathcal{O}(n^{2d})$, (2) $\top \Leftrightarrow \bigvee_{i=1}^{o} O_i$, (3) $\bot \Leftrightarrow O_i \wedge O_j$ whenever $i \neq j$.*

Lemma 2 is proven analogously to [13, Proposition 5.1].
The second step of our QE procedure is as follows:

5 Let T be the set of all y-free terms t such that t, $y - t$ or $-y + t$ belongs to $\mathrm{lin}(\Phi_1)$.
6 Using Lemma 2, build a set $\{O_1, \ldots, O_o\}$ of orderings for the terms T.
7 Let $Z \stackrel{\text{def}}{=} \mathrm{vars}(\Phi)$ and $m \stackrel{\text{def}}{=} \mathrm{lcm}(\mathrm{mod}(\Phi_1))$.
8 For every $i \in [1, o]$ and every $r \colon Z \to [m]$, let $\Gamma_{i,r} \stackrel{\text{def}}{=} O_i \wedge (\bigwedge_{z \in Z} z \equiv_m r(z))$.
9 Define $\Phi_2 \stackrel{\text{def}}{=} \bigvee_{i=1}^{o} \bigvee_{r \colon Z \to [m]} (\Gamma_{i,r} \wedge \Phi_1)$.

Claim 2. $\Phi_1 \Leftrightarrow \Phi_2$.

In Steps III to V of the procedure, we focus on each disjunct of Φ_2 separately, iterating over all $i \in [1, o]$, hence over all orderings, and all $r \colon Z \to [m]$, i.e., functions assigning residue classes modulo m to the variables in Z.

Step III: Split the range of y into segments. Recall that $\Phi_1 = \exists^{=x} y\, \Psi$, where Ψ is some Boolean combination of inequalities and modulo constraints with variables from $\mathrm{vars}(\Phi)$ in which the non-zero coefficients of y are either 1 or -1. Let $T|_{O_i} \stackrel{\text{def}}{=} (t'_1, \cdots, t'_\ell)$ be the tuple of all the terms in T that the formula O_i asserts pairwise non-equal, taken in the ascending order. In other words, we obtain $t'_1, \ldots, t'_\ell$ by removing from the sequence $t_1, \ldots, t_n$ in Equation (1) all terms t_{j+1} for which $\lhd_j$ is $=$. Let $seg(y, O_i)$ be the set of formulae

$$\big\{\, y < t_1', \ \ y = t_1', \ \ (t_{i-1}' < y \wedge y < t_i'), \ \ y = t_i', \ \ t_\ell' < y \ : \ i \in [2, \ell] \big\}.$$

We have $\#seg(y, O_i) = 2\ell + 1$. Given $\kappa \in seg(y, O_i)$, the formula $O_i \wedge \kappa$ imparts a linear ordering on the terms $T \cup \{y\}$. This enables us to "almost evaluate" Ψ:

Lemma 3. *For every $\kappa \in seg(y, O_i)$, there is a Boolean combination $\Psi_\kappa^{i,r}$ of simple modulo constraints such that* $\mathrm{vars}(\Psi_\kappa^{i,r}) = \{y\}$, $\mathrm{mod}(\Psi_\kappa^{i,r}) \subseteq \mathrm{mod}(\Psi)$ *and*

$$\Gamma_{i,r} \wedge \kappa \wedge \Psi \ \Leftrightarrow \ \Gamma_{i,r} \wedge \kappa \wedge \Psi_\kappa^{i,r}.$$

Our QE procedure manipulates Φ_2 as follows:

10   For every $i \in [1, o]$ and every $r \colon Z \to [m]$:
11   Let $seg(y, O_i) = \{\kappa_0, \ldots, \kappa_{2\ell}\}$.
12   For every $j \in [0, 2\ell]$, consider the formula $\Psi_{\kappa_j}^{i,r}$ from Lemma 3.
13   Let $\Phi_3^{i,r} = \exists x_0 \ldots \exists x_{2\ell}\big(x = \sum_{j=0}^{2\ell} x_j \ \wedge \ \bigwedge_{j=0}^{2\ell} \exists^{=x_j} y(\kappa_j \wedge \Psi_{\kappa_j}^{i,r})\big)$.
14   Define $\Phi_3 \overset{\text{def}}{=} \bigvee_{i=1}^{o} \bigvee_{r \colon Z \to [m]} (\Gamma_{i,r} \wedge \Phi_3^{i,r})$.

Claim 3. $\Phi_2 \Leftrightarrow \Phi_3$.

Step IV: Compute the number of solutions for each segment. We next aim at eliminating the counting quantifiers introduced in Step III in the sub-formulae $\exists^{=x_j} y(\kappa_j \wedge \Psi_{\kappa_j}^{i,r})$. We go over each $\kappa \in seg(y, O_i)$, and consider three cases depending on whether it specifies (syntactically) an infinite interval, a finite segment, or a single value for y.

Notice that r is in fact an assignment to variables, so $r(t) \in \mathbb{Z}$ is well-defined for every term t with free variables Z. For all $i \in [1, o]$ and $r \colon Z \to [m]$, given $T|_{O_i} = (t_1', \ldots, t_\ell')$ the procedure computes the following numbers $c_1, \ldots, c_\ell$, $p_2, \ldots, p_\ell$ and $r_2, \ldots, r_\ell$.

15   For every $j \in [1, \ell]$:
16   If $\Psi_\kappa^{i,r}[r(t_j')/y]$ is true, where $\kappa = (y = t_j')$, then let $c_j \overset{\text{def}}{=} 1$, else let $c_j \overset{\text{def}}{=} 0$.
17   For every $j \in [2, \ell]$:
18   Let $p_j \in [0, m]$ be the number of $y \in [m]$ satisfying $\Psi_\kappa^{i,r}(y)$.
19   Let $\underline{u}_j = (r(t_{j-1}') \bmod m)$.
20   Let $\overline{u}_j$ be the smallest integer congruent to $r(t_j')$ modulo m and greater than $\underline{u}_j$.
21   Let $r_j' \in [0, m]$ be the number of $y \in [\underline{u}_j + 1, \overline{u}_j - 1]$ satisfying $\Psi_\kappa^{i,r}(y)$.
22   Let $r_j \in [-m^2, m^2]$ be such that $r_j = -p_j \cdot (\overline{u}_j - \underline{u}_j) + m \cdot r_j'$.

Lemma 4. *Given a formula $\Psi_\kappa^{i,r}$ and $m, \underline{u}_j, \overline{u}_j$, the numbers p_j and r_j' can be computed in $\#\mathrm{P}$, or by a deterministic algorithm with running time $\mathcal{O}(m \cdot |\Psi_\kappa^{i,r}|)$.*

The numbers c_j, p_j, r_j determine, for each $\kappa \in seg(y, O_i)$, how many assignments to the variable y satisfy the formula $\Psi_\kappa^{i,r}$ in the conjunction $\Gamma_{i,r} \wedge \kappa \wedge \Psi_\kappa^{i,r}$. Intuitively, this is c_j for κ of the form $y = t_j'$, and $(p_j(t_j' - t_{j-1}') + r_j)/m$ for κ of the form $t_{j-1}' < y \wedge y < t_j'$. We say "intuitively" here, because in the latter case the expression above depends on other variables so is not, strictly speaking, a number. The following claims formalize this intuition:

Claim 4. Let $\kappa \in \{y < t_1', \; t_\ell' < y\}$. If $\Psi_\kappa^{i,r}(y)$ is satisfiable, then $\Phi_3^{i,r} \Leftrightarrow \bot$.

Claim 5. Let $j \in [1, \ell]$, $\kappa = (y = t_j')$, $z \in X$. Then, $\exists^{=z} y \, (\kappa \wedge \Psi_\kappa^{i,r}) \; \Leftrightarrow \; z = c_j$.

Claim 6. Let $\kappa = (t_{j-1}' < y \wedge y < t_j')$ for some $j \in [2, \ell]$ and let z be a fresh variable. Then, $\Gamma_{i,r} \wedge \exists^{=z} y \, (\kappa \wedge \Psi_\kappa^{i,r}) \; \Leftrightarrow \; \Gamma_{i,r} \wedge mz = p_j(t_j' - t_{j-1}') + r_j$.

The procedure manipulates the formula Φ_3 as follows:

23 For every $i \in [1, o]$ and every $r \colon Z \to [m]$:
24   If $\Psi_\kappa^{i,r}(y)$ is satisfiable for some $\kappa \in \{y < t_1', \; t_\ell' < y\}$, then let $\Phi_4^{i,r} \overset{\text{def}}{=} \bot$,
25   else $\Phi_4^{i,r} \overset{\text{def}}{=} \exists x_2 \ldots \exists x_\ell \, \big(x = \sum_{j=2}^{\ell} x_j + \sum_{j=1}^{\ell} c_j \wedge \bigwedge_{j=2}^{\ell} mx_j = p_j(t_j' - t_{j-1}') + r_j\big)$.
26 Define $\Phi_4 \overset{\text{def}}{=} \bigvee_{i=1}^{o} \bigvee_{r \colon Z \to [m]} (\Gamma_{i,r} \wedge \Phi_4^{i,r})$.

Claim 7. $\Phi_3 \Leftrightarrow \Phi_4$.

Step V: Sum up the solutions. It remains to get rid of the variables x_i introduced earlier. For each disjunct $\Gamma_{i,r} \wedge \Phi_4^{i,r}$ of Φ_4, we use the notation from Step IV.

27 For every $i \in [1, o]$ and every $r \colon Z \to [m]$:
28   If $\Phi_4^{i,r} = \bot$, then let $\Phi_5^{i,r} \overset{\text{def}}{=} \bot$,
29   else let $\Phi_5^{i,r} \overset{\text{def}}{=} mx = \sum_{j=2}^{\ell}(p_j(t_j' - t_{j-1}') + r_j) + m \cdot \sum_{j=1}^{\ell} c_j$.
30 Let $\Phi_5 \overset{\text{def}}{=} \bigvee_{i=1}^{o} \bigvee_{r \colon Z \to [m]} (\Gamma_{i,r} \wedge \Phi_5^{i,r})$.

The procedure outputs Φ_5. The following claim implies Proposition 2.

Claim 8. $\Phi_4 \Leftrightarrow \Phi_5$. The formula Φ_5 is quantifier-free.

4   Discussion, summary of results and roadmap

The QE procedure for a single counting quantifier $\exists^{=x} y$ from Section 3 forms the basis of our results. In this section we discuss its use and lay out its applications.

Analysis of the procedure. The next lemma establishes the growth of the formulae and their parameters in our quantifier elimination procedure.

Lemma 5. *Let Φ_5 be obtained from applying the QE procedure of Section 3 to a formula $\exists^{=y} x \, \Phi$, where Φ is quantifier-free and $\#\mathrm{vars}(\Phi) = d$. Then:*

$$\mathrm{mod}(\Phi_5) = \{m\} \quad \text{with } m = k \cdot \mathrm{lcm}(\mathrm{mod}(\Phi)) \text{ and } k \leq \|\mathrm{hom}(\Phi)\|^{\#\mathrm{hom}(\Phi)},$$

$$\#\mathrm{lin}(\Phi_5) \leq N^{O(d)}, \quad \|\mathrm{lin}(\Phi_5)\| \leq \mathcal{O}(N) \cdot \|\mathrm{lin}(\Phi)\|,$$

$$\#\mathrm{hom}(\Phi_5) \leq N^{O(d)}, \quad \|\mathrm{hom}(\Phi_5)\| \leq \mathcal{O}(N) \cdot \|\mathrm{hom}(\Phi)\|, \quad \text{with } N = m^2 \cdot \#\mathrm{lin}(\Phi).$$

Remark 1. With minor changes to our procedure, one can obtain a QE procedure for the quantifier $\exists^{\geq x} y$. In particular, since $\exists^{\geq x} y \, \Phi$ is true if there are infinitely many values for y that satisfy Φ, Claim 4 needs to be updated so that $\Phi_3^{i,r} \Leftrightarrow \top$ is deduced, instead of $\Phi_3^{i,r} \Leftrightarrow \bot$. Other minor adaptations are required, e.g. equalities "$x = \ldots$" and counting quantifiers $\exists^{=x_j} y$ appearing in Line 13 must be updated to "$x \leq \ldots$" and $\exists^{\geq x_j} y$. The resulting QE procedure for $\exists^{\geq x} y$ still adheres to the bounds in Lemma 5.

A consequence of Lemma 5 is that our QE procedure gives an algorithm for deciding a formula Φ from PAC featuring multiple counting quantifiers $\exists^{=x}y$ in time $2^{\cdot^{\cdot^{\cdot^{2}}}}$, where the height of the tower is linear in the quantifier rank of Φ. Indeed, in view of the upper bounds and equations given by Lemma 5 for $\#\mathrm{hom}(\Phi_5)$, N, m, and k, we observe that the upper bound for $\#\mathrm{hom}(\Phi_5)$ is exponential in $\#\mathrm{hom}(\Phi)$. This means that more fine-grained bounds are necessary for decision procedures with elementary complexity, i.e., with a running time bounded from above by a k-fold exponential in the size of the input formula.

Elementary decision procedures. In view of this growth of the parameters, it is natural to ask ourselves whether our QE procedure is perhaps naïvely disregarding important properties of the underlying arithmetic theory that could lead to better bounds. A good test in this direction is to check whether improved bounds can be achieved when the procedure runs on restricted forms of counting quantifiers. In the remainder of the paper we show that this is the case, and explain how the growth of parameters can be countered for restricted quantifiers, obtaining 3ExpTime quantifier elimination procedures as well as 2ExpSpace decision procedures for extensions of PA with a variety of counting quantifiers.

As an example, let us consider Presburger arithmetic enriched with threshold quantifiers $\exists^{\geq c}y\,\Phi$, where $c \in \mathbb{N}$ is written in binary. These are satisfied whenever there are at least c distinct values for the variable y that make the formula Φ true. Notice that the threshold counting quantifiers $\exists^{\geq c}y$ are a syntactic generalization of the first-order quantifiers, as $\exists^{\geq 1}y\,\Phi \Leftrightarrow \exists y\,\Phi$. Interestingly enough, one can translate threshold quantifiers into standard Presburger arithmetic with just a polynomial increase in the size of the formula. For simplicity, assume that the threshold c is a power of 2. Then, the quantifier $\exists^{\geq c}y$ can be internalized in PA by relying on the equivalence

$$\exists^{\geq 2g}y\,\Phi(y,\boldsymbol{z}) \Leftrightarrow \exists u\,\forall v\,\exists^{\geq g}y : (v=0 \leftrightarrow y<u) \wedge \Phi(y,\boldsymbol{z})$$

as well as $\exists^{\geq 1}y\,\Phi \Leftrightarrow \exists y\,\Phi$. However, in terms of decision procedures, this is an inadequate solution, as it comes at the cost of introducing $2\log_2 c$ many quantifier alternations. Building upon the QE procedure from Section 3, we show how to directly eliminate threshold quantifiers. This proves that the increase in alternation depth that depends on the threshold c is unnecessary.

Theorem 1. *The validity of a formula Φ from Presburger arithmetic with threshold counting quantifiers can be decided in* $\mathrm{STA}(*, 2^{2^{|\Phi|^{\mathcal{O}(1)}}}, \mathcal{O}(\mathrm{fd}(\Phi)))$.

This result matches the complexity of deciding standard PA in the case of unbounded alternation depth. Thus, PA can be enriched with threshold quantifiers with almost no computational overhead. Note that a slight increase in number of alternations is still required, and goes from $\mathcal{O}(\mathrm{nr}(\Phi))$ for PA to $\mathcal{O}(\mathrm{fd}(\Phi))$ for PA with threshold counting quantifiers.

We further strengthen Theorem 1, extending it to the case where the threshold c is encoded even more succinctly, as the unique solution of a PA formula

$\Phi(x)$ as long as this solution is bounded doubly-exponentially in $|\Phi|$. An example of such a formula is $\Phi(x) = \exists z : z = 1 \land \Psi_n(x, z)$, where

$$\Psi_0(x, z) \stackrel{\text{def}}{=} x = 2z,$$

$$\Psi_{n+1}(x, z) \stackrel{\text{def}}{=} \exists y \forall a \forall b : (a = x \land b = y) \lor (a = y \land b = z) \to \Psi_n(a, b),$$

and the only solution is given by $x = 2^{2^n}$ [7, Lecture 23], whilst $|\Phi| = \mathcal{O}(n)$. The crux of our results lies in the identification of a fragment of PAC that we call *monadically-guarded*, for which the following theorem can be established.

Theorem 2. *Monadically-guarded PAC is decidable in* 2ExpSpace.

In the next section, we introduce the monadically-guarded fragment of PAC and discuss extensions of PA that can be captured by this fragment. In Section 6, by adding post-processing to the procedure from Section 3, we show how to deal with any monadically-guarded counting quantifiers in 3ExpTime. In Section 7 we establish Theorem 2 by designing a quantifier relativization argument, continuing the direction of research due to [12]. In Section 8 we prove Theorem 1.

5   The monadically-guarded fragment of PAC

Fix a logic $\mathcal{L}$. A formula $\Phi(x, \boldsymbol{z})$ from $\mathcal{L}$, where $\boldsymbol{z}$ is a tuple of variables not including x, is said to be *monadically decomposable* on the variable x whenever

$$\Phi \Leftrightarrow \Psi, \text{ for some } \Psi \stackrel{\text{def}}{=} \bigvee_{i \in I}(\Delta_i(x) \land \Gamma_i(\boldsymbol{z})),$$

where Δ_i and Γ_i are formulae from $\mathcal{L}$. In this case, Ψ is said to be a *monadic decomposition of* Φ on the variable x.

The notion of monadic decomposition has been put forward by Veanes et al. in [11], as a general simplification technique that improves the performance of solvers. Here, our interest lies in studying whether the notion of monadic decomposability can bring complexity advantages for Presburger arithmetic with counting quantifiers. With this in mind, we consider formulae of PAC that we call *monadically-guarded*: those in which the quantifiers $\exists^{=x} y$ only appear in subformulae of the form $\exists x (\Psi \land \exists^{=x} y \Phi)$, where Φ and Ψ are themselves from the monadically-guarded fragment of PAC, x does not occur in Φ, and Ψ is monadically decomposable on the variable x. The *monadically-guarded fragment* of PAC is understood as the set of all formulae from PAC that are monadically-guarded. This fragment captures several interesting extensions of PA:

– It can express that the number of different y satisfying $\Phi(y, \boldsymbol{z})$ lies in an arithmetic progression $b, b+p, b+2 \cdot p, b+i \cdot p, \ldots$, with $b, p \in \mathbb{N}$. That is,

$$\exists x(x \geq b \land x \equiv_p b \land \exists^{=x} y \, \Phi(y, \boldsymbol{z})).$$

This type of monadically-guarded formulae extends the *modulo counting* quantifiers studied by Habermehl and Kuske [4]. Modulo counting quantifiers are written as $\exists^{(r,q)} y \, \Phi$ and hold whenever the number of different y satisfying Φ is congruent to r modulo q. Hence, $\exists^{(r,q)} y \, \Phi \Leftrightarrow \exists x (x \equiv_p r \land \exists^{=x} y \Phi)$.

Moreover, in the monadically-guarded fragment, we can replace the integer r with an arbitrary linear term t with variables from $\mathbf{z}$, since the modulo constraint $x \equiv_p t$ can be monadically decomposed into $\bigvee_{r \in [p]} (x \equiv_p r \wedge t \equiv_p r)$.

– As we recalled in the previous section with the formula $\Psi_n(x, z)$, it is known that PA allows one to succinctly encode numbers that are doubly or triply exponentially large with respect to the size of the formula. For instance, one can define a formula $L_n(x)$, again of size polynomial in n, that is true whenever x is the product of all primes in the interval $[2, 2^{2^n}]$ (see [7, Lecture 24]). In this case, $x \geq 2^{c2^{2^n}}$ for some fixed $c > 0$. The monadically-guarded fragment of PAC allows one to use these succinct representations as guards of counting quantifiers. For instance, $\exists x (L_n(x) \wedge \exists^{=x} y\, \Psi(y, \mathbf{z}))$ is true whenever the number of y satisfying $\Psi(y, \mathbf{z})$ is the product of all primes in $[2, 2^{2^n}]$.

Hague et al. [5] proved that constructing the monadic decomposition of a quantifier-free formula can be done in exponential time. More precisely, given a q.f. formula $\Phi(x, \mathbf{y})$ from PA that is monadically decomposable on x, in [5] it is shown that there is a natural number B of magnitude exponential in $|\Phi|$ that makes the following formula $\Psi_B(x, \mathbf{y})$ a monadic decomposition of Φ on x:

$$\Psi_B \stackrel{\text{def}}{=} \bigvee_{c=0}^{m-1} \left((x \geq B \wedge x \equiv_m c \wedge \Phi(B + c, \mathbf{y})) \vee (x \leq -B \wedge x \equiv_m c \wedge \Phi(-B - c, \mathbf{y})) \right)$$
$$\vee \bigvee_{c=-B+1}^{B-1} (x = c \wedge \Phi(c, \mathbf{y})),$$

where $m = \text{lcm}(\text{mod}(\Phi))$. We study the arguments presented in [5] and refine the bound B, tracking dependencies on several formula parameters separately. We find that B is polynomial in $\|\Phi\|$; it is only exponential in $\#\text{mod}(\Phi)$ and in the number of variables of the tuple $\mathbf{y}$.

Proposition 3. *Let $\Phi(x, \mathbf{y})$ be a q.f. formula from PA, where $\mathbf{y} = (y_1, \ldots, y_d)$. Let $m = \text{lcm}(\text{mod}(\Phi))$ and $B = 2^{48d^2}(m \cdot \|\text{lin}(\Phi)\|)^{6d} + 1$. If Φ is monadically decomposable on x, then the formula Ψ_B is such a decomposition.*

Together with our QE procedure, Proposition 3 shows that it is decidable to check whether a formula of PAC is monadically decomposable (on a certain variable). Due to Theorem 2, this problem is in 2ExpSpace for formulae of the monadically-guarded fragment of PAC. Besides, notice that all formulae having one free variable are monadic decompositions of themselves.

Our QE procedure for the monadically-guarded fragment of PAC, outlined below, makes use of the sharper bound obtained in Proposition 3.

6 Eliminating monadically-guarded counting quantifiers

Consider a formula $\Phi_0 = \exists x (\Psi \wedge \exists^{=x} y\, \Phi)$, where Φ and Ψ are quantifier-free formulae, x does not occur in Φ, and Ψ is monadically decomposable on x. By relying on the QE procedure introduced in Section 3, we show how to obtain a quantifier-free formula equivalent to Φ_0. W.l.o.g., we assume that all free variables distinct from x and y and occurring in Φ and Ψ come from the tuple of variables $\mathbf{z}$.

Below, let $\Psi' = \bigvee_{k \in K} \Delta_k(x) \wedge \Psi_k(\boldsymbol{z})$ be the monadic decomposition of Ψ on the variable x computed according to Proposition 3. Recall that this means that each Δ_k is a formula having one among the following three forms:

$$x \geq B \wedge x \equiv_q c; \qquad\qquad x \leq -B \wedge x \equiv_q c; \text{ or} \qquad\qquad x = r,$$

where $q \stackrel{\text{def}}{=} \mathrm{lcm}(\mathrm{mod}(\Psi))$, $c \in [q]$, $r \in [-B+1, B-1]$ and B is a fixed natural number. Let us also consider the formula Φ_5 obtained from performing the QE procedure for the $\exists^{=x} y$ counting quantifier on $\exists^{=x} y\, \Phi$, so that $\Phi_0 \Leftrightarrow \exists x(\Psi' \wedge \Phi_5)$. In particular, recall that $\Phi_5 \stackrel{\text{def}}{=} \bigvee_{i=1}^{o} \bigvee_{r:\, Z \to [m]} (\Gamma_{i,r} \wedge \Phi_5^{i,r})$, where Z is the set of variables appearing in $\boldsymbol{z}$, $m = \mathrm{lcm}(\mathrm{mod}(\Phi))$ and $\Gamma_{i,r} = O_i \wedge (\bigwedge_{w \in Z} w \equiv_m r(w))$ is a conjunction of an ordering O_i and simple modulo constraints with variables from Z. Hence, $\Gamma_{i,r}$ is x-free. Moreover, $\Phi_5^{i,r}$ is either $\bot$ or a formula of the form

$$mx = \sum_{j=2}^{\ell} (p_j(t'_j - t'_{j-1}) + r_j) + m \cdot \sum_{j=1}^{\ell} c_j. \tag{2}$$

where the terms $t'_1, \dots, t'_\ell$ are from T (where T is defined as in Step II of Section 3), and hence x-free. Therefore, the following property holds.

Claim 9. In Φ_5, x only appears on the left-hand side of equalities of the form (2).

This inconspicuous claim, together with the shape of Δ_k, is at the heart of our QE procedure eliminating x from the formula $\exists x(\Psi' \wedge \Phi_5)$. Indeed, after distributing the existential quantifier $\exists x$ and all conjunctions over disjunctions of $\Psi' \wedge \Phi_5$, we end up with a disjunction of formulae of the form $\exists x : \Delta_k(x) \wedge \Psi_k(\boldsymbol{z}) \wedge \Gamma_{i,r} \wedge \Phi_5^{i,r}$, and let us consider one such disjunct with $\Delta_k(x) = (x \geq B \wedge x \equiv_q c)$ and $\Phi_5^{i,r}$ as in Equation (2). The variable x can be eliminated with a simple substitution, rewriting $\Delta_k(x) \wedge \Phi_5^{i,r}$ as the new formula $\widetilde{t} \geq m \cdot B \wedge \widetilde{t} \equiv_{m \cdot q} m \cdot c$, where $\widetilde{t}$ is the right-hand side of Equation (2). The correctness of this rewrite step follows simply from the equivalences $x \geq B \Leftrightarrow m \cdot x \geq m \cdot B$ and $x \equiv_q c \Leftrightarrow m \cdot x \equiv_{m \cdot q} m \cdot c$, with $m \geq 1$. In a similar way, we can treat all possible cases for the different forms of $\Delta_k(x)$ and $\Phi_5^{i,r}$. We obtain a formula

$$\Psi_k(\boldsymbol{z}) \wedge \Gamma_{i,r} \wedge \widetilde{t} \geq m \cdot B \wedge \widetilde{t} \equiv_{m \cdot q} m \cdot c. \tag{3}$$

The number of homogeneous terms across all such disjuncts is still prohibitive as it was in Φ_5. Now comes the key simplification step; we deal with the inequality $\widetilde{t} \geq m \cdot B$ and with the modulo constraint $\widetilde{t} \equiv_{m \cdot q} m \cdot c$.

Consider the former first. By definition, all the coefficients p_j of Equation (2) are non-negative, and thanks to the ordering O_i appearing in $\Gamma_{i,r}$, in every valuation ν satisfying the formula in Equation (3) we have $\nu(t'_j - t'_{j-1}) \geq 0$. Therefore, the inequality $\widetilde{t} \geq m \cdot B$ can be translated into a formula of the form $\bigvee_{g \in G} \bigwedge_{j=2}^{\ell} t'_j - t'_{j-1} \geq d_{g,j}$, where each $d_{g,j}$ is non-negative and, for every $g \in G$, the sum $\sum_{j=2}^{\ell} p_j d_{g,j}$ is at least $e \stackrel{\text{def}}{=} m(B - \sum_{j=1}^{\ell} c_j) - \sum_{j=2}^{\ell} r_j$. To compute this formula efficiently, we appeal to Lemma 2, with respect to the set of terms $\{t'_j - t'_{j-1} \mid j \in [2, \ell]\} \cup [0, e]$.

Lemma 6. *Let $d = |\text{fv}(O_i \wedge \widetilde{t} \geq m \cdot B)|$. In time $(e + \ell)^{\mathcal{O}(d)} \log(B \cdot \|O_i\|)^{\mathcal{O}(1)}$ one can compute a formula $\Theta = \bigvee_{g \in G} \bigwedge_{j=2}^{\ell} t'_j - t'_{j-1} \geq d_{g,j}$ s.t. (1) $d_{g,j} \in [0, e + 1]$, (2) $\#G \leq \mathcal{O}((e + \ell)^{2d})$, and (3) $O_i \wedge \widetilde{t} \geq m \cdot B \Leftrightarrow O_i \wedge \Theta$.*

A similar simplification can be done for the modulo constraint $\widetilde{t} \equiv_{m \cdot q} m \cdot c$: we guess residue classes of variables in $\widetilde{t}$ modulo $m \cdot q$, rewriting $\widetilde{t} \equiv_{m \cdot q} m \cdot c$ into $\bigvee_{s:\, Z \to [m \cdot q]} (\widetilde{t} \equiv_{m \cdot q} m \cdot c \wedge \bigwedge_{z \in Z} z \equiv_{m \cdot q} s(z))$ and then replace, in each disjunct, $\widetilde{t} \equiv_{m \cdot q} m \cdot c$ by $\top$ or $\bot$, according to the satisfaction of $s(\widetilde{t}) \equiv_{m \cdot q} m \cdot c$.

The steps just discussed forms the post-processing phase of our QE procedure for the monadically-guarded fragment of PAC. Thanks to Lemma 6, we can show that the set of homogeneous terms of the resulting quantifier free formula Φ', equivalent to Φ_0, is the set of homogeneous terms in the monadic decomposition Ψ', together with terms of the form $t - t'$ with t and t' belong to the set T defined in Line 5. But $\#\text{hom}(\Psi') = \mathcal{O}(\#\text{hom}(\Phi_0))$, and thus:

Lemma 7. $\#\text{hom}(\Phi') \leq \mathcal{O}(\#\text{hom}(\Phi_0)^2)$.

Running time. Lemma 7 is the key to obtaining an elementary QE procedure. In particular, this improvement over the exponential dependence of $\#\text{hom}(\Phi_5)$ on $\#\text{hom}(\Phi)$ from our "baseline" Lemma 5 leads to the following bounds on the elimination of an arbitrary number of monadically-guarded quantifiers.

Lemma 8. *Let Ω be a formula from the monadically-guarded fragment of PAC, with quantifier rank d. There is an equivalent quantifier-free formula Υ such that*

- $\#\text{hom}(\Upsilon) \leq |\Omega|^{2^{\mathcal{O}(d)}}$ *and* $\#\text{mod}(\Upsilon) \leq \mathcal{O}(|\Omega|)$;
- $\#\text{lin}(\Upsilon), \|\text{const}(\Upsilon)\|, \|\text{hom}(\Upsilon)\|$ *and* $\|\text{mod}(\Upsilon)\|$ *are at most* $2^{|\Omega|^{2^{\mathcal{O}(d)}}}$.

Proof idea. In a nutshell, the bounds of Lemma 8 are obtained by first iterating Lemma 7 across all quantifier elimination rounds. This results in the doubly exponential bound $|\Omega|^{2^{\mathcal{O}(d)}}$ on the cardinality of the set of homogeneous terms throughout the entire procedure. With this bound in hand, exponentiation on the right-hand side of the inequalities of Section 3 does not blow the parameters above triple exponential. $\square$

Subsequent analysis leads to the following result.

Theorem 3. *There is a* 3EXPTIME *quantifier elimination procedure for the monadically-guarded fragment of PAC.*

Theorem 3 follows by combining Lemma 8 with upper bounds on the running time of a single quantifier elimination round. These upper bounds are all subsumed by the size of the obtained formulae, except possibly for the subdivision procedure of Step II (Lemma 2), the model counting procedure of Step IV (Lemma 4), and the further subdivision performed by Lemma 6. For Lemmas 2 and 6, the running time is only exponential in the size of the original formula, and thus polynomial time in the size of the obtained formula, as long as the latter

has at least exponential size. For Lemma 4, observe that $m \leq \|\mathrm{mod}(\Upsilon)\|$, where Υ is the quantifier-free formula of Lemma 8. Therefore, the bounds of Lemma 8 suffice for a triply exponential time overall.

Remark 2. Only small updates are necessary to treat monadically-guarded formulae of the form $\exists x(\Psi(x, \boldsymbol{z}) \wedge \exists^{\geq x} y \Phi(y, \boldsymbol{z}))$. Again, these updates deal with the fact that, contrary to $\exists^{=x} y \Phi$, the formula $\exists^{\geq x} y \Phi$ is true whenever there are infinitely many y satisfying Φ, or alternatively when x corresponds to a non-positive number. Then, Lemma 8 can be established for formulae of PAC containing both monadically-guarded quantifiers $\exists^{=x}$ and $\exists^{\geq x}$.

7 The monadically-guarded fragment is in doubly exponential space

In this section, we prove Theorem 2. Theorem 3 shows that our QE procedure has the same asymptotic running time as the standard QE procedures for PA. Historically, bounds obtained from the latter lead to computationally optimal decision procedures based on quantifier relativisation [12,4]. More precisely, given a formula Φ from PA, the QE procedures allow us to conclude that there is a bound C, of bitsize at most doubly exponential in $|\Phi|$, such that $\exists x \, \Phi \Leftrightarrow \exists x : -C \leq x \leq C \wedge \Phi$ holds (a *small-model property*). Then, a *quantifier relativisation procedure* follows the semantics of the formula and naïvely tries all the possible assignments to x in $[-C, C]$ whenever a quantifier $\exists x$ is encountered. With some bookkeeping, this procedure runs in 2ExpSpace. In this section, we show that this is also the case for our QE procedure, leading to a 2ExpSpace relativisation procedure for the monadically-guarded fragment of PAC, proving Theorem 2.

First of all, we need to recall a folklore result regarding the existence of infinitely many solutions of a quantifier-free Presburger formula.

Lemma 9. *Let ν be an assignment and $\Phi(y, \boldsymbol{z})$ be a q.f. formula of PA, where $\boldsymbol{z}$ has d variables. Let $C \stackrel{\text{def}}{=} \|\Phi\| \cdot d \cdot \max\{1, |\nu(z)| : z \text{ is in } \boldsymbol{z}\} + \|\Phi\|^{\#\mathrm{mod}(\Phi)} + 1$.*
1. *If there are finitely many $n \in \mathbb{Z}$ s.t. $\nu[n/y] \models \Phi$, then they all satisfy $|n| \leq C$.*
2. *If there are infinitely many $n \in \mathbb{Z}$ such that $\nu[n/y] \models \Phi$, then for every $j \in \mathbb{N}_+$ there is such an n satisfying $j \cdot C < |n| \leq (j + 1) \cdot C$.*

Together with Lemma 8, this result leads to the relativisation of first-order quantifiers in the context of PAC.

Lemma 10. *There is a constant c with the following property. Let ν be an assignment, $\Phi(y, \boldsymbol{z})$ be a monadically-guarded formula of PAC, where $\boldsymbol{z}$ has d variables, and let $C \stackrel{\text{def}}{=} 2^{|\Psi|^{2^{c \cdot d}}} \cdot \max\{1, |\nu(z)| : z \text{ is in } \boldsymbol{z}\}$. Then, $\nu \models \exists y \, \Phi$ if and only if $\nu[n/y] \models \Phi$ holds for some $n \in \mathbb{Z}$ with $|n| \leq 3 \cdot C$.*

We want to derive a similar lemma for monadically guarded counting quantifiers. First of all, we consider a formula $\Phi = \exists^{=x} y \, \Psi(y, \boldsymbol{z})$ where Ψ is a monadically guarded formula. Recall that Φ is satisfied by an assignment ν whenever the number of distinct values $n \in \mathbb{Z}$ such that $\nu[n/y] \models \Psi$ is finite and equal to $\nu(x)$. By relying on Lemmas 8 and 9, we show the following lemma.

Lemma 11. *There is a constant c with the following property. Let ν be an assignment, and consider a formula $\Phi = \exists^{=x} y\,\Psi(y, \boldsymbol{z})$ such that Ψ is a monadically guarded formula of quantifier rank d. Let $C \stackrel{\text{def}}{=} 2^{|\Psi|^{2^{c \cdot d}}} \cdot \max\{1, |\nu(z)| : z \text{ is in } \boldsymbol{z}\}$. Then, $\nu \models \Phi$ iff (i) $\nu[n/y] \not\models \Psi$, for every $n \in \mathbb{Z}$ with $C < |n| \le 3 \cdot C$; and (ii) $\#\{n \in \mathbb{Z} : |n| \le C \text{ and } \nu[n/y] \models \Psi\} = \nu(x)$.*

We now consider the outermost quantifier x of a monadically-guarded formula $\Theta = \exists x\,(\Psi(x, \boldsymbol{z}) \wedge \exists^{=x} y\,\Phi(y, \boldsymbol{z}))$, and aim at finding relativisation bounds for the variable x. Notice that the subformula $\Psi(x, \boldsymbol{z}) \wedge \exists^{=x} y\,\Phi(y, \boldsymbol{z})$ is not, strictly speaking, in the monadically-guarded fragment of PAC. However, we can first apply Lemma 8 and obtain quantifier-free formulae $\widehat{\Psi}$ and Φ' equivalent to Ψ and Φ, respectively. Then, we apply the QE procedure of Section 3 on input $\exists^{=x} y\,\Phi'$, producing an equivalent quantifier-free formula $\widehat{\Phi}$. We have $\Theta \Leftrightarrow \exists x\,(\widehat{\Psi} \wedge \widehat{\Phi})$, where $\widehat{\Psi} \wedge \widehat{\Phi}$ is quantifier-free. Similarly to Lemma 10, we can now obtain relativisation bounds from $\exists x\,(\widehat{\Psi} \wedge \widehat{\Phi})$ by relying on Lemma 9:

Lemma 12. *There is a constant c with the following property. Let ν be an assignment, and let $\Theta = \exists x\,(\Psi(x, \boldsymbol{z}) \wedge \exists^{=x} y\,\Phi(y, \boldsymbol{z}))$ be a monadically-guarded formula of quantifier rank d. Define $C \stackrel{\text{def}}{=} 2^{|\Theta|^{2^{c \cdot d}}} \cdot \max\{1, |\nu(z)| : z \text{ is in } \boldsymbol{z}\}$. Then, $\nu \models \Theta$ if and only if there is $n \in \mathbb{N}$ s.t. $n \le C$ and $\nu[n/x] \models \Psi \wedge \exists^{=x} y\,\Phi$.*

Lemmas 10 to 12 allow to evaluate the truth of a sentence of the monadically-guarded fragment of PAC by recursively evaluating the truth of its subformulae, and iterating over a finite set of values when considering first-order and counting quantifiers. As all the considered values admit a binary encoding that is doubly exponential in the size of the input formula, this proves Theorem 2.

8   A complexity characterisation

By Theorem 2, for deterministic machines, the monadically-guarded fragment of PAC is no harder than standard Presburger arithmetic, and the same is true when considering monadically-guarded quantifiers $\exists^{\ge x} y$ (Remark 2). However, by Proposition 1, PA is not complete for 2EXPSPACE, but rather for the complexity class $\mathrm{STA}(*, 2^{2^{n^{\mathcal{O}(1)}}}, \mathcal{O}(n))$. This leads to the natural question on whether the monadically-guarded fragment of PAC is also complete for the same STA class. While we leave this question open, in this section we show a completeness result in the restricted case where all monadically-guarded quantifiers appear in the form $\exists x(\Psi(x) \wedge \exists^{\ge x} y\,\Phi)$, where $\Psi(x)$ is any formula from PAC having all models bounded by $2^{2^{|\Psi|}}$, in absolute value. For brevity, let us denote this fragment by F. As F extends PA, proving the following upper bound suffices.

Theorem 4. *The validity of a sentence Φ in F can be decided by an alternating Turing machine with runtime $2^{2^{|\Phi|^{\mathcal{O}(1)}}}$ and performing $\mathcal{O}(\mathrm{fd}(\Phi))$ alternations.*

Since the equivalence $\exists^{\ge c} y\,\Phi \Leftrightarrow \exists x : x = c \wedge \exists^{\ge x} y\,\Phi$, where $c \in \mathbb{Z}$ is written in binary, shows that F contains PA enriched with threshold counting quantifiers,

Function check(V : non-empty set of assignments, Φ : formula from F) $\to \{\top, \bot\}$

1 check($V, t < 0$) $=$ if $\nu \models t < 0$ holds for all $\nu \in V$ then return $\top$ else return $\bot$.
2 check($V, \Phi_1 \vee \Phi_2$) $=$ if $\exists V_1, V_2 : V = V_1 \cup V_2$ and check(V_1, Φ_1) $=$ check(V_2, Φ_2) $= \top$
3   then return $\top$ else return $\bot$.
4 check($V, \neg\Psi$) $=$ if $\exists \nu \in V$: check($\{\nu\}, \Psi$) $= \top$ then return $\bot$ else return $\top$.
5 check($V, \exists x(\Psi(x) \wedge \exists^{\geq x} y\, \Theta(y, \boldsymbol{z})))$ $=$ if there is a family $(W_\nu)_{\nu \in V}$ such that
6   • check($\bigcup_{\nu \in V} W_\nu, \Psi \wedge \Theta$) $= \top$, and
7   • each W_ν is either $\{\nu[k^{(\nu)}/x][n^{(\nu)}/y]\}$ or $\{\nu[k^{(\nu)}/x][n_i^{(\nu)}/y] : i \in [1, k^{(\nu)}]\}$
8   where $|k^{(\nu)}| \leq 2^{2^{|\Psi|}}$
9   let C be defined as in Lemma 13, w.r.t. ν and Ψ;
10                 $n^{(\nu)} \in \mathbb{Z}$ such that $C < |n^{(\nu)}| \leq 3 \cdot C$;
11                 $|n_i^{(\nu)}| \leq C$ for every $i \in [1, k^{(\nu)}]$; and
12                 $n_i^{(\nu)} \neq n_j^{(\nu)}$ for every two distinct $i, j \in [1, k^{(\nu)}]$
13   then return $\top$ else return $\bot$.

Fig. 1. Deciding whether a formula Φ from F is satisfied by all assignments in V.

this result implies Theorem 1. To establish Theorem 4, the first step is to rely on Lemmas 8 and 9 and adapt the proof of Lemma 11 to obtain a quantifier relativisation argument for the counting quantifier $\exists^{\geq x} y$.

Lemma 13. *There is a constant c with the following property. Let ν be an assignment, and consider a formula $\Phi = \exists^{\geq x} y\, \Psi(y, \boldsymbol{z})$ such that Ψ is a monadically guarded formula of quantifier rank d. Let $C \stackrel{\text{def}}{=} 2^{|\Psi|^{2^{c \cdot d}}} \cdot \max\{1, |\nu(z)| : z \text{ is in } \boldsymbol{z}\}$. Then, $\nu \models \Phi$ iff (i) there is $n \in \mathbb{Z}$ s.t. $\nu[n/y] \models \Psi$ and $C < |n| \leq 3 \cdot C$, or (ii) $\#\{n \in \mathbb{Z} : |n| \leq C \text{ and } \nu[n/y] \models \Psi\} \geq \nu(x)$.*

With Lemma 13 at hand, designing an algorithm that can be implemented as an alternating Turing machine with resources bounded as in Theorem 4 is simple. The function check($\cdot, \cdot$) given in Figure 1 provides such an algorithm.

Lemma 14. check(V, Φ) *returns $\top$ if and only if for all $\nu \in V$, $\nu \models \Phi$.*

When Φ is a sentence, i.e. $\text{fv}(\Phi) = \emptyset$, this lemma implies that Φ is valid if and only if check($\{\nu\}, \Phi$) $= \top$, where ν is an arbitrary assignment. Then, Theorem 4 follows by establishing that check($\cdot, \cdot$) can be implemented with an alternating Turing machine that, on input $(\{\nu\}, \Phi)$ where Φ is a sentence in F, runs in time $2^{2^{|\Phi|^{\mathcal{O}(1)}}}$ and performs $\mathcal{O}(\text{fd}(\Phi))$ many alternations. We see the existential quantifications on V_1, V_2, ν and $(W_\nu)_{\nu \in V}$ in Lines 2, 4 and 5 as guesses done by the alternating Turing machine. The computation in Line 1 is done deterministically in time polynomial in the encoding of V and $t < 0$. In Line 2, the alternating Turing machine decides which branch among check(V_1, Φ_1) and check(V_2, Φ_2) must be evaluated, at the cost of one alternation. In this way, alternations occur only in the case of check($V, \Phi_1 \vee \Phi_2$) and check($V, \neg\Psi$), as the latter returns the negation of the assertion "$\exists \nu \in V$: check($\{\nu\}, \Psi$) $= \top$". This leads to $\mathcal{O}(\text{fd}(\Phi))$

many alternations overall. Let us now discuss the runtime of check$(\cdot, \cdot)$, again on alternating Turing machines. Assume that, after a certain number of recursive calls including at most $r \leq \mathrm{qr}(\Phi)$ calls to Line 5, the algorithm evaluates the input (V', Ψ). Then, the number of assignments in V' is bounded by $2^{r \cdot 2^{|\Phi|}}$ (this correspond to the case where each W_ν in Line 7 contains the maximum amount of assignments, according to $k^{(\nu)}$), and following the bounds on the numbers $n^{(\nu)}$ and $n_i^{(\nu)}$ in Lines 10 and 11 and by Lemma 13, all these assignments map each variable to an integer that is, in absolute value, bounded by $2^{r \cdot |\Phi|^{2^{c \cdot d}}}$, where c is the constant of Lemma 13 and d is the number of variables in Φ. So, as the number of recursive calls to check$(\cdot, \cdot)$ is bounded by $|\Phi|$, no more than $|\Phi| \cdot 2^{\mathrm{qr}(\Phi) \cdot 2^{|\Phi|}} \cdot \log_2(2^{\mathrm{qr}(\Phi) \cdot |\Phi|^{2^{c \cdot d}}}) \leq 2^{2^{(c+3)|\Phi|}}$ space is required to represent all possible sets of assignments that are generated throughout the evaluation of check$(\cdot, \cdot)$. All the assignments are guessed by the alternating Turing machine and thus, when also accounting for the computation done in Line 1, we conclude that check$(\{\nu\}, \Phi)$ runs in time $2^{2^{|\Phi|^{\mathcal{O}(1)}}}$.

9   Conclusion

We developed a new quantifier elimination procedure for Presburger arithmetic extended with the unary counting quantifiers (PAC), and adapted it for its monadically-guarded fragment. While the existence of an algorithm for PAC running in elementary time is wide open, our procedure runs in 3ExpTime on the monadically-guarded fragment and leads to the small-model property and relativisation argument, which show that this logic is decidable in 2ExpSpace. When it comes to deterministic algorithms, this matches the complexity of deciding standard Presburger arithmetic. However, fully settling the complexity of the monadically-guarded fragment of Presburger arithmetic seems to require a generalisation of the STA complexity framework to capture counting mechanisms, which we leave as an avenue for further investigation. In this direction, we have shown that Presburger arithmetic is still STA$(*, 2^{2^{n^{\mathcal{O}(1)}}}, O(n))$-complete when enriched with threshold quantifiers $\exists^{\geq c} y$, for the case of c written in binary but also even for the case of c represented succinctly as a solution of a Presburger formula Φ, characterising a number that may be doubly exponential in $|\Phi|$.

With respect to our QE procedure for (general) unary counting quantifiers $\exists^{=x}$, we have pinpointed precisely where the non-elementary growth occurs. It remains to be seen whether our procedure can be further improved, or if, possibly based on insights obtained from it, a non-elementary lower bound for Presburger arithmetic extended with the $\exists^{=x} y$ quantifier can be established.

References

1. Apelt, H.: Axiomatische Untersuchungen über einige mit der Presburgerschen Arithmetik verwandte Systeme. Math. Log. Q. **12**(1), 131–168 (1966)
2. Barvinok, A.I.: A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed. Math. Oper. Res. **19**(4), 769–779 (1994)
3. Berman, L.: The complexity of logical theories. Theor. Comput. Sci. **11**(1), 71–77 (1980)
4. Habermehl, P., Kuske, D.: On Presburger arithmetic extended with modulo counting quantifiers. In: Proc. Foundations of Software Science and Computation Structures, FoSSaCS. Lect. Notes Comput. Sc., vol. 9034, pp. 375–389. Springer (2015)
5. Hague, M., Lin, A.W., Rümmer, P., Wu, Z.: Monadic decomposition in integer linear arithmetic. In: Proc. International Joint Conference on Automated Reasoning, IJCAR. Lect. Notes Comput. Sc., vol. 12166, pp. 122–140. Springer (2020)
6. Herre, H., Krynicki, M., Pinus, A., Väänänen, J.: The Härtig quantifier: A survey. J. Symb. Comput. **56**(4), 1153–1183 (1991)
7. Kozen, D.: Theory of Computation. Texts in Computer Science, Springer (2006)
8. Oppen, D.C.: A $2^{2^{2^{pn}}}$ upper bound on the complexity of Presburger arithmetic. J. Comput. Syst. Sci. **16**(3), 323–332 (1978)
9. Presburger, M.: Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In: Comptes Rendus du I congres de Mathematiciens des Pays Slaves, pp. 92–101. American Mathematical Society (1929)
10. Schweikardt, N.: Arithmetic, first-order logic, and counting quantifiers. ACM Trans. Comput. Log. **6**(3), 634–671 (2005)
11. Veanes, M., Bjørner, N., Nachmanson, L., Bereg, S.: Monadic decomposition. J. ACM **64**(2), 14:1–14:28 (2017)
12. Weispfenning, V.: The complexity of almost linear diophantine problems. J. Symb. Comput. **10**(5), 395–404 (1990)
13. Woods, K.: Presburger arithmetic, rational generating functions, and quasi-polynomials. J. Symb. Log. **80**(2), 433–449 (2015)

A first-order logic characterisation of safety and co-safety languages

Alessandro Cimatti[1], Luca Geatti[3](✉), Nicola Gigante[3](✉),
Angelo Montanari[2], and Stefano Tonetta[1]

[1] Fondazione Bruno Kessler, Trento, Italy
{cimatti,tonettas}@fbk.eu
[2] University of Udine, Italy
angelo.montanari@uniud.it
[3] Free University of Bozen-Bolzano, Italy
{geatti,gigante}@inf.unibz.it

Abstract. *Linear Temporal Logic* (LTL) is one of the most popular temporal logics, that comes into play in a variety of branches of computer science. Its widespread use is also due to its strong foundational properties. One of them is Kamp's theorem, showing that LTL and the *first-order theory of one successor* (S1S[FO]) are expressively equivalent. Safety and co-safety languages, where a finite prefix suffices to establish whether a word does not or does belong to the language, respectively, play a crucial role in lowering the complexity of problems like model checking and reactive synthesis for LTL. Safety-LTL (resp., coSafety-LTL) is a fragment of LTL where only universal (resp., existential) temporal modalities are allowed, that recognises safety (resp., co-safety) languages only.
In this paper, we introduce a fragment of S1S[FO], called Safety-FO, and its dual coSafety-FO, which are *expressively complete* with regards to the LTL-definable safety languages. In particular, we prove that they respectively characterise exactly Safety-LTL and coSafety-LTL, a result that joins Kamp's theorem, and provides a clearer view of the charactisations of (fragments of) LTL in terms of first-order languages. In addition, it gives a direct, compact, and self-contained proof that any safety language definable in LTL is definable in Safety-LTL as well. As a by-product, we obtain some interesting results on the expressive power of the *weak tomorrow* operator of Safety-LTL interpreted over finite and infinite traces.

1 Introduction

Linear Temporal Logic (LTL) is the de-facto standard logic for system specifications [14]. It is a modal logic that is usually interpreted over infinite state sequences, but the finite-trace semantics has recently gained attention as well [6,7]. The widespread use of LTL is due to its simple syntax and semantics, and to its strong foundational properties. Among them, we would like to mention the seminal work by Kamp [10] and Gabbay *et al.* [8], on its expressive completeness, i.e., LTL-definable languages are exactly those definable in the first-order fragment of the monadic second-order theory of one successor [3] (S1S[FO] for short).

In formal verification, an important class of specifications is that of *safety languages*. They are languages of infinite words where a finite prefix suffices to tell whether a word does not belong to the language. As an example, the set of all and only those infinite sequences where some particular bad event never happens can be regarded as a safety language. In their duals, *co-safety* languages (sometimes called *guarantee* languages), a finite prefix is sufficient to tell whether a word *belongs* to the language, e.g., when some desired event is mandated to eventually happen. Safety and co-safety languages are important for verification, model-checking, monitoring, and automated synthesis because they capture a variety of real-world requirements while being much simpler to deal with algorithmically [1, 11, 20].

Safety-LTL is the fragment of LTL where only *universal* temporal modalities are allowed. Similarly, its dual coSafety-LTL is obtained by only allowing *existential* modalities. It has been proved by Chang *et al.* [5] that Safety-LTL and coSafety-LTL define exactly the safety and co-safety languages that are definable in LTL, respectively.

In this paper, we provide a novel characterization of LTL-definable safety languages, and of their duals, in terms of a fragment of S1S[FO], called Safety-FO, and its dual coSafety-FO. The presented fragments have a very natural syntax, and we prove they are *expressively complete* with regards to LTL-definable safety and co-safety languages. We prove the correspondence between coSafety-FO and coSafety-LTL, which extends naturally to their duals and can be considered as a version of Kamp's theorem [10] specialized for safety and co-safety properties, helping to create a clearer picture of the correspondence between (fragments of) temporal and first-order logics. We exploit such a result to prove the correspondence between co-safety languages definable in LTL and coSafety-FO, thus establishing also the equivalence between the former and coSafety-LTL. This provides a proof of the fact that Safety-LTL captures exactly the set of LTL-definable safety languages [5], which can be regarded as another contribution of the paper. The interest of our proof is twofold: on the one hand, the original proof by Chang *et al.* [5] is only sketched and it relies on two non-trivial translations scattered across different sources [16, 21]; on the other hand, such an equivalence result seems not to be very much known, as some authors presented the problem as open as lately as 2017 [20].[4] Thus, a compact and self-contained proof of the result seems to be a useful contribution for the community. It is worth to note that both proofs build on the fact that safety/co-safety languages can be captured by formulas of the form $G\alpha/F\alpha$ with α pure-past, but after that, the two proofs significantly diverge. Finally, as a by-product of this proof, we provide some results that assess the expressive power of the *weak tomorrow* operator of Safety-LTL when interpreted over finite *vs.* infinite traces.

The paper is organized as follows. After recalling necessary background knowledge in Section 2, Section 3 introduces Safety-FO and coSafety-FO and proves their correspondence with Safety-LTL and coSafety-LTL. Then, Section 4 proves

---

[4] As a matter of fact, we discovered about Chang *et al.* [5] after setting up the proof shown in this paper.

their correspondence with the set of safety and co-safety languages definable in LTL, thus providing a compact and self-contained proof of the equivalence between Safety-LTL and LTL-definable safety languages. Some properties of the *weak next* operator are outlined as well. Finally, Section 5 concludes the paper with some final considerations and a discussion of future work.

2   Preliminaries

Let A be a finite alphabet. We denote as A^* and A^ω the set of all finite and infinite words, respectively, over A. We let $A^+ = A^* \setminus \{\varepsilon\}$, where ε is the empty word. Given a word $\sigma \in A^*$ we denote as $|\sigma|$ the length of σ. For an infinite word $\sigma \in A^\omega$, $|\sigma| = \omega$. For a (finite or infinite) word σ, we denote as $\sigma_i \in A$, for $0 \leq i < |\sigma|$, the letter at the i-th position of the word. With $\sigma_{[i,j]}$, for $0 \leq i \leq j < |\sigma|$, we denote the subword that goes from the i-th to the j-th letter of the word, extrema included. With $\sigma_{[i,\infty]}$ we denote the suffix of σ starting from the i-th letter. Given a word $\sigma \in A^*$ and $\sigma' \in A^* \cup A^\omega$, we denote the *concatenation* of the two words as $\sigma \cdot \sigma'$, or simply $\sigma\sigma'$. A *language* $\mathcal{L}$, either $\mathcal{L} \subseteq A^*$ or $\mathcal{L} \subseteq A^\omega$, is a set of words. Given two languages $\mathcal{L}$ and $\mathcal{L}'$ with $\mathcal{L} \subseteq A^*$ and either $\mathcal{L}' \subseteq A^*$ or $\mathcal{L}' \subseteq A^\omega$, we define $\mathcal{L} \cdot \mathcal{L}' = \{\sigma \cdot \sigma' \mid \sigma \in \mathcal{L} \text{ and } \sigma' \in \mathcal{L}'\}$. For a finite word $\sigma = \sigma_0 \dots \sigma_k$ let $\sigma^r = \sigma_k \dots \sigma_0$ be the reverse of σ, and for a language of finite words $\mathcal{L}$ let $\mathcal{L}^r = \{\sigma^r \mid \sigma \in \mathcal{L}\}$. We can now define *safety* and *co-safety* languages.

Definition 1 (Safety language [11, 19]). *Let $\mathcal{L} \subseteq A^\omega$. We say that $\mathcal{L}$ is a safety language if and only if for all the words $\sigma \in A^\omega$ it holds that, if $\sigma \notin \mathcal{L}$, then there exists an $i \in \mathbb{N}$ such that, for all $\sigma' \in A^\omega$, $\sigma_{[0,i]} \cdot \sigma' \notin \mathcal{L}$. The class of safety languages is denoted as* SAFETY.

Definition 2 (Co-safety language [11, 19]). *Let $\mathcal{L} \subseteq A^\omega$. We say that $\mathcal{L}$ is a* co-safety *language if and only if for all the words $\sigma \in A^\omega$ it holds that, if $\sigma \in \mathcal{L}$, then there exists an $i \in \mathbb{N}$ such that, for all $\sigma' \in A^\omega$, $\sigma_{[0,i]} \cdot \sigma' \in \mathcal{L}$. The class of co-safety languages is denoted as* coSAFETY.

Linear Temporal Logic with Past (LTL+P) is a modal logic interpreted over infinite or finite words. Given a set Σ of proposition variables, the syntax of an LTL formula ϕ is generated by the following grammar:

$$\phi := p \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \qquad \text{Boolean connectives}$$
$$\mid \mathsf{X}\phi_1 \mid \widetilde{\mathsf{X}}\phi_1 \mid \phi_1 \, \mathcal{U} \, \phi_2 \mid \phi_1 \, \mathcal{R} \, \phi_2 \qquad \text{future modalities}$$
$$\mid \mathsf{Y}\phi_1 \mid \mathsf{Z}\phi_1 \mid \phi_1 \, \mathcal{S} \, \phi_2 \mid \phi_1 \, \mathcal{T} \, \phi_2 \qquad \text{past modalities}$$

where ϕ_1 and ϕ_2 are LTL+P formulas and $p \in \Sigma$. An LTL+P formula is a *pure future* formula if it does not make use of past modalities, and it is *pure past* if it does not make use of future modalities. We denote with LTL the set of pure future formulas, and with LTL$_\mathsf{P}$ the set of pure past formulas. Most of the temporal operators of the language can be defined in terms of a small number

of basic ones. In particular, conjunction can be defined in terms of disjunction ($\phi_1 \wedge \phi_2 \equiv \neg(\neg\phi_1 \vee \neg\phi_2)$), the *release* operator can be defined in terms of the *until* operator ($\phi_1 \mathcal{R} \phi_2 \equiv \neg(\neg\phi_1 \mathcal{U} \neg\phi_2)$), and the *triggered* operator can be defined in terms of the *since* operator ($\phi_1 \mathcal{T} \phi_2 \equiv \neg(\neg\phi_1 \mathcal{S} \neg\phi_2)$). Nevertheless, we consider all these connectives and operators as primitive in order to be able to put any formula in *negated normal form* (NNF), *i.e.*, a form where negations are only applied to proposition letters. Note that the syntax includes both a *tomorrow* ($\mathsf{X}\phi$) and *weak tomorrow* ($\widetilde{\mathsf{X}}\phi$) operators, as well as a *yesterday* ($\mathsf{Y}\phi$) and *weak yesterday* ($\mathsf{Z}\phi$) operators, for the same reason. Moreover, standard shortcut operators are available such as the *eventually* ($\mathsf{F}\phi \equiv \top\, \mathcal{U} \phi$), and *always* ($\mathsf{G}\phi \equiv \neg\mathsf{F}\neg\phi$) future operators, and the *once* ($\mathsf{O}\phi \equiv \top\, \mathcal{S}\, \phi$), and *historically* ($\mathsf{H}\phi \equiv \neg\mathsf{O}\neg\phi$) past operators.

LTL+P is interpreted over *state sequences*, which are finite or infinite words over 2^Σ. Given a state sequence $\sigma \in (2^\Sigma)^+$ or $\sigma \in (2^\Sigma)^\omega$, the *satisfaction* of a formula ϕ by σ at a time point $i \geq 0$, denoted as $\sigma, i \models \phi$, is defined as follows:

1. $\sigma, i \models p$ iff $p \in \sigma_i$;
2. $\sigma, i \models \neg\phi$ iff $\sigma, i \not\models \phi$;
3. $\sigma, i \models \phi_1 \vee \phi_2$ iff $\sigma, i \models \phi_1$ or $\sigma, i \models \phi_2$;
4. $\sigma, i \models \phi_1 \wedge \phi_2$ iff $\sigma, i \models \phi_1$ and $\sigma, i \models \phi_2$;
5. $\sigma, i \models \mathsf{X}\phi$ iff $i + 1 < |\sigma|$ and $\sigma, i + 1 \models \phi$;
6. $\sigma, i \models \widetilde{\mathsf{X}}\phi$ iff   either $i + 1 = |\sigma|$ or $\sigma, i + 1 \models \phi$;
7. $\sigma, i \models \mathsf{Y}\phi$ iff $i > 0$ and $\sigma, i - 1 \models \phi$;
8. $\sigma, i \models \mathsf{Z}\phi$ iff   either $i = 0$ or $\sigma, i - 1 \models \phi$;
9. $\sigma, i \models \phi_1 \mathcal{U} \phi_2$ iff   there exists $i \leq j < |\sigma|$ such that $\sigma, j \models \phi_2$, and $\sigma, k \models \phi_1$ for all k, with $i \leq k < j$;
10. $\sigma, i \models \phi_1 \mathcal{S} \phi_2$ iff   there exists $j \leq i$ such that $\sigma, j \models \phi_2$, and $\sigma, k \models \phi_1$ for all k, with $j < k \leq i$;
11. $\sigma, i \models \phi_1 \mathcal{R} \phi_2$ iff   either $\sigma, j \models \phi_2$ for all $i \leq j < |\sigma|$, or there exists $k \geq i$ such that $\sigma, k \models \phi_1$ and $\sigma, j \models \phi_2$ for all $i \leq j \leq k$;
12. $\sigma, i \models \phi_1 \mathcal{T} \phi_2$ iff   either $\sigma, j \models \phi_2$ for all $0 \leq j \leq i$, or there exists $k \leq i$ such that $\sigma, k \models \phi_1$ and $\sigma, j \models \phi_2$ for all $i \geq j \geq k$

We say that a state sequence σ satisfies ϕ, written $\sigma \models \phi$, if $\sigma, 0 \models \phi$. Note that, when interpreted over an infinite word, the *tomorrow* and *weak tomorrow* operators have the same semantics. The *language* of ϕ, denoted as $\mathcal{L}(\phi)$, is the set of words $\sigma \in (2^\Sigma)^\omega$ such that $\sigma \models \phi$. The *language of finite words* of ϕ, denoted as $\mathcal{L}^{<\omega}(\phi)$, is the set of finite words $\sigma \in (2^\Sigma)^+$ such that $\sigma \models \phi$. Given a logic L (*e.g.*, LTL), we denote as $[\![L]\!]$ the set of languages $\mathcal{L}$ such that there is a formula $\phi \in L$ such that $\mathcal{L} = \mathcal{L}(\phi)$, and we denote as $[\![L]\!]^{<\omega}$ the set of languages of finite words $\mathcal{L}$ such that there is a formula $\phi \in L$ such that $\mathcal{L} = \mathcal{L}^{<\omega}(\phi)$. Note that $[\![\mathsf{LTL}]\!]^{<\omega}$ is usually called LTLf in the literature [6].

We now define the two fragments of LTL that are the subject of this paper.

Definition 3 (Safety-LTL and coSafety-LTL [17])**.** *The logic* Safety-LTL *(resp.* coSafety-LTL*) is the fragment of* LTL *where, for formulas in* negated normal

form, *only the* tomorrow, weak tomorrow *and* release *(resp.* until*) temporal operators are allowed.*

We also define the logic coSafety-LTL($-\widetilde{\mathsf{X}}$) as the logic coSafety-LTL devoid of the *weak tomorrow* operator (this logic will play a central role in our proofs).

In the next Section we present two fragments of the *first-order theory of one successor* [2, 3], namely S1S[FO], or simply FO in the following. Fixed an alphabet Σ, FO is a first-order language with equality over the signature $\langle <, \{P\}_{p \in \Sigma} \rangle$, and is interpreted over structures $\mathcal{M} = \langle D^{\mathcal{M}}, <^{\mathcal{M}}, \{P^{\mathcal{M}}\}_{p \in \Sigma} \rangle$ where $D^{\mathcal{M}}$, for our goals, is either the set $\mathbb{N}$ of natural numbers or a prefix $\{0, \ldots, n\}$ thereof, and $<^{\mathcal{M}}$ is the usual ordering relation between natural numbers. Given an FO formula $\phi(x_0, \ldots, x_m)$ with $m + 1$ free variables, the satisfaction of ϕ by a first-order structure $\mathcal{M}$ when $x_0 = n_0, \ldots, x_m = n_m$, denoted as $\mathcal{M}, n_0, \ldots, n_m \models \phi(x_0, \ldots, x_m)$, is defined following the standard first-order semantics. State sequences over Σ map naturally into such structures. Given a word $\sigma \in (2^{\Sigma})^*$ or $\sigma \in (2^{\Sigma})^{\omega}$, we denote as $(\sigma)^s$ the corresponding first-order structure. Given a formula $\phi(x)$ with exactly one free variable, the *language* of ϕ, denoted as $\mathcal{L}(\phi)$, is the set of words $\sigma \in (2^{\Sigma})^{\omega}$ such that $(\sigma)^s, 0 \models \phi$. Similarly, the *language of finite words* of ϕ, denoted as $\mathcal{L}^{<\omega}(\phi)$, is the set of finite words $\sigma \in (2^{\Sigma})^+$ such that $(\sigma)^s \models \phi$. We denote as $[\![\mathsf{FO}]\!]$ and $[\![\mathsf{FO}]\!]^{<\omega}$ the set of languages of infinite and finite words, respectively, definable by a FO formula.

Given a class of languages of finite words $[\![\mathsf{L}]\!]^{<\omega}$, we denote as $[\![\mathsf{L}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$ the set of languages $[\![\mathsf{L}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega} = \{\mathcal{L} \cdot (2^{\Sigma})^{\omega} \mid \mathcal{L} \in [\![\mathsf{L}]\!]^{<\omega}\}$. We recall now some known results.

Proposition 1 (Kamp [10] and Gabbay [8]).
$[\![\mathsf{LTL}]\!] = [\![\mathsf{FO}]\!]$ *and* $[\![\mathsf{LTL}]\!]^{<\omega} = [\![\mathsf{FO}]\!]^{<\omega}$.

Finally, we state a normal form for LTL-definable safety/co-safety languages.

Proposition 2 (Chang *et al.* [5], Thomas [19]). *A language $\mathcal{L} \in [\![\mathsf{LTL}]\!]$ is safety (resp. co-safety) if and only if it is the language of a formula of the form $\mathsf{G}\alpha$ (resp. $\mathsf{F}\alpha$), where $\alpha \in \mathsf{LTL_P}$.*

3   Safety-FO and coSafety-FO

In this section we introduce the core contribution of the paper, *i.e.*, two fragments of FO that precisely capture Safety-LTL and coSafety-LTL, respectively, and we prove this relationship. A summary of the results provided by the paper is given in Fig. 1.

Definition 4 (Safety-FO). *The logic* Safety-FO *is generated by the following grammar:*

$$atomic := x < y \mid x = y \mid x \neq y \mid P(x) \mid \neg P(x)$$
$$\phi := atomic \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \exists y (x < y < z \wedge \phi_1) \mid \forall y (x < y \rightarrow \phi_1)$$

where x, y, and z are first-order variables, P is a unary predicate, and ϕ_1 and ϕ_2 are Safety-FO *formulas.*

$$\llbracket \mathsf{LTL} \rrbracket \cap \mathsf{SAFETY} \quad = \quad \llbracket \mathsf{Safety\text{-}LTL} \rrbracket$$
$$\llbracket \mathsf{LTL} \rrbracket \cap \mathsf{coSAFETY} \quad = \quad \llbracket \mathsf{coSafety\text{-}LTL} \rrbracket \qquad\qquad \llbracket \mathsf{LTL} \rrbracket^{<\omega}$$
Chang *et al.* [5]

Lemmas 6 and 7   Lemma 1   Lemma 7

$$\llbracket \mathsf{coSafety\text{-}LTL} \rrbracket^{<\omega} \cdot (2^{\Sigma})^{\omega} = \llbracket \mathsf{coSafety\text{-}LTL}(-\widetilde{\mathsf{X}}) \rrbracket^{<\omega} \cdot (2^{\Sigma})^{\omega} \qquad \llbracket \mathsf{coSafety\text{-}LTL} \rrbracket^{<\omega} \supsetneq \llbracket \mathsf{coSafety\text{-}LTL}(-\widetilde{\mathsf{X}}) \rrbracket^{<\omega}$$
Theorem 4

Lemma 8   Lemma 2 Corollary 1   Kamp   Corollary 1

$$\llbracket \mathsf{coSafety\text{-}FO} \rrbracket \qquad\qquad\qquad \llbracket \mathsf{FO} \rrbracket^{<\omega} \quad \supsetneq \quad \llbracket \mathsf{coSafety\text{-}FO} \rrbracket^{<\omega}$$
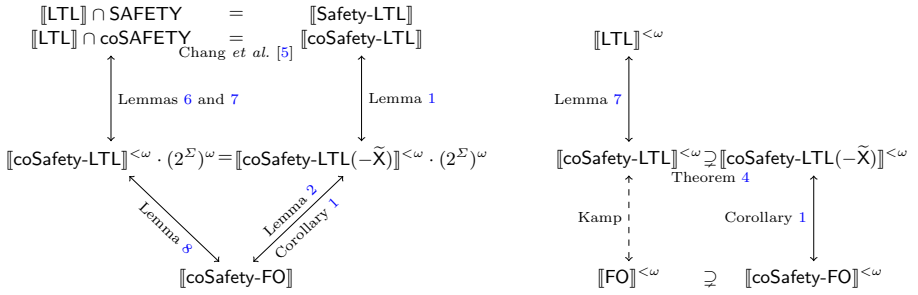
Fig. 1. Summary of the results of the paper, about languages over infinite words on the left, and over finite words on the right. Solid arrows are own results. Dashed arrows are known from literature.

Definition 5 (coSafety-FO). *The logic* coSafety-FO *is generated by the following grammar:*

$$atomic := x < y \mid x = y \mid x \neq y \mid P(x) \mid \neg P(x)$$
$$\phi := atomic \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \exists y(x < y \wedge \phi_1) \mid \forall y(x < y < z \rightarrow \phi_1)$$

where x, y, *and* z *are first-order variables,* P *is a unary predicate, and* ϕ_1 *and* ϕ_2 *are* coSafety-FO *formulas.*

We need to make a few observations on the syntax of the two fragments. First of all, note how any formula of Safety-FO is the negation of a formula of coSafety-FO and *vice versa*. Then, note that the two fragments are defined in *negated normal form, i.e.*, negation only appears on atomic formulas. The particular kind of existential and universal quantifications allowed are the culprit of these fragments. In particular Safety-FO restricts any existentially quantified variable to be bounded between two already quantified variables. The same applies to universal quantification in coSafety-FO. Moreover Safety-FO and coSafety-FO formulas are *future formulas, i.e.*, the quantifiers can only range over values *greater* than already quantified variables. These two features are essential to precisely capture Safety-LTL and coSafety-LTL. Finally, note that the comparisons in the guards of the quantifiers are strict, but non-strict comparisons can be used as well. In particular, $\exists y(x \leq y \wedge \phi)$ can be rewritten as $\phi[y/x] \vee \exists y(x < y \wedge \phi)$, where $\phi[y/x]$ is the formula obtained by replacing all occurrences of y with x. Similarly, $\forall z(x \leq z \leq y \rightarrow \phi)$ can be rewritten as $\phi[z/x] \wedge \phi[z/y] \wedge \forall z(x < z < y \rightarrow \phi)$.

To prove the relationship between Safety-LTL, coSafety-LTL, and these fragments, we focus now on coSafety-FO. By duality, all the results transfer to Safety-FO. We focus on coSafety-FO because the unbounded quantification is existential, and it is easier to reason about the existence of prefixes than on all the prefixes at once. We start by observing that, since the *weak tomorrow* operator, over infinite words, coincides with the *tomorrow* operator, the following holds.

Observation 1. $\llbracket \mathsf{coSafety\text{-}LTL} \rrbracket = \llbracket \mathsf{coSafety\text{-}LTL}(-\widetilde{\mathsf{X}}) \rrbracket$

When reasoning over finite words, the *weak tomorrow* operator plays a crucial role, since it can be used to recognize when we are at the last position of a word. In fact, the formula $\sigma, i \models \widetilde{X}\bot$ is true if and only if $i = |\sigma| - 1$, for any $\sigma \in (2^\Sigma)^*$.

Now, let us note that, thanks to the absence of the *weak tomorrow* operator, we can in some sense reduce ourselves to reasoning over finite words.

Lemma 1. $[\![\text{coSafety-LTL}(-\widetilde{X})]\!] = [\![\text{coSafety-LTL}(-\widetilde{X})]\!]^{<\omega} \cdot (2^\Sigma)^\omega$

Proof. We have to prove that, for each formula $\phi \in \text{coSafety-LTL}(-\widetilde{X})$, it holds that:

$$\mathcal{L}(\phi) = \mathcal{L}^{<\omega}(\phi) \cdot (2^\Sigma)^\omega$$

We proceed by induction on the structure of ϕ. For the base case, consider $\phi \equiv p \in \Sigma$. The case for $\phi \equiv \neg p$ is similar. Let $\sigma \in \mathcal{L}(p)$. It holds that $\sigma_0 \models p$ and $\sigma_0 \cdot \sigma' \models p$, for all $\sigma' \models (2^\Sigma)^\omega$, and in particular for $\sigma' = \sigma_{[1,\infty)}$. This is equivalent to say that $\sigma \in \mathcal{L}^{<\omega}(\phi) \cdot (2^\Sigma)^\omega$. For the inductive step:

1. Let $\phi \equiv \phi_1 \wedge \phi_2$. Suppose that $\sigma \in \mathcal{L}(\phi)$. Obviously, $\sigma \models \phi_1$ and $\sigma \models \phi_2$, and therefore $\sigma \in \mathcal{L}(\phi_1)$ and $\sigma \in \mathcal{L}(\phi_2)$. By the inductive hypothesis, $\sigma \in \mathcal{L}^{<\omega}(\phi_1) \cdot (2^\Sigma)^\omega$ and $\sigma \in \mathcal{L}^{<\omega}(\phi_2) \cdot (2^\Sigma)^\omega$. This means that there exist two indices $i, j \in \mathbb{N}$ such that $\sigma_{[0,i]} \models \phi_1$ and $\sigma_{[0,j]} \models \phi_2$. Let m be the greatest between i and j. It holds that $\sigma_{[0,m]} \models \phi_1 \wedge \phi_2$. Therefore $\sigma \in \mathcal{L}^{<\omega}(\phi_1 \wedge \phi_2) \cdot (2^\Sigma)^\omega$.

2. Let $\phi \equiv \phi_1 \vee \phi_2$ and let $\sigma \in \mathcal{L}(\phi)$. We have that $\sigma \models \phi_1$ or $\sigma \models \phi_2$. Without loss of generality, we consider the case that $\sigma \models \phi_1$ (the other case is specular). By the inductive hypothesis, $\sigma \in \mathcal{L}^{<\omega}(\phi_1) \cdot (2^\Sigma)^\omega$. Therefore, it also holds that $\sigma \in \mathcal{L}^{<\omega}(\phi_1 \vee \phi_2) \cdot (2^\Sigma)^\omega$.

3. Let $\phi \equiv X\phi_1$ and let $\sigma \in \mathcal{L}(X\phi_1)$. By the semantics of the *tomorrow* operator, it holds that $\sigma_{[1,\infty)} \models \phi_1$. By the inductive hypothesis, $\sigma_{[1,\infty)} \in \mathcal{L}^{<\omega}(\phi_1) \cdot (2^\Sigma)^\omega$. This means that there exists an index $i \geq 1$ such that $\sigma_{[1,i]} \models \phi_1$. Therefore, it also holds that the state sequence $\sigma_{[0,i]} = \sigma_0 \cdot \sigma_{[1,i]}$ satisfies $X\phi_1$ over finite words, that is, $\sigma_{[0,i]} \models X\phi_1$. This means that $\sigma \in \mathcal{L}^{<\omega}(X\phi_1) \cdot (2^\Sigma)^\omega$.

4. Let $\phi \equiv \phi_1 \mathcal{U} \phi_2$. Let $\sigma \in \mathcal{L}(\phi)$. By the semantics of the *until* operator, it holds that there exists an index $i \in \mathbb{N}$ such that $\sigma_{[i,\infty)} \models \phi_2$ and $\sigma_{[j,\infty)} \models \phi_1$ for all $0 \leq j < i$. By the inductive hypothesis, we have that $\sigma_{[i,\infty)} \in \mathcal{L}^{<\omega}(\phi_2) \cdot (2^\Sigma)^\omega$ and $\sigma_{[j,\infty)} \in \mathcal{L}^{<\omega}(\phi_1) \cdot (2^\Sigma)^\omega$ for all $0 \leq j < i$. This means that there exists an index $i \in \mathbb{N}$ and $i + 1$ indices $k_0, \ldots, k_i \in \mathbb{N}$ such that $\sigma_{[i,k_i]} \models \phi_2$ and $\sigma_{[j,k_j]} \models \phi_1$ for all $0 \leq j < i$. Let m be the greatest between $k_0, \ldots, k_i$. It holds that there exists an index $i \in \mathbb{N}$ such that $\sigma_{[i,m]} \models \phi_2$ and $\sigma_{[j,m]} \models \phi_1$ for all $0 \leq j < i$. Therefore, $\sigma \in \mathcal{L}^{<\omega}(\phi_1 \mathcal{U} \phi_2) \cdot (2^\Sigma)^\omega$.

The same property applies to coSafety-FO as well.

Lemma 2. $[\![\text{coSafety-FO}]\!] = [\![\text{coSafety-FO}]\!]^{<\omega} \cdot (2^\Sigma)^\omega$

Proof. We have to prove that, for each formula $\psi \in \text{coSafety-FO}$ with one free variable, it holds that $\mathcal{L}(\psi) = \mathcal{L}^{<\omega}(\psi) \cdot (2^\Sigma)^\omega$. We proceed by induction,

but with a more general statement. Let $\phi(x_1,\ldots,x_k)$ have k free variables. We prove by induction on ϕ that for any infinite state sequence σ such that $(\sigma)^s, n_1,\ldots,n_k \models \phi(x_1,\ldots,x_k)$, there exists a prefix $\sigma_{[0,i]}$ of σ such that for all $\sigma' \in (2^\Sigma)^\omega$, $(\sigma_{[0,i]}\sigma')^s, n_1,\ldots,n_k \models \phi(x_1,\ldots,x_k)$. The base case considers the four kinds of atomic formulas. If $(\sigma)^s, n_1, n_2 \models x_1 < x_2$, then $n_1 < n_2$ and we know that $(\sigma_{[0,n_2]}\sigma')^s, n_1, n_2 \models x_1 < x_2$ for all $\sigma' \in (2^\Sigma)^*$. The case of $x_1 = x_2$ is similar. Now, if $(\sigma)^s, n_1 \models P(x_1)$, then $p \in \sigma_{n_1}$ and we know that $(\sigma_{[0,n_1]}\sigma')^s, n_1 \models P(x_1)$ for all $\sigma' \in (2^\Sigma)^*$. The case for $\neg P(x_1)$ is similar. For the inductive step:

1. if $(\sigma)^s, n_1,\ldots,n_k \models \phi_1(x_1,\ldots,x_k) \wedge \phi_2(x_1,\ldots,x_k)$, by the induction hypothesis we know that there are two prefixes $\sigma_{[0,i]}$ and $\sigma_{[0,j]}$ such that, respectively, $(\sigma_{[0,i]}\sigma')^s, n_1,\ldots,n_k \models \phi_1(x_1,\ldots,x_k)$ and $(\sigma_{[0,j]}\sigma'')^s, n_1,\ldots,n_k \models \phi_2(x_1,\ldots,x_k)$, for all $\sigma', \sigma'' \in (2^\Sigma)^*$. Then, supposing $w.l.o.g.$ that $i \leq j$, we know that $(\sigma_{[0,j]}\sigma''), n_1,\ldots,n_k \models \phi_1(x_1,\ldots,x_k) \wedge \phi_2(x_1,\ldots,x_k)$. The case for $\phi_1(x_1,\ldots,x_k) \vee \phi_2(x_1,\ldots,x_k)$ is similar.

2. If $(\sigma)^s, n_1,\ldots,n_k \models \exists x_{k+1}(x_u < x_{k+1} \wedge \phi_1(x_1,\ldots,x_{k+1}))$ for some $1 \leq u \leq k$, then there exists an $n_{k+1} > n_u$ such that $(\sigma)^s, n_1,\ldots,n_{k+1} \models \phi_1(x_1,\ldots,x_{k+1})$. This implies that $(\sigma_{[0,i]}\sigma'), n_1,\ldots,n_{k+1} \models \phi_1(x_1,\ldots,x_{k+1})$ for some $i \geq 0$ and all $\sigma' \in (2^\Sigma)^*$, by the induction hypothesis. It follows that $(\sigma_{[0,i]}\sigma'), n_1,\ldots,n_k \models \exists x_{k+1}(x_i < x_{k+1} \wedge \phi_1(x_1,\ldots,x_{k+1}))$.

3. if $(\sigma)^s, n_1,\ldots,n_k \models \forall x_{k+1}(x_u < x_{k+1} < x_v \rightarrow \phi_1(x_1,\ldots,x_{k+1}))$ for some $1 \leq u,v \leq k$, then for all n_{k+1} with $n_u < n_{k+1} < n_v$ it holds that $(\sigma)^s, n_1 \ldots,n_{k+1} \models \phi_1(x_1,\ldots,x_{k+1})$. Then, for the induction hypothesis, for all n_{k+1} with $n_u < n_{k+1} < n_v$ there is a prefix $\sigma_{[0,i_{n_{k+1}}]}$ such that $(\sigma_{[0,i_{n_{k+1}}]}\sigma')^s, n_1,\ldots,n_{k+1} \models \phi_1(x_1,\ldots,x_{k+1})$ for all $\sigma' \in (2^\Sigma)^*$. Then, if $n_* = \max_{n_u < n_{k+1} < n_v}(i_{n_{k+1}})$, it holds that:

$$(\sigma_{[0,n_*]}\sigma')^s, n_1,\ldots,n_k \models \forall x_{k+1}(x_u < x_{k+1} < x_v \rightarrow \phi_1(x_1,\ldots,x_{k+1}))$$

Now, let $\psi(x)$ be a coSafety-FO formula with exactly one free variable x. Thanks to the above induction we can conclude that each infinite state sequence σ such that $(\sigma)^s, 0 \models \phi(x)$ is of the form $\sigma_{[0,i]} \cdot \sigma'$, where $(\sigma_{[0,i]})^s \models \phi(x)$, and this implies that $\mathcal{L}(\psi) = \mathcal{L}^{<\omega}(\psi) \cdot (2^\Sigma)^\omega$.

It is worth to note that Lemmas 1 and 2 show that coSafety-LTL($-\widetilde{X}$) and coSafety-FO are *insensitive to infiniteness* as defined by De Giacomo *et al.* [9]. Then, we can focus on coSafety-LTL($-\widetilde{X}$) and coSafety-FO on finite words. If we can prove that $[\![\text{coSafety-LTL}(-\widetilde{X})]\!]^{<\omega} = [\![\text{coSafety-FO}]\!]^{<\omega}$, we are done. At first, we show how to encode coSafety-LTL($-\widetilde{X}$) formulas into coSafety-FO.

Lemma 3. $[\![\text{coSafety-LTL}(-\widetilde{X})]\!]^{<\omega} \subseteq [\![\text{coSafety-FO}]\!]^{<\omega}$

Proof. Let $\mathcal{L} \in [\![\text{coSafety-LTL}(-\widetilde{X})]\!]^{<\omega}$, and let $\phi \in \text{coSafety-LTL}(-\widetilde{X})$ such that $\mathcal{L} = \mathcal{L}^{<\omega}(\phi)$. By following the semantics of the operators in ϕ, we can obtain an equivalent coSafety-FO formula ϕ_{FO}. We inductively define the formula $FO(\phi, x)$, where x is a variable, as follows:

- $FO(p, x) = P(x)$, for each $p \in \Sigma$
- $FO(\neg p, x) = \neg P(x)$, for each $p \in \Sigma$
- $FO(\phi_1 \wedge \phi_2, x) = FO(\phi_1, x) \wedge FO(\phi_2, x)$
- $FO(\phi_1 \vee \phi_2, x) = FO(\phi_1, x) \vee FO(\phi_2, x)$
- $FO(\mathsf{X}\phi_1, x) = \exists y(x < y \wedge y = x + 1 \wedge FO(\phi_1, y))$
  where $y = x + 1$ can be expressed as $\forall z(x < z < y \rightarrow \bot)$.
- $FO(\phi_1 \, \mathcal{U} \, \phi_2, x) = \exists y(x \leq y \wedge FO(\phi_2, y) \wedge \forall z(x \leq z < y \rightarrow FO(\phi_1, z)))$

For each $\phi \in \mathsf{coSafety\text{-}LTL}(-\widetilde{\mathsf{X}})$, the formula $FO(\phi, x)$ has exactly one free variable x. It is easy to see that for all finite state sequences $\sigma \in (2^\Sigma)^*$, it holds that $\sigma \models \phi$ if and only if $(\sigma)^s, 0 \models FO(\phi, x)$, and $FO(\phi, x) \in \mathsf{coSafety\text{-}FO}$. Therefore, $\mathcal{L} \in [\![\mathsf{coSafety\text{-}FO}]\!]^{<\omega}$.

It is time to show the opposite direction, *i.e.*, that any $\mathsf{coSafety\text{-}FO}$ formula can be translated into a $\mathsf{coSafety\text{-}LTL}(-\widetilde{\mathsf{X}})$ formula which is equivalent over finite words. To prove this fact we adapt a proof of Kamp's theorem by Rabinovich [15]. Kamp's theorem is one of the fundamental results about temporal logics, which states that LTL corresponds to FO in terms of expressiveness. Here, we prove a similar result in the context of co-safety languages. The proof goes by introducing a *normal form* for FO formulas, and showing that (i) any $\mathsf{coSafety\text{-}FO}$ formula can be translated into such normal form and (ii) any formula in normal form can be straightforwardly translated into a $\mathsf{coSafety\text{-}LTL}(-\widetilde{\mathsf{X}})$ formula. We start by introducing such a normal form.

Definition 6 ($\exists\forall$-formulas). *An $\exists\forall$-formula $\phi(z_0, \ldots, z_m)$ with m free variables is a formula of this form:*

$$
\phi(z_0, \ldots, z_m) := \exists x_0 \ldots \exists x_n \big(
$$

$$
x_0 < x_1 < \cdots < x_n \qquad\qquad \textit{ordering constraints}
$$

$$
\wedge \, z_0 = x_0 \wedge \bigwedge_{k=1}^{m} (z_k = x_{i_k}) \qquad \textit{binding constraints}
$$

$$
\wedge \bigwedge_{j=0}^{n} \alpha_j(x_j) \qquad\qquad\qquad \textit{punctual constraints}
$$

$$
\wedge \bigwedge_{j=1}^{n} \forall y(x_{j-1} < y < x_j \rightarrow \beta_j(y))) \quad \textit{interval constraints}
$$

where $i_k \in \{0, \ldots, n\}$ for each $0 \leq k \leq m$, and α_j and β_j, for each $1 \leq j \leq n$, are quantifier-free formulas with exactly one free variable.

Some explanations are due. Each $\exists\forall$-formula states a number of requirements for its free variables and for its quantified variables. Through the binding constraints, the free variables are identified with a subset of the quantified variables in order to uniformly state the punctual and interval constraints, and the ordering constraints which sort all the variable in a total order. Note that there is no relationship between n and m: there might be more quantified variables

than free variables, or less. Note as well that the binding constraint $z_0 = x_0$ is always present, *i.e.*, at least one free variable has to be the minimal element of the ordering. This ensures that $\exists\forall$-formulas are always *future* formulas.

We say that a formula of coSafety-FO is in *normal form* if and only if it is a disjunction of $\exists\forall$-formulas. To see how formulas in normal form make sense, let us immediately show how to translate them into coSafety-LTL$(-\widetilde{X})$ formulas.

Lemma 4. *For any formula $\phi(z) \in$ coSafety-FO in normal form, with a single free variable, there exists a formula $\psi \in$ coSafety-LTL$(-\widetilde{X})$ such that $\mathcal{L}^{<\omega}(\phi(z)) = \mathcal{L}^{<\omega}(\psi)$.*

Proof. We show how any $\exists\forall$-formula is equivalent to an coSafety-LTL$(-\widetilde{X})$-formula, over finite words. Since each formula in normal form is a disjunction of $\exists\forall$-formulas, and since coSafety-LTL$(-\widetilde{X})$ is closed under disjunction, this implies the proposition. Let $\phi(z)$ be a $\exists\forall$-formula with a single free variable. Having only one free variable, $\phi(z)$ is of the form:

$$\exists x_0 \ldots \exists x_n \big(x_0 < \cdots < x_n \wedge z = x_0$$
$$\wedge \bigwedge_{j=0}^{n} \alpha_j(x_j) \wedge \bigwedge_{j=1}^{n} \forall y(x_{j-1} < y < x_j \rightarrow \beta_j(y)))$$

Now, let A_i be the temporal formulas corresponding to α_i and B_i be the ones corresponding to β_i. Recall that α_i and β_i are quantifier free with only one free variable, hence this correspondence is trivial. Since z is the first time point of the ordering mandated by the formula, we only need future temporal operators to encode ϕ into a coSafety-LTL$(-\widetilde{X})$ formula ψ defined as follows:

$$\psi \equiv A_0 \wedge X(B_0 \, \mathcal{U} \, (A_1 \wedge X(B_1 \, \mathcal{U} \, A_2 \wedge \ldots X(B_{n-1} \, \mathcal{U} \, A_n) \ldots)))$$

It can be seen that $\sigma, k \models \psi$ if and only if $(\sigma)^s, k \models \phi(z)$, for each $\sigma \in (2^\Sigma)^+$ and each $k \geq 0$. Thus, $\mathcal{L}^{<\omega}(\phi(z)) = \mathcal{L}^{<\omega}(\psi)$.

Two differences between our $\exists\forall$-formulas and those used by Rabinovich [15] are crucial: first, we do not have unbounded universal requirements, but all interval constraints use bounded quantifications, hence we do not need the *always* operator to encode them; second, our $\exists\forall$-formulas are *future* formulas, hence we only need future operators to encode them.

We now show that any coSafety-FO formula can be translated into normal form, that is, into a *disjunction* of $\exists\forall$-formulas.

Lemma 5. *Any coSafety-FO formula is equivalent to a disjunction of $\exists\forall$-formulas.*

Proof. Let ϕ be a coSafety-FO formula. We proceed by structural induction on ϕ. For the base case, for each atomic formula $\phi(z_0, z_1)$ we provide an equivalent $\exists\forall$-formula $\psi(z_0, z_1)$:

1. if $\phi \equiv z_0 < z_1$ then $\psi \equiv \exists x_0 \exists x_1(z_0 = x_0 \wedge z_1 = x_1 \wedge x_0 < x_1)$;
2. if $\phi \equiv z_0 = z_1$, then $\psi \equiv \exists x_0(z_0 = x_0 \wedge z_1 = x_0)$.

3. if $\phi \equiv z_0 \neq z_1$, we can note that $\phi \equiv z_0 < z_1 \vee z_1 < z_0$ and then apply Item 1;
4. If $\phi \equiv P(z_0)$ then we define $\psi := \exists x_0 (z_0 = x_0 \wedge P(x_0))$. Similarly if $\phi \equiv \neg P(z_0)$.

For the inductive step:

1. The case of a disjunction is trivial.
2. If $\phi(z_0, \ldots, z_k)$ is a conjunction, by the inductive hypothesis each conjunct is equivalent to a disjunction of $\exists\forall$-formulas. By distributing the conjunction over the disjunction we can reduce ourselves to the case of a conjunction $\psi_1(z_0, \ldots, z_k) \wedge \psi_2(z_0, \ldots, z_k)$ of two $\exists\forall$-formulas. In this case we have that:

$$\psi_1 \equiv \exists x_0 \ldots \exists x_n \big(x_0 < \cdots < x_n \wedge z_0 = x_0 \wedge \ldots\big)$$
$$\psi_2 \equiv \exists x_{n+1} \ldots \exists x_m (x_{n+1} < \cdots < x_m \wedge z_0 = x_{n+1} \wedge \ldots)$$

Since the set of quantified variables in ψ_1 is disjoint from the set of quantified variables in ψ_2, we can distribute the existential quantifiers over the conjunction $\psi_1 \wedge \psi_2$, obtaining:

$$\psi_1 \wedge \psi_2 \equiv \exists x_0 \ldots \exists x_n \exists x_{n+1} \ldots \exists x_m$$
$$\big(x_0 < \cdots < x_n \wedge x_{n+1} < \cdots < x_m \wedge z_0 = x_0 \wedge z_0 = x_{n+1} \wedge \ldots\big)$$

Note that we can identify x_0 and x_{n+1}, obtaining:

$$\psi_1 \wedge \psi_2 \equiv \exists x_0 \ldots \exists x_n \exists x_{n+2}, \ldots \exists x_m$$
$$\big(x_0 < \cdots < x_n \wedge x_0 < x_{n+2} < \cdots < x_m \wedge$$
$$z_0 = x_0 \wedge \bigwedge_{i=1}^{k}(z_i = x_{j_i''}) \wedge \bigwedge_{i=0, i \neq n+1}^{m} \alpha_i(x_i) \wedge$$
$$\bigwedge_{\substack{i=1, i \neq n+1 \\ i \neq n+2}}^{m} \forall y(x_{i-1} < y < x_i \to \beta_i(y)) \wedge \forall y(x_0 < y < x_{n+2} \to \beta_{n+2}))$$

Now, to turn this formula into a disjunction of $\exists\forall$-formulas, we consider all the possible interleavings of the variables that respect the two imposed orderings and explode the formula into a disjunction that consider each such interleaving. Let $X = \{x_0, \ldots, x_n, x_{n+2}, \ldots, x_m\}$ and let Π be the set of all the permutations of X compatible with the orderings $x_0 < \cdots < x_n$ and $x_0 < x_{n+1} < \cdots < x_m$. For any $\pi \in \Pi$, $\pi(0) = 0$. Now, $\psi_1 \wedge \psi_2$ becomes the disjunction of a set of $\exists\forall$-formulas ψ_π, for each $\pi \in \Pi$, defined as:

$$\psi_\pi \equiv \exists x_{\pi(0)} \ldots \exists x_{\pi(m)}$$
$$\big(x_{\pi(0)} < \cdots < x_{\pi(m)} \wedge$$
$$z_0 = x_0 \wedge \bigwedge_{i=1}^{k}(z_i = x_{\pi(j_i'')}) \wedge \bigwedge_{i=0}^{m} \alpha_i(x_i) \wedge$$
$$\bigwedge_{i=0}^{m} \forall y(x_{\pi(i-1)} < y < x_{\pi(i)} \to \beta_i^*(y)))$$

where β_i^* suitably combines the formulas β according to the interleaving of the orderings of the original variables, and is defined as follows:

$$\beta_i^* = \begin{cases} \beta_{\pi(i)} & \text{if both } \pi(i), \pi(i-1) \leq n \text{ or both } \pi(i), \pi(i-1) > n \\ \beta_{\pi(i)} \wedge \beta_{\pi(i-1)} & \text{if } \pi(i) \leq n \text{ and } \pi(i-1) > n \text{ or } \textit{vice versa} \end{cases}$$

Then we have that $\psi_1 \wedge \psi_2 \equiv \bigvee_{\pi \in \Pi}(\psi_\pi)$, which is a disjunction of $\exists\forall$-formulas.

3. Let $\phi(z_0, \ldots, z_m) \equiv \exists z_{m+1} \, . \, (z_i < z_{m+1} \wedge \phi_1(z_0, \ldots, z_m, z_{m+1}))$, for some $0 \leq i \leq m$. By the inductive hypothesis, this is equivalent to the formula $\exists z_{m+1}(z_i < z_{m+1} \wedge \bigvee_{k=0}^{j} \psi_k(z_0, \ldots, z_m, z_{m+1}))$, where $\psi_k(z_0, \ldots, z_m, z_{m+1})$ is a $\exists\forall$-formula, for each $0 \leq k \leq j$, that is:

$$\exists z_{m+1} \, . \, (z_i < z_{m+1} \wedge \bigvee_{k=0}^{j} (\exists x_0 \ldots \exists x_{n_k} \psi_k'(z_0, \ldots, z_{m+1}, x_0, \ldots, x_{n_k})))$$

By distributing the conjunction over the disjunction, we obtain:

$$\exists z_{m+1} \, . \, (\bigvee_{k=0}^{j} ((z_i < z_{m+1}) \wedge \exists x_0 \ldots \exists x_{n_k} \psi_k'(z_0, \ldots, z_{m+1}, x_0, \ldots, x_{n_k})))$$

and by distributing the existential quantifier over the disjunction, we have:

$$\bigvee_{k=0}^{j} (\exists z_{m+1}((z_i < z_{m+1}) \wedge \exists x_0 \ldots \exists x_{n_k} \psi_k'(z_0, \ldots, z_{m+1}, x_0, \ldots, x_{n_k})))$$

Since the subformula $z_i < z_{m+1}$ does not contain the variables $x_0, \ldots, x_n$, we can push it inside the existential quantification, obtaining:

$$\bigvee_{k=0}^{j} (\exists z_{m+1} \, . \, \exists x_0 \ldots \exists x_{n_k} \, . \, ((z_i < z_{m+1}) \wedge \psi_k'(z_0, \ldots, z_{m+1}, x_0, \ldots, x_{n_k})))$$

Now we divide in cases:

(a) suppose that the formula $\psi_k'(z_0, \ldots, z_{m+1}, x_0, \ldots, x_{n_k})$ contains the following conjuncts: $z_i = x_{l_i}$ and $z_{m+1} = x_{l_{m+1}}$, with $l_i = l_{m+1}$. It holds that these formulas are in contradiction with the formula $z_i < z_{m+1}$, that is:

$$(z_i < z_{m+1}) \wedge (z_i = x_{l_i}) \wedge (z_{m+1} = x_{l_{m+1}}) \equiv \bot$$

Therefore, the disjunct $(z_i < z_{m+1}) \wedge \psi_k'(z_0, \ldots, z_{m+1}, x_0, \ldots, x_{n_k})$ is equivalent to $\bot$, and thus can be safely removed from the disjunction.

(b) suppose that the formula $\psi_k'(z_0, \ldots, z_{m+1}, x_0, \ldots, x_{n_k})$ contains the following conjuncts: $z_i = x_{l_i}$, $z_{m+1} = x_{l_{m+1}}$ (with $l_i \neq l_{m+1}$), and $x_{l_{m+1}} < \cdots < x_{l_i}$. As in the previous case, it holds that:

$$(z_i < z_{m+1}) \wedge (z_i = x_{l_i}) \wedge (z_{m+1} = x_{l_{m+1}}) \wedge (x_{l_{m+1}} < \cdots < x_{l_i}) \equiv \bot$$

Thus, also in this case, this disjunct can be safely removed from the disjunction.

(c) otherwise, it holds that the formula $\psi_k'(z_0, \ldots, z_{m+1}, x_0, \ldots, x_{n_k})$ contains the following conjuncts: $z_i = x_{l_i}$, $z_{m+1} = x_{l_{m+1}}$ (with $l_i \neq l_{m+1}$), and $x_{l_i} < \cdots < x_{l_{m+1}}$. Therefore, the subformula $z_i < z_{m+1}$ is redundant, and can be safely removed from $\psi_k'(z_0, \ldots, z_{m+1}, x_0, \ldots, x_{n_k})$. The resulting formula is a $\exists\forall$-formula.

After the previous transformation, we obtain:

$$\bigvee_{k=0}^{j'} (\exists z_{m+1} . \exists x_0 \ldots \exists x_{n_k} . \psi_k''(z_0, \ldots, z_{m+1}, x_0, \ldots, x_{n_k}))$$

Finally, since each formula $\psi_k''(z_0, \ldots, z_{m+1}, x_0, \ldots, x_{n_k})$ contains the conjunct $z_{m+1} = x_{l_{m+1}}$, we can safely remove the quantifier $\exists z_{m+1}$. We obtain the formula:

$$\bigvee_{k=0}^{j'} (\exists x_0 \ldots \exists x_{n_k} . \psi_k''(z_0, \ldots, z_m, x_0, \ldots, x_{n_k}))$$

which is a disjunction of $\exists\forall$-formulas.

4. Let $\phi(z_0, \ldots, z_m) \equiv \forall z_{m+1}(z_i < z_{m+1} < z_j \to \phi_1(z_0, \ldots, z_m, z_{m+1}))$, for some $0 \leq i, j \leq m$. By the induction hypothesis we know that ϕ_1 is equivalent to a disjunction $\bigvee_k \psi_k$ where ψ_k are $\exists\forall$-formulas, i.e., each ψ_k is of the form:

$$\psi_k \equiv \exists x_0, \ldots, x_n \Big(x_0 < \ldots < x_n \wedge z_0 = x_0 \wedge \bigwedge_{l=1}^{m+1} (z_l = x_{u_l}) \wedge$$
$$\bigwedge_{l=0}^{n} \alpha_l(x_l) \wedge \bigwedge_{l=1}^{n} \forall y(x_{l-1} < y < x_l \to \beta_l(y)) \Big)$$

We now note that we can suppose *w.l.o.g.* that the ordering constraint and the binding constraint of ψ_k imply that z_i, z_{m+1} and z_j are ordered consecutively, i.e., $z_i < z_{m+1} < z_j$ with no other variable in between. That is because otherwise the constraints would be in conflict with the guard of the universal quantification and the disjunct could be removed from the disjunction. Take for example a disjunct of ψ_k with an ordering constraint of the type $z_i < z_h < z_{m+1}$, for some h. The existence of such a z_h is not guaranteed for each z_{m+1} between z_i and z_j because when $z_{m+1} = z_i + 1$ there is no value between z_i and $z_i + 1$ (we are on discrete time models). That said, we can now isolate all the parts of ψ_k that talk about z_{m+1}, bringing them out of the existential quantification, obtaining $\psi_k \equiv \theta_k \wedge \eta_k$, where:

$$\theta_k \equiv z_i < z_{m+1} < z_j$$
$$\wedge \, \alpha(z_{m+1}) \wedge \forall y(z_i < y < z_{m+1} \to \beta(y)) \wedge \forall y(z_{m+1} < y < z_i \to \beta'(y))$$

$$\eta_k \equiv \exists x_0, \ldots, x_n \big(x_0 < \ldots < x_n \wedge z_0 = x_0 \wedge \bigwedge_{l=1}^{m} (z_l = x_{u_l}) \wedge$$

$$\bigwedge_{\substack{l=0 \\ l \neq u_{m+1}}}^{n} \alpha_l(x_l) \wedge \bigwedge_{\substack{l=1 \\ l-1 \neq u_i \\ l \neq u_j}}^{n} \forall y (x_{l-1} < y < x_l \rightarrow \beta_l(y)) \big) \big)$$

Now, we have $\phi \equiv \forall z_{m+1}(z_i < z_{m+1} < z_j \rightarrow \bigvee_k (\theta_k \wedge \eta_k))$. We can distribute the head of the implication over the disjunction, and then over the conjunction, obtaining:

$$\phi \equiv \forall z_{m+1} \big(\bigvee_k ((z_i < z_{m+1} < z_j \rightarrow \theta_k) \wedge (z_i < z_{m+1} < z_j \rightarrow \eta_k)) \big)$$

In order to simplify the exposition, we now show how to proceed in the case of two disjuncts, which is easily generalizable. So suppose we have:

$$\phi \equiv \forall z_{m+1} \left(\bigvee \begin{array}{c} (z_i < z_{m+1} < z_j \rightarrow \theta_1) \wedge (z_i < z_{m+1} < z_j \rightarrow \eta_1) \\ (z_i < z_{m+1} < z_j \rightarrow \theta_2) \wedge (z_i < z_{m+1} < z_j \rightarrow \eta_2) \end{array} \right)$$

Now we can a) distribute the disjunction over the conjunction (*i.e.*, convert in conjunctive normal form in the case of multiple disjuncts), b) factor out the head of the implications and c) distribute the universal quantification over the conjunction, obtaining:

$$\phi \equiv \left(\begin{array}{c} \forall z_{m+1}(z_i < z_{m+1} < z_j \rightarrow \theta_1 \vee \theta_2) \\ \wedge \forall z_{m+1}(z_i < z_{m+1} < z_j \rightarrow \theta_1 \vee \eta_2) \\ \wedge \forall z_{m+1}(z_i < z_{m+1} < z_j \rightarrow \eta_1 \vee \theta_2) \\ \wedge \forall z_{m+1}(z_i < z_{m+1} < z_j \rightarrow \eta_1 \vee \eta_2) \end{array} \right)$$

Now, note that η_1 and η_2 do not contain z_{m+1} as a free variable, because we factored out all the parts mentioning z_{m+1} into θ_1 and θ_2 before. Therefore we can push them out from the universal quantifications, obtaining:

$$\phi \equiv \left(\begin{array}{c} \forall z_{m+1}(z_i < z_{m+1} < z_j \rightarrow \theta_1 \vee \theta_2) \\ \wedge \forall z_{m+1}(z_i < z_{m+1} < z_j \rightarrow \theta_1) \vee \eta_2 \\ \wedge \forall z_{m+1}(z_i < z_{m+1} < z_j \rightarrow \theta_2) \vee \eta_1 \\ \wedge \neg \exists z_{m+1}(z_i < z_{m+1} < z_j) \vee \eta_1 \vee \eta_2 \end{array} \right)$$

Now, note that $\neg \exists z_{m+1}(z_i < z_{m+1} < z_j)$ is equivalent to $z_i = z_j \vee z_j = z_i + 1$, which is the disjunction of two formulas that can be turned into $\exists \forall$-formulas. Since both η_1 and η_2 are already $\exists \forall$-formulas and since we already know how to deal with conjunctions and disjunctions of $\exists \forall$-formulas, it remains to show that the universal quantifications in the formula above can be turned into

$\exists\forall$-formulas. Take $\forall z_{m+1}(z_i < z_{m+1} < z_j \rightarrow \theta_1)$, *i.e.*:

$$\forall z_{m+1} \left(z_i < z_{m+1} < z_j \rightarrow \begin{matrix} z_i < z_{m+1} < z_j \\ \wedge\, \alpha(z_{m+1}) \\ \wedge\, \forall y(z_i < y < z_{m+1} \rightarrow \beta(y)) \\ \wedge\, \forall y(z_{m+1} < y < z_j \rightarrow \beta'(y)) \end{matrix} \right)$$

Note that the first conjunct of the consequent can be removed, since it is redundant. Now, this formula is requesting $\beta(y)$ for all y between z_i and z_{m+1}, but with z_{m+1} that ranges between z_i and $z_j - 1$, hence effectively requesting $\beta(y)$ to hold between z_i and z_j. Similarly for $\beta'(y)$, which has to hold for all y between $z_i + 1$ and z_j.

Hence, it is equivalent to:

$$
\begin{aligned}
& z_i = z_j \\
\vee\ & z_j = z_i + 1 \\
\vee\ & \exists x_{i+1}(z_i < x_{i+1} \wedge x_{i+1} = z_i + 1 \wedge z_j = x_{i+1} + 1 \wedge \alpha(x_{i+1})) \\
\vee\ & \exists x_i \exists x_{i+1} \exists x_{j-1} \exists x_j \left(\begin{matrix} x_i < x_{i+1} < x_{j-1} < x_j \\ \wedge\, z_i = x_i \wedge z_j = x_j \\ \wedge\, \alpha(x_{i+1}) \wedge \alpha(x_{j-1}) \\ \wedge\, \forall y(x_i < y < x_{i+1} \rightarrow \bot) \\ \wedge\, \forall y(x_{j-1} < y < x_j \rightarrow \bot) \\ \wedge\, \forall y(x_i < y < x_{j-1} \rightarrow \alpha(y) \wedge \beta(y)) \\ \wedge\, \forall y(x_{i+1} < y < x_j \rightarrow \alpha(y) \wedge \beta'(y)) \end{matrix} \right)
\end{aligned}
$$

which is a disjunction of a $\exists\forall$-formula and others that can be turned into disjunctions of $\exists\forall$-formulas. The reasoning is at all similar for $\forall z_{m+1}(z_i < z_{m+1} < z_j \rightarrow \theta_1 \vee \theta_2)$.

Any coSafety-FO formula can be translated into a disjunction of $\exists\forall$-formulas by Lemma 5, and then to a coSafety-LTL($-\widetilde{X}$) formula by Lemma 4. Together with Lemma 3, we obtain the following.

Corollary 1. $\llbracket \text{coSafety-FO} \rrbracket^{<\omega} = \llbracket \text{coSafety-LTL}(-\widetilde{X}) \rrbracket^{<\omega}$

We are now ready to state the main result of this section.

Theorem 1. $\llbracket \text{coSafety-LTL} \rrbracket = \llbracket \text{coSafety-FO} \rrbracket$

Proof. We know that $\llbracket \text{coSafety-LTL} \rrbracket = \llbracket \text{coSafety-LTL}(-\widetilde{X}) \rrbracket^{<\omega} \cdot (2^\Sigma)^\omega$ by Observation 1 and Lemma 1. Since $\llbracket \text{coSafety-LTL}(-\widetilde{X}) \rrbracket^{<\omega} = \llbracket \text{coSafety-FO} \rrbracket^{<\omega}$ by Corollary 1, we have that $\llbracket \text{coSafety-LTL}(-\widetilde{X}) \rrbracket^{<\omega} \cdot (2^\Sigma)^\omega = \llbracket \text{coSafety-FO} \rrbracket^{<\omega} \cdot (2^\Sigma)^\omega$. Then, by Lemma 2 we have that $\llbracket \text{coSafety-FO} \rrbracket^{<\omega} \cdot (2^\Sigma)^\omega = \llbracket \text{coSafety-FO} \rrbracket$, hence $\llbracket \text{coSafety-LTL} \rrbracket = \llbracket \text{coSafety-FO} \rrbracket$.

Corollary 2. $\llbracket \text{Safety-LTL} \rrbracket = \llbracket \text{Safety-FO} \rrbracket$

4   Safety-FO captures LTL-definable safety languages

In this section, we prove that coSafety-FO captures LTL-definable co-safety languages. By duality, we have that Safety-FO captures LTL-definable safety languages, and by the equivalence shown in the previous Section, this provides a novel proof of the fact that Safety-LTL captures LTL-definable safety languages. We start by characterizing co-safety languages in terms of LTL over finite words.

Lemma 6. $[\![\mathsf{LTL}]\!] \cap \mathsf{coSAFETY} = [\![\mathsf{LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$

Proof. ($\subseteq$) By Proposition 2 we know that each language $\mathcal{L} \in [\![\mathsf{LTL}]\!] \cap \mathsf{coSAFETY}$ is definable by a formula of the form $\mathsf{F}\alpha$ where $\alpha \in \mathsf{LTL_P}$. Hence for each $\sigma \in \mathcal{L}$ there exists an n such that $\sigma, n \models \alpha$, hence $\sigma_{[0,n]}, n \models \alpha$. Note that $\sigma_{[n+1,\infty]}$ is unconstrained. By replacing all the *since/yesterday/weak yesterday* operators in α with *until/tomorrow/weak tomorrow* operators, we obtain an LTL formula α^r such that $(\sigma_{[0,n]})^r, 0 \models \alpha^r$ (where σ^r is the reverse of σ). Since LTL captures star-free languages [12] and star-free languages are closed by reversal, there is also an LTL formula β such that $\sigma_{[0,n]}, 0 \models \beta$. Hence $\mathcal{L} = \mathcal{L}^{<\omega}(\beta) \cdot (2^{\Sigma})^{\omega}$, and we proved that $[\![\mathsf{LTL}]\!] \cap \mathsf{coSAFETY} \subseteq [\![\mathsf{LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$.

($\supseteq$) Given $\mathcal{L} \in [\![\mathsf{LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$, we know $\mathcal{L} = \mathcal{L}^{<\omega}(\beta) \cdot (2^{\Sigma})^{\omega}$ for some LTL formula β. Hence, for each $\sigma \in \mathcal{L}$ there is an n such that $\sigma_{[0,n]}, 0 \models \beta$. Since LTL captures star-free languages and star-free languages are closed by reversal, there is an LTL formula α^r such that $(\sigma_{[0,n]})^r, 0 \models \alpha^r$. Now, by replacing all the *until/tomorrow/weak tomorrow* operators in α^r with *since/yesterday/weak yesterday* operators, we obtain an LTL_P formula α such that $\sigma_{[0,n]}, n \models \alpha$. Hence, σ is such that there is an n such that $\sigma, n \models \alpha$, *i.e.*, $\sigma \models \mathsf{F}\alpha$. Therefore, by Proposition 2, $\mathcal{L} \in [\![\mathsf{LTL}]\!] \cap \mathsf{coSAFETY}$, and this in turn implies that $[\![\mathsf{LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega} \subseteq [\![\mathsf{LTL}]\!] \cap \mathsf{coSAFETY}$.

Now, we show that, over finite words, universal temporal operators are unneeded.

Lemma 7. $[\![\mathsf{LTL}]\!]^{<\omega} = [\![\mathsf{Safety\text{-}LTL}]\!]^{<\omega} = [\![\mathsf{coSafety\text{-}LTL}]\!]^{<\omega}$

Proof. Since Safety-LTL and coSafety-LTL are fragments of LTL, we only need to show one direction, *i.e.*, that $[\![\mathsf{LTL}]\!]^{<\omega} \subseteq [\![\mathsf{Safety\text{-}LTL}]\!]^{<\omega}$ and $[\![\mathsf{LTL}]\!]^{<\omega} \subseteq [\![\mathsf{coSafety\text{-}LTL}]\!]^{<\omega}$. At first, we show that universal temporal operators are not needed over finite words. For each LTL formula ϕ, we can build an equivalent coSafety-LTL formula with only existential temporal operators. The *globally* operator can be replaced by means of an *until* operator whose existential part always refers to the last position of the word. In turn, this can be done with the formula $\widetilde{\mathsf{X}}\bot$, which is true only at the final position:

$$\mathsf{G}\phi \equiv \phi \,\mathcal{U}\, (\phi \wedge \widetilde{\mathsf{X}}\bot)$$

Similarly, the *release* operator can be expressed by means of a *globally* operator in disjunction with an *until* operator:

$$\phi_1 \,\mathcal{R}\, \phi_2 \equiv \mathsf{G}\phi_2 \vee (\phi_2 \,\mathcal{U}\, (\phi_1 \wedge \phi_2)) \equiv \big(\phi_2 \,\mathcal{U}\, (\phi_2 \wedge \widetilde{\mathsf{X}}\bot)\big) \vee \big(\phi_2 \,\mathcal{U}\, (\phi_1 \wedge \phi_2)\big)$$

Hence $[\![\mathsf{LTL}]\!]^{<\omega} = [\![\mathsf{coSafety\text{-}LTL}]\!]^{<\omega}$. Now, if we exploit the duality between the *eventually/until* and the *globally/release* operators, we obtain:

$$\mathsf{F}\phi \equiv \phi \,\mathcal{R}\, (\phi \vee \mathsf{X}\top)$$
$$\phi_1 \,\mathcal{U}\, \phi_2 \equiv \phi_2 \,\mathcal{R}\, (\phi_2 \vee \mathsf{X}\top) \wedge \phi_2 \,\mathcal{R}\, (\phi_1 \vee \phi_2)$$

Hence $[\![\mathsf{LTL}]\!]^{<\omega} = [\![\mathsf{Safety\text{-}LTL}]\!]^{<\omega}$.

Then, we relate coSafety-LTL on finite words and coSafety-FO.

Lemma 8. $[\![\mathsf{coSafety\text{-}LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega} = [\![\mathsf{coSafety\text{-}FO}]\!]$

Proof. ($\subseteq$) We have that $[\![\mathsf{coSafety\text{-}LTL}]\!]^{<\omega} = [\![\mathsf{LTL}]\!]^{<\omega}$ by Lemma 7, and this implies that $[\![\mathsf{coSafety\text{-}LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega} = [\![\mathsf{LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$, and $[\![\mathsf{coSafety\text{-}LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega} = [\![\mathsf{FO}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$ by Proposition 1. Now, let $\phi \in \mathsf{FO}$, and suppose *w.l.o.g.* that ϕ is in *negated normal form*. We define the formula $\phi'(x, y)$, where x and y are two fresh variables that do not occur in ϕ, as the formula obtained from ϕ by a) replacing each subformula of ϕ of type $\exists z \phi_1$ with $\exists z(x \leq z \wedge \phi_1)$, and b) by replacing each subformula of ϕ of type $\forall z \phi_1$ with $\forall z(x \leq z < y \rightarrow \phi_1)$. Now, consider the formula $\psi \equiv \exists y(x \leq y \wedge \phi'(x, y))$. Note that ψ is a coSafety-FO formula. When interpreted over *infinite* words, the models of ψ are exactly those containing a prefix that belongs to $\mathcal{L}^{<\omega}(\phi)$, with the remaining suffix unconstrained, that is $\mathcal{L}(\psi) = \mathcal{L}^{<\omega}(\phi) \cdot (2^{\Sigma})^{\omega}$, hence $[\![\mathsf{FO}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega} \subseteq [\![\mathsf{coSafety\text{-}FO}]\!]$, and this implies that $[\![\mathsf{coSafety\text{-}LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega} \subseteq [\![\mathsf{coSafety\text{-}FO}]\!]$.

($\supseteq$) We know by Lemma 2 that $[\![\mathsf{coSafety\text{-}FO}]\!] = [\![\mathsf{coSafety\text{-}FO}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$. Since coSafety-FO formulas are also FO formulas, we have $[\![\mathsf{coSafety\text{-}FO}]\!] \subseteq [\![\mathsf{FO}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$. By Proposition 1 and Lemma 7, we obtain that $[\![\mathsf{coSafety\text{-}FO}]\!] \subseteq [\![\mathsf{coSafety\text{-}LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$.

We are ready now to state the main result.

Theorem 2. $[\![\mathsf{LTL}]\!] \cap \mathsf{coSAFETY} = [\![\mathsf{coSafety\text{-}FO}]\!]$

Proof. We know that $[\![\mathsf{LTL}]\!] \cap \mathsf{coSAFETY} = [\![\mathsf{LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$ by Lemma 6. Then, by Lemma 7 we know that $[\![\mathsf{LTL}]\!]^{<\omega} = [\![\mathsf{coSafety\text{-}LTL}]\!]^{<\omega}$, and this in turn implies that $[\![\mathsf{LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega} = [\![\mathsf{coSafety\text{-}LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$. Since $[\![\mathsf{coSafety\text{-}LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega} = [\![\mathsf{coSafety\text{-}FO}]\!]$ by Lemma 8, we conclude that $[\![\mathsf{LTL}]\!] \cap \mathsf{coSAFETY} = [\![\mathsf{coSafety\text{-}FO}]\!]$.

This result together with Theorem 1 allow us to conclude the following.

Theorem 3. $[\![\mathsf{Safety\text{-}LTL}]\!] = [\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY}$

Note that by Observation 1 and Lemma 1 on one hand, and by Lemmas 6 and 7 on the other, the question of whether $[\![\mathsf{Safety\text{-}LTL}]\!] = [\![\mathsf{LTL}]\!] \cap \mathsf{SAFETY}$ can be reduced to whether $[\![\mathsf{coSafety\text{-}LTL}]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega} = [\![\mathsf{coSafety\text{-}LTL}(-\widetilde{\mathsf{X}})]\!]^{<\omega} \cdot (2^{\Sigma})^{\omega}$. If coSafety-LTL and coSafety-LTL($-\mathsf{X}$) were equivalent over finite words, this would already prove Theorem 3. However, we can prove this is not the case.

Theorem 4. $[\![\mathsf{coSafety\text{-}LTL}]\!]^{<\omega} \neq [\![\mathsf{coSafety\text{-}LTL}(-\widetilde{\mathsf{X}})]\!]^{<\omega}$

Proof. Note that in coSafety-LTL($-\widetilde{X}$) we only have existential temporal modalities and we cannot hook the final position of the word without the *weak tomorrow* operator. For these reasons, given a coSafety-LTL($-\widetilde{X}$) formula ϕ, with a simple structural induction we can prove that for each $\sigma \in (2^\Sigma)^+$ such that $\sigma \models \phi$, it holds that $\sigma\sigma' \models \phi$ for any $\sigma' \in (2^\Sigma)^+$, *i.e.*, all the extensions of σ satisfy ϕ as well. This implies that $\mathcal{L}^{<\omega}(\phi)$ is either empty (*i.e.*, if ϕ is unsatisfiable) or infinite. Instead, by using the *weak tomorrow* operator to hook the last position of the word, we can describe a finite non-empty language, for example as in the formula $\phi \equiv a \wedge X(a \wedge \widetilde{X}\bot)$. The language of ϕ is $\mathcal{L}(\phi) = \{\mathsf{aa}\}$, including exactly one word, hence $\mathcal{L}(\phi)$ cannot be described without the *weak tomorrow* operator.

Note that Theorem 4 does *not* contradict Theorem 3, that is, it does not imply that $[\![\mathsf{coSafety\text{-}LTL}]\!]^{<\omega} \cdot (2^\Sigma)^\omega \neq [\![\mathsf{coSafety\text{-}LTL}(-\widetilde{X})]\!]^{<\omega} \cdot (2^\Sigma)^\omega$. For example, consider again the formula $a \wedge X(a \wedge \widetilde{X}\bot)$. It cannot be expressed without the *weak tomorrow* operator, yet it holds that: $\mathcal{L}^{<\omega}(a \wedge X(a \wedge \widetilde{X}\bot)) \cdot (2^\Sigma)^\omega = \mathcal{L}^{<\omega}(a \wedge Xa) \cdot (2^\Sigma)^\omega$.

5   Conclusions

In this paper, we gave a first-order characterization of safety and co-safety languages, by means of two fragments of first-order logic, Safety-FO and coSafety-FO. These fragments of S1S[FO] provide a very natural syntax and are *expressively complete* with regards to LTL-definable safety and co-safety languages.

The core theorem establishes a correspondence between Safety-FO (resp., coSafety-FO) and Safety-LTL (resp., coSafety-LTL), and thus it can be viewed as a special version of Kamp's theorem for safety (resp., co-safety) properties. Thanks to these new fragments, we were able to provide a novel, compact, and self-contained proof of the fact that Safety-LTL captures LTL-definable safety languages. Such a result was previously proved by Chang *et al.* [5], but in terms of the properties of a non-trivial transformation from star-free languages to LTL by Zuck [21]. As a by-product, we provided a number of results that relate the considered languages when interpreted over finite and infinite words. In particular, we highlighted the expressive power of the *weak tomorrow* temporal modality, showing it to be essential in coSafety-LTL over finite words.

Different equivalent characterizations of LTL are known, in terms of (i) first-order logic, (ii) regular expressions, (iii) automata, and (iv) monoids (see the summary by Thomas in [19]). This work focuses on the first item, but for LTL-definable safety languages. A natural follow-up would be to investigate the other items, looking for what kind of automata (resp., regular expressions, monoids) captures exactly safety and co-safety LTL-definable languages. While on finite traces simple characterizations in terms of automata and syntactic monoids exist, the infinite-traces scenario is more complex: there exists a characterization of LTL in terms of counter-free automata [13] and the one for safety ω-regular languages seems not to be difficult (see *e.g.*, terminal automata [4, 18]), but their combination requires to have a canonical (minimal) representation of a (Muller/Rabin/Streett) automata corresponding to any ω-regular language.

References

1. Biere, A., Artho, C., Schuppan, V.: Liveness checking as safety checking. Electronic Notes in Theoretical Computer Science **66**(2), 160–177 (2002)
2. Buchi, J.R.: Weak second-order arithmetic and finite automata. Journal of Symbolic Logic **28**(1) (1963)
3. Büchi, J.R.: On a decision method in restricted second order arithmetic. In: The collected works of J. Richard Büchi, pp. 425–435. Springer (1990)
4. Cerná, I., Pelánek, R.: Relating hierarchy of temporal properties to model checking. In: Rovan, B., Vojtás, P. (eds.) Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science 2003. Lecture Notes in Computer Science, vol. 2747, pp. 318–327. Springer (2003). https://doi.org/10.1007/978-3-540-45138-9_26
5. Chang, E.Y., Manna, Z., Pnueli, A.: Characterization of temporal property classes. In: Kuich, W. (ed.) Proceedings of the 19th International Colloquium on Automata, Languages and Programming. Lecture Notes in Computer Science, vol. 623, pp. 474–486. Springer (1992). https://doi.org/10.1007/3-540-55719-9_97
6. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: Rossi, F. (ed.) Proceedings of the 23rd International Joint Conference on Artificial Intelligence. pp. 854–860. IJCAI/AAAI (2013)
7. De Giacomo, G., Vardi, M.Y.: Synthesis for LTL and LDL on finite traces. In: Yang, Q., Wooldridge, M.J. (eds.) Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence. pp. 1558–1564. AAAI Press (2015)
8. Gabbay, D., Pnueli, A., Shelah, S., Stavi, J.: On the temporal analysis of fairness. In: Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. pp. 163–173 (1980)
9. Giacomo, G.D., Masellis, R.D., Montali, M.: Reasoning on LTL on finite traces: Insensitivity to infiniteness. In: Brodley, C.E., Stone, P. (eds.) Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence. pp. 1027–1033. AAAI Press (2014)
10. Kamp, J.A.W.: Tense logic and the theory of linear order. University of California, Los Angeles (1968)
11. Kupferman, O., Vardi, M.Y.: Model checking of safety properties. Formal Methods in System Design **19**(3), 291–314 (2001)
12. Lichtenstein, O., Pnueli, A., Zuck, L.: The glory of the past. In: Workshop on Logic of Programs. pp. 196–218. Springer (1985)
13. McNaughton, R., Papert, S.A.: Counter-Free Automata (MIT research monograph no. 65). The MIT Press (1971)
14. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science (sfcs 1977). pp. 46–57. IEEE (1977)
15. Rabinovich, A.: A Proof of Kamp's theorem. Logical Methods in Computer Science **Volume 10, Issue 1** (Feb 2014). https://doi.org/10.2168/LMCS-10(1:14)2014, https://lmcs.episciences.org/730
16. Sherman, R., Pnueli, A., Harel, D.: Is the interesting part of process logic uninteresting? A translation from PL to PDL. SIAM J. Comput. **13**(4), 825–839 (1984). https://doi.org/10.1137/0213051
17. Sistla, A.P.: Safety, liveness and fairness in temporal logic. Formal Aspects of Computing **6**(5), 495–511 (1994)
18. Strejcek, J.: Linear temporal logic: Expressiveness and model checking. Ph.D. thesis, Faculty of Informatics, Masaryk University in Brno (2004)

19. Thomas, W.: Safety-and liveness-properties in propositional temporal logic: characterizations and decidability. Banach Center Publications **1**(21), 403–417 (1988)
20. Zhu, S., Tabajara, L.M., Li, J., Pu, G., Vardi, M.Y.: A Symbolic Approach to Safety LTL Synthesis. In: Strichman, O., Tzoref-Brill, R. (eds.) Proceedings of the 13th International Haifa Verification Conference. Lecture Notes in Computer Science, vol. 10629, pp. 147–162. Springer (2017). https://doi.org/10.1007/978-3-319-70389-3_10
21. Zuck, L.: Past temporal logic. Weizmann Institute of Science **67** (1986)

First-order separation over countable ordinals[*]

Thomas Colcombet[1] , Sam van Gool[1] , and Rémi Morvan[2](✉)

[1] IRIF, Université de Paris & CNRS, Paris, France
{thomas.colcombet,vangool}@irif.fr
[2] École normale supérieure Paris-Saclay, Gif-sur-Yvette, France
fistname.lastname@ens-paris-saclay.fr

Abstract. We show that the existence of a first-order formula separating two monadic second order formulas over countable ordinal words is decidable. This extends the work of Henckell and Almeida on finite words, and of Place and Zeitoun on ω-words. For this, we develop the algebraic concept of monoid (resp. ω-semigroup, resp. ordinal monoid) with aperiodic merge, an extension of monoids (resp. ω-semigroup, resp. ordinal monoid) that explicitly includes a new operation capturing the loss of precision induced by first-order indistinguishability. We also show the computability of FO-pointlike sets, and the decidability of the covering problem for first-order logic on countable ordinal words.

Keywords: Regular languages · Separation, Pointlike sets · Countable Ordinals · First-order logic · Monadic second-order logic

A full version of this paper can be found on arXiv. This document contains internal hyperlinks, and is best read on an electronic device.

1 Introduction

In this paper, we establish the decidability of FO-separability over countable ordinal words:

Theorem 1. *There is an algorithm which, given two regular languages of countable ordinal words K, L, either:*

- *answers 'yes', and outputs an FO-separator which is an FO-formula φ which separates K from L, i.e. such that $u \models \varphi$ for all $u \in K$, and $v \models \neg\varphi$ for all $v \in L$, or*
- *answers 'no', and outputs a witness function, i.e., a computable function taking as input an FO-sentence φ and returning a pair of words $(u, v) \in K \times L$ such that $u \models \varphi$ if and only if $v \models \varphi$.*

The decidability of FO-separability was previously only known for finite words [19,2,25,17] and for words of length ω [25]. Countable ordinal words are sequences of letters that are indexed by a countable total well-ordering, *i.e.*, up to isomorphism, by a countable ordinal. There is a natural notion of regular languages over these objects which can be equivalently described in terms of logic (either monadic second-order logic or weak monadic second-order logic), automata (Büchi introduced a notion of automata for countable ordinal words [13], which was studied in more detail by Wojciechowski [39] and which generalises Choueka's automata [15] for words of length at most ω^n—the fact that Choueka's automata can be seen as a restriction of Büchi's automata for countable ordinals was proven by Bedon [5]), rational expressions (introduced by Wojciechowski [40]), or algebra (recognisable by finite ordinal monoids—introduced by Bedon and Carton [8]). A detailed survey of the equivalence between all these notions can be found in Bedon's thesis [6].

Our algorithm follows the approach initiated by Henckell, and constructs the FO-pointlike sets in an ordinal monoid that recognises the two input languages simultaneously. FO-pointlike sets are subsets of a monoid whose elements are inherently indistinguishable by first-order logic. Our completeness proof for the algorithm follows a scheme similar to the one followed by Place and Zeitoun in the context of finite and ω-words [25], which was inspired by Wilke's characterisation of FO-definable languages [38]. We had to make several substantial changes to this approach for the proofs to generalize from finite and ω-words to the setting of countable ordinal words. A seemingly slight modification of the notion of saturation (Definition 8) allows for a careful redesign of several of the core lemmas in the proof of completeness, and in particular the construction of an FO-approximant in Section 5 below.

Related work This work lies in a line of research that aims to obtain a decidable understanding of the expressive power of subclasses of the class of regular languages. The seminal work in this area is the Schützenberger-McNaughton-Papert theorem [34,22] which effectively characterizes the languages of finite words definable in first-order logic as the ones which have an aperiodic syntactic monoid. This theorem was at the origin of a large body of work that studies classes of languages through the corresponding classes of monoids, including for instance Simon's result characterising piecewise-testable languages via $\mathcal{J}$-trivial monoids [36]. FO-pointlike sets are also known in the literature as aperiodic pointlike sets, and were first studied and shown to be computable by Henckell [19], in the context of the Krohn-Rhodes semigroup complexity problem. The computability of pointlike sets was shown to be equivalent to the decidability of the covering problem by Almeida [2]. Alternative proofs of separation and covering problems for FO were given recently in [25,17], and, ever since Henckell's work, the computability of FO-pointlike sets was also extended to pointlike sets for other varieties—for example [4] for the variety of finite groups, [3] for the variety of $\mathcal{J}$-trivial finite semigroups and [18] for varieties of finite semigroups determined by a variety of finite groups; also see [18] for further references. Place and Zeitoun recently used pointlike sets, in the form of covering problems [27],

to resolve long-standing open membership problems for the lower levels of the dot-depth and of the Straubing-Thérien hierarchies [26,28,29].

Another, orthogonal, line of research consists in the extension of the notions of regularity (logic/automata/rational expressions/algebra) to models beyond finite words. This is the case for finite or infinite trees [30]. In this paper, we are concerned with words that go beyond finite, such as words of length ω [12,37,24], of countable ordinal length [6,5], of countable scattered[3] length [31,32], or of general countable length [30,35,14].

These two branches have also been studied jointly, and first-order logic was characterised on words of length ω [23], of countable ordinal length [7], of countable scattered length [10] (and in [9] for first-order augmented with quantifiers over Dedekind cuts), and for words of countable length [16] (as well as other logics [16,21,1]). Prior to the current work, the questions of computing the FO-pointlike sets and deciding FO-separation for languages of infinite words had only been investigated for words of length ω [25].

Structure of the document In Section 2, we introduce important definitions for manipulating infinite words in algebraic terms (ordinal monoids and their pow-erset), and in logical terms (first-order logic and first-order definable maps). In Section 3, we describe the algorithm, and in particular its core, a saturation construction. The correctness of the algorithm is then proved in Section 4, and the completeness in Section 5. In Section 6, we show two stronger results that arise from the same technique: the decidability of the covering problem and the computability of pointlikes. Section 7 concludes.

2   Preliminaries

2.1   Ordinals

A *linear ordering* is a set equipped with a total order. It is *countable* (resp. *finite*) if the underlying set is countable (resp. finite). Let α and β be two linear orderings. A *morphism* from α to β is a monotonic function, and an *isomorphism* between α and β is a bijective morphism. The (ordered) *sum* of two linear orders α and β is denoted by $\alpha + \beta$ and is defined, as usual, on the disjoint union of the linear orders α and β, by further postulating that every element of α is below every element of β. The *product* of two linear orders is denoted by $\alpha \cdot \beta$ and is defined to be the right-to-left lexicographic ordering on the Cartesian product of the two orders, *i.e.*, $(x, y) \leqslant (x', y')$ iff $y < y'$ or $y = y'$ and $x \leqslant x'$. The n-fold product of α with itself is denoted by α^n. A linear ordering is *well-founded* when it does not contain an infinite strictly decreasing sequence. An *ordinal* is a well-founded linear ordering, considered only up to isomorphism of linear orderings. The empty linear ordering, the linear ordering with a single element and the linear ordering of natural numbers are all ordinals, and are denoted 0, 1 and ω, respectively. The class of all ordinals is itself totally ordered by the *embedding*

---

[3] A linear ordering is *scattered* if it does not contain a dense subordering.

relation: $\alpha \preccurlyeq \beta$ means that there exists an injective monotonic function from α to β. The relation $\prec$ denotes the strict ordering associated with $\preccurlyeq$. An ordinal is a *successor ordinal* if it has a maximum, and a *limit ordinal* otherwise.

2.2   Ordinal words

Given a set X, a *word* w over X is a map from some linear ordering to X. The linear ordering is called the *domain* of w, and denoted $\mathrm{dom}(w)$. A word is *countable* (resp. *finite*, resp. *scattered*, resp. *ω-word*), if its domain is countable (resp. finite, resp. scattered, resp. ω). In this paper, a *countable ordinal word* is a word that has a countable and ordinal domain (hence, the countability assumption in silently assumed throughout the paper). The set of all finite words over X is denoted by X^*, and the collection of all countable ordinal words over X is denoted by X^{ord}. Similarly, the set of finite non-empty words is denoted by X^+ and the collection of non-empty countable ordinal words is denoted by $X^{\mathrm{ord}+}$. The concatenation of two countable ordinal words u and v over X is the word $u \cdot v : \mathrm{dom}(u) + \mathrm{dom}(v) \to X$ over X defined by $(u \cdot v)_\iota := u_\iota$ if $\iota \in \mathrm{dom}(u)$ and $(u \cdot v)_\iota := v_\iota$ if $\iota \in \mathrm{dom}(v)$. If w is a countable ordinal word, we define its *omega iteration*, denoted by w^ω, as the word with domain $\mathrm{dom}(w) \cdot \omega$ defined by $(w^\omega)_{(\iota,n)} := w_\iota$ for every $\iota \in \mathrm{dom}(w)$ and $n \in \omega$. For example, if $a, b \in X$, then the omega iteration $(ab)^\omega$ of the two-letter word ab is the word $ababab\cdots$ with domain $2 \cdot \omega = \omega$.

2.3   Ordinal monoids

A *semigroup* is a set S equipped with an associative binary product, denoted by $\cdot$. A *monoid* is a semigroup with a distinguished neutral element for the product, denoted as 1. An element $x \in S$ is called *idempotent* if $x^2 = x$. In a finite finite semigroup S, every element $x \in S$ has a unique *idempotent power*, denoted by[4] x^{idem}, which we recall is the limit of the ultimately constant series $n \mapsto x^{n!}$. We also denote $x^{\mathrm{idem}+k}$, for k integer, the limit of the ultimately constant series $n \mapsto x^{n!+k}$. Note that x^{idem} is the identity element of the unique maximal group inside the subsemigroup generated by x. A finite semigroup is *aperiodic* (we equivalently write *group-trivial*) if $a^{\mathrm{idem}} = a^{\mathrm{idem}+1}$ for all of its elements a.

   We now extend the notion of monoid to obtain an algebraic structure in which one can evaluate a product indexed by any countable ordinal. Let Σ be any set, and α a countable ordinal. For any word $(w_\iota)_{\iota<\alpha}$ over the set Σ^{ord} of countable ordinal words—i.e. $(w_\iota)_{\iota<\alpha}$ is a word whose letters are words over Σ— we define $\mathrm{flat}(w_\iota \mid \iota < \alpha)$ to be the word over Σ with domain $\sum_{\iota<\alpha} \mathrm{dom}(w_\iota)$, which has the letter $(w_\iota)_\kappa \in \Sigma$ at position (ι, κ), for every $\iota \in \alpha$ and $\kappa \in \mathrm{dom}(w_\iota)$.

---

[4] The standard notation is x^ω, but this notation conflicts with the linear ordering ω. It is sometimes denoted x^π or $x^!$ when in the context of infinite words. We find the notation x^{idem} more self-explanatory.

Definition 2. *An* ordinal monoid[5] *is a pair* $\mathcal{M} = (M, \pi)$ *where M is a set and* $\pi : M^{\mathrm{ord}} \to M$ *is a function, called* generalised product, *such that:*

- $\pi(x) = x$ *for every $x \in M$, and*
- $\pi((\pi(u_\iota))_{\iota < \alpha}) = \pi(\mathrm{flat}((u_\iota)_{\iota < \alpha}))$ *for every word $(u_\iota)_{\iota < \alpha} \in (M^{\mathrm{ord}})^{\mathrm{ord}}$.*

The second axiom is called *generalised associativity*. An *ordinal monoid morphism* is a map between ordinal monoids preserving the generalised product. An ordinal monoid is *ordered* if it is equipped with an order $\leqslant$ that makes π monotonic, *i.e.* such that $u \leqslant v$ implies $\pi(u) \leqslant \pi(v)$, in which $\leqslant$ is extended letter-by-letter to words in M^{ord}.

Given a set Σ (the *alphabet*), an ordinal monoid $\mathcal{M} = (M, \pi)$, a letter-to-letter map $\sigma \colon \Sigma \to M$ extended to $\sigma^{\mathrm{ord}} \colon \Sigma^{\mathrm{ord}} \to M^{\mathrm{ord}}$, and $F \subseteq M$, the language $L \subseteq \Sigma^{\mathrm{ord}}$ *recognised* by $(\mathcal{M}, \sigma, F)$ is

$$L = \{u \in \Sigma^{\mathrm{ord}} \ : \ \pi(\sigma^{\mathrm{ord}}(u)) \in F\},$$

and a language $L \subseteq \Sigma^{\mathrm{ord}}$ is called *recognisable* if it is recognised by some such tuple $(\mathcal{M}, \sigma, F)$. We recall that recognisable languages of ordinal words coincide with the ones definable in monadic second-order logic, or definable by suitable automata. These languages are called *regular*. Example 9 below will illustrate this concept.

We now recall a finite presentation of finite ordinal monoids (originally for ordinal semigroups), first given by Bedon [6] by extending a similar result established by Perrin and Pin [24, prop II.5.2] for ω-semigroups[6]. Let (S, π) be an ordinal monoid. We define the constant $\underline{1}$ and two functions $\underline{\cdot} : S \times S \to S$ and $-^{\underline{\omega}} : S \to S$ by

$$\underline{1} := \pi(\varepsilon) \qquad x \underline{\cdot} y := \pi(xy) \qquad \text{and} \qquad x^{\underline{\omega}} := \pi(x^\omega) = \pi(\overbrace{xxx\cdots}^{\omega \text{ times}}) \ .$$

The following proposition lets us interchangeably regard an ordinal monoid $\mathcal{M}$ as either a pair (M, π) or as a quadruple $(M, \underline{1}, \underline{\cdot}, -^{\underline{\omega}})$, that we refer to as its *presentation*.

Proposition 3 ([6, Thm. 3.5.6], originally for ordinal semigroups). *In a finite ordinal monoid the generalised product is uniquely determined by the operations $\underline{1}, \underline{\cdot}$ and $-^{\underline{\omega}}$.*

An important construction on which our proof relies is the *power ordinal monoid*: given an ordinal monoid (M, π), we equip the powerset $\mathcal{P}(M)$ of M with a generalised product $\pi : \mathcal{P}(M)^{\mathrm{ord}} \to \mathcal{P}(M)$ defined by

$$\pi((X_\iota)_{\iota < \kappa}) := \{\pi((x_\iota)_{\iota < \kappa}) \mid x_\iota \in X_\iota \text{ for all } \iota < \kappa\}$$

$$\text{for all words } (X_\iota)_{\iota < \kappa} \in (\mathcal{P}(M))^{\mathrm{ord}}.$$

---

[5] The object should probably be called a 'countable ordinal monoid' since its intent is to model countable ordinal words. However the naming becomes clumsy for 'finite countable ordinal monoids'...

[6] The finitary reprensation of ω-semigroups is usually called a Wilke algebra, which is the algebraic structure introduced by Wilke in [37] to recognise regular ω-languages.

Observe that if M is a finite ordinal monoid, then so is $\mathcal{P}(M)$. We can compute a finite representation of the power ordinal monoid $\mathcal{P}(M)$ of M from a finite representation of M. Indeed,

$$\underline{1} = \{\underline{1}\}, \quad X \cdot Y = \{x \cdot y \mid x \in X, y \in Y\}, \quad \text{and} \quad X^{\underline{\omega}} = \{u \cdot v^{\underline{\omega}} \mid u, v \in X^+\}$$

for all $X, Y \in \mathcal{P}(M)$. The two first properties are trivial while the third one can be proven using the infinite Ramsey's theorem—this is a classical argument used to give finite representation of infinite structures, see e.g. [24, Theorem II.2.1]. Note that this power ordinal monoid is indeed an ordinal monoid. It is even an ordered ordinal monoid when equipped with the inclusion ordering.

2.4   First-order logic

Over a fixed (finite) alphabet Σ, we define the set of *first-order logic formulæ* or FO-*formulæ* for short, by the grammar:

$$\varphi ::= \exists x. \varphi \quad | \quad \forall x. \varphi \quad | \quad \varphi \wedge \varphi \quad | \quad \varphi \vee \varphi \quad | \quad \neg \varphi \quad | \quad x < y \quad | \quad a(x)$$

where x, y range over some fixed infinite set of variables, and a over Σ. *Free variables* are defined as usual, and an FO-*sentence* is a formula with no free variables. In our setting, a model is a countable ordinal word, and a *valuation* over this model is a total map from variables to the domain of the word. We define, for any word w and any valuation ν, the semantic relation $w, \nu \models \varphi$ of first-order logic on countable ordinal words by structural induction on the FO-formula φ, by interpreting variables as positions in the word and propositions of the form $a(x)$ as "the letter at position x is an a". If φ is an FO-sentence, then the semantics of φ over a word w does not depend on the valuation, and thus we write $w \models \varphi$ or $w \not\models \varphi$. When $w \models \varphi$ we say that w *satisfies* φ, or also that φ *accepts* w.

A language $L \subseteq \Sigma^{\mathrm{ord}}$ is said to be FO-*definable* if $L = \{w \in \Sigma^{\mathrm{ord}} \mid w \models \varphi\}$ for some FO-sentence φ. For example, the language of words over the alphabet $\{a, b, c\}$ such that every 'a' is at a finite distance from a 'b' is defined by the FO-sentence $\forall x. a(x) \to \exists y. b(y) \wedge \mathrm{finite}(x, y)$, where:

$$\mathrm{isSuccessor}(z) ::= \exists y. y < z \wedge (\forall x. x < z \to x \leqslant y)$$
$$\mathrm{finite}(x, y) ::= \forall z. (x < z \leqslant y \vee y < z \leqslant x) \to \mathrm{isSuccessor}(z) \ .$$

Bedon [7] extended the Schützenberger-McNaughton-Papert theorem [34,22] to countable ordinal words.

Proposition 4 (*Bedon's theorem* [7, Theorem 3.4]). *A language of countable ordinal words is* FO-*definable if and only if it is recognised by a finite aperiodic ordinal monoid.*

Let $L \subseteq \Sigma^{\mathrm{ord}}$. A function $f : L \to X$ whose codomain X is a finite set is said to be FO-*definable* when every preimage $f^{-1}[x]$, with $x \in X$, is an FO-definable

language. Note that if f is FO-definable, then its domain L is necessarily an FO-definable language.

For example, the function $\Sigma^* \to \mathbb{Z}/2\mathbb{Z}$, sending a word $w \in \Sigma^*$ to its length modulo 2, is not FO-definable. On the other hand, for a fixed letter $a \in \Sigma$, the total function sending a word $w \in \Sigma^{\mathrm{ord}+}$ to $\top$ if w contains the letter 'a' and to $\bot$ otherwise is FO-definable.

A useful tool to manipulate words is the notion of condensation — see, e.g., [33, §4] for an introduction to the subject. A *condensation* of a countable ordinal α is an equivalence relation $\sim$ over α whose equivalence classes are convex. Note that the quotient of an countable ordinal by a condensation is still a countable ordinal.

A *condensation formula* $\varphi(x, y)$ is a formula which is interpreted as a condensation of the domain over all countable ordinal words, *i.e.* for every word $w \in \Sigma^{\mathrm{ord}}$, the relation defined on $\mathrm{dom}(w)$ by $\iota \sim_\varphi \kappa$ if and only if $w, [x \mapsto \iota, y \mapsto \kappa] \models \varphi(x, y)$ is a condensation. A condensation formula $\varphi(x, y)$ induces a map:

$$\hat{\varphi} \colon \Sigma^{\mathrm{ord}} \to (\Sigma^{\mathrm{ord}+})^{\mathrm{ord}}$$

where for every $u \in \Sigma^{\mathrm{ord}}$, $\hat{\varphi}(u)$ is a word whose domain is $\mathrm{dom}(w)/\sim_\varphi$, and such that for every class $I \in \mathrm{dom}(w)/\sim_\varphi$, the I-th letter of $\hat{\varphi}(u)$ is the word $(u_\iota)_{\iota \in I}$—hence $\mathrm{flat}(\hat{\varphi}(u)) = u$.

For example, the formula $\mathrm{finite}(x, y)$ is a condensation formula, called *finite condensation*. The function $\hat{\varphi}_{\mathrm{finite}} \colon \Sigma^{\mathrm{ord}} \to (\Sigma^{\mathrm{ord}})^{\mathrm{ord}}$ that it induces sends the word $ababab \cdots cdcdcd \cdots abc \in \Sigma^{\mathrm{ord}}$ of length $\omega \cdot 2 + 3$ to the 3-letter word $(ababab \cdots)(cdcdcd \cdots)(abc)$. Observe that for every word $w \in \Sigma^{\mathrm{ord}}$, every letter of $\hat{\varphi}_{\mathrm{finite}}(w)$ is a word of length ω, except possibly for the last letter (if the word has one), which can be finite.

Given two FO-definable functions—one that describes "local transformations" and another that described how to glue these local transformations together— the following lemma allows us to build a new FO-definable function. It is one of the key ingredients in our proof of Theorem 1.

Lemma 5. *Let A, B, C be finite sets. Let $\varphi(x, y)$ be a condensation FO-formula over A, let $f \colon A^{\mathrm{ord}+} \to B$ and $g \colon B^{\mathrm{ord}} \to C$ be FO-definable functions. Then, the map*

$$g \circ_\varphi f \colon A^{\mathrm{ord}} \to \quad C$$

$$u \mapsto g \left(\prod_{i \in dom(\hat{\varphi}(u))} f(\hat{\varphi}(u)_i) \right)$$

is FO-definable.

3   The algorithm

In this section we describe the algorithm behind Theorem 1. We first introduce the key notion of saturation in Section 3.1, and formalise the algorithm in Section 3.2.

3.1   The saturation construction

Until the end of Section 3.1, we fix a finite ordinal monoid $\mathcal{M} = (M, \cdot, 1, -^{\omega})$.

The saturation construction is at the heart of the algorithm, both in this paper, and in previous work. We introduce the necessary definitions. Note however that in our case, we do not close the definition under subsets as is usually done. This change, which may look minor, is in fact key for our proof to go through in the case of countable ordinals, and we find it also simplifies some points in the setting of finite words. We first recall an essential operation on $\mathcal{P}(M)$ that we denote $-^{\mathrm{grp}}$. Applied to a set $X \subseteq M$, it computes the union of all the elements that belong to the maximal group in the subsemigroup of $\mathcal{P}(M)$ generated by X.

Definition 6. *Let $X \subseteq M$. Define*

$$X^{\mathrm{grp}} = \bigcup_{k \in \mathbb{N}} X^{\mathrm{idem}+k} =^{\star} \bigcap_{n \in \mathbb{N}} \bigcup_{m \geq n} X^m.$$

Note that the $\star$ equality holds: Left to right inclusion comes from the fact that $X^{\mathrm{idem}+k} = X^m$ holds for infinitely many values of m, while the other inclusion stems from the fact that X^m can be written as $X^{\mathrm{idem}+k}$ for some k whenever m is sufficiently large.

Some important properties of this operation are the following.

Lemma 7. *The operation $-^{\mathrm{grp}}$ is monotonic, and for all $A, B \subseteq M$, and all integers k,*

$$A^{\mathrm{idem}+k} \subseteq A^{\mathrm{grp}}, \qquad\qquad (A \cdot B)^{\mathrm{grp}} = A \cdot (B \cdot A)^{\mathrm{grp}} \cdot B \ ,$$

and $A^{\mathrm{grp}} \cdot A^{\mathrm{grp}} = (A^{\mathrm{grp}})^{\mathrm{grp}} = A^{\mathrm{grp}}.$

The core of the algorithm computes the closure under $-^{\mathrm{grp}}$ and all the operations of the algebra of the images of the letters.

Definition 8. *Let $A \subseteq \mathcal{P}(M)$. The set $\langle A \rangle^{\mathrm{grp,ord}} \subseteq \mathcal{P}(M)$ is defined to be the least set containing A, $\{\underline{1}\}$, and closed under $\cdot$, grp and $\underline{\omega}$.*[7]

This definition is close in spirit to what is called saturation in previous works, with the difference that we do not take the downward closure, and that we close under the operation $-^{\underline{\omega}}$. Despite this difference, we sometimes call $\langle A \rangle^{\mathrm{grp,ord}}$ the *saturation*.

Observe that the ordinal monoid $\mathcal{M}$ is aperiodic if and only if

$$\langle \{\{x\} \mid x \in \mathcal{M}\} \rangle^{\mathrm{grp,ord}} = \{\{x\} \mid x \in \mathcal{M}\} \ .$$

---

[7] Recall that we showed that in a power ordinal monoid, the operation $-^{\underline{\omega}}$ is computable.

3.2   The algorithm

We are now ready to describe the core of the algorithm that is claimed to exist in Theorem 1. Let K and L be two regular languages of countable ordinal words over the alphabet Σ. The *algorithm* is:

1. Let $\mathcal{M}$, σ, F_K, F_L be such that K is recognised by $(\mathcal{M}, \sigma, F_K)$ and L by $(\mathcal{M}, \sigma, F_L)$.
2. Compute $\mathrm{Sat} := \langle\{\{\sigma(a)\} \mid a \in \Sigma\}\rangle^{\mathrm{grp,ord}}$ (inside $\mathcal{P}(\mathcal{M})$).
3. If $F_K \cap X \neq \varnothing$ and $F_L \cap X \neq \varnothing$ for some $X \in \mathrm{Sat}$, answer 'no'. Otherwise answer 'yes'.

$\cdot$	1	a	aa	a^ω	$a^\omega a$	$a^\omega aa$
1	1	a	aa	a^ω	$a^\omega a$	$a^\omega aa$
a	a	aa	a	a^ω	$a^\omega a$	$a^\omega aa$
aa	aa	a	aa	a^ω	$a^\omega a$	$a^\omega aa$
a^ω	a^ω	$a^\omega a$	$a^\omega aa$	a^ω	$a^\omega a$	$a^\omega aa$
$a^\omega a$	$a^\omega a$	$a^\omega aa$	$a^\omega a$	a^ω	$a^\omega a$	$a^\omega aa$
$a^\omega aa$	$a^\omega aa$	$a^\omega a$	$a^\omega aa$	a^ω	$a^\omega a$	$a^\omega aa$
$\_^\omega$	1	a^ω	a^ω	a^ω	a^ω	a^ω
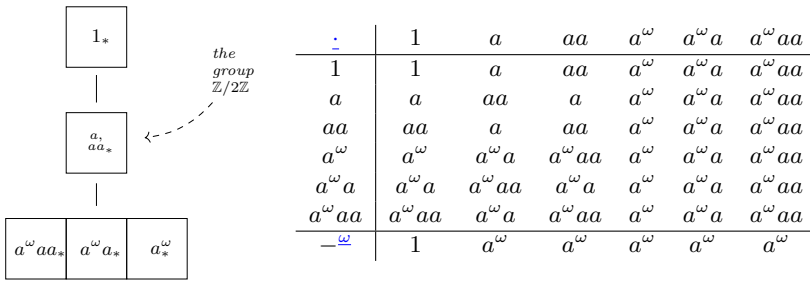
$$\langle\{\{a\}\}\rangle^{\mathrm{grp,ord}} = \{\{1\}, \{a\}, \{aa\}, \{a, aa\}, \{a^\omega\}, \{a^\omega a\}, \{a^\omega aa\}, \{a^\omega a, a^\omega aa\}\}$$

Fig. 1. Egg-box diagram of a finite ordinal monoid $\mathcal{M}$ recognising J, K and L (left), multiplication table and ω-iteration of $\mathcal{M}$ (right) and saturation (bottom).

Example 9. We illustrate the saturation construction and the algorithm on the following three languages over the singleton alphabet $\{a\}$:

$$J = \{\text{infinite words whose longest finite suffix has even length}\},$$
$$K = \{\text{infinite words whose longest finite suffix has odd length}\},$$
$$\text{and} \quad L = \{\qquad \text{words that do not have a last letter}\}.$$

It is classical that J and K are not FO-definable, while L is defined by the formula $\forall x.\, \exists y.\, y > x$. We can build a finite ordinal monoid $\mathcal{M}$ recognising all three languages: it has six elements, 1, a, aa, a^ω, $a^\omega a$ and $a^\omega aa$. Its presentation its described Figure 1. Naturally, the letter a is mapped to $\sigma(a) = a$. Then J, K and L are recognised by $F_J := \{a^\omega, a^\omega aa\}$, by $F_K := \{a^\omega a\}$ and by $F_L := \{1, a^\omega\}$, respectively.

The languages K and L are FO-separable: in fact L is an FO-separator of K and L. On the other hand, J and K are not FO-separable, as witnessed

by the saturation algorithm. Indeed, the saturation $\langle\{\{\sigma(a)\} \mid a \in \Sigma\}\rangle^{\mathrm{grp,ord}}$ contains all singletons, and furthermore $\{a, aa\} = \{a\}^{\mathrm{grp}}$. As a consequence, it also contains $\{a^\omega a, a^\omega aa\} = \{a\}^{\underline{\omega}} \cdot \{a, aa\}$. This last set intersects both F_J and F_K.

The rest of the paper is dedicated to establishing the validity of this approach. In Section 4, we prove Proposition 12 stating that if the algorithm answers 'no', then the languages cannot be separated, as described in Theorem 1. In Section 5, we prove Corollary 16 stating that if the algorithm answers 'yes', then it is possible to construct an FO-separator sentence as described in Theorem 1. In Section 6, we shall package the results of Sections 4 and 5 differently, concluding that we have in fact computed the pointlike sets, and that we can also decide the more general covering problem.

4   When the algorithm says 'no'

In this section, we establish the correctness of the algorithm, i.e., when the algorithm answers 'no', we have to prove that the two input languages cannot be separated by an FO-definable language, and that we can produce a witness function. This is established in Proposition 12. The proof follows standard arguments.

The *quantifier depth*, a.k.a. quantifier rank, of an FO-formula is the maximal number of nested quantifiers in the formula. Two words $u, v \in \Sigma^{\mathrm{ord}}$ are said to be FO_k-*equivalent*, denoted by $u \equiv_{\mathrm{FO}_k} v$, if every FO-sentence of quantifier depth at most k accepts u if and only if it accepts v.

Proposition 10. *Let $k \in \mathbb{N}$.*

- *For $u, u', v, v' \in \Sigma^{\mathrm{ord}}$, if $u \equiv_{\mathrm{FO}_k} u'$ and $v \equiv_{\mathrm{FO}_k} v'$ then $uv \equiv_{\mathrm{FO}_k} u'v'$,*
- *for all Σ^{ord}-valued sequences $(u_n)_{n\in\mathbb{N}}$ and $(v_n)_{n\in\mathbb{N}}$, if $u_n \equiv_{\mathrm{FO}_k} v_n$ for all $n \in \mathbb{N}$, then $\mathrm{flat}(u_n \mid n \in \mathbb{N}) \equiv_{\mathrm{FO}_k} \mathrm{flat}(v_n \mid n \in \mathbb{N})$, and*
- *for all $n \geqslant 2^k - 1$, for all $u \in \Sigma^{\mathrm{ord}}$, $u^n \equiv_{\mathrm{FO}_k} u^{n+1}$.*

This can be proved, for example, by using Ehrenfeucht-Fraïssé games—see e.g. [33, Lemma 6.5 & Corollary 6.9] for a proof of the first and third items ; the proof of the second item is similar[8]. Note that the first two items are also immediate corollaries of the Feferman-Vaught theorem [20, Theorem 1.3]. Note that the third property can be used to prove that every FO-definable language is recognised by an aperiodic finite ordinal monoid—this is the easy direction of Bedon's theorem [7].

Throughout the rest of this section, we fix K and L, two regular languages of countable ordinal words over an alphabet Σ. Recall that the algorithm computes the subset $\mathrm{Sat} := \langle\{\{\sigma(a)\} \mid a \in \Sigma\}\rangle^{\mathrm{grp,ord}}$ of $\mathcal{P}(\mathcal{M})$, where $\mathcal{M}$ is a finite ordinal monoid recognizing both K and L.

---

[8] Moreover, note that the first item can be deduced from the second item by taking $u_n = v_n = \varepsilon$ for $n \geq 2$.

We begin with a lemma which states that to all sets that belong to Sat can be effectively associated witnesses of indisinguishability (we shall see in Proposition 30 that what we have proved is that the elements in Sat are pointlike sets).

Lemma 11. *There exists a computable function which takes as input a number $k \in \mathbb{N}$ and an element $X \in$ Sat, and produces an X-indexed sequence of ordinal words $(u_x)_{x \in X} \in (\Sigma^{\mathrm{ord}})^X$ such that,*

- $\pi(\sigma^{\mathrm{ord}}(u_x)) = x$ *for all $x \in X$, and*
- $u_x \equiv_{\mathrm{FO}_k} u_{x'}$ *for all $x, x' \in X$.*

The proof is by structural induction on the definition of Sat, making use of the two first items of Proposition 10 for composing witnesses, and of furthermore the third item for treating the $-^{\mathrm{grp}}$ operation.

From the above lemma, one can easily deduce that when the algorithm answers 'no', there is indeed an obstruction to the fact that K and L can be FO-separated.

Proposition 12. *Assume that the algorithm answers 'no' when run with input languages K and L. Then there is a witness function which computes, for any FO-sentence φ, a pair of words $(u, u') \in K \times L$ such that $u \models \varphi$ if and only if $u' \models \varphi$. In particular, K and L cannot be FO-separated.*

Proof. Since the algorithm answered 'no', pick a pair $(x, x') \in F_K \times F_L$ such that $x, x' \in X$ for some $X \in$ Sat. Now, for any FO-sentence φ, using the function of Lemma 11 with k the quantifier depth of φ, we can compute a sequence $(u_x)_{x \in X}$ of ordinal words. Now define $u := u_x$ and $u' := u_{x'}$. Then $u \equiv_{\mathrm{FO}_k} u'$, so that $u \models \varphi$ if and only if $u' \models \varphi$. Also, $\pi(\sigma^{\mathrm{ord}}(u)) = x \in F_K$ and $\pi(\sigma^{\mathrm{ord}}(u')) = x' \in F_L$, so $u \in K$ and $u' \in L$. $\square$

Example 13 (Continuing Example 9). Recall that J and K are not FO-separable. Because of the set $\{a^\omega a, a^\omega aa\} \in \langle\{\sigma(a) \mid a \in \Sigma\}\rangle^{\mathrm{grp},\mathrm{ord}}$, the algorithm outputs 'no', and can return, to witness the FO-inseparability of the two languages the computable map $\varphi \mapsto (a^\omega a^{2^k+1}, a^\omega a^{2^k+2}) \in J \times K$, where k denoted the quantifier depth of φ. To prove that $a^\omega a^{2^k+1} \equiv_{\mathrm{FO}_k} a^\omega a^{2^k+2}$, one can simply use the first and third items of Proposition 10.

5   When the algorithm says 'yes'

We now establish the completeness part of the proof of the main theorem, Theorem 1. The goal of this proof is to establish that if the algorithm answers 'yes', it is indeed possible to produce an FO-separator (Corollary 16).

This is the part of the proof that differs most substantially from previous works on separation. In Section 5.1, we abstract the question with the notion of ordinal monoids with merge, and we introduce the notion of FO-approximants which are FO-definable over-approximations of the product. The key result,

Lemma 15, states their existence for all finite ordinal monoid with merge. Corollary 16 follows immediately. The proof of Lemma 15 is then established in Section 5.2 for words of finite or ω length. Building on these simpler cases, the general case is the subject Section 5.3.

5.1   Merge operators and FO-approximants

We abstract in this section the ordinal $\mathcal{P}(M)$ equipped with the $-^{\mathrm{grp}}$ operator into a new algebraic structure. A finite *ordinal monoid with merge* $\mathcal{M} = (M, \underline{1}, \leqslant, \underline{\cdot}, \underline{\cdot}^{\omega}, ^{\mathrm{grp}})$ consists of:

- a presentation of an ordered ordinal monoid $(M, \underline{1}, \leqslant, \underline{\cdot}, \underline{\cdot}^{\omega})$, together with
- a monotonic *merge operator* $-^{\mathrm{grp}} \colon M \to M$ such that for all $a, b \in M$, and all integers k,

$$a^{\mathrm{idem}+k} \leqslant a^{\mathrm{grp}}, \qquad\qquad (a^{\mathrm{idem}})^{\mathrm{grp}} = a^{\mathrm{idem}},$$
$$a^{\mathrm{grp}} \underline{\cdot} a^{\mathrm{grp}} = (a^{\mathrm{grp}})^{\mathrm{grp}} = a^{\mathrm{grp}}, \quad \text{and} \quad (a \underline{\cdot} b)^{\mathrm{grp}} = a \underline{\cdot} (b \underline{\cdot} a)^{\mathrm{grp}} \underline{\cdot} b \ .$$

The following lemma is an immediate consequence of Lemma 7.

Lemma 14. *Both* $(\mathcal{P}(\mathcal{M}), \{1\}, \subseteq, \underline{\cdot}, \underline{\cdot}^{\omega}, ^{\mathrm{grp}})$ *and* $(\mathrm{Sat}, \{1\}, \subseteq, \underline{\cdot}, \underline{\cdot}^{\omega}, ^{\mathrm{grp}})$ *are ordinal monoids with merge.*

The idea behind ordinal monoids with merge is that not only there is a product operation as for every ordinal monoid, but also an FO-definable overapproximation for it. This is the concept of FO-approximant that we introduce now. Given a an FO-definable language $L \subseteq M^{\mathrm{ord}}$, an FO-*approximant* of π over L is an FO-definable map $\rho \colon L \to M$ such that:

$$\pi(u) \leqslant \rho(u), \qquad \text{for all } u \in L.$$

The key result concerning ordinal monoids with merge is the existence of a total FO-approximant:

Lemma 15. *There is an* FO-*approximant* ρ *over* M^{ord} *for all ordinal monoids with merge* $\mathcal{M}$.

An example of an FO-approximant can be found in Example 26. Before establishing Lemma 15, let us explain why it is sufficient for concluding the proof of Theorem 1 in the case the algorithm answers 'yes'.

Corollary 16. *If the algorithm answers 'yes', there exists an* FO-*separator.*

Proof. By Lemmas 14 and 15, there exists an FO-approximant $\rho : A^{\mathrm{ord}} \to \langle A \rangle^{\mathrm{grp,ord}}$ over the power ordinal monoid $\mathcal{P}(\mathcal{M})$, where $A = \{\{\sigma(a)\} \mid a \in \Sigma\}$. Now define the language

$$S := \{u \in \Sigma^{\mathrm{ord}} \mid \rho(\tilde{\sigma}^{\mathrm{ord}}(u)) \cap F_K \neq \varnothing\}$$
$$\text{where } \tilde{\sigma}^{\mathrm{ord}}(u) := (\{\sigma(u_i)\})_{i \in \mathrm{dom}(u)} \in A^{\mathrm{ord}} \text{ for all } u \in \Sigma^{\mathrm{ord}}.$$

Note first that since ρ is FO-definable, this language is FO-definable. Let us show that it separates K from L.

For every $u \in K$, $F_K \ni \pi(\sigma^{\mathrm{ord}}(u)) \subseteq \rho(\tilde{\sigma}^{\mathrm{ord}}(u))$, and as a consequence $\rho(\tilde{\sigma}^{\mathrm{ord}}(u)) \cap F_K \neq \varnothing$. We have proved $K \subseteq S$.

Conversely, consider some $u \in L$. We have $F_L \ni \pi(\sigma^{\mathrm{ord}}(u)) \in \rho(\tilde{\sigma}^{\mathrm{ord}}(u)) \in \langle A \rangle^{\mathrm{grp,ord}}$, and thus $\rho(\tilde{\sigma}^{\mathrm{ord}}(u)) \cap F_L \neq \varnothing$. Since the algorithm returns 'yes', this means that there is no set in $\langle A \rangle^{\mathrm{grp,ord}}$ that intersects both F_K and F_L. In our case, this means that $\rho(\tilde{\sigma}^{\mathrm{ord}}(u)) \cap F_K = \varnothing$, proving that $u \notin S$. We have proved $L \cap S = \varnothing$.

Overall, S is an FO-separator for K and L. □

Remark 17. Notice how the "difficult" implication of Bedon's theorem (Proposition 4) can be easily deduced from Lemma 15[9]: recall that this implication consists in showing that a regular language $L \subseteq \Sigma^{\mathrm{ord}}$, recognised by some triplet $(\mathcal{M}, \sigma, F)$ with $\mathcal{M}$ is aperiodic is definable in first-order logic. Indeed, by aperiodicity of $\mathcal{M}$, the operation $^{\mathrm{grp}}$ applied to a singleton $\{a\}$ yields the singleton $\{a^{\mathrm{idem}}\}$. Hence, the set $\langle \{\{\sigma(a)\} \mid a \in \Sigma\} \rangle^{\mathrm{grp,ord}} = \{\{\pi \circ \sigma^{\mathrm{ord}}(u)\} \mid u \in \Sigma^{\mathrm{ord}}\}$ consists only of singletons, and as a consequence, all FO-approximants ρ (and in particular the one constructed in Lemma 15) maps a word u to $\pi(u)$. Hence, π is an FO-definable map, and thus L is an FO-definable language.

The rest of this section is devoted to establishing Lemma 15. The construction is based on subresults showing the existence of FO-approximants over subsets of M^{ord}; first for finite and ω-words in Section 5.2, and finally for words of any countable ordinal length in Section 5.3. But beforehand, we shall introduce some more definitions and elementary results.

In what follows we use the notation $\langle - \rangle^{\mathrm{grp,ord}}$ from Definition 8, interpreted in a generic ordinal monoid with merge, as well as some variants. Let $A \subseteq M$. We define $\langle A \rangle^+$ as the closure of A under $\cdot$, $\langle A \rangle^{\mathrm{grp}+}$ as the closure of A under $\cdot$ and $-^{\mathrm{grp}}$, and $\langle A \rangle^{\mathrm{grp}*}$ as $\langle A \rangle^{\mathrm{grp}+} \cup \{\underline{1}\}$. We define $\langle A \rangle^{\mathrm{grp,ord}+}$ as the closure of A under $\cdot$, $^{\mathrm{grp}}$ and $\underline{\omega}$. Note that thanks to the identities of ordinal monoids with merge, we have $\langle A \rangle^{\mathrm{grp,ord}} = \langle A \rangle^{\mathrm{grp,ord}+} \cup \{\underline{1}\}$. Moreover, we have the following identities[10]:

Proposition 18. *Let $\mathcal{M}$ be an ordinal monoid with merge. For every $A \subseteq \mathcal{M}$,*

$$\langle A \rangle^{\mathrm{grp}+} = A \langle A \rangle^{\mathrm{grp}*} = \langle A \rangle^{\mathrm{grp}*} A \qquad and \qquad \langle A \rangle^{\mathrm{grp,ord}+} = A \langle A \rangle^{\mathrm{grp,ord}} .$$

Proof. Note, by definition, that $\langle A \rangle^{\mathrm{grp}*} = \langle A \rangle^{\mathrm{grp}+} \cup \{1\}$, so

$$A \langle A \rangle^{\mathrm{grp}*} = A \langle A \rangle^{\mathrm{grp}+} \cup A \subseteq \langle A \rangle^{\mathrm{grp}+}.$$

The converse inclusion $\langle A \rangle^{\mathrm{grp}+} \subseteq A \langle A \rangle^{\mathrm{grp}*}$ is obtained by induction. Let $b \in \langle A \rangle^{\mathrm{grp}+}$. If $b \in A$, then $b \in A \langle A \rangle^{\mathrm{grp}*}$ since $1 \in \langle A \rangle^{\mathrm{grp}*}$. If $c = cd$ with $c, d \in$

---

[9] Similarly, for finite words, Schützenberger-McNaughton-Papert's theorem is a consequence of Henckell's algorithm for aperiodic pointlikes—see e.g. [25, Corollary 4.8]

[10] Notice the similarity with the (trivial) identities $A^+ = AA^* = A^*A$ and $A^{\mathrm{ord}+} = AA^{\mathrm{ord}}$.

$\langle A \rangle^{\mathrm{grp}+}$, then, by induction, $c = ac'$ for some $a \in A$ and $c' \in \langle A \rangle^{\mathrm{grp}*}$, thus $b = a(c'd) \in A \langle A \rangle^{\mathrm{grp}*}$ since $a \in A$ and $c'd \in \langle A \rangle^{\mathrm{grp}*}$. Finally, if $b = c^{\mathrm{grp}}$, then, again by induction, $c = ac'$ for some $a \in A$ and $c' \in \langle A \rangle^{\mathrm{grp}*}$, and thus $b = c^{\mathrm{grp}} = cc^{\mathrm{grp}} = a(c'c^{\mathrm{grp}}) \in A \langle A \rangle^{\mathrm{grp}*}$.

The equality $\langle A \rangle^{\mathrm{grp}+} = \langle A \rangle^{\mathrm{grp}*} A$ is symmetric.

The identity $\langle A \rangle^{\mathrm{grp,ord}+} = A \langle A \rangle^{\mathrm{grp,ord}}$ is similar. The new case in the induction is if some $b \in \langle A \rangle^{\mathrm{grp,ord}+}$ is of the form c^{ω}, then, by induction hypothesis, $c = ac'$ for some $a \in A$ and $c' \in \langle A \rangle^{\mathrm{grp,ord}}$, and thus $b = c^{\omega} = cc^{\omega} = a(c'c^{\omega}) \in A \langle A \rangle^{\mathrm{grp,ord}}$. □

Proposition 19. *If there are* FO-*approximants over K and L respectively, then there exist effectively* FO-*approximants over $K \cup L$ and KL.*

5.2 Construction of FO-approximants for words of finite and ω-length

First, we show how to construct FO-approximants for finite words. It serves at the same time as a building block for more complex cases, as a way to show the proof mechanisms in simpler cases, as well as to comment on differences with previous works.

Lemma 20. *Let $A \subseteq M$, then either*

- $a \cdot \langle A \rangle^{\mathrm{grp}+} \subsetneq \langle A \rangle^{\mathrm{grp}+}$, *for some $a \in A$,*
- $\langle A \rangle^{\mathrm{grp}+} \cdot a \subsetneq \langle A \rangle^{\mathrm{grp}+}$, *for some $a \in A$, or*
- $\langle A \rangle^{\mathrm{grp}+}$ *has a maximum.*

Proof. Assume the two first items do not hold. Because of the non-first-one, the map $x \mapsto a \cdot x$ is surjective on $\langle A \rangle^{\mathrm{grp}+}$, for all $a \in A$. Since $\langle A \rangle^{\mathrm{grp}+}$ is finite, this means that it is bijective on $\langle A \rangle^{\mathrm{grp}+}$. Hence it is also bijective on $\langle A \rangle^{+}$. The negation of the second item has a symmetric consequence. Together we get that $\langle A \rangle^{+}$ is a group. Let I be its neutral element. Note first that for all $x \in \langle A \rangle^{+}$, $I = x^k$ for some k, and hence, $I \leqslant x^{\mathrm{grp}}$. Set now $a_1, \ldots, a_n$ to be the elements in A, and define: $M = (a_1^{\mathrm{grp}} \cdot a_2^{\mathrm{grp}} \cdots a_n^{\mathrm{grp}})^{\mathrm{grp}}$.

By the above remark $a_i = I^{i-1} \cdot a_i \cdot I^{n-i} \leqslant a_1^{\mathrm{grp}} \cdot a_2^{\mathrm{grp}} \cdots a_n^{\mathrm{grp}} \leqslant M$ for all i. Since furthermore for all $x, y \leqslant M$, $x \cdot y \leqslant M$ and $x^{\mathrm{grp}} \leqslant M$, it follows that $z \leqslant M$ for all $z \in \langle A \rangle^{\mathrm{grp}+}$. □

A similar lemma is used in [25], but concludes with the existence of a pseudo-group as the third item.

Lemma 21. *For all $a \in M$ there exists an* FO-*approximant from a^{+} to $\langle \{a\} \rangle^{\mathrm{grp}+}$.*

Construction. Let k be such that $a^{\mathrm{idem}} = a^k$. Define

$$\rho(\overbrace{a \cdots a}^{\text{length } n}) = \begin{cases} a^n & \text{if } n < k, \\ a^{\mathrm{grp}} & \text{otherwise.} \end{cases}$$

□

We can now use this for proving the finite word case.

Lemma 22. *For all $A \subseteq M$ there exists an FO-approximant from A^+ to $\langle A \rangle^{\mathrm{grp}+}$.*

Proof. We use a double induction on $|\langle A \rangle^{\mathrm{grp}+}|$ and $|A|$. The induction is guided by Lemma 20. The base case is $A = \varnothing$, and the nowhere defined FO-approximant proves it.

First case: $a \cdot \langle A \rangle^{\mathrm{grp}+} \subsetneq \langle A \rangle^{\mathrm{grp}+}$ for some $a \in A$. This part of the proof is similar to [25, Lemma 6.7]. Let $B ::= A \smallsetminus \{a\}$.

We first construct an FO-approximant from $a^+ B^+$ to $a \cdot \langle A \rangle^{\mathrm{grp}+}$. Indeed, we know by Lemma 21 that there is an FO-approximant from a^+ to $\langle \{a\} \rangle^{\mathrm{grp}+} \subseteq a \cdot \langle A \rangle^{\mathrm{grp}*}$. We also know by induction[11] that there is an FO-approximant from B^+ to $\langle B \rangle^{\mathrm{grp}+} \subseteq \langle A \rangle^{\mathrm{grp}+}$. Thus by Proposition 19, there exists effectively an FO-approximant τ from $a^+ B^+$ to $a \cdot \langle A \rangle^{\mathrm{grp}*} \cdot \langle A \rangle^{\mathrm{grp}+} \subseteq a \cdot \langle A \rangle^{\mathrm{grp}+}$.

We now provide an FO-approximant for $(a^+ B^+)^+$ (which is FO-definable), and for this, define the condensation FO-formula $\varphi(x, y)$ that expresses that "two positions x and y are equivalent if the subword on the interval $[x, y]$ belongs to $a^* B^*$" (this can be expressed in first-order logic). Over a word $u \in (a^+ B^+)^+$, each of the condensation classes belong to $a^+ B^+$ and its image under τ belongs to $a \cdot \langle A \rangle^{\mathrm{grp}+}$. Furthermore, still by induction hypothesis[12], there is an FO-approximant from $(a \cdot \langle A \rangle^{\mathrm{grp}+})^+$ to $\langle A \rangle^{\mathrm{grp}+}$. By Lemma 5, we thus obtain an FO-definable map from $(a^+ B^+)^+$ to $\langle A \rangle^{\mathrm{grp}+}$. It is an FO-approximant by construction.

Using the above case and Proposition 19, it can be easily extended to an FO-approximant from $A^+ = AB^*(a^+ B^+)^* a^*$ to $\langle A \rangle^{\mathrm{grp}+}$.

Second case: $\langle A \rangle^{\mathrm{grp}+} \cdot a \subsetneq \langle A \rangle^{\mathrm{grp}+}$. This case is symmetric to the first case.

Third case: $\langle A \rangle^{\mathrm{grp}+}$ has a maximum M. Then the constant map that sends every word $u \in A^*$ to M is an FO-approximant over A^*.   □

Following similar ideas, we can treat the case of ω-words. We define here $\langle A \rangle^{\mathrm{grp},\omega}$ as the elements of the form $\{a \cdot b^{\omega} \mid a, b \in \langle A \rangle^{\mathrm{grp}+}\}$—or, equivalently, $\langle A \rangle^{\mathrm{grp},\omega} = (\langle A \rangle^{\mathrm{grp}+})^{\omega}$.

Lemma 23. *Let M be an ordinal monoid with merge. For all $A \subseteq M$, there exists an FO-approximant from A^{ω} to $\langle A \rangle^{\mathrm{grp},\omega}$.*

5.3   Construction of FO-approximants for countable ordinal words

As for the finite case, the proof revolves around a carefully designed case distinction. This one is more complex to establish, and makes use of Green's relations and a precise understanding of the properties of ordinal monoids with merge.

Lemma 24 (Trichotomy principle). *Let M be a finite ordinal monoid with merge and $A \subseteq M$, then either*

---

[11] Indeed, $|B| < |A|$.

[12] This time, we can use the induction hypothesis because $|\langle (a \cdot \langle A \rangle^{\mathrm{grp}+})^+ \rangle^{\mathrm{grp}+}| < |\langle A \rangle^{\mathrm{grp}+}|$. Indeed, by Proposition 18, $\langle (a \cdot \langle A \rangle^{\mathrm{grp}+})^+ \rangle^{\mathrm{grp}} \subseteq (a \cdot \langle A \rangle^{\mathrm{grp}+})^+ \langle (a \cdot \langle A \rangle^{\mathrm{grp}+})^+ \rangle^{\mathrm{grp}*} \subseteq a \cdot \langle A \rangle^{\mathrm{grp}+} \subsetneq \langle A \rangle^{\mathrm{grp}+}$.

- $a \underline{\cdot} \langle A \rangle^{\mathrm{grp,ord+}} \subsetneq \langle A \rangle^{\mathrm{grp,ord+}}$, *for some* $a \in A$,
- $\langle \langle A \rangle^{\mathrm{grp},\omega} \rangle^{\mathrm{grp,ord+}} \subsetneq \langle A \rangle^{\mathrm{grp,ord+}}$, *or*
- $x \underline{\cdot} y = y$ *and* $x^{\underline{\omega}} = y^{\underline{\omega}}$, *for all* $x, y \in \langle A \rangle^{\mathrm{grp,ord+}}$.

The above lemma is key in the proof of the existence of an FO-approximant.

Lemma 25. *For all* $a \in \mathcal{M}$, *there exists an* FO-*approximant over* a^{ord}.

The proof follows a similar structure as the one for Lemma 22 for the finite case. This time, Lemma 24 is the key argument that makes the induction progress, playing the same role as Lemma 20 in the finite case. Note, however, that the second items in Lemmas 20 and 24 are very different in structure. And indeed, this entails a different argument for constructing the FO-approximant. It is based on performing in one step the condensation of all the maximal factors of order-type ω.

Example 26 (Continuing Example 13). An FO-approximant ρ of π over a^{ord} in the ordinal monoid defined in Example 9 can be defined for all $u \in \{a\}^{\mathrm{ord}}$ as:

$$\rho(u) := \begin{cases} \{1\} & \text{if } \mathrm{dom}(u) \text{ is empty,} \\ \{a, aa\} & \text{if } \mathrm{dom}(u) \text{ is finite and non-empty,} \\ \{a^{\omega}\} & \text{if } \mathrm{dom}(u) \text{ is a non-zero limit ordinal,} \\ \{a^{\omega}a, a^{\omega}aa\} & \text{if } \mathrm{dom}(u) \text{ is an infinite successor ordinal.} \end{cases}$$

Lemma 27. *For all* $A \subseteq \mathcal{M}$, *there exists an* FO-*approximant from* $A^{\mathrm{ord+}}$ *to* $\langle A \rangle^{\mathrm{grp,ord+}}$.

Proof. We prove the result by induction on $|\langle A \rangle^{\mathrm{grp,ord+}}|$ and $|A^{\mathrm{ord+}}|$. The base case $A = \varnothing$ is trivial. If A is non-empty, following Lemma 24, there are three cases to treat.

First case: There exists $a \in A$ *such that* $a \underline{\cdot} \langle A \rangle^{\mathrm{grp,ord+}} \subsetneq \langle A \rangle^{\mathrm{grp,ord+}}$. This case is as in the proof for finite words, Lemma 22, using Lemma 25 in place of Lemma 21. The key reason why the proof remains valid is because the hypothesis $a \underline{\cdot} \langle A \rangle^{\mathrm{grp,ord+}} \subsetneq \langle A \rangle^{\mathrm{grp,ord+}}$ implies $|\langle (a \underline{\cdot} \langle A \rangle^{\mathrm{grp,ord+}})^{\mathrm{ord+}} \rangle^{\mathrm{grp,ord+}}| < |\langle A \rangle^{\mathrm{grp,ord+}}|$ by Proposition 18[13].

Second case[14]: $\langle \langle A \rangle^{\mathrm{grp},\omega} \rangle^{\mathrm{grp,ord+}} \subsetneq \langle A \rangle^{\mathrm{grp,ord+}}$. By Lemma 23, there is an FO-approximant from A^{ω} to $\langle A \rangle^{\mathrm{grp},\omega}$. By induction hypothesis[15], we have an FO-approximant from $(\langle A \rangle^{\mathrm{grp},\omega})^{\mathrm{ord+}}$ to $\langle \langle A \rangle^{\mathrm{grp},\omega} \rangle^{\mathrm{grp,ord+}} \subseteq \langle A \rangle^{\mathrm{grp,ord+}}$. Since

---

[13] More precisely, we are using the property $\langle B \rangle^{\mathrm{grp,ord+}} = B \langle B \rangle^{\mathrm{grp,ord}}$ of Proposition 18. By thinking of elements of $\langle B \rangle^{\mathrm{grp,ord+}}$ as "countable ordinal words with merge", this property is simply saying that every "countable ordinal word with merge" has a first letter. However, countable ordinal words need not have a last letter: this is what makes an hypothesis of the form $\langle A \rangle^{\mathrm{grp,ord+}} \underline{\cdot} a \subsetneq \langle A \rangle^{\mathrm{grp,ord+}}$ unusable—and this is the motivation behind the trichotomy principle Lemma 24.

[14] Note here that it is different from the second case in the proof of Lemma 22.

[15] Indeed, $\langle \langle A \rangle^{\mathrm{grp},\omega} \rangle^{\mathrm{grp,ord+}} \subsetneq \langle A \rangle^{\mathrm{grp,ord+}}$.

the formula finite(x, y) is a condensation FO-formula, we obtain by Lemma 5 an FO-approximant from $(A^\omega)^{\mathrm{ord}+} \to \langle A \rangle^{\mathrm{grp,ord}+}$. Using Proposition 19 and Lemma 22, we easily extend it to an FO-approximant from $A^{\mathrm{ord}+} = A(A^\omega)^{\mathrm{ord}}A^*$ to $\langle A \rangle^{\mathrm{grp,ord}+}$.

Third case: $x \cdot y = y$ and $x^{\underline{\omega}} = y^{\underline{\omega}}$, for all $x, y \in \langle A \rangle^{\mathrm{grp,ord}+}$. Then the product over A sends a countable ordinal word $u \in A^{\mathrm{ord}+}$ to its last letter if the word has a last letter, and to the unique omega power of $\langle A \rangle^{\mathrm{grp,ord}+}$ if the word has no last letter. Since the languages of the form $A^{\mathrm{ord}+}a$ where $a \in A$ and $\{u \in A^{\mathrm{ord}+} \mid \mathrm{dom}(u)$ is a limit ordinal$\}$ all are FO-definable, it follows that the product over A is FO-definable. □

6 Related problems

In this section, we solve two related problems: the decidability of the covering problem (Proposition 28), and the computability of pointlike sets (Proposition 30). Both are direct applications of the key lemmas presented above.

The FO-*covering problem* asks, given regular languages, in our case of countable ordinal words, $L, K_1, \ldots, K_n$, to determine if there exist FO-definable languages $C_1, \ldots, C_n$ such that $L \subseteq \cup_i C_i$ and $C_i \cap K_i = \varnothing$ for all i—see [27] for more details. In general, separation problems trivially reduce to covering problems, since L and K are separable if and only if there is a solution to the covering problem for the instance (L, K). In the other direction, there is no known example of a variety with decidable separation problem but undecidable covering problem. We show that a further consequence of the above results is that the FO-pointlike sets in a finite ordinal monoid (see Definition 29) are computable, from which we deduce:

Proposition 28. *The FO-covering problem for countable ordinal words is decidable.*

Let us now introduce, and explain, the relation with pointlike sets. The FO_k-*closure* of a word u is the set $[u]_{FO_k}$ which contains all words that are FO_k-equivalent to u.

Definition 29. *Given a finite ordinal monoid $\mathcal{M}$ the FO-pointlike sets of a map $\sigma \colon \Sigma \to M$ are defined by*

$$\mathrm{Pl}_{\mathrm{FO}}(\sigma) ::= \bigcap_{k \in \mathbb{N}} \downarrow \left\{ \pi(\sigma^{\mathrm{ord}}([u]_{FO_k})) \mid u \in \Sigma^{\mathrm{ord}} \right\},$$

where $\downarrow X$ denotes the downward closure of X.

The definition of pointlike sets is in fact more general[16]: given a variety of finite semigroups $\mathbb{V}$ one can define a notion of pointlike sets with respect to this

---

[16] In the following discussion, we focus on finite words, but the notion of variety—of algebras, or of languages—can be extended to countable ordinal words [8] and many other settings [11, §4].

variety. Almeida observed that the separation problem for the variety $\mathbb{V}$—given two regular languages, can they be separated by a $\mathbb{V}$-recognisable language?—is decidable if and only if the $\mathbb{V}$-pointlikes of size 2 of every morphism are computable [2, Prop. 3.4]. The covering problem also has an algebraic counterpart: it is decidable for the variety $\mathbb{V}$ if and only if, for every morphism, the collection of all $\mathbb{V}$-pointlike sets of this morphism is computable [2, Prop. 3.6][17]. Hence, the fact that FO-covering and FO-separation are decidable for finite words is simply a corollary of Henckell's theorem on aperiodic pointlikes [19, Fact 3.7 & Fact 5.31], stating that they are computable. Place & Zeitoun's simpler proof of the decidability of FO-covering for finite words and for ω-words [25] relies on the same principle.[18] Unsurprisingly, our result can be interpreted in the same way: we are implicitly showing the following property, from which one can immediately deduce the computability of $\mathrm{Pl}_{\mathrm{FO}}(\sigma)$.

Proposition 30. *Given a finite ordinal monoid $\mathcal{M}$ and $\sigma\colon \Sigma \to M$,*

$$\mathrm{Pl}_{\mathrm{FO}}(\sigma) = \downarrow \langle\{\{\sigma(a)\} \mid a \in \Sigma\}\rangle^{\mathrm{grp,ord}}.$$

7   Conclusion

In this paper, we have studied the problem of FO-separation over words of countable ordinal length. Our proof is based on the work of Place and Zeitoun over words of length ω [25]. We build an FO-approximant using essentially the same technique as Place and Zeitoun. However a key difference is that for finite words and ω-words, the proof relies on a case distinction (Lemma 20) which is conceptually similar to the characterisation of groups as semigroups whose translations are bijective. This was no longer sufficient for countable ordinal words because of ω-iterations. In this situation, our new case distinction (Lemma 24) captures the subtle interaction of ω-iteration with groups in finite ordinal monoids. In particular, a difference with previously known algorithms is that we do not close the saturation under subset. This a priori innocuous difference has significant consequences on the proof of completeness, yielding some simplifications in the finite and ω-case, and necessary for the proof to be extendable to all ordinals.

Of course, the next step is to go to longer words, in particular scattered countable words, or even better to all countable words. Here, there are conceptual difficulties, and let us stress also that, starting from scattered countable words, first-order logic and first-order logic with access to Dedekind cuts begin to have a different expressiveness. Thus several notions of separation have to be studied.

References

1. Adsul, B., Sarkar, S., Sreejith, A.V.: First-order logic and its infinitary quantifier extensions over countable words (2021)

---

[17] Beware: there is a typo in the statement of the first item of the proposition.

[18] There is a difference in terminology: they refer to the $\mathrm{Pl}_{\mathrm{FO}}(\varphi)$ as "optimal imprint with respect to FO on φ".

2. Almeida, J.: Some algorithmic problems for pseudovarieties. Publ. Math. Debrecen **54**(1), 531–552 (1999)

3. Almeida, J., Zeitoun, M.: The pseudovariety J is hyperdecidable. RAIRO-Theoretical Informatics and Applications **31**(5), 457–482 (1997)

4. Ash, C.J.: Inevitable graphs: a proof of the type II conjecture and some related decision procedures. International Journal of Algebra and Computation **1**(01), 127–146 (1991)

5. Bedon, N.: Finite automata and ordinals. Theoretical Computer Science **156**(1), 119–144 (1996). https://doi.org/10.1016/0304-3975(95)00006-2

6. Bedon, N.: Langages reconnaissables de mots indexés par des ordinaux. Theses, Université de Marne la Vallée (Jan 1998), https://tel.archives-ouvertes.fr/tel-00003586

7. Bedon, N.: Logic over words on denumerable ordinals. Journal of Computer and System Sciences **63**(3), 394–431 (2001). https://doi.org/10.1006/jcss.2001.1782

8. Bedon, N., Carton, O.: An Eilenberg theorem for words on countable ordinals. In: Lucchesi, C.L., Moura, A.V. (eds.) LATIN'98: Theoretical Informatics. pp. 53–64. Springer Berlin Heidelberg, Berlin, Heidelberg (1998). https://doi.org/10.1007/BFb0054310

9. Bedon, N., Rispal, C.: Schützenberger and Eilenberg theorems for words on linear orderings. Journal of Computer and System Sciences **78**(2), 517–536 (Mar 2012). https://doi.org/10.1016/j.jcss.2011.06.003

10. Bès, A., Carton, O.: Algebraic Characterization of FO for Scattered Linear Orderings. In: Bezem, M. (ed.) Computer Science Logic (CSL'11) - 25th International Workshop/20th Annual Conference of the EACSL. Leibniz International Proceedings in Informatics (LIPIcs), vol. 12, pp. 67–81. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2011). https://doi.org/10.4230/LIPIcs.CSL.2011.67

11. Bojańczyk, M.: Recognisable languages over monads. In: Potapov, I. (ed.) Developments in Language Theory. pp. 1–13. Springer International Publishing, Cham (2015), https://arxiv.org/abs/1502.04898v1

12. Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Logic, Methodology and Philosophy of Science (Proc. 1960 Internat. Congr .), pp. 1–11. Stanford Univ. Press, Stanford, Calif. (1962)

13. Büchi, J.R.: The monadic second order theory of ω_1, pp. 1–127. Springer Berlin Heidelberg (1973). https://doi.org/10.1007/BFb0082721

14. Carton, O., Colcombet, T., Puppis, G.: An algebraic approach to MSO-definability on countable linear orderings (May 2018). https://doi.org/10.1017/jsl.2018.7

15. Choueka, Y.: Finite automata, definable sets, and regular expressions over ωn-tapes. Journal of Computer and System Sciences **17**(1), 81–97 (1978)

16. Colcombet, T., Sreejith, A.V.: Limited set quantifiers over countable linear orderings. In: Proceedings, Part II, of the 42nd International Colloquium on Automata, Languages, and Programming - Volume 9135. pp. 146–158. ICALP 2015, Springer-Verlag, Berlin, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47666-6_12

17. van Gool, S.J., Steinberg, B.: Merge decompositions, two-sided Krohn–Rhodes, and aperiodic pointlikes. Canadian Mathematical Bulletin **62**(1), 199–208 (2019). https://doi.org/10.4153/CMB-2018-014-8

18. Gool, S., Steinberg, B.: Pointlike sets for varieties determined by groups. Advances in Mathematics **348**, 18–50 (2019). https://doi.org/10.1016/j.aim.2019.03.020

19. Henckell, K.: Pointlike sets: the finest aperiodic cover of a finite semigroup. Journal of Pure and Applied Algebra **55**(1), 85–126 (1988). https://doi.org/10.1016/0022-4049(88)90042-4

20. Makowsky, J.A.: Algorithmic uses of the Feferman–Vaught theorem. Annals of Pure and Applied Logic **126**(1-3), 159–213 (2004). https://doi.org/10.1016/j.apal.2003.11.002

21. Manuel, A., Sreejith, A.V.: Two-variable logic over countable linear orderings. In: Faliszewski, P., Muscholl, A., Niedermeier, R. (eds.) 41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland. LIPIcs, vol. 58, pp. 66:1–66:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016). https://doi.org/10.4230/LIPIcs.MFCS.2016.66, https://doi.org/10.4230/LIPIcs.MFCS.2016.66

22. McNaughton, R., Papert, S.A.: Counter-Free Automata. The MIT Press (1971)

23. Perrin, D.: Recent results on automata and infinite words. In: International Symposium on Mathematical Foundations of Computer Science. pp. 134–148. Springer (1984). https://doi.org/10.1007/BFb0030294

24. Pin, J.E., Perrin, D.: Infinite Words: Automata, Semigroups, Logic and Games. Elsevier (2004), https://hal.archives-ouvertes.fr/hal-00112831

25. Place, T., Zeitoun, M.: Separating regular languages with first-order logic. Logical Methods in Computer Science **12** (2016). https://doi.org/10.2168/LMCS-12(1:5)2016

26. Place, T., Zeitoun, M.: The complexity of separation for levels in concatenation hierarchies. In: Ganguly, S., Pandya, P. (eds.) 38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018). Leibniz International Proceedings in Informatics (LIPIcs), vol. 122, pp. 47:1–47:17. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2018). https://doi.org/10.4230/LIPIcs.FSTTCS.2018.47

27. Place, T., Zeitoun, M.: The covering problem. Logical Methods in Computer Science **Volume 14, Issue 3** (Jul 2018). https://doi.org/10.23638/LMCS-14(3:1)2018

28. Place, T., Zeitoun, M.: On all things star-free. In: 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2019), https://arxiv.org/abs/1904.11863v1

29. Place, T., Zeitoun, M.: Separation for dot-depth two. Logical Methods in Computer Science **Volume 17, Issue 3** (Sep 2021). https://doi.org/10.46298/lmcs-17(3:24)2021

30. Rabin, M.O.: Decidability of second-order theories and automata on infinite trees. Trans. Amer. Math. Soc. **141**, 1–35 (1969)

31. Rispal, C.: Automates sur les ordres linéaires : Complémentation. Theses, Université de Marne la Vallée (Dec 2004), https://tel.archives-ouvertes.fr/tel-00720658

32. Rispal, C., Carton, O.: Complementation of Rational Sets on Countable Scattered Linear Orderings. International Journal of Foundations of Computer Science **16**(4), 767–786 (2005), https://hal.archives-ouvertes.fr/hal-00160985

33. Rosenstein, J.G.: Linear orderings. Academic press (1982)

34. Schützenberger, M.: On finite monoids having only trivial subgroups. Information and Control **8**(2), 190–194 (1965). https://doi.org/10.1016/S0019-9958(65)90108-7

35. Shelah, S.: The monadic theory of order. Ann. of Math. (2) **102**(3), 379–419 (1975)

36. Simon, I.: Piecewise testable events. In: Brakhage, H. (ed.) Automata Theory and Formal Languages. pp. 214–222. Springer Berlin Heidelberg, Berlin, Heidelberg (1975)

37. Wilke, T.: An algebraic theory for regular languages of finite and infinite words. International Journal of Algebra and Computation **03**(04), 447–489 (1993). https://doi.org/10.1142/S0218196793000287

38. Wilke, T.: Classifying discrete temporal properties. In: Meinel, C., Tison, S. (eds.) STACS 99. pp. 32–46. Springer Berlin Heidelberg, Berlin, Heidelberg (1999). https://doi.org/10.1007/3-540-49116-3_3
39. Wojciechowski, J.: Classes of transfinite sequences accepted by nondeterministic finite automata. Fundamenta informaticæ **7**(2), 191–223 (1984)
40. Wojciechowski, J.: Finite automata on transfinite sequences and regular expressions. Fundamenta informaticæ **8**(3-4), 379–396 (1985)

A Faithful and Quantitative Notion of Distant Reduction for Generalized Applications

José Espírito Santo[1](✉) [iD], Delia Kesner[2,3](✉) [iD], and Loïc Peyrot[2](✉) [iD]

[1] Centro de Matemática, Universidade do Minho, Portugal
jes@math.uminho.pt
[2] Université de Paris, CNRS, IRIF, Paris, France
{kesner,lpeyrot}@irif.fr
[3] Institut Universitaire de France (IUF), France

Abstract. We introduce a call-by-name lambda-calculus λJ with generalized applications which integrates a notion of distant reduction that allows to unblock β-redexes without resorting to the permutative conversions of generalized applications. We show strong normalization of simply typed terms, and we then fully characterize strong normalization by means of a quantitative typing system. This characterization uses a non-trivial inductive definition of strong normalization –that we relate to others in the literature–, which is based on a weak-head normalizing strategy. Our calculus relates to explicit substitution calculi by means of a translation between the two formalisms which is faithful, in the sense that it preserves strong normalization. We show that our calculus λJ and the well-know calculus ΛJ determine equivalent notions of strong normalization. As a consequence, ΛJ inherits a faithful translation into explicit substitutions, and its strong normalization can be characterized by the quantitative typing system designed for λJ, despite the fact that quantitative subject reduction fails for permutative conversions.

Keywords: Lambda-calculus · Generalized applications · Quantitative types

1 Introduction

(Pure) functional programming can be understood by means of a universal model of computation known as the λ-calculus, which is in tight correspondence, by means of the so-called Curry-Howard isomorphism, with propositional intuitionistic logic in Gentzen's natural deduction style. The Curry-Howard isomorphism emphasizes the fact that proof systems on one hand, and programming languages on the other, are two mathematical and computational facets of the same object. The λ-calculus with *generalized applications* (ΛJ), introduced by Joachimski and Matthes [8], is an extension of the λ-calculus which can be seen as the Curry-Howard counterpart of van Plato's natural deduction with generalized elimination rules [11].

A generalized application in ΛJ is written $t(u, y.r)$. It intuitively means that t is applied to u in the context of the substitution $\{\_/y\}r$. The conversion of the β-redex $(\lambda x.t)(u, y.r)$ then produces two (nested) substitutions $\{\{u/x\}t/y\}r$. But some β-redexes can be *blocked* by the syntax, *e.g.* in the term $t(u, y.r)(u', y'.r')$, where the (potential) application of $r = \lambda x.s$ to u' remains hidden. An iterated generalized application $t(u, y.r)(u', y'.r')$ may be rearranged as $t(u, y.r(u', y'.r'))$ by a permutative conversion called π. Rule π is then an unblocker of stuck β-redexes: the contractum $t(u, y.(\lambda x.s)(u', y'.r'))$ unveils the desired application of r to u'. Rule π, together with rule β, allows natural deduction proofs to be brought to a "fully normal" form [11] enjoying the subformula property. Computationally, ΛJ defines a call-by-name operational semantics; a call-by-value variant has been proposed in [5], but this is out of the scope of this paper.

Strong normalization w.r.t. the two rules β and π has been characterized by typability with (idempotent) intersection types by Matthes [10]: a term is typable if and only if it is strongly normalizing. However, this characterization is just *qualitative*. A different flavor of intersection types, called *non-idempotent*, offers a more powerful *quantitative* characterization of strong normalization, in the sense that the length of the longest reduction sequence to normal form starting at a typable term t is bound by the size of its type derivation. However, quantitative types were never used in the framework of generalized applications, and it is our purpose to propose and study one such typing system.

Quantitative types allow for simple combinatorial proofs of strong normalization, without any need to use reducibility or computability arguments. More remarkably, they also provide a refined tool to understand permutative rules. For instance, in ΛJ, rule π is not quantitatively sound (*i.e.* π does not enjoy quantitative subject reduction), although π becomes valid in an idempotent framework. Hence, a good question is: how can we unblock redexes to reach normal forms in a quantitative model of computation based on generalized applications?

Our solution is to adopt the paradigm of *distant* reduction [2] coming from explicit substitution (ES) calculi, which extends the key concept of β-redex, so that we may find the λ-abstraction hidden under a sequence of nested generalized applications. This is essentially similar to adopting a different permutation rule, converting $t(u, y.\lambda z.s)$ to $\lambda z.t(u, y.s)$. However, the permutation rule is mostly a way to overcome syntactical limitations, while distant β is a way to put emphasis on the computational behavior of the calculus: it is at the β-step that resources are consumed, not during the permutations.

The syntax of the ΛJ-calculus will thus be equipped with an operational call-by-name semantics given by distant β, but without π. The resulting calculus is called λJ. As a major contribution, we prove a characterization of strong normalization in terms of typability in our quantitative system. In such proof, the soundness result (typability implies strong normalization) is obtained by combinatorial arguments, with the size of typing derivations decreasing at each step. For the completeness result (strong normalization implies typability) we need an inductive characterization of the terms that are strongly normalizing for distant β: this is a non-trivial technical contribution of the paper.

As mentioned above, we draw inspiration for our distant β rule from calculi with explicit substitutions, having in mind the usual translation of $t(u, y.r)$ to the explicit substitution $[tu/y]r$ (a let-binding of tu over y in r). As such, we expect the dynamic behavior of our calculus to be *faithful* to explicit substitutions. Such translation, however, does not in general preserve strong normalization. Indeed, in a β-redex $(\lambda x.t)(u, y.r)$, the interaction of $\lambda x.t$ with the argument u is materialized by the internal substitution in the contractum term $\{\{u/x\}t/y\}r$, as mentioned before. But such interaction is elusive: if the external substitution is vacuous (that is, if y is not free in r), β-reduction will simply throw away the λ-abstraction $\lambda x.t$ and its argument u, whereas $(\lambda x.t)u$ may reduce in the context of the explicit substitution $[(\lambda x.t)u/y]r$. The different interaction between the abstraction and its argument in the two mentioned models of computation has important consequences. For instance, let $\delta^\circ := \lambda x.x(x, w.w)$ be the encoding of $\delta = \lambda x.xx$ as a ΛJ-term. Then, if $y \notin r$ and r is normal, the only thing the term $\delta^\circ(\delta^\circ, y.r)$ can do is to reduce to r, whereas $\delta\delta$ may reduce forever in the context of the vacuous explicit substitution $[\delta\delta/y]r$.

That is why we propose a new, type-preserving, encoding of terms with generalized applications into terms with explicit substitutions. Using this new encoding and quantitative types, we show that strong normalization of the source term with generalized applications is equivalent to the strong normalization of the target term with explicit substitutions.

As a final contribution, we compare λJ-strong normalization to that of other calculi, including the original ΛJ. We extract new results for the latter, as a faithful translation to ES, and a new normalizing strategy. Moreover, we obtain a quantitative characterization of ΛJ-strong normalization, where the bound for reduction given by the size of type derivations only holds for β (and not for π).

Plan of the paper. Sec. 2 presents our calculus with distant β. Sec. 3 provides an inductive characterization of strongly normalizing terms. Sec. 4 is about non-idempotent intersection types. Sec. 5 shows the faithful translation to ES. Sec. 6 contains the comparisons with other calculi. Sec. 7 concludes. Full proofs are available in [6].

2   A Calculus with Generalized Applications

In this section we define our calculus λJ with generalized applications and give some introductory observations on strong normalization in that system.

2.1   Syntax and Semantics

We start with some general notations. Given a reduction relation $\rightarrow_{\mathcal{R}}$, we write $\rightarrow_{\mathcal{R}}^{*}$ (resp. $\rightarrow_{\mathcal{R}}^{+}$) for the reflexive-transitive (resp. transitive) closure of $\rightarrow_{\mathcal{R}}$. A term t is said to be in $\mathcal{R}$-**normal form** (written $\mathcal{R}$-nf) iff there is no t' such that $t \rightarrow_{\mathcal{R}} t'$. A term t is said to be $\mathcal{R}$-**strongly normalizing** (written $t \in \mathcal{SN}(\mathcal{R})$) iff there is no infinite $\mathcal{R}$-sequence starting at t. $\mathcal{R}$ is strongly normalizing iff every term is $\mathcal{R}$-strongly normalizing. When $\mathcal{R}$ is finitely branching, $\|t\|_{\mathcal{R}}$ denotes the

maximal length of an $\mathcal{R}$-reduction sequence to $\mathcal{R}$-nf starting at t, for every $t \in \mathcal{SN}(\mathcal{R})$.

The set of terms generated by the following grammar is denoted by $\mathtt{T}_J$.

$$(\textbf{Terms}) \quad t, u, r, s ::= x \mid \lambda x.t \mid t(u, x.r)$$

The term $t(u, x.r)$ is called a generalized application, and the part $x.r$ is sometimes referred as the *continuation* of that application. Free variables of terms are defined as usual, notably $\mathrm{fv}(t(u, x.r)) := \mathrm{fv}(t) \cup \mathrm{fv}(u) \cup \mathrm{fv}(r) \setminus \{x\}$. We also work modulo α-conversion, denoted $=_\alpha$, so that bound variables can be systematically renamed. We use $\mathtt{I}$ to denote the identity function $\lambda z.z$.

We introduce contexts (terms with one occurrence of the hole $\Diamond$) and the special distant contexts:

$$
\begin{array}{ll}
(\textbf{Contexts}) & \mathtt{C} ::= \Diamond \mid \lambda x.\mathtt{C} \mid \mathtt{C}(u, x.r) \mid t(\mathtt{C}, x.r) \mid t(u, x.\mathtt{C}) \\
(\textbf{Distant Contexts}) & \mathtt{D} ::= \Diamond \mid t(u, x.\mathtt{D})
\end{array}
$$

The term $\mathtt{C}\langle t \rangle$ denotes $\mathtt{C}$ where $\Diamond$ is replaced by t, so that capture of variables may eventually occur. Given a rewriting rule $\mathcal{R} \subseteq \mathtt{T}_J \times \mathtt{T}_J$, $\to_\mathcal{R}$ denotes the reduction relation generated by the closure of $\mathcal{R}$ under all contexts.

We say that t has an **abstraction shape** iff $t = \mathtt{D}\langle \lambda x.u \rangle$. The substitution operation is capture-avoiding and defined as usual, in particular $\{u/x\}(t(s, y.r)) := (\{u/x\}t)(\{u/x\}s, y.\{u/x\}r)$.

2.2   Towards a Call-by-Name Operational Semantics

The $\mathtt{T}_J$-syntax can be equipped with different rewriting rules, as discussed in the introduction. We use the generic notation $\mathtt{T}_J[\mathcal{R}]$ to denote the calculus given by the syntax $\mathtt{T}_J$ equipped with the reduction relation $\to_\mathcal{R}$.

Now, if we consider $t_0 := t(u', y'.\lambda x.s)(u, y.r)$ in the calculus $\mathtt{T}_J[\beta]$, where

$$(\lambda x.s)(u, y.r) \mapsto_\beta \{\{u/x\}s/y\}r$$

we can see that the term t_0 is stuck since the subterm $\lambda x.s$ is not close to u. This is when the following rule π, plays the role of an unblocker of β-redexes:

$$t(u, y.r)(u', y'.r') \mapsto_\pi t(u, y.r(u', y'.r'))$$

Indeed, $t_0 \to_\pi t(u', y'.(\lambda x.s)(u, y.r)) \to_\beta t(u', y'.\{\{u/x\}s/y\}r)$. More generally, given $t_1 := \mathtt{D}\langle \lambda x.s \rangle(u, y.r)$, with $\mathtt{D} \neq \Diamond$, a sequence of π-steps reduces the term t_1 above to $\mathtt{D}\langle (\lambda x.s)(u, y.r) \rangle$. A further β-step produces $\mathtt{D}\langle \{\{u/x\}s/y\}r \rangle$. So, the original ΛJ-calculus [8], which is exactly $\mathtt{T}_J[\beta, \pi]$, has a derived notion of *distant* β rule, based on π, which can be specified by the following rule:

$$\mathtt{D}\langle \lambda x.s \rangle(u, y.r) \mapsto \mathtt{D}\langle \{\{u/x\}s/y\}r \rangle \tag{1}$$

However, π-reduction is not only about unblocking redexes, as witnessed by $\mathtt{D}\langle x \rangle(u, y.r) \to_\pi^* \mathtt{D}\langle x(u, y.r) \rangle$. So it is reasonable to keep terms of the form

$D\langle x\rangle(u, y.r)$ without reducing them further, as those π-steps do not contribute to unblock more β-redexes. The absence of terms of the form $D\langle\lambda x.s\rangle(u, y.r)$ gives already a reasonable notion of normal form which, in particular, already enjoy the subformula property, as will be seen in Sec. 2.3.

Still, we will not reduce as in (1) because such rule, as well as π itself, does not admit a quantitative semantics (*c.f.* Sec. 4.3). We then choose to unblock β-redexes with the following rule p_2 instead[4]:

$$t(u', y'.\lambda x.s) \mapsto_{p_2} \lambda x.t(u', y'.s)$$

so that t_1 given above reduces in several p_2-steps to $(\lambda x.D\langle s\rangle)(u, y.r)$, which can now be further reduced with β since it is no longer stuck. If we reduce it, we obtain $\{\{u/x\}D\langle s\rangle/y\}r$; and since free variables in u cannot be captured by D, this is equal to $\{D\langle\{u/x\}s\rangle/y\}r$. We thus obtain our distant rule:

Definition 1. *We write λJ for our new calculus $T_J[d\beta]$, where the distant β-rule is defined as follows:*

$$D\langle\lambda x.t\rangle(u, y.r) \mapsto_{d\beta} \{D\langle\{u/x\}t\rangle/y\}r$$

A reduction step $t_1 \to_{d\beta} t_2$ is said to be **erasing** iff the reduced $d\beta$-redex in t_1 is of the form $D\langle\lambda x.t\rangle(u, y.r)$ with $x \notin \text{fv}(t)$ or $y \notin \text{fv}(r)$.

It is obvious that $\to_{d\beta} \subset \to^+_{\beta, p_2}$. Some other variants of the p_2-rule are possible, like $D\langle\lambda x.t\rangle(u, y.r) \mapsto (\lambda x.D\langle t\rangle)(u, y.r)$ or $D\langle\lambda x.t\rangle \mapsto_{p_2} \lambda x.D\langle t\rangle$, in both cases for $D \neq \Diamond$, but we do not develop them. However, while most of the paper is about λJ, brief comparisons with the calculi ΛJ and $T_J[\beta, p_2]$ are considered in Sec.6.

2.3 Some (Un)typed Properties of λJ

Lemma 1. *The grammar m characterizes $d\beta$-normal forms.*

$$m ::= x \mid \lambda x.m \mid m_{\text{var}}(m, x.m) \qquad m_{\text{var}} ::= x \mid m_{\text{var}}(m, x.m_{\text{var}})$$

We already saw that, once β is generalized to $d\beta$, π is not needed anymore to unblock β-redexes; the next Lemma says that π preserves $d\beta$-nfs, so it does not bring anything new to $d\beta$-nfs either. The proof uses Lem. 1, and it proceeds by simultaneous induction on m and m_{var}.

Lemma 2. *If t is a $d\beta$-nf, and $t \to_\pi t'$, then t' is a $d\beta$-nf.*

Let us discuss now some properties related to (simple) typability for generalized applications [8], a system that we call ST. Recall the following typing rules, where $\sigma, \rho, \tau ::= a \mid \sigma \to \rho$, and a belongs to a set of base type variables:

$$\frac{}{\Gamma, x : \sigma \vdash x : \sigma} \qquad \frac{\Gamma, x : \sigma \vdash t : \rho}{\Gamma \vdash \lambda x.t : \sigma \to \rho} \qquad \frac{\Gamma \vdash t : \rho \to \tau \quad \Gamma \vdash u : \rho \quad \Gamma, y : \tau \vdash r : \sigma}{\Gamma \vdash t(u, y.r) : \sigma}$$

We write $\Gamma \Vdash_{ST} t : \sigma$ if there is a type derivation in system ST ending in $\Gamma \vdash t : \sigma$. In the following result, we refer to simple types as formulas.

---

[4] Rule p_2 is used in [7,3] along with two other permutation rules p_1 and p_3 to reduce T_J-terms to a fragment isomorphic to natural deduction.

Lemma 3 (Subformula Property). *If $\Phi = \Gamma \Vdash_{ST} \mathtt{m} : \tau$ then every formula in the derivation Φ is a subformula of τ or a subformula of some formula in Γ.*

Proof. The lemma is proved together with another statement: If $\Psi = \Gamma \Vdash_{ST} \mathtt{m}_{\mathtt{var}} : \tau$ then every formula in Ψ is a subformula of some formula in Γ. The proof is by simultaneous induction of Φ and Ψ.

We close this section with the following:

Theorem 1. *If t is simply typable, i.e. $\Gamma \Vdash_{ST} t : \sigma$, then $t \in \mathcal{SN}(\mathtt{d}\beta)$.*

The proof is by a map into the λ-calculus which produces a simulation when the λ-calculus is equipped with the following σ-rules [13]:

$$(\lambda x.M)NN' \mapsto_{\sigma_1} (\lambda x.MN')N \qquad (\lambda x.\lambda y.M)N \mapsto_{\sigma_2} \lambda y.(\lambda x.M)N$$

3   Inductive Characterization of Strong Normalization

In this section we give an inductive characterization of strong normalization (ISN) for λJ and prove it correct. This characterization will be useful to show completeness of the type system that we are going to present in Sec. 4.1, as well as to compare strong normalization of λJ to the ones of $\mathtt{T}_\lambda[\beta, \mathtt{p_2}]$ and ΛJ.

3.1   ISN in the λ-Calculus Through Weak-Head Contexts

As an introduction, we first look at the case of the ISN for the λ-calculus ($\mathcal{ISN}(\beta)$), on which our forthcoming definition of $\mathcal{ISN}(\mathtt{d}\beta)$ elaborates. A usual way to define $\mathcal{ISN}(\beta)$ is by the following rules [12], where the general notation $t\boldsymbol{r}$ abbreviates $(\dots(tr_1)\dots)r_n$ for some $n \geq 0$.

$$\frac{r_1,\dots,r_n \in \mathcal{ISN}(\beta)}{x\boldsymbol{r} \in \mathcal{ISN}(\beta)} \qquad \frac{t \in \mathcal{ISN}(\beta)}{\lambda x.t \in \mathcal{ISN}(\beta)} \qquad \frac{\{u/x\}t\boldsymbol{r}, u \in \mathcal{ISN}(\beta)}{(\lambda x.t)u\boldsymbol{r} \in \mathcal{ISN}(\beta)}$$

One shows that $t \in \mathcal{SN}(\beta)$ if and only if $t \in \mathcal{ISN}(\beta)$.

The reduction strategy underlying the definition of $\mathcal{ISN}(\beta)$ is the following one: reduce terms to weak-head normal form, and then iterate reduction inside the components of the weak-head normal form, without any need to come back to the head of the term. **Weak-head normal terms** are of two kinds: (**neutral terms**) $\mathtt{n} ::= x \mid \mathtt{n}t$ and (**answers**) $\mathtt{a} ::= \lambda x.t$. Neutral terms cannot produce any head β-redex. On the contrary, answers can create a β-redex when given at least one argument. In the case of the λ-calculus, these are only abstractions. If the term is not a weak-head term, a redex can be located with a *weak-head context* $\mathtt{W} ::= \lozenge \mid \mathtt{W}t$. These concepts allow a different definition of $\mathcal{ISN}(\beta)$.

$$\frac{}{x \in \mathcal{ISN}(\beta)} \qquad \frac{\mathtt{n}, t \in \mathcal{ISN}(\beta)}{\mathtt{n}t \in \mathcal{ISN}(\beta)} \qquad \frac{t \in \mathcal{ISN}(\beta)}{\lambda x.t \in \mathcal{ISN}(\beta)} \qquad \frac{\mathtt{W}\langle\{u/x\}t\rangle, u \in \mathcal{ISN}(\beta)}{\mathtt{W}\langle(\lambda x.t)u\rangle \in \mathcal{ISN}(\beta)}$$

Weak-head contexts are an alternative to the meta-syntactic notation $\boldsymbol{r}$ of vectors of arguments. Notice that there is one rule for each kind of neutral term, one rule for answers and one rule for terms which are not weak-head normal forms.

3.2   ISN for dβ

We define $\mathcal{ISN}(\text{d}\beta)$ with the same methodology as before. Hence, we first have to define neutral terms, answers and weak-head contexts.

Definition 2. *We consider the following grammars:*

$$
\begin{aligned}
(\textbf{Neutral terms})\ \text{n} \ &::= x \mid \text{n}(u, x.\text{n}) \\
(\textbf{Answers})\ \text{a} \ &::= \lambda x.t \mid \text{n}(u, x.\text{a}) \\
(\textbf{Neutral distant contexts})\ \text{D}_{\text{n}} \ &::= \Diamond \mid \text{n}(u, x.\text{D}_{\text{n}}) \\
(\textbf{Weak-head contexts})\ \text{W} \ &::= \Diamond \mid \text{W}(u, x.r) \mid \text{n}(u, x.\text{W})
\end{aligned}
$$

Notice that n *and* a *are disjoint and stable by* dβ*-reduction. Also* $\text{D}_{\text{n}} \subsetneq \text{W}$.

Example 1 (Decomposition).   Let $t = x_1(x_2, y_1.I(I, z.I))(x_3, y.II)$. Then, there are two decompositions of t in terms of a redex r and a weak-head context W: either $\text{W} = \Diamond$ and $r = t$, or $\text{W} = x_1(x_2, y_1.\Diamond)(x_3, y.II)$ and $r = I(I, z.I)$. In both cases $t = \text{W}\langle r \rangle$. We will rule out the first possibility by defining next a restriction of the β-rule, securing uniqueness of such kind of decomposition in all cases.

The strategy underlying our definition of $\mathcal{ISN}(\text{d}\beta)$ will be the **weak-head strategy** $\to_{\text{wh}}$, defined as the closure under W of the following restricted β-rule:

$$
\text{D}_{\text{n}}\langle \lambda x.t \rangle(u, y.r) \mapsto \{\text{D}_{\text{n}}\langle \{u/y\}t \rangle/y\}r.
$$

The restriction of D to a neutral distant context D_{n} is what allows determinism of our forthcoming Def. 3.

Lemma 4. *The reduction* $\to_{\text{wh}}$ *is deterministic.*

As in the case of the λ-calculus, weak-head normal forms are either neutral terms or answers. This time, answers are not only abstractions, but also abstractions under a (neutral) distant context. Because of distance, these terms can also create a dβ-redex when applied to an argument, as seen in the next example.

Example 2. Consider again term t of Ex. 1. If the third form in the grammar of W was disallowed, then it would not be possible to write t as $\text{W}\langle r \rangle$, with r a restricted redex. In that case, the reduction strategy associated with $\mathcal{ISN}(\text{d}\beta)$ would consider t as a weak-head normal form, and start reducing the subterms of t, including $I(I, z.I)$. Now, the latter would eventually reach I and suddenly the whole term $t' = x_1(x_2, y_1.I)(x_3, y.r')$ would be a weak-head redex again: the typical separation between an initial weak-head reduction phase and a later internal reduction phase, as it is the case in the λ-calculus, would be lost in our framework. This is a subtle point due to the *distant* character of rule dβ which explains the complexity of Def. 2.

Lemma 5. *Let* $t \in \text{T}_J$. *Then* t *is in* wh-*normal form iff* $t \in \text{n} \cup \text{a}$.

Our inductive definition of strong normalization follows.

Definition 3 (Inductive Strong Normalization). *We consider the following inductive predicate:*

$$\frac{}{x \in \mathcal{ISN}(\mathtt{d}\beta)}(\mathtt{snvar}) \qquad \frac{\mathtt{n}, u, r \in \mathcal{ISN}(\mathtt{d}\beta) \qquad r \in \textit{wh-nf}}{\mathtt{n}(u, x.r) \in \mathcal{ISN}(\mathtt{d}\beta)}(\mathtt{snapp})$$

$$\frac{t \in \mathcal{ISN}(\mathtt{d}\beta)}{\lambda x.t \in \mathcal{ISN}(\mathtt{d}\beta)}(\mathtt{snabs}) \qquad \frac{\mathtt{W}\langle\{\mathtt{D_n}\langle\{u/x\}t\rangle/y\}r\rangle, \mathtt{D_n}\langle t\rangle, u \in \mathcal{ISN}(\mathtt{d}\beta)}{\mathtt{W}\langle\mathtt{D_n}\langle\lambda x.t\rangle(u, y.r)\rangle \in \mathcal{ISN}(\mathtt{d}\beta)}(\mathtt{snbeta})$$

Notice that every term can be written according to the conclusions of the previous rules, so that the grammar $t, u, r ::= x \mid \lambda x.t \mid \mathtt{n}(t, x.r) \mid \mathtt{W}\langle\mathtt{D_n}\langle\lambda x.t\rangle(u, y.s)\rangle$, with $r \in$ wh-nf, also defines the syntax $\mathtt{T}_J$. Moreover, at most one rule in the previous definition applies to each term, *i.e.* the rules are deterministic. An equivalent, but non-deterministic definition, can be given by removing the side condition "$r \in$ wh-nf" in rule (snapp). Indeed, this (weaker) rule would overlap with rule (snbeta) for terms in which the weak-head context lies in the last continuation, as for instance in $x(u, y.y)(u', y'.\mathtt{II})$. Notice the difference with the λ-calculus: the head of a term with generalized applications can be either on the left of the term (as in the λ-calculus), or recursively on the left in a continuation. We conclude with the following result.

Theorem 2. $\mathcal{SN}(\mathtt{d}\beta) = \mathcal{ISN}(\mathtt{d}\beta)$.

4   Quantitative Types Characterize Strong Normalization

We proved that simply typable terms are strongly normalizing in Sec. 2.3. In this section we use non-idempotent intersection types to fully characterize strong normalization, so that strongly normalizing terms are also typable. First we introduce the typing system, next we prove the characterization and finally we study the quantitative behavior of π and give in particular an example of failure.

4.1   The Typing System

We now define our quantitative type system $\cap J$ for $\mathtt{T}_J$-terms and we show that strong normalization in λJ exactly corresponds to $\cap J$ typability.

Given a countable infinite set BTV of base type variables $a, b, c, \ldots$, we define the following sets of types:

$$\textbf{(types)} \ \sigma, \tau, \rho ::= a \in BTV \mid \mathcal{M} \to \sigma$$
$$\textbf{(multiset types)} \ \mathcal{M}, \mathcal{N} ::= [\sigma_i]_{i \in I} \text{ where } I \text{ is a finite set}$$

The empty multiset is denoted $[\,]$. We use $|\mathcal{M}|$ to denote the size of the multiset, thus if $\mathcal{M} = [\sigma_i]_{i \in I}$ then $|\mathcal{M}| = |I|$. We introduce a **choice operator** on multiset types: if $\mathcal{M} \neq [\,]$, then $\#(\mathcal{M}) = \mathcal{M}$, otherwise $\#([\,]) = [\sigma]$, where σ is an arbitrary type. This operator is used to guarantee that there is always a typing witness for all the subterms of typed terms.

Typing environments (or just **environments**), written Γ, Δ, Λ, are functions from variables to multiset types assigning the empty multiset to all but a finite set of variables. The domain of Γ is given by $\mathrm{dom}(\Gamma) := \{x \mid \Gamma(x) \neq [\,]\}$. The **union of environments**, written $\Gamma \wedge \Delta$, is defined by $(\Gamma \wedge \Delta)(x) := \Gamma(x) \sqcup \Delta(x)$, where $\sqcup$ denotes multiset union. This notion is extended to several environments as expected, so that $\wedge_{i \in I} \Gamma_i$ denotes a finite union of environments ($\wedge_{i \in I} \Gamma_i$ is to be understood as the empty environment when $I = \emptyset$). We write $\Gamma \backslash\!\backslash x$ for the environment such that $(\Gamma \backslash\!\backslash x)(y) = \Gamma(y)$ if $y \neq x$ and $(\Gamma \backslash\!\backslash x)(x) = [\,]$. We write $\Gamma ; \Delta$ for $\Gamma \wedge \Delta$ when $\mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Delta) = \emptyset$. A sequent has the form $\Gamma \vdash t : \sigma$, where Γ is an environment, t is a term, and σ is a type.

The type system $\cap J$ is given by the following typing rules.

$$\frac{}{x : [\sigma] \vdash x : \sigma} \,(\mathtt{var}) \qquad \frac{\Gamma ; x : \mathcal{M} \vdash t : \sigma}{\Gamma \vdash \lambda x.t : \mathcal{M} \to \sigma} \,(\mathtt{abs}) \qquad \frac{(\Gamma_i \vdash t : \sigma_i)_{i \in I} \quad I \neq \emptyset}{\wedge_{i \in I} \Gamma_i \vdash t : [\sigma_i]_{i \in I}} \,(\mathtt{many})$$

$$\frac{\Gamma \vdash t : \#([\mathcal{M}_i \to \tau_i]_{i \in I}) \qquad \Delta \vdash u : \#(\sqcup_{i \in I} \mathcal{M}_i) \qquad \Lambda ; x : [\tau_i]_{i \in I} \vdash r : \sigma}{\Gamma \wedge \Delta \wedge \Lambda \vdash t(u, x.r) : \sigma} \,(\mathtt{app})$$

The use of the choice operator in rule $(\mathtt{app})$ is subtle. If I is empty, then the multiset $[\mathcal{M}_i \to \tau_i]_{i \in I}$ typing t as well as the multiset $\sqcup_{i \in I} \mathcal{M}_i$ typing u are both empty, so that the choice operator must be used to type both terms. If I is not empty, then the multiset typing t is non-empty as well. However, the multiset typing u may or not be empty, *e.g.* if $[[\,] \to \alpha]$ types t.

System $\cap J$ lacks weakening: it is *relevant*.

Lemma 6 (Relevance). *If $\Gamma \Vdash t : \sigma$, then $\mathrm{fv}(t) = \mathrm{dom}(\Gamma)$.*

Notice that the typing rules (and the choice operator) force all the subterms of a typed term to be also typed. Moreover, if $I = \emptyset$ in rule $(\mathtt{app})$, then the types of t and u are not necessarily related. Indeed, let $\delta^\circ := \lambda y.y(y, w.w)$ in $t_0 := \delta^\circ(\delta^\circ, x.z)$. Then t_0 is $\mathsf{d}\beta$-strongly-normalizing so it must be typed in system $\cap J$. However, since the set I of $x : [\tau_i]_{i \in I}$ in the typing of $r = z$ is necessarily empty (*c.f.* Lem. 6), then the unrelated types $\#([\mathcal{M}_i \to \tau_i]_{i \in I})$ and $\#(\sqcup_{i \in I} \mathcal{M}_i)$ of the two occurrences of δ° witness to the fact that these subterms will not interact during the reduction of t_0. Indeed, the term t_0 can be typed as follows, where $\rho_i := [[\sigma_i] \to \sigma_i, \sigma_i] \to \sigma_i$ and $\tau_i := [\sigma_i] \to \sigma_i$, for $i = 1, 2$:

$$\frac{\dfrac{\emptyset \vdash \delta^\circ : \rho_1}{\emptyset \vdash \delta^\circ : [\rho_1]}\,(\mathtt{many}) \qquad \dfrac{\emptyset \vdash \delta^\circ : \rho_2}{\emptyset \vdash \delta^\circ : [\rho_2]}\,(\mathtt{many}) \qquad \dfrac{}{z : [\tau]; x : [\,] \vdash z : \tau}\,(\mathtt{var})}{z : [\tau] \vdash \delta^\circ(\delta^\circ, x.z) : \tau}\,(\mathtt{app})$$

where δ° is typed with ρ_i as follows:

$$\frac{\dfrac{\dfrac{}{y : [\tau_i] \vdash y : \tau_i}\,(\mathtt{var})}{y : [\tau_i] \vdash y : [\tau_i]}\,(\mathtt{many}) \qquad \dfrac{\dfrac{}{y : [\sigma_i] \vdash y : \sigma_i}\,(\mathtt{var})}{y : [\sigma_i] \vdash y : [\sigma_i]}\,(\mathtt{many}) \qquad \dfrac{}{w : [\sigma_i] \vdash w : \sigma_i}\,(\mathtt{var})}{\dfrac{y : [[\sigma_i] \to \sigma_i, \sigma_i] \vdash y(y, w.w) : \sigma_i}{\emptyset \vdash \lambda y.y(y, w.w) : [[\sigma_i] \to \sigma_i, \sigma_i] \to \sigma_i}\,(\mathtt{abs})}\,(\mathtt{app})$$

We write $\Gamma \Vdash_{\cap J} t : \sigma$ or simply $\Gamma \Vdash t : \sigma$ if there is a derivation in system $\cap J$ ending in $\Gamma \vdash t : \sigma$. For $n \geq 1$, we write $\Gamma \Vdash_{\cap J}^n t : \sigma$ or simply $\Gamma \Vdash^n t : \sigma$ if there is a derivation in system $\cap J$ ending in $\Gamma \vdash t : \sigma$ and containing n occurrences of rules in the set $\{(\mathtt{var}), (\mathtt{abs}), (\mathtt{app})\}$.

4.2   The Characterization of dβ-Strong Normalization

The soundness Lem. 9 is based on Lem. 8, based in turn on Lem. 7.

Lemma 7 (Substitution Lemma). *Let $t, u \in \mathtt{T}_J$ with $x \in \mathrm{fv}(t)$. If both $\Gamma; x : \mathcal{M} \Vdash^n t : \sigma$ and $\Delta \Vdash^m u : \mathcal{M}$ hold, then $\Gamma \wedge \Delta \Vdash^k \{u/x\}t : \sigma$ where $k = n + m - |\mathcal{M}|$.*

Lemma 8 (Non-Erasing Subject Reduction). *Let $\Gamma \Vdash_{\cap J}^{n_1} t_1 : \sigma$. If $t_1 \to_{\mathtt{d}\beta} t_2$ is a non-erasing step, then $\Gamma \Vdash_{\cap J}^{n_2} t_2 : \sigma$ with $n_1 > n_2$.*

Lemma 9 (Soundness for λJ). *If t is $\cap J$-typable, then $t \in \mathcal{SN}(\mathtt{d}\beta)$.*

The completeness Lem. 13 is based on Lem. 10 and Lem. 12, this last based in turn on Lem. 11.

Lemma 10 (Typing Normal Forms).

1. *For all $t \in \mathtt{m}$, there exists Γ, σ such that $\Gamma \Vdash_{\cap J} t : \sigma$.*
2. *For all $t \in \mathtt{m}_{\mathtt{var}}$, for all σ, there exists Γ such that $\Gamma \Vdash_{\cap J} t : \sigma$.*

Lemma 11 (Anti-Substitution). *If $\Gamma \Vdash \{u/x\}t : \sigma$ where $x \in \mathrm{fv}(t)$, then there exist Γ_t, Γ_u and $\mathcal{M} \neq []$ such that $\Gamma_t; x : \mathcal{M} \Vdash t : \sigma$, $\Gamma_u \Vdash u : \mathcal{M}$ and $\Gamma = \Gamma_t \wedge \Gamma_u$.*

Lemma 12 (Non-Erasing Subject Expansion). *If $\Gamma \Vdash_{\cap J} t_2 : \sigma$ and $t_1 \to_{\mathtt{d}\beta} t_2$ is a non-erasing step, then $\Gamma \Vdash_{\cap J} t_1 : \sigma$.*

Lemma 13 (Completeness for λJ). *If $t \in \mathcal{SN}(\mathtt{d}\beta)$, then t is $\cap J$-typable.*

We finally obtain:

Theorem 3 (Characterization). *System $\cap J$ characterizes strong normalization, i.e. t is $\cap J$-typable if and only if t is $\to_{\mathtt{d}\beta}$-normalizing. Moreover, if $\Gamma \Vdash^n t : \sigma$ then the number of reduction steps in any reduction sequence from t to normal form is bounded by n.*

Proof. Soundness holds by Lem. 9, while completeness holds by Lem. 13. The bound is given by Thm. 9 in the long version [6].

4.3   Why π Is Not Quantitative

In the introduction we discussed that π is rejected by the quantitative type systems $\cap J$ for CBN. This happens in the critical case when $x \notin \mathrm{fv}(r)$ and $y \in \mathrm{fv}(r')$ in $t_0 = t(u, x.r)(u', y.r') \to_\pi t(u, x.r(u', y.r')) = t_1$. Let us see a concrete example.

Example 3. We take $t_1 = x(y, a.z)(w, b.b(b, c.c)) \to_\pi x(y, a.z(w, b.b(b, c.c))) = t_2$. Let $\rho_1 = [\sigma] \to \tau$ and $\rho_2 = [\sigma] \to [\tau] \to \tau$. For each $i \in \{1, 2\}$ let $\Delta_i = x : [\sigma_1]; y : [\sigma_2]; z : [\rho_i]$. Consider

$$\Psi = \dfrac{b : [[\tau] \to \tau] \Vdash b : [[\tau] \to \tau] \qquad b : [\tau] \Vdash b : [\tau] \qquad \overline{c : [\tau] \vdash c : \tau}}{b : [[\tau] \to \tau, \tau] \vdash b(b, c.c) : \tau}$$

and the derivation Φ_i for $i \in \{1, 2\}$:

$$\Phi_i = \dfrac{x : [\sigma_1] \Vdash x : [\sigma_1] \qquad y : [\sigma_2] \Vdash y : [\sigma_2] \qquad \overline{z : [\rho_i] \vdash z : \rho_i}}{\Delta_i \vdash x(y, a.z) : \rho_i}$$

Then, for the term t_1, we have the following derivation:

$$\dfrac{\dfrac{\Phi_1 \qquad \Phi_2}{\Delta_1 \wedge \Delta_2 \vdash x(y, a.z) : [\rho_1, \rho_2]} \qquad w : [\sigma, \sigma] \Vdash w : [\sigma, \sigma] \qquad \Psi}{\Gamma_1 \vdash x(y, a.z)(w, b.b(b, c.c)) : \tau}$$

where $\Gamma_1 = z : [\rho_1, \rho_2]; w : [\sigma, \sigma]; x : [\sigma_1, \sigma_1]; y : [\sigma_2, \sigma_2]$.

While for the term t_2, we have:

$$\dfrac{x : [\sigma_1] \Vdash x : [\sigma_1] \qquad y : [\sigma_2] \Vdash y : [\sigma_2] \qquad \Phi}{\Gamma_2 \vdash x(y, a.z(w, b.b(b, c.c))) : \tau}$$

where

$$\Phi = \dfrac{z : [\rho_1, \rho_2] \Vdash z : [\rho_1, \rho_2] \qquad w : [\sigma, \sigma] \Vdash w : [\sigma, \sigma] \qquad \Psi}{\Gamma_2 \vdash z(w, b.b(b, c.c)) : \tau}$$

and $\Gamma_2 = z : [\rho_1, \rho_2]; w : [\sigma, \sigma]; x : [\sigma_1]; y : [\sigma_2]$.

Thus, the multiset types of x and y in Γ_1 and Γ_2 resp. are not the same. Despite the fact that the step $t_1 \to_\pi t_2$ does not erase any subterm, the typing environment is losing quantitative information.

Notice that by replacing non-idempotent types by idempotent ones, subject reduction (and expansion) would work for π-reduction: by assigning sets to variables instead of multisets, Γ_1 and Γ_2 would now represent the same object.

Despite the fact that quantitative subject reduction fails for some π-steps, the following weaker property is sufficient to recover (qualitative) soundness of our typing system $\cap J$ w.r.t. the reduction relation $\to_{\beta,\pi}$. Soundness will be used later in Sec. 6 to show equivalence between $\mathcal{SN}(\mathsf{d}\beta)$ and $\mathcal{SN}(\beta, \pi)$.

Lemma 14 (Typing Behavior of π-Reduction). *Let $\Gamma \Vdash_{\cap J}^{n_1} t_1 : \sigma$. If $t_1 = t(u, x.r)(u', y.r') \mapsto_\pi t_2 = t(u, x.r(u', y.r'))$, then there are n_2 and $\Sigma \sqsubseteq \Gamma$ such that $\Sigma \Vdash_{\cap J}^{n_2} t_2 : \sigma$ with $n_1 \geq n_2$.*

Lemma 15 (Soundness for ΛJ).   *If t is $\cap J$-typable, then $t \in \mathcal{SN}(\beta, \pi)$.*

5   Faithfulness of the Translation

As discussed in the introduction, the natural translation [4] of generalized applications into ES is not faithful. In this section we define an alternative encoding and prove it faithful: a term in T_J is $d\beta$-strongly normalizing *iff* its alternative encoding is strongly normalizing in the ES framework. In a later subsection, we use this connection with ES to establish the equivalence between strong normalization w.r.t. $d\beta$ and (β, p_2).

5.1   Explicit Substitutions

We define the syntax and semantics of an ES calculus borrowed from [1] to which we relate λJ. It is a simple calculus where β is implemented in two independent steps: one creating a let-binding, and another one substituting the term bound. It has a notion of distance which allows to reduce redexes such as $([N/x](\lambda y.M))P \to_{dB} [N/x][P/y]M$, where the ES $[N/x]$ lies between the abstraction and its argument. Terms and list contexts are given by:

$$(T_{ES})\ M, N, P, Q ::= x \mid \lambda x.M \mid MN \mid [N/x]M$$
$$(\textbf{List contexts}) \qquad \texttt{L} ::= \Diamond \mid [N/x]\texttt{L}$$

The calculus λES is defined by $T_{ES}[dB, s]$ (closed under all contexts) where:

$$\texttt{L}\langle \lambda x.M \rangle N \mapsto_{dB} \texttt{L}\langle [N/x]M \rangle \qquad [N/x]M \mapsto_s \{N/x\}M$$

Now, consider the (naive) translation from T_J to T_{ES} [4]:

$$x^\star := x \qquad (\lambda x.t)^\star := \lambda x.t^\star \qquad t(u, y.r)^\star := [t^\star u^\star / y] r^\star$$

According to this translation, the notion of distance in λES corresponds to our notion of distance for λJ. For instance, the application $t(u, x.\cdot)$ in the term $t(u, x.\lambda y.r)(u', z.r')$ can be seen as a substitution $[t^\star u^\star / x]\cdot$ inserted between the abstraction $\lambda y.r$ and the argument u'. But how can we now (informally) relate π to the notions of existing permutations for λES? Using the previous translation, we can see that $t_0 = t(u, x.r)(u', y.r') \mapsto_\pi t(u, x.r(u', y.r')) = t_1$ simulates as

$$t_0^\star = [([t^\star u^\star / x]r^\star)u'^\star / y]r'^\star \to [[t^\star u^\star / x](r^\star u'^\star)/y]r'^\star \to [t^\star u^\star / x][r^\star u'^\star / y]r'^\star = t_1^\star.$$

The first step is an instance of a rule in ES known as σ_1: $([u/x]t)v \mapsto [u/x](tv)$, and the second one of a rule we call σ_4: $[[u/x]t/y]v \mapsto [u/x][t/y]v$. Quantitative types for ES tell us that only rule σ_1, but not rule σ_4, is valid for a call-by-name calculus. This is why it is not surprising that π is rejected by our type system, as detailed in Sec. 4.3.

The alternative encoding we propose is as follows (noted $\_^*$ instead of $\_^\star$):

Definition 4 (Translation from T_J to T_{ES}).

$$x^* := x \qquad (\lambda x.t)^* := \lambda x.t^* \qquad t(u, x.r)^* := [t^*/x^l][u^*/x^r]\{x^l x^r/x\}r^*$$

Notice the above π-reduction $t_0 \to t_1$ is still simulated: $t_0^* \to_{\sigma_4}^2 t_1^*$.

Consider again the counterexample to faithfulness already discussed in the introduction, given by $t := \delta^\circ(\delta^\circ, y.r)$ with $y \notin \mathrm{fv}(r)$, where $\delta^\circ = \lambda x.x(x, w.w)$. The term t is a $\mathsf{d}\beta$-redex, whose contraction throws away the two copies of δ°. The naive translation of t gives $[\delta^{\circ*}\delta^{\circ*}/y]r^\star$, which clearly diverges in λES. The alternative encoding of t is $[\delta^{\circ*}/y^l][\delta^{\circ*}/y^r]\{y^l y^r/y\}r^\star$, which is just $[\delta^{\circ*}/y^l][\delta^{\circ*}/y^r]r^\star$, because $y \notin \mathrm{fv}(r^*)$. The only hope to have an interaction between the two copies of $\delta^{\circ*}$ in the previous term is to execute the ES, but such executions will just throw away those two copies, because $y^l, y^r \notin \mathrm{fv}(r^*)$. This gives an intuitive idea of the faithfulness of our encoding.

5.2 Proof of Faithfulness

We need to prove the equivalence between two notions of strong normalization: the one of a term in λJ and the one of its encoding in λES. While this proof can be a bit involved using traditional methods, quantitative types will make it very straightforward. Indeed, since quantitative types correspond exactly to strong normalization, we only have to show that a term t is typable exactly when its encoding is typable, for two appropriate quantitative type systems.

For λES, we will use the following system [9]:

Definition 5 (The Type System $\cap ES$).

$$\frac{}{x : [\sigma] \vdash x : \sigma} \ (\texttt{var}) \qquad\qquad \frac{\Gamma; x : \mathcal{M} \vdash t : \sigma}{\Gamma \vdash \lambda x.M : \mathcal{M} \to \sigma} \ (\texttt{abs})$$

$$\frac{(\Gamma_i \vdash M : \sigma_i)_{i \in I} \quad I \neq []}{\wedge_{i \in I} \Gamma_i \vdash M : [\sigma_i]_{i \in I}} \ (\texttt{many}) \qquad \frac{\Gamma \vdash M : \mathcal{M} \to \sigma \quad \Delta \vdash N : \#(\mathcal{M})}{\Gamma \wedge \Delta \vdash MN : \sigma} \ (\texttt{app})$$

$$\frac{\Gamma; x : \mathcal{M} \vdash M : \sigma \quad \Delta \vdash N : \#(\mathcal{M})}{\Gamma \wedge \Delta \vdash [N/x]M : \sigma} \ (\texttt{sub})$$

Theorem 4. *Let $M \in T_{ES}$. Then M is typable in $\cap ES$ iff $M \in \mathcal{SN}(\mathtt{dB}, \mathtt{s})$.*

A simple induction on the type derivation shows that the encoding is sound.

Lemma 16. *Let $t \in T_J$. Then $\Gamma \Vdash_{\cap J} t : \sigma \implies \Gamma \Vdash_{\cap ES} t^* : \sigma$.*

We show completeness by a detour through the encoding of T_{ES} to T_J:

Definition 6 (Translation from T_{ES} to T_J).

$$\begin{aligned} x^\circ &:= x & (MN)^\circ &:= M^\circ(N^\circ, x.x) \\ (\lambda x.M)^\circ &:= \lambda x.M^\circ & ([N/x]M)^\circ &:= \mathtt{I}(N^\circ, x.M^\circ) \end{aligned}$$

The two following lemmas, shown by induction on the type derivations, give in particular that t^* typable implies t typable.

Lemma 17. *Let $M \in \mathtt{T}_{ES}$. Then $\Gamma \Vdash_{\cap ES} M : \sigma \implies \Gamma \Vdash_{\cap J} M^\circ : \sigma$.*

Lemma 18. *Let $t \in \mathtt{T}_J$. Then $\Gamma \Vdash_{\cap J} t^{*\circ} : \sigma \implies \Gamma \Vdash_{\cap J} t : \sigma$.*

Putting all together, we get this equivalence:

Corollary 1. *Let $t \in \mathtt{T}_J$. Then $\Gamma \Vdash_{\cap J} t : \sigma \iff \Gamma \Vdash_{\cap ES} t^* : \sigma$.*

This corollary, together with the two characterization theorems 3 and 4, provides the main result of this section:

Theorem 5 (Faithfulness). *Let $t \in \mathtt{T}_J$. Then $t \in \mathcal{SN}(\mathtt{d}\beta) \iff t^* \in \mathcal{SN}(\mathtt{dB}, \mathtt{s})$.*

6   Equivalent Notions of Strong Normalization

In the previous section, we related strong $\mathtt{d}\beta$-normalization with strong normalization of ES. In this section we will compare the various concepts of strong normalization that are induced on $\mathtt{T}_J$ by β, $\mathtt{d}\beta$, $(\beta, \mathtt{p}_2)$ and (β, π). This comparison will make use of several results obtained in the previous sections, and will obtain new results about the original calculus ΛJ.

6.1 β-Normalization Is Not Enough

We discussed in Sec. 2.2 about the unblocking property of π and $\mathtt{p}_2$. From the point of view of normalization, this means that $\mathtt{T}_J[\beta]$ has *premature* normal forms and that $\mathcal{SN}(\beta) \subsetneq \mathcal{SN}(\mathtt{d}\beta)$. To illustrate this purpose we give an example of a $\mathtt{T}_J$-term which normalizes when only using rule β, but diverges when adding permutation rules or distance. We write Ω the term $\delta^\circ(\delta^\circ, x.x)$, where $\delta^\circ = \lambda y.y(y, z.z)$, so that $\Omega \to_\beta \Omega$. Now, let us take $t := w(u, w'.\delta^\circ)(\delta^\circ, x.x)$. Although this term is normal in $\mathtt{T}_J[\beta]$, the second δ° is actually an argument for the first one, as we can see with a π permutation:

$$t \to_\pi w(u, w'.\delta^\circ(\delta^\circ, x.x)) = w(u, w'.\Omega) := t'$$

Thus $t \to_\pi t' \to_\beta t'$ which implies $t \notin \mathcal{SN}(\beta, \pi)$. We can also unblock the redex in t by a $\mathtt{p}_2$-permutation moving the inner λx up:

$$t \to_{\mathtt{p}_2} (\lambda y.w(u, w'.y(y, z.z)))(\delta^\circ, x.x) \to_\beta t'$$

Thus $t \to_{\mathtt{p}_2} \to_\beta t' \to_\beta t'$ and thus $t \notin \mathcal{SN}(\beta, \mathtt{p}_2)$. We get the same thing in a unique $\mathtt{d}\beta$-step: $t \to_{\mathtt{d}\beta} t'$.

In all the three cases, β-strong normalization is not preserved by the permutation rules, as there is a term $t \in \mathcal{SN}(\beta)$ such that $t \notin \mathcal{SN}(\beta, \pi)$, $t \notin \mathcal{SN}(\beta, \mathtt{p}_2)$ and $t \notin \mathcal{SN}(\mathtt{d}\beta)$.

6.2   Comparison with $\beta + \mathtt{p_2}$

We now formalize the fact that our calculus $\mathtt{T}_J[\mathsf{d}\beta]$ is a version with distance of $\mathtt{T}_J[\beta, \mathtt{p_2}]$, so that they are equivalent from a normalization point of view. For this, we will establish the equivalence between strong normalization w.r.t. $\mathsf{d}\beta$ and $(\beta, \mathtt{p_2})$, through a long chain of equivalences. One of them is Thm. 5, that we have proved in the previous section; the other is a result about σ-rules in the λ-calculus – which is why we have to go through the λ-calculus again.

Definition 7 (Translation from $\mathtt{T}_{ES}$ to $\mathtt{T}_\lambda$).

$$x^\sharp := x \quad (\lambda x.M)^\sharp := \lambda x.M^\sharp \quad (MN)^\sharp := M^\sharp N^\sharp \quad [N/x]M^\sharp := (\lambda x.M^\sharp)N^\sharp$$

Lemma 19. *Let $M \in \mathtt{T}_{ES}$. Then $M \in \mathcal{SN}(\mathtt{dB}, \mathtt{s}) \implies M^\sharp \in \mathcal{SN}(\beta)$.*

Proof. For typability in the λ-calculus, we use the type system $\mathcal{S}'_\lambda$ with choice operators in [9], which we rename here $\cap S$. It can be seen as a restriction of our system $\cap ES$ to λ-terms. Suppose $M \in \mathcal{SN}(\mathtt{dB}, \mathtt{s})$. By Thm. 4 M is typable in $\cap ES$, and it is straightforward to show that $M^\sharp$ is typable in $\cap S$. Moreover, $M^\sharp$ typable implies that $M^\sharp \in \mathcal{SN}(\beta)$ [9], which is what we want.

For $t \in \mathtt{T}_J$, let $t^\square := t^{*\sharp}$. So, we are just composing the alternative encoding of generalized application into ES with the map into λ-calculus just introduced. The λ-term $t^\square$ may be given by recursion on t as follows:

$$x^\square = x \qquad (\lambda x.t)^\square = \lambda x.t^\square \qquad t(u, y.r)^\square = (\lambda y^\mathrm{r}.(\lambda y^\mathrm{l}.\{y^\mathrm{l}y^\mathrm{r}/y\}r^\square)t^\square)u^\square$$

Lemma 20. $t^\square \in \mathcal{SN}(\beta, \sigma_2) \implies t \in \mathcal{SN}(\beta, \mathtt{p_2})$.

Proof. Because $(\cdot)^\square$ produces a strict simulation from $\mathtt{T}_J$ to $\mathtt{T}_\lambda$. More precisely: (i) if $t_1 \to_\beta t_2$ then $t_1^\square \to_\beta^+ t_2^\square$; (ii) if $t_1 \to_{\mathtt{p_2}} t_2$ then $t_1^\square \to_{\sigma_2}^2 t_2^\square$.

Theorem 6. *Let $t \in \mathtt{T}_J$. Then $t \in \mathcal{SN}(\beta, \mathtt{p_2})$ iff $t \in \mathcal{SN}(\mathsf{d}\beta)$.*

Proof. We prove that the following conditions are equivalent: 1) $t \in \mathcal{SN}(\beta, \mathtt{p_2})$. 2) $t \in \mathcal{SN}(\mathsf{d}\beta)$. 3) $t^* \in \mathcal{SN}(\mathtt{dB}, \mathtt{s})$. 4) $t^\square \in \mathcal{SN}(\beta)$. 5) $t^\square \in \mathcal{SN}(\beta, \sigma_2)$. Now, 1) $\implies$ 2) is because $\to_{\mathsf{d}\beta} \subset \to_{\beta, \mathtt{p_2}}^+$. 2) $\implies$ 3) is by Thm. 5. 3) $\implies$ 4) is by Lem. 19. 4) $\implies$ 5) is showed in [13]. 5) $\implies$ 1) is by Lem. 20.

6.3   Comparison with $\beta + \pi$

We now prove the equivalence between strong normalization for $\mathsf{d}\beta$ and for (β, π). One of the implications already follows from the properties of the typing system.

Lemma 21. *Let $t \in \mathtt{T}_J$. If $t \in \mathcal{SN}(\mathsf{d}\beta)$ then $t \in \mathcal{SN}(\beta, \pi)$.*

Proof. Follows from the completeness of the typing system (Lem. 13) and soundness of $\cap J$ for (β, π) (Lem. 15).

The proof of the other implication requires more work, organized in 4 parts: 1) A remark about ES. 2) A remark about translations of ES into the ΛJ-calculus. 3) Two new properties of strong normalization for (β, π) in ΛJ. 4) Preservation of strong (β, π)-normalization by a certain map from the set $\mathtt{T}_J$ into itself.

The remark about explicit substitutions is this:

Lemma 22. *For all* $M \in \mathtt{T}_{ES}$, $M \in \mathcal{SN}(\mathtt{dB}, \mathtt{s})$ *iff* $M \in \mathcal{SN}(\mathtt{B}, \mathtt{s})$.

The translation $\_^\circ$ in Def. 6 induces a simulation of each $\mathtt{s}$-reduction step on $\mathtt{T}_{ES}$ into a β-reduction step on $\mathtt{T}_J$, but cannot simulate the creation of an ES effected by rule $\mathtt{B}$. A solution is to refine the translation $\_^\circ$ for applications, yielding the following alternative translation:

$$x^\bullet := x \qquad\qquad (\lambda x.M)^\bullet := \lambda x.M^\bullet$$
$$(MN)^\bullet := I(N^\bullet, y.M^\bullet(y, z.z)) \qquad [N/x]M^\bullet := I(N^\bullet, x.M^\bullet)$$

Since the clause for ES is not changed, simulation of each $\mathtt{s}$-reduction step by a β-reduction step holds as before. The improvement lies in the simulation of each $\mathtt{B}$-reduction step:

$$((\lambda x.M)N)^\bullet = I(N^\bullet, y.(\lambda x.M^\bullet)(y, z.z)) \to_\beta I(N^\bullet, y.\{y/x\}M^\bullet) =_\alpha ([N/x]M)^\bullet$$

This strict simulation gives immediately:

Lemma 23. *For all* $M \in \mathtt{T}_{ES}$, *if* $M^\bullet \in \mathcal{SN}(\beta)$ *then* $M \in \mathcal{SN}(\mathtt{B}, \mathtt{s})$.

We now prove two properties of strong normalization for (β, π) in ΛJ. Following [10], $\mathcal{SN}(\beta, \pi)$ admits an inductive characterization $\mathcal{ISN}(\beta, \pi)$, which uses the following inductive generation for $\mathtt{T}_J$-terms:

$$t, u, r ::= x\boldsymbol{S} \mid \lambda x.t \mid (\lambda x.t)S\boldsymbol{S} \qquad S ::= (u, y.r)$$

Hence S stands for a *generalized* argument, while $\boldsymbol{S}$ denotes a possibly empty list of S's. The definition of $\mathcal{ISN}(\beta, \pi)$ is given below. Notice that at most one rule applies to a given term, so the rules are deterministic (and thus invertible).

$$\frac{}{x \in \mathcal{ISN}(\beta, \pi)} (var) \qquad \frac{u, r \in \mathcal{ISN}(\beta, \pi)}{x(u, z.r) \in \mathcal{ISN}(\beta, \pi)} (hvar) \qquad \frac{t \in \mathcal{ISN}(\beta, \pi)}{\lambda x.t \in \mathcal{ISN}(\beta, \pi)} (lambda)$$

$$\frac{x(u, y.r S)\boldsymbol{S} \in \mathcal{ISN}(\beta, \pi)}{x(u, y.r)S\boldsymbol{S} \in \mathcal{ISN}(\beta, \pi)} (pi) \qquad \frac{\{\{u/x\}t/y\}r\boldsymbol{S} \in \mathcal{ISN}(\beta, \pi) \quad t, u \in \mathcal{ISN}(\beta, \pi)}{(\lambda x.t)(u, y.r)\boldsymbol{S} \in \mathcal{ISN}(\beta, \pi)} (beta)$$

A preliminary fact is the following:

Lemma 24. $\mathcal{SN}(\beta, \pi)$ *is closed under prefixing of arbitrary* π-*reduction steps:*

$$\frac{t \to_\pi t' \text{ and } t' \in \mathcal{SN}(\beta, \pi)}{t \in \mathcal{SN}(\beta, \pi)}$$

Given that $\mathcal{SN}(\beta, \pi) = \mathcal{ISN}(\beta, \pi)$, the "rule" in Lem. 24, when written with $\mathcal{ISN}(\beta, \pi)$, is admissible for the predicate $\mathcal{ISN}(\beta, \pi)$. Now, consider:

$$\frac{u, r \in \mathcal{ISN}(\beta, \pi)}{\{y(u, z.z)/x\}r \in \mathcal{ISN}(\beta, \pi)} \, (I)$$

$$\frac{\{\{\{u/y\}t/z\}r/x\}r \in \mathcal{ISN}(\beta, \pi) \qquad t, u \in \mathcal{ISN}(\beta, \pi) \qquad x \notin \mathrm{fv}(t, u, r)}{\{(\lambda y.t)(u, z.r)/x\}r \in \mathcal{ISN}(\beta, \pi)} \, (II)$$

Notice rule II generalizes rule (*beta*): just take $r = x\mathbf{S}$, with $x \notin \mathbf{S}$.

The two new properties of strong normalization for (β, π) in ΛJ are contained in the following Lemma.

Lemma 25. *Rules I and II are admissible rules for the predicate $\mathcal{ISN}(\beta, \pi)$.*

We now move to the fourth part of the ongoing reasoning. Consider the map from $\mathtt{T}_J$ to itself obtained by composing $(\cdot)^* : \mathtt{T}_J \to \mathtt{T}_{ES}$ with $(\cdot)^\bullet : \mathtt{T}_{ES} \to \mathtt{T}_J$. Let us write $t^\dagger := t^{*\bullet}$. A recursive definition is also possible, as follows:

$$x^\dagger = x \qquad \lambda x.t^\dagger = \lambda x.t^\dagger \qquad t(u, y.v)^\dagger = I(t^\dagger, y_1.I(u^\dagger, y_2.\{y_1(y_2, z.z)/y\}v^\dagger))$$

Lemma 26. *If $t \in \mathcal{SN}(\beta, \pi)$ then $t^\dagger \in \mathcal{SN}(\beta, \pi)$.*

Proof. Heavy use is made of Lem. 24 and Lem. 25.

All is in place to obtain the desired result:

Theorem 7. *Let $t \in \mathtt{T}_J$. $t \in \mathcal{SN}(\mathtt{d}\beta)$ iff $t \in \mathcal{SN}(\beta, \pi)$.*

Proof. The implication from left to right is Lem. 21. For the converse, suppose $t \in \mathcal{SN}(\beta, \pi)$. By Lem. 26, $t^\dagger \in \mathcal{SN}(\beta, \pi)$. Trivially, $t^\dagger \in \mathcal{SN}(\beta)$. Since $t^\dagger = t^{*\bullet}$, Lem. 23 gives $t^* \in \mathcal{SN}(\mathtt{B}, \mathtt{s})$. By Lem. 22, $t^* \in \mathcal{SN}(\mathtt{dB}, \mathtt{s})$. By an equivalence in the proof of Thm. 6, $t \in \mathcal{SN}(\mathtt{d}\beta)$.

6.4   Consequences for ΛJ

The comparison with λJ gives new results about the original ΛJ (a quantitative typing system characterizing strong normalization, and a faithful translation into ES) as immediate consequences of Thms. 3, 5, and 7.

Corollary 2. *Let $t \in \mathtt{T}_J$. (1) $t \in \mathcal{SN}(\beta, \pi)$ iff t is $\cap J$-typable. (2) $t \in \mathcal{SN}(\beta, \pi)$ iff $t^* \in \mathcal{SN}(\mathtt{dB}, \mathtt{s})$.*

Beyond strong normalization, ΛJ gains a new normalizing strategy, which reuses the notion of weak-head normal form introduced in Sec. 3.2. We take the definitions of neutral terms, answer and weak-head context $\mathtt{W}$ there for λJ, in order to define a new weak-head strategy and a new predicate ISN for ΛJ. The strategy is defined as the closure under $\mathtt{W}$ of rule β and of the particular case of rule π where the redex has the form $\mathtt{n}(u, x.\mathtt{a})S$[5].

---

[5] Notice how a redex has the two possible forms $(\lambda x.t)S$ or $\mathtt{n}(u, x.\mathtt{a})S$, that can be written as $\mathtt{a}S$, that is, the form $\mathtt{D}_\mathtt{n}\langle \lambda x.t \rangle S$ of a weak-head redex in λJ

Definition 8. *Predicate ISN is defined by the rules* (snvar)*,* (snapp)*,* (snabs) *in Def. 3, together with the following two rules (which replace rule* (snbeta)*):*

$$\frac{\mathtt{W}\langle\mathtt{n}(u,y.\mathtt{a}S)\rangle \in ISN}{\mathtt{W}\langle\mathtt{n}(u,y.\mathtt{a})S\rangle \in ISN}\,(\mathtt{snredex1}) \qquad \frac{\mathtt{W}\langle\{\{u/x\}t/y\}r\rangle, t, u \in ISN}{\mathtt{W}\langle(\lambda x.t)(u,y.r)\rangle \in ISN}\,(\mathtt{snredex2})$$

The corresponding normalization strategy is organized as usual: an initial phase obtains a weak-head normal form, whose components are then reduced by internal reduction. Is this new strategy any good? The last theorem of the paper answers positively:

Theorem 8. *Let* $t \in \mathtt{T}_J$. $t \in ISN$ *iff* $t \in \mathcal{ISN}(\beta, \pi)$.

7   Conclusion

Contributions. This paper presents and studies several properties of the call-by-name λJ-calculus, a formalism implementing an appropriate notion of distant reduction to unblock the β-redexes arising in generalized application notation.

Strong normalization of simple typed terms was shown by translating the λJ into the λ-calculus. A full characterization of strong normalization was developed by means of a quantitative type system, where the length of $\mathtt{d}\beta$-reduction to normal form is bound by the size of the type derivation of the starting term. An inductive definition of $\mathtt{d}\beta$-strong normalization was defined and proved correct in order to achieve this characterization. It was also shown how the traditional permutative rule π is rejected by the quantitative system, thus emphasizing the choice of $\mathtt{d}\beta$-reduction for a quantitative generalized application framework.

We have also defined a faithful translation from the λJ-calculus into ES. The translation preserves strong normalization, in contrast to the traditional translation to ES *e.g.* in [4]. Last but not least, we related strong normalization of λJ with that of other calculi, including in particular the original ΛJ. New results for the latter were found by means of the techniques developed for λJ. In particular, a quantitative characterization of strong normalization was developed for ΛJ, where the bound of reduction given by the size of type derivations only holds for β-steps (and not for π-steps).

Future work. Regarding call-by-name for generalized applications, this paper opens new questions. We studied a new calculus λJ, proposed as an alternative to the original ΛJ, but we also mentioned some possible variants in Sec. 2.2, notably a calculus based on rule (1), and $\beta + \mathtt{p_2}$ (used as a technical tool in Sec. 6). The first option seems to have the flavor of ΛJ whereas the $\beta + \mathtt{p_2}$ option seems to have the flavor of λJ. It remains to be seen what are the advantages and drawbacks of the latter one with respect to λJ.

Regarding call-by-value, we plan to develop the quantitative semantics in the presence of generalized applications, starting from the calculus proposed in [5]. Further unification between call-by-name and call-by-value with the help of generalized applications could be considered in the setting of the polarized lambda-calculus [4].

References

1. Accattoli, B.: An abstract factorization theorem for explicit substitutions. In: Tiwari, A. (ed.) 23rd International Conference on Rewriting Techniques and Applications (RTA'12) , RTA 2012, May 28 - June 2, 2012, Nagoya, Japan. LIPIcs, vol. 15, pp. 6–21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2012)
2. Accattoli, B., Kesner, D.: The structural λ-calculus. In: Dawar, A., Veith, H. (eds.) Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6247, pp. 381–395. Springer (2010)
3. Espírito Santo, J., Frade, M.J., Pinto, L.: Structural proof theory as rewriting. In: Pfenning, F. (ed.) Term Rewriting and Applications. pp. 197–211. Lecture Notes in Computer Science, Springer (2006)
4. Espírito Santo, J.: Delayed substitutions. In: Lecture Notes in Computer Science, pp. 169–183. Springer Berlin Heidelberg (2007)
5. Espírito Santo, J.: The call-by-value lambda-calculus with generalized applications. In: CSL. LIPIcs, vol. 152, pp. 35:1–35:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)
6. Espírito Santo, J., Kesner, D., Peyrot, L.: A faithful and quantitative notion of distant reduction for generalized applications. CoRR **abs/2201.04156** (2022)
7. Espírito Santo, J., Pinto, L.: Permutative conversions in intuitionistic multiary sequent calculi with cuts. In: Hofmann, M. (ed.) Typed Lambda Calculi and Applications. pp. 286–300. Lecture Notes in Computer Science, Springer (2003)
8. Joachimski, F., Matthes, R.: Standardization and confluence for a lambda calculus with generalized applications. In: Rewriting Techniques and Applications, pp. 141–155. Springer Berlin Heidelberg (2000)
9. Kesner, D., Vial, P.: Non-idempotent types for classical calculi in natural deduction style. Log. Methods Comput. Sci. **16**(1) (2020)
10. Matthes, R.: Characterizing strongly normalizing terms of a calculus with generalized applications via intersection types. In: Rolim, J.D.P., Broder, A.Z., Corradini, A., Gorrieri, R., Heckel, R., Hromkovic, J., Vaccaro, U., Wells, J.B. (eds.) ICALP Workshops 2000, Proceedings of the Satelite Workshops of the 27th International Colloquium on Automata, Languages and Programming, Geneva, Switzerland, July 9-15, 2000. pp. 339–354. Carleton Scientific, Waterloo, Ontario, Canada (2000)
11. von Plato, J.: Natural deduction with general elimination rules. Arch. Math. Log. **40**(7), 541–567 (2001)
12. van Raamsdonk, F.: Confluence and normalisation for higher-order rewriting. Ph.D. thesis, Vrije Universiteit Amsterdam (May 1996)
13. Regnier, L.: Une équivalence sur les lambda-termes. Theor. Comput. Sci. **126**(2), 281–292 (1994)

Modal Logics and Local Quantifiers: A Zoo in the Elementary Hierarchy

Raul Fervari[1] and Alessio Mansutti[2](✉)

[1] CONICET and Universidad Nacional de Córdoba, Córdoba, Argentina
`rfervari@unc.edu.ar`
[2] Department of Computer Science, University of Oxford, Oxford, UK
`alessio.mansutti@cs.ox.ac.uk`

Abstract. We study a family of *modal logics* interpreted on tree-like structures, and featuring *local quantifiers* $\exists^k p$ that bind the proposition p to worlds that are accessible from the current one in at most k steps. We consider a first-order and a second-order semantics for the quantifiers, which enables us to relate several well-known formalisms, such as *hybrid logics*, S5Q and *graded modal logic*. To better stress these connections, we explore fragments of our logics, called herein *round-bounded* fragments. Depending on whether first or second-order semantics is considered, these fragments populate the hierarchy $2\text{NEXP} \subset 3\text{NEXP} \subset \cdots$ or the hierarchy $2\text{AEXP}_{pol} \subset 3\text{AEXP}_{pol} \subset \cdots$, respectively. For formulae up-to modal depth k, the complexity improves by one exponential.

1 Introduction

From a traditional perspective, *modal logics* [10] are formalisms to reason about different modes of truth. However, another view consists of seeing these logics as computationally well-behaved fragments of *first-order logic* and *second-order logic* (see e.g., [1] for a discussion). Some examples of well-known modal logics with a good balance between expressivity and computational complexity are *graded modal logic* (GML) [5,28], whose satisfiability problem is PSPACE-complete; and the *temporal logics* LTL, CTL and CTL* whose satisfiability problems are complete for PSPACE, EXP and 2EXP, respectively [31,19,25].

A family of logics that elude this nice computational picture is that made of modal logics enriched with first-order or second-order propositional quantifiers $\exists p$, which update the set of worlds of a Kripke structure that satisfy the propositional symbol p. The literature of modal logics featuring quantification over propositional symbols can be traced back to [12,26,18]. All these works show that, in spite of the simplicity of the principle, propositional quantification leads to undecidability very quickly. One of the few exceptions is the logic S5Q, i.e. S5 enriched with *second-order propositional quantifiers*, which enjoys an exponential-size small model property, and is thus decidable [22,18]. Here, the success in finding a well-behaved framework for propositional quantification is due to the fact that S5 has a very restricted class of models. In modern literature, the family of *hybrid logics* [2] is one of the most relevant approaches

offering *first-order propositional quantification*. Most hybrid logics provide *operators* $\downarrow i$ that binds the current world to the proposition i, and $@_i$ that allows to jump to the world bound to i. This form of quantification is very expressive, and leads to undecidability over standard Kripke structures [3]. To regain decidability, one can restrict the logic to syntactical fragments that avoid the quantification patters $\Box\downarrow$ and $\Diamond\downarrow\Diamond$, or restrict the interpretation to models in which each world has at most two successors [14]. Again, one can also simply consider S5 models: the hybrid logic with $\downarrow$ and @ on S5 is known to admit an NExp-complete satisfiability problem [30].

Recent works shed new lights on the role of propositional quantifiers. From a model theoretical perspective, a revision about the different forms of propositional quantification has been put forward in [9]. Novel algebraic insights on S5 with propositional quantification have been discovered in [17]. From a computational perspective, [6] shows that second-order propositional quantification is enough to obtain Tower-complete (hence, non-elementary decidable, [29]) logics on tree-like structures. This last result is of interest, as the second-order logic $\mathsf{QCTL}_\mathbf{X}^t$ considered in [6] subsumes several other modal logics with forms of quantification "in disguise", such as the aforementioned GML, as well as *modal separation logics* [16], *ambient logics* [13] and *team logics* [21]. However, when translated into $\mathsf{QCTL}_\mathbf{X}^t$, the good computational properties of these logics are lost, and the Tower-hardness of $\mathsf{QCTL}_\mathbf{X}^t$ prevents us to grasp the real capabilities of their (often restricted) form of propositional quantifications.

Contributions. The overall message of [6] is that the computational power of propositional quantification in the context of modal logic deserves to be better understood. Driven by this message, we investigate from a unified perspective a family of logics interpreted on tree-like models, featuring a very intuitive form of propositional quantification: the *local quantifier* $\exists^k p$, with $k \geq 1$ integer, that binds the propositional symbol p to world(s) occurring within distance k from the current point of evaluation. More precisely, we look at two families of modal logics: the family $\mathsf{ML}(\exists_{FO}^1), \mathsf{ML}(\exists_{FO}^2), \cdots$, where $\mathsf{ML}(\exists_{FO}^k)$ extends the basic modal logic ML with the *first-order* local quantifier $\exists^k p$ binding p to exactly one world occurring within distance k of the current world; and the family $\mathsf{ML}(\exists_{SO}^1), \mathsf{ML}(\exists_{SO}^2), \cdots$, where $\mathsf{ML}(\exists_{SO}^k)$ extends ML with the *second-order* local quantifier $\exists^k p$ binding p to a set of worlds occurring within distance k.

As previously mentioned, in introducing these logics our aim is to better understand the similarities and differences between the various modal logics featuring propositional quantification, especially when it comes to their complexity. This analysis cannot be done using Tower-complete logics like $\mathsf{QCTL}_\mathbf{X}^t$, as finer complexity classes are required. In this sense, it is worth to notice that our framework features the logic $\mathsf{ML}(\exists_{SO}^\infty)$, whose quantifier $\exists^\infty p$ binds p to arbitrary worlds reachable from the current one. This is exactly the logic $\mathsf{QCTL}_\mathbf{X}^t$. Because of this connection and of similarities with other frameworks, e.g. [7], we argue that even if we restrict ourselves to quantifiers $\exists^k$ with small k, the complexity does not improve. In fact, $\mathsf{ML}(\exists_{FO}^2)$ is already Tower-complete, although we defer this result to an extended version of the paper, due to the lack of space.

Consequently, to pursue our goal of a fine-grained analysis of the computational power of propositional quantification in modal logic, in this paper we focus on a syntactical restriction for $\mathsf{ML}(\exists_{FO}^k)$ and $\mathsf{ML}(\exists_{SO}^k)$ where the local quantifiers are *round-bounded* (Sec. 2). Roughly speaking, under the round-bounded condition, $\mathsf{ML}(\exists_{FO}^k)$ and $\mathsf{ML}(\exists_{SO}^k)$ formulae can be split into parts having k nested modalities. Quantifiers belonging to one part of the formula do not interact with quantifiers from other parts of the formula. The following results are established.

Theorem 1. *The sat. problem for round-bounded $\mathsf{ML}(\exists_{FO}^k)$ is $(k+1)\mathrm{NEXP}$-complete. It is $k\mathrm{NEXP}$-complete for formulae of $\mathsf{ML}(\exists_{FO}^k)$ of modal depth k.*

Theorem 2. *The sat. problem for round-bounded $\mathsf{ML}(\exists_{SO}^k)$ is $(k+1)\mathrm{AEXP}_{pol}$-complete. It is $k\mathrm{AEXP}_{pol}$-complete for formulae of $\mathsf{ML}(\exists_{SO}^k)$ of modal depth k.*

Here and along the paper, given natural numbers $k, n \geq 1$, we write t for the tetration function inductively defined as $\mathsf{t}(0,n) \overset{\text{def}}{=} n$ and $\mathsf{t}(k,n) = 2^{\mathsf{t}(k-1,n)}$. Intuitively, $\mathsf{t}(k,n)$ defines a tower of exponentials of height k. Then, $k\mathrm{NEXP}$ is the class of all problems decidable by a non-deterministic Turing machine running in time $\mathsf{t}(k, f(n))$, for some polynomial f, on each input of length n; whereas $k\mathrm{AEXP}_{pol}$ is the class of all problems decidable with an alternating Turing machine [15] in time $\mathsf{t}(k, f(n))$ and performing at most $g(n)$ alternations, for some polynomials f, g, on each input of length n. For all $k \geq 1$, $k\mathrm{NEXP} \subseteq k\mathrm{AEXP}_{pol} \subseteq \mathrm{TOWER}$, as we recall that TOWER is the class of all problems decidable with a Turing machine running in time $\mathsf{t}(g(n), f(n))$ for some polynomial f and elementary function g, on each input of length n [29]. The lower bounds of Thms. 1 and 2 are established by reduction from suitable tiling problems (Sec. 3). The upper bounds are established by designing a quantifier elimination procedure that yields a $(k+1)\mathrm{EXPSPACE}$ small-model property for round-bounded $\mathsf{ML}(\exists_{SO}^k)$, and a $k\mathrm{EXPSPACE}$ small-model property for the set of formulae of $\mathsf{ML}(\exists_{SO}^k)$ of modal depth k (Sec. 4). The round-bounded condition does not change the set of formulae of $\mathsf{ML}(\exists_{FO}^1)$ and $\mathsf{ML}(\exists_{SO}^1)$, and thus, as a corollary, we characterise the complexity of these logics:

Corollary 1. *(I) The sat. problem for $\mathsf{ML}(\exists_{FO}^1)$ is $2\mathrm{NEXP}$-complete.*
          (II) The sat. problem for $\mathsf{ML}(\exists_{SO}^1)$ is $2\mathrm{AEXP}_{pol}$-complete.

As promised, our framework yields a refined analysis on the power of propositional quantification in modal logic, which we compare to previous known results in Sec. 2. Quite surprisingly, we show that, on tree-like models, modal logic enriched with propositional quantifiers is as expressive as graded modal logic. Moreover, we establish that $\mathsf{S5Q}$ is AEXP_{pol}-complete (refining the previous results from [22,18]), and that hybrid logic with $\downarrow$ and @ on trees is TOWER-complete.

2   Preliminaries

The symbol $\mathbb{N}$ (resp. $\mathbb{N}_+$) denotes the set of *natural numbers* including (resp. excluding) zero, $\overline{\mathbb{N}}$ denotes the set $\mathbb{N} \cup \{\infty\}$, where $n < \infty$, $\infty + n = \infty$ and $n \bmod \infty = n$ for all $n \in \mathbb{N}$, and $\overline{\mathbb{N}}_+ \overset{\text{def}}{=} \overline{\mathbb{N}} \setminus \{0\}$. We write $|S| \in \overline{\mathbb{N}}$ for the size of a set S. Finally, let $\mathsf{AP} = \{p, q, r, \dots\}$ be a countable set of *atomic propositions*.

Kripke structures. A *Kripke structure* is a triple $\mathcal{K} = (\mathcal{W}, R, \mathcal{V})$ where $\mathcal{W}$ is a non-empty set of *worlds*, $\mathcal{V} \colon \mathsf{AP} \to 2^{\mathcal{W}}$ is a *valuation*, and $R \subseteq \mathcal{W} \times \mathcal{W}$ is a binary *accessibility relation*. A *Kripke-style forest* is a Kripke structure whose accessibility relation R is such that its inverse R^{-1} is functional and acyclic. In particular, the graph described by $\mathcal{K}$ is a collection of disjoint trees, where R encodes the child relation. We write $R(w)$ for the set of *children* of w, i.e. $\{w' \in \mathcal{W} : (w, w') \in R\}$. For $i \in \mathbb{N}$, R^i is the *i-th composition* of R: R^0 is the identity map on $\mathcal{W}$, and $R^{i+1} \stackrel{\text{def}}{=} \{(w, w') \in \mathcal{W} \times \mathcal{W} : (w, w'') \in R^i$ and $(w'', w') \in R$, for some $w'' \in \mathcal{W}\}$. For $n, m \in \overline{\mathbb{N}}$, $R^{[n,m]} \stackrel{\text{def}}{=} \bigcup_{j=n}^{m} R^j$, and $R^* \stackrel{\text{def}}{=} R^{[0,\infty]}$ is the *Kleene closure* of R. For $\mathcal{W}' \subseteq \mathcal{W}$, $\mathcal{V}[p \leftarrow \mathcal{W}']$ is the valuation obtained from $\mathcal{V}$ by updating to $\mathcal{W}'$ the set assigned to $p \in \mathsf{AP}$. A *pointed forest* $(\mathcal{K}, w)$ is a Kripke-style finite forest $\mathcal{K}$ together with one of its worlds w.

Modal logic with local quantifiers. For $k \in \overline{\mathbb{N}}_+$ written in unary, we introduce the modal logic $\mathsf{ML}(\exists^k)$, whose formulae φ, ψ, χ, etc., are from the grammar below:

$$\varphi, \psi \ := \ \top \ | \ p \ | \ \varphi \wedge \psi \ | \ \neg\varphi \ | \ \Diamond\varphi \ | \ \exists^k p \, \varphi, \qquad \text{where } p \in \mathsf{AP}.$$

We call $\exists^k p$ a *local (existential) quantifier*. We are interested in two interpretations for the logic $\mathsf{ML}(\exists^k)$, one where the local quantifier $\exists^k p$ performs a first-order quantification, and one where it performs a second-order one. For simplicity, $\mathsf{ML}(\exists^k_{FO})$ (resp. $\mathsf{ML}(\exists^k_{SO})$) stands for $\mathsf{ML}(\exists^k)$ interpreted under first-order (resp. second-order) semantics. The basic modal logic ML is obtained by removing the constructor $\exists^k p \, \varphi$ from the grammar.

Let $(\mathcal{K}, w)$ be a pointed forest, where $\mathcal{K} = (\mathcal{W}, R, \mathcal{V})$. For formulae of $\mathsf{ML}(\exists^k_{FO})$, the satisfaction relation $\models$ is defined as follows (Boolean cases are omitted):

$$\mathcal{K}, w \models p \quad \Leftrightarrow w \in \mathcal{V}(p); \quad \mathcal{K}, w \models \Diamond\varphi \ \Leftrightarrow \ \text{there is } w' \in R(w) \text{ s.t. } \mathcal{K}, w' \models \varphi;$$

$$\mathcal{K}, w \models \exists^k p \, \varphi \Leftrightarrow \text{there is } w' \in R^{[0,k]}(w) \text{ such that } (\mathcal{W}, R, \mathcal{V}[p \leftarrow \{w'\}]), w \models \varphi.$$

An atomic proposition p is said to be a *nominal* for $(\mathcal{K}, w)$ whenever $|\mathcal{V}(p)| = 1$. Additionally, p is *i-local* whenever $\mathcal{V}(p) \subseteq R^i(w)$. In particular, the first-order quantification $\exists^k p \, \varphi$ leads to φ being evaluated in a pointed forest where p is an *i-local nominal* for some $i \in [0, k]$. Given a nominal p, we call $w \in \mathcal{V}(p)$ the *world corresponding to p*, and often denote it by w_p.

For formulae of the *second-order logic* $\mathsf{ML}(\exists^k_{SO})$, the interpretation of the ML fragment remains as for $\mathsf{ML}(\exists^k_{FO})$, whereas we reinterpret the local quantifier as:

$$\mathcal{K}, w \models \exists^k p \, \varphi \Leftrightarrow \text{there is a set } \mathcal{W}' \subseteq R^{[0,k]}(w) \text{ s.t. } (\mathcal{W}, R, \mathcal{V}[p \leftarrow \mathcal{W}']), w \models \varphi.$$

The contradiction $\bot$ and connectives $\vee$, $\Rightarrow$ and $\Leftrightarrow$ are defined as usual. Below, let φ and ψ be two formulae of $\mathsf{ML}(\exists^k)$. The *local universal quantifier* $\forall^k p \, \varphi$ and the modality $\Box\varphi$ are defined as $\neg\exists^k p \, \neg\varphi$ and $\neg\Diamond\neg\varphi$, respectively. We define $\Diamond^0\varphi \stackrel{\text{def}}{=} \varphi$, and given $i \in \mathbb{N}$, $\Diamond^{i+1}\varphi \stackrel{\text{def}}{=} \Diamond^i\Diamond\varphi$. Similarly, $\Box^i\varphi \stackrel{\text{def}}{=} \neg\Diamond^i\neg\varphi$. We write $@^i_p\varphi$ for $\Diamond^i(p \wedge \varphi)$. If p is a nominal, the formula $@^i_p\varphi$ states that p is *i-local*, and that its corresponding world satisfies φ. We define $\spadesuit^0\varphi \stackrel{\text{def}}{=} \varphi$ and $\boxplus^0\varphi \stackrel{\text{def}}{=} \varphi$, and given $i \in \mathbb{N}$, $\spadesuit^{i+1}\varphi \stackrel{\text{def}}{=} \varphi \vee \Diamond \spadesuit^i \varphi$ and $\boxplus^{i+1}\varphi \stackrel{\text{def}}{=} \varphi \wedge \Box \boxplus^i \varphi$. We use the operator precedence $\{\neg, \Diamond, \Box, \exists^k, \forall^k, @^i_p\} < \{\wedge, \vee\} < \{\Rightarrow, \Leftrightarrow\}$, and sometimes

write ":" after a local quantifier with the intuitive meaning that the formula on the right of ":" should be enclosed in brackets, e.g. $\exists^2 p : \varphi \wedge \psi$ abbreviates $\exists^2 p\,(\varphi \wedge \psi)$. Given $i \in \mathbb{N}$, we write $\varphi[\psi \leftarrow_i \chi]$ for the formula obtained from φ by simultaneously substituting with χ each occurrence of the formula ψ appearing under the scope of exactly i nested modalities.

The *length* of φ, denoted with $|\varphi|$, is the number of symbols needed to represent φ. The *modal depth* $\mathsf{md}(\varphi)$ of φ is the maximal number of nested modalities occurring in φ. We write $\mathsf{bp}(\varphi)$ for the set of *bound propositions* of φ, i.e. propositions p that occur in a quantifier $\exists^k p$ inside φ. We say that φ is *well-quantified* whenever each subformula $\exists^k p\,\psi$ of φ quantifies on a different $p \in \mathsf{AP}$, and every occurrence of p in ψ appears under the scope of at most k modalities. One can translate every formula into a well-quantified one at no cost: atomic propositions can be renamed, and occurrences of a quantified atomic proposition that are under the scope of more than k modalities can be replaced with $\bot$.

We write $\varphi \equiv_{FO} \psi$ (resp. $\varphi \equiv_{SO} \psi$) whenever φ and ψ are equivalent under their first-order (resp. second-order) semantics, i.e. they are satisfied by the same pointed forests. When clear from the context or true under both semantics, we drop the subscripts and write $\varphi \equiv \psi$. Notice that $\exists^k p\,\varphi \equiv \exists^{k+1} p\,(\varphi \wedge \Box^{k+1} \neg p)$, and thus $\mathsf{ML}(\exists^k)$ is a syntactical fragment of $\mathsf{ML}(\exists^{k+1})$, and it is able to express all the local quantifiers $\exists^1 p, \ldots, \exists^k p$.

Round-bounded fragment. As discussed in Sec. 1, in this paper we focus on a syntactical restriction for $\mathsf{ML}(\exists^k)$ where the local quantifiers are *round-bounded*. The round-bounded formulae of $\mathsf{ML}(\exists^k)$ are those generated from the symbol φ_0^k of the grammar below ($j \in \mathbb{N}$):

$$\varphi_j^k, \psi_j^k := \top \ \mid \ p \ \mid \ \varphi_j^k \wedge \psi_j^k \ \mid \ \neg\varphi_j^k \ \mid \ \Diamond\varphi_{j+1}^k \ \mid \ \exists^{k-(j \bmod k)} p\,\varphi_j^k, \ \text{where } p \in \mathsf{AP}.$$

In a round-bounded formula of $\mathsf{ML}(\exists^k)$, quantifiers appearing under the scope of j modalities are restricted to $\exists^{k-(j \bmod k)}$, e.g. $\exists^3 p\,\Diamond\exists^2 q\,\Diamond\exists^1 r\,\Diamond\exists^3 p\,\varphi$ is a round-bounded formula of $\mathsf{ML}(\exists^3)$, provided that φ is also in this fragment, whereas $\exists^3 p\,\Diamond\exists^3 q\,\varphi$ is not round-bounded. The round-bounded condition does not change the set of formulae of $\mathsf{ML}(\exists^1)$ and $\mathsf{ML}(\exists^\infty)$. Besides, every formula of $\mathsf{ML}(\exists^\infty)$ of modal depth k is equivalent to a round-bounded formula of $\mathsf{ML}(\exists^k)$, of similar size, since given a formula φ of $\mathsf{ML}(\exists^\infty)$, we have $\exists^\infty p\,\varphi \equiv \exists^{\mathsf{md}(\varphi)} p\,\varphi$.

Our framework of local quantifiers enables us to derive connections with other modal logics featuring some form of quantification, which we now briefly discuss.

Graded modal logic. A logic that has been shown related to different forms of quantification is the *graded modal logic* GML [5], that extends ML with modalities $\Diamond_{\geq \ell}$ ($\ell \in \mathbb{N}$), with semantics: $\mathcal{K}, w \models \Diamond_{\geq \ell}\varphi \Leftrightarrow |\{w' \in R(w) \mid \mathcal{K}, w' \models \varphi\}| \geq \ell$. GML has a tree model property, i.e., each of its satisfiable formulae is satisfied by a pointed forest. Then, by syntactically replacing each $\Diamond_{\geq \ell}\varphi$ occurring in a GML formula by $\exists^1 \mathsf{x}_1, \ldots, \mathsf{x}_\ell : (\bigwedge_{i=0}^{\ell} \bigwedge_{j=i+1}^{\ell} @_{\mathsf{x}_i}^1 \neg\mathsf{x}_j) \wedge \Box((\bigvee_{i=0}^{\ell} \mathsf{x}_i) \Rightarrow \varphi)$, one shows that GML embeds in $\mathsf{ML}(\exists^1_{FO})$. At this point, it is worth noting that, for all $k \in \mathbb{N}_+$, $\mathsf{ML}(\exists^k_{FO})$ can be embedded into $\mathsf{ML}(\exists^k_{SO})$ by replacing, in a well-quantified formula of $\mathsf{ML}(\exists^k_{FO})$, each occurrence of $\exists^k p\,\varphi$ with the $\mathsf{ML}(\exists^k_{SO})$ formula

$\exists^k p : \varphi \wedge \mathtt{uniq}_k(p)$, where $\mathtt{uniq}_k(p) \overset{\text{def}}{=} \diamondsuit^k p \wedge \forall^k q : \diamondsuit^k(p \wedge q) \Rightarrow \boxplus^k(p \Rightarrow q)$ states that there is at most one world satisfying p that is reachable from the current one in at most k steps. Hence, $\mathsf{ML}(\exists^k_{SO})$ captures GML, and in fact the converse also holds, as we discover when proving Thm. 2. The corollary below is established.

Corollary 2. *For $k \in \overline{\mathbb{N}}_+$, $\mathsf{ML}(\exists^k_{FO})$, $\mathsf{ML}(\exists^k_{SO})$ and GML are equally expressive.*

This result is surprising, as it implies that $\mathsf{QCTL}^t_{\mathbf{X}}$ from [6] is as expressive as GML, and that in the context of modal logics, second-order propositional quantifiers do not yield any additional expressive power compared to first-order ones.

Connections with $\mathsf{S5Q}$. The sat. problem of $\mathsf{S5Q}$ [18,22] is equireducible to the sat. problem for formulae of $\mathsf{ML}(\exists^1_{SO})$ of modal depth 1. Briefly, any satisfiable formula of $\mathsf{S5Q}$ is satisfied by a Kripke structure $(\mathcal{W}, R, \mathcal{V})$ where $R = \mathcal{W} \times \mathcal{W}$, and $\mathsf{S5Q}$ enriches ML with quantifiers $\exists p$ which, by virtue of the relation R, are essentially the quantifiers $\exists^1 p$ from $\mathsf{ML}(\exists^1_{SO})$. We can simulate the models of $\mathsf{S5Q}$ by using a pointed forest $(\mathcal{K}, w)$ with accessibility relation R' such that $R'(w) = \mathcal{W}$. The current world of the $\mathsf{S5Q}$ model is simulated with a 1-local nominal $\mathtt{x}$ for $(\mathcal{K}, w)$. Then, the translation τ from $\mathsf{S5Q}$ to $\mathsf{ML}(\exists^1_{SO})$ is simple: $\tau(\diamondsuit\varphi) = \exists^1\mathtt{x} : \diamondsuit\mathtt{x} \wedge \mathtt{uniq}_1(\mathtt{x}) \wedge \tau(\varphi)$, binding the nominal $\mathtt{x}$ to a new world; $\tau(p) = @^1_{\mathtt{x}}p$, and otherwise τ is homomorphic. A similar translation can be given from formulae of $\mathsf{ML}(\exists^1_{SO})$ with modal depth 1 to $\mathsf{S5Q}$. Following Thm. 2, this allows us to characterise the complexity of $\mathsf{S5Q}$ left open in [18].

Corollary 3. *The sat. problem for $\mathsf{S5Q}$ is AExp_{pol}-complete.*

Connections with hybrid logics. Hybrid logics [3] are among the most studied modal logics featuring first-order propositional quantification. Given a set of nominals $\mathsf{NOM} \subseteq \mathsf{AP}$, the hybrid logic $\mathsf{HL}(\downarrow, @)$ extends ML with the *binder* $\downarrow i$ and the satisfaction operator $@_i$ (where $i \in \mathsf{NOM}$), having the semantics below:

$$(\mathcal{W}, R, \mathcal{V}), w \models \downarrow i.\varphi \Leftrightarrow (\mathcal{W}, R, \mathcal{V}[i \leftarrow \{w\}]), w \models \varphi;$$
$$(\mathcal{W}, R, \mathcal{V}), w \models @_i\varphi \Leftrightarrow (\mathcal{W}, R, \mathcal{V}), w_i \models \varphi, \text{ where } \mathcal{V}(i) = \{w_i\}.$$

$\mathsf{ML}(\exists^k_{FO})$ embeds in $\mathsf{HL}(\downarrow, @)$ by replacing with $\downarrow i.\diamondsuit^k \downarrow p.@_i\varphi$ each occurrence of $\exists^k p \varphi$ appearing in an $\mathsf{ML}(\exists^k_{FO})$ formula. This translation is (only) exponential in k, and so by uniform reduction for all $k \in \mathbb{N}_+$, and by Rabin's theorem [27] for the upper bound, Thm. 1 implies the following result.

Corollary 4. *The sat. problem for $\mathsf{HL}(\downarrow, @)$ on forests is* TOWER-*complete.*

3   Lower bounds for $\mathsf{ML}(\exists^k_{FO})$ and $\mathsf{ML}(\exists^k_{SO})$

In this section, we establish the lower bounds of Thms. 1 and 2, which follow by reduction from the *k-exp alternating multi-tiling problem*. While we will introduce this problem in due time, the main difficulty in establishing the reduction is defining, for all $k, n \in \mathbb{N}_+$ given in unary, a formula $type(k, n)$ that, whenever satisfied by a pointed forest $(\mathcal{K}, w)$, forces w to have $\mathfrak{t}(k, n)$ children, each of
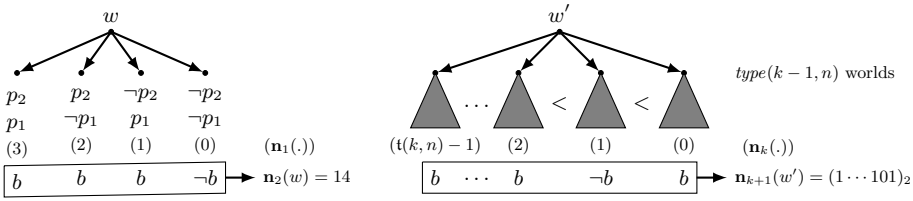
Fig. 1: Two worlds w and w' satisfying $type(1,2)$ and $type(k,n)$, respectively.

them encoding a different number in $[0, \mathsf{t}(k,n)-1]$. To establish Thms. 1 and 2, it is essential that $type(k,n)$ is of size polynomial in k and n, has modal depth k, it is in $\mathsf{ML}(\exists^1_{FO})$ for $k = 1$, and is in round-bounded $\mathsf{ML}(\exists^{k-1}_{FO})$ for all $k \geq 2$. The formula $type(k,n)$ is inspired by the homonymous formula defined in [6] to show that $\mathsf{QCTL}^t_{\mathbf{X}}$ is TOWER-hard, and later adapted in [7] to modal separation logics. With respect to both these works, our definition of $type(k,n)$ poses two serious challenges. First, [6,7] rely on second-order quantification, whereas we only use first-order. Second, in [6,7] the formula $type(k,n)$ is of size exponential in k, whereas our formula is of polynomial size. To achieve both improvements, we rely on a novel gadget that simulates binary addition with carry.

Numeric encoding. First of all, let us define how numbers are encoded by worlds of a pointed forest, following the presentation of [6]. Fix $n + 1$ distinct atomic propositions $p_1, \ldots, p_n, b$, and consider a Kripke-style forest $\mathcal{K} = (\mathcal{W}, R, \mathcal{V})$. Given $j \in [1, k]$ and $w \in \mathcal{W}$, we write $\mathbf{n}_j(w)$ for the number in $[0, \mathsf{t}(j,n)-1]$ encoded by w. For $j = 1$, we represent $\mathbf{n}_1(w) \in [0, 2^n - 1]$ by using the truth values of the propositions $p_1, \ldots, p_n$, where the proposition p_i is responsible for the i-th least significant bit of the number. That is, $\mathbf{n}_1(w) \stackrel{\text{def}}{=} \sum \{2^{i-1} : i \in [1, n] \text{ and } w \in \mathcal{V}(p_i)\}$. For $j > 1$, the number $\mathbf{n}_j(w)$ is represented by the binary encoding of the truth values of the atomic proposition b on the children of w, where a child $w' \in R(w)$ with $\mathbf{n}_{j-1}(w') = i$ from $[0, \mathsf{t}(j-1,n)-1]$ is responsible for the $(i+1)$-th least significant bit of the number encoded by w. Formally, $\mathbf{n}_j(w) \stackrel{\text{def}}{=} \sum \{2^i : \mathbf{n}_{j-1}(w') = i \text{ and } w' \in \mathcal{V}(b), \text{ for some } w' \in R(w)\}$.

With respect to this encoding of numbers, the forthcoming formula $type(k,n)$ shall satisfy the specification given by the lemma below, which guarantees that in a pointed forest $(\mathcal{K}, w)$ satisfying $type(k,n)$, the numbers encoded by the children of w span all over $[0, \mathsf{t}(k,n)-1]$. This is illustrated in Fig. 1.

Lemma 1. *A pointed forest $(\mathcal{K}, w)$, with $\mathcal{K} = (\mathcal{W}, R, \mathcal{V})$, satisfies $type(k,n)$ iff*
1. *for all $i \in [0, \mathsf{t}(k,n)-1]$ there is exactly one world $w' \in R(w)$ s.t. $\mathbf{n}_k(w') = i$;*
2. *if $k > 1$, then for every $w' \in R(w)$, $\mathcal{K}, w' \models type(k-1, n)$.*

Addition with carry. In defining $type(k,n)$, the main challenge lies in how to express the condition (1) of Lemma 1. In [6,7], this boils down to the definition of formulae that express (in)equalities between the numbers encoded by distinct $w_1, w_2 \in R(w)$, e.g. $\mathbf{n}_k(w_1) < \mathbf{n}_k(w_2)$ or $\mathbf{n}_k(w_1) = \mathbf{n}_k(w_2) + 1$. Unfortunately, these formulae are tree-recursive on k, meaning that multiple (possibly negated) occurrences of the inequalities for the case $k - 1$ are required to

Formula:	Expected Semantics:	Assumptions:
0_j	$\mathbf{n}_j(w) = 0$	The world w is the current world, which is assumed
1_j	$\mathbf{n}_j(w) = 1$	to satisfy $type(j,n)$. The world w_p corresponds to
$\mathcal{E}_j$	$\mathbf{n}_j(w) = \mathbf{t}(j,n) - 1$	the i-local nominal $p \in \{\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{c}\}$, and is assumed
$add_k^i(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{c})$	$+_{k-i+1}(w_{\mathbf{x}}, w_{\mathbf{y}}, w_{\mathbf{z}}, w_{\mathbf{c}})$	to satisfy $type(k-i,n)$.

Fig. 2: Auxiliary formulae used in the definition of $type(k,n)$, where $i = k = 1$ or $i < k$.

define the inequalities for the case k. Overall, this induces an exponential blow-up on $|type(k,n)|$. To avoid this blow-up, instead of relying on these inequalities we consider a quaternary relation $+_k(w_1, w_2, w_3, w_4)$ that holds whenever $\mathbf{n}_k(w_1) + \mathbf{n}_k(w_2) = \mathbf{n}_k(w_3)$ and $\mathbf{n}_k(w_4)$ represents the sequence of carries needed to perform $\mathbf{n}_k(w_1) + \mathbf{n}_k(w_2)$ in binary, on $\mathbf{t}(k-1,n)$ bits. For instance, for 4-bits numbers $\mathbf{n}_1(w_1) = 3 = (0011)_2$, $\mathbf{n}_1(w_2) = 5 = (0101)_2$, $\mathbf{n}_1(w_3) = 8 = (1000)_2$ and $\mathbf{n}_1(w_4) = 14 = (1110)_2$, the tuple (w_1, w_2, w_3, w_4) is in $+_1$, as

$$
\begin{array}{ll}
1\ 1\ 1\ 0 & : w_4 \text{ (sequence of carries of the sum)} \\
\underline{0\ 0\ 1\ 1} + & : w_1 \\
\underline{0\ 1\ 0\ 1} & : w_2 \\
1\ 0\ 0\ 0 & : w_3
\end{array}
$$

corresponds to the table for the binary addition with carry of $3 + 5 = 8$. By looking at the elementary algorithm for addition, a direct characterisation of $+_k$ is as follows. Let $\mathbf{n}_k(w_1) = (x_m \ldots x_1)_2$, $\mathbf{n}_k(w_2) = (y_m \ldots y_1)_2$, $\mathbf{n}_k(w_3) = (z_m \ldots z_1)_2$, $\mathbf{n}_k(w_4) = (c_m \ldots c_1)_2$, where $m = \mathbf{t}(k-1,n)$, and x_i, y_i, z_i and c_i are the i-th least significant digits in the binary encoding of $\mathbf{n}_k(w_1)$, $\mathbf{n}_k(w_2)$, $\mathbf{n}_k(w_3)$, $\mathbf{n}_k(w_4)$, respectively. Then, $+_k(w_1, w_2, w_3, w_4)$ holds if and only if

> **A.** $c_1 = 0$ and at most one among c_m, x_m and y_m is 1,
> **B.** for every $i \in [2, m]$, $c_i = maj(x_{i-1}, y_{i-1}, c_{i-1})$,   (†)
> **C.** for every $i \in [1, m]$, $z_i = (x_i \oplus y_i) \oplus c_i$,

where $maj(\varphi, \psi, \chi) \stackrel{\text{def}}{=} (\varphi \wedge \psi) \vee (\varphi \wedge \chi) \vee (\psi \wedge \chi)$ and $\varphi \oplus \psi \stackrel{\text{def}}{=} (\varphi \vee \psi) \wedge \neg(\varphi \wedge \psi)$ are the standard Boolean functions *majority* and *exclusive or*, respectively. When it comes to capturing $+_k$ with an $\mathsf{ML}(\exists_{FO}^k)$ formula, the key property is that the conditions (A), (B) and (C) can be checked with first-order quantification, by going through the binary encodings of $\mathbf{n}_k(w_1)$, $\mathbf{n}_k(w_2)$, $\mathbf{n}_k(w_3)$ and $\mathbf{n}_k(w_4)$ bit by bit, as one would do to check if an addition with carry was performed correctly.

A schema for $type(k,n)$. We move to the definition of $type(k,n)$. In view of its specification given in Lemma 1, the formula is defined recursively on k. For simplicity, we extend $type(k,n)$ to $k = 0$, and define it as $\top$. To express the condition (1) of Lemma 1, we rely on the auxiliary formulae presented in Fig. 2, which we later define. For $k, n \in \mathbb{N}_+$, we define $type(k,n)$ as:

$\Box type(k-1,n) \wedge \Diamond 0_k \wedge \Diamond 1_k \wedge \Diamond \mathcal{E}_k \wedge$
$\quad \forall^1 \mathbf{x} \forall^1 \mathbf{y} (\Diamond \mathbf{y} \wedge @_{\mathbf{x}}^1 \neg \mathbf{y} \Rightarrow \exists^1 \mathbf{z} \exists^1 \mathbf{c} : \Diamond \mathbf{c} \wedge @_{\mathbf{z}}^1 \neg 0_k \wedge (add_k^1(\mathbf{x}, \mathbf{z}, \mathbf{y}, \mathbf{c}) \vee add_k^1(\mathbf{y}, \mathbf{z}, \mathbf{x}, \mathbf{c}))).$

Whereas the first conjunct of $type(k,n)$ clearly encodes the condition (2) of Lemma 1, the remaining part of the formula forces the condition (1) by saying that the current world w has three children encoding the numbers 0, 1 and

$t(k, n) - 1$, respectively, and that for every two children $w_\mathbf{x}, w_\mathbf{y}$ of w, if $w_\mathbf{x} \neq w_\mathbf{y}$ (subformula $\Diamond \mathbf{y} \wedge @_\mathbf{x}^1 \neg \mathbf{y}$) then there is a child $w_\mathbf{z}$ of w such that $\mathbf{n}_k(w_\mathbf{z}) \neq 0$, and $\mathbf{n}_k(w_\mathbf{x}) + \mathbf{n}_k(w_\mathbf{z}) = \mathbf{n}_k(w_\mathbf{y})$ or $\mathbf{n}_k(w_\mathbf{y}) + \mathbf{n}_k(w_\mathbf{z}) = \mathbf{n}_k(w_\mathbf{x})$. Hence, in combination with $\Diamond 0_k$, $\Diamond 1_k$ and $\Diamond \mathcal{E}_k$, the last conjunct of $type(k, n)$ not only states that distinct children of w must encode different numbers, but also that every number of $[0, t(k, n) - 1]$ must be encoded by some child of w.

To effectively construct $type(k, n)$, what is left is to define the formulae in Fig. 2. Given how the numbers $\mathbf{n}_k(.)$ are encoded, the definitions of 0_k, 1_k and $\mathcal{E}_k$ are simple. For the case $k = 1$, we define $0_1 \stackrel{\text{def}}{=} \bigwedge_{j=1}^n \neg p_j$, $1_1 \stackrel{\text{def}}{=} (p_1 \wedge \bigwedge_{j=2}^n \neg p_j)$ and $\mathcal{E}_1 \stackrel{\text{def}}{=} \bigwedge_{j=1}^n p_j$. For $k \geq 2$, we define instead: $0_k \stackrel{\text{def}}{=} \Box \neg b$, $1_k \stackrel{\text{def}}{=} \Box(b \Rightarrow 0_{k-1})$, and $\mathcal{E}_k \stackrel{\text{def}}{=} \Box b$. The main difficulty lies in how to define add_k^i, which requires a recursive definition. Below, we consider three cases. First, we consider the base case $i = k = 1$ and define add_1^1 by only using the local quantifiers $\exists^1$. Afterwards, we consider the case $1 \leq i < k - 1$ and define the formula add_k^i by using local quantifiers $\exists^1, \ldots, \exists^{k-1}$. This formula relies on the definition of add_k^{i+1}, which we assume to be defined by inductive reasoning. Lastly, we consider the only remaining case of $i = k - 1$, and define add_k^{k-1} by using quantifiers $\exists^{k-1}$ and $\exists^1$, and without relying on the definition of add_1^1. This case is left for last as it is somewhat more involved than the other two cases, and some ingenuity is required to define add_k^{k-1} without relying on the local quantifiers $\exists^k$. The ad-hoc treatment of this case is however fundamental, as it leads to $type(k, n)$ being a round-bounded formula of the logic $\mathsf{ML}(\exists_{FO}^{k-1})$, for every $k \geq 2$.

Case: $i = k = 1$. Recall that the numbers $\mathbf{n}_1(.)$ are encoded using the truth values of $p_1, \ldots, p_n \in \mathsf{AP}$. Then, add_1^1 simply follows the constraints (†) of $+_1$:

$$add_1^1(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{c}) \stackrel{\text{def}}{=} @_\mathbf{c}^1 \neg p_1 \wedge \bigwedge_{q \in \{\mathbf{x}, \mathbf{y}, \mathbf{c}\}} \left(@_q^1 p_n \Rightarrow \bigwedge_{r \in \{\mathbf{x}, \mathbf{y}, \mathbf{c}\} \setminus \{q\}} @_r^1 \neg p_n \right) \quad \text{(A)}$$

$$\wedge \bigwedge_{i=2}^n \left(@_\mathbf{c}^1 p_i \Leftrightarrow maj(@_\mathbf{x}^1 p_{i-1}, @_\mathbf{y}^1 p_{i-1}, @_\mathbf{c}^1 p_{i-1}) \right) \quad \text{(B)}$$

$$\wedge \bigwedge_{i=1}^n \left(@_\mathbf{z}^1 p_i \Leftrightarrow \left((@_\mathbf{x}^1 p_i \oplus @_\mathbf{y}^1 p_i) \oplus @_\mathbf{c}^1 p_i \right) \right) \quad \text{(C)}$$

Case: $1 \leq i < k - 1$. To define add_k^i, we assume by inductive reasoning that the formula add_k^{i+1} is correctly defined, following its specification in Fig. 2. We specialise add_k^{i+1} to define the two auxiliary formulae below:

$$eq_k^{i+1}(\mathbf{x}, \mathbf{y}) \stackrel{\text{def}}{=} \exists^{i+1} \mathbf{z}, \mathbf{c} : \Diamond^{i+1} \mathbf{c} \wedge @_\mathbf{z}^{i+1} 0_{k-i} \wedge add_k^{i+1}(\mathbf{y}, \mathbf{z}, \mathbf{x}, \mathbf{c});$$
$$succ_k^{i+1}(\mathbf{x}, \mathbf{y}) \stackrel{\text{def}}{=} \exists^{i+1} \mathbf{z}, \mathbf{c} : \Diamond^{i+1} \mathbf{c} \wedge @_\mathbf{z}^{i+1} 1_{k-i} \wedge add_k^{i+1}(\mathbf{y}, \mathbf{z}, \mathbf{x}, \mathbf{c}).$$

Given $\mathbf{x}$ and $\mathbf{y}$ be two $(i+1)$-local nominals for $(\mathcal{K}, w)$, with corresponding worlds $w_\mathbf{x}$ and $w_\mathbf{y}$, if $\mathcal{K}, w' \models type(k - i, n)$ for some $w' \in R^i(w)$, then:

- $\mathcal{K}, w \models eq_k^{i+1}(\mathbf{x}, \mathbf{y})$ if and only if $\mathbf{n}_{k-i}(w_\mathbf{x}) = \mathbf{n}_{k-i}(w_\mathbf{y})$;
- $\mathcal{K}, w \models succ_k^{i+1}(\mathbf{x}, \mathbf{y})$ if and only if $\mathbf{n}_{k-i}(w_\mathbf{x}) = \mathbf{n}_{k-i}(w_\mathbf{y}) + 1$.

Notice that the semantics of $succ_k^{i+1}$ and eq_k^{i+1} is given under the hypothesis that a world in $R^i(w)$ satisfies $type(k - i, n)$. This extra hypothesis ensures that the local quantifiers $\exists^{i+1} \mathbf{z}$ and $\exists^{i+1} \mathbf{c}$ used to define $succ_k^{i+1}$ and eq_k^{i+1} quantify over a set of worlds encoding all the numbers in $[0, t(k - (i+1), n) - 1]$,

so that no possible addition with carry is missing. In defining $add_k^i(\mathsf{x},\mathsf{y},\mathsf{z},\mathsf{c})$, this hypothesis is clearly satisfied, as the worlds corresponding to the i-local nominals x, y, z and c are assumed to satisfy $type(k-i,n)$.

By relying on $succ_k^{i+1}$ and eq_k^{i+1}, we define $add_k^i(\mathsf{x},\mathsf{y},\mathsf{z},\mathsf{c})$ again by following the characterisation (†) of $+_{k-i+1}$, as shown below (where $X \overset{\text{def}}{=} \{\overline{\mathsf{x}},\overline{\mathsf{y}},\overline{\mathsf{c}}\}$):

$$\forall^{i+1}\overline{\mathsf{x}},\overline{\mathsf{y}},\overline{\mathsf{z}},\overline{\mathsf{c}},\mathsf{g} \,:\, @_{\mathsf{x}}^i\Diamond\overline{\mathsf{x}} \wedge @_{\mathsf{y}}^i\Diamond\overline{\mathsf{y}} \wedge @_{\mathsf{z}}^i\Diamond\overline{\mathsf{z}} \wedge @_{\mathsf{c}}^i(\Diamond\overline{\mathsf{c}} \wedge \Diamond\mathsf{g}) \;\Rightarrow$$

(A): $\qquad @_{\overline{\mathsf{c}}}^{i+1}(0_{k-i} \Rightarrow \neg b) \wedge \left((\bigwedge_{q \in X} @_q^{i+1}\mathcal{E}_{k-i}) \Rightarrow \bigwedge_{q \in X}(@_q^{i+1}b \Rightarrow \bigwedge_{r \in X\setminus\{q\}} @_r^{i+1}\neg b) \right)$

(B): $\qquad \wedge \left(eq_k^{i+1}(\overline{\mathsf{x}},\overline{\mathsf{y}}) \wedge eq_k^{i+1}(\overline{\mathsf{y}},\overline{\mathsf{c}}) \wedge succ_k^{i+1}(\mathsf{g},\overline{\mathsf{c}}) \Rightarrow (@_{\mathsf{g}}^{i+1}b \Leftrightarrow maj(@_{\overline{\mathsf{x}}}^{i+1}b, @_{\overline{\mathsf{y}}}^{i+1}b, @_{\overline{\mathsf{c}}}^{i+1}b)) \right)$

(C): $\qquad \wedge \left(eq_k^{i+1}(\overline{\mathsf{x}},\overline{\mathsf{y}}) \wedge eq_k^{i+1}(\overline{\mathsf{y}},\overline{\mathsf{z}}) \wedge eq_k^{i+1}(\overline{\mathsf{z}},\overline{\mathsf{c}}) \Rightarrow (@_{\overline{\mathsf{z}}}^{i+1}b \Leftrightarrow ((@_{\overline{\mathsf{x}}}^{i+1}b \oplus @_{\overline{\mathsf{y}}}^{i+1}b) \oplus @_{\overline{\mathsf{c}}}^{i+1}b)) \right).$

The first line of add_k^i binds the propositions $\overline{\mathsf{x}}$, $\overline{\mathsf{y}}$, $\overline{\mathsf{z}}$, and $\overline{\mathsf{c}}$ and g to children of x, y, z and c, respectively. Afterwards, the formula follows closely the constraints in (†). For instance, the last conjunct characterises the condition (C) by saying that whenever we consider children $w_{\overline{\mathsf{x}}}$, $w_{\overline{\mathsf{y}}}$, $w_{\overline{\mathsf{z}}}$ and $w_{\overline{\mathsf{c}}}$ of w_{x}, w_{y}, w_{z} and w_{c} respectively, if $j = \mathbf{n}_{k-i}(w_{\overline{\mathsf{x}}}) = \mathbf{n}_{k-u}(w_{\overline{\mathsf{y}}}) = \mathbf{n}_{k-i}(w_{\overline{\mathsf{z}}}) = \mathbf{n}_{k-i}(w_{\overline{\mathsf{c}}})$ for some $j \in \mathbb{N}$, then $\mathbf{n}_2(w_{\mathsf{z}})[j] = ((\mathbf{n}_2(w_{\mathsf{x}})[j] \oplus \mathbf{n}_2(w_{\mathsf{y}})[j]) \oplus \mathbf{n}_2(w_{\mathsf{c}})[j])$, where $\mathbf{n}_2(w)[j]$ is the $(j+1)$-th least significant digit of the number encoded by a world w.

Case: $i = k - 1$. To complete the definition of add_k^i, what is left is to define add_k^{k-1} by only using quantifiers $\exists^{k-1}$ and $\exists^1$. Below, the worlds w_{x}, w_{y}, w_{z} and w_{c}, corresponding to the $(k-1)$-local nominals x, y, z and c, satisfy $type(1,n)$, and so accordingly with $\mathbf{n}_2(.)$ they encode a number by looking at the value of the proposition b in their children, which themselves encode a number $\mathbf{n}_1(.)$. To properly define $add_k^{k-1}(\mathsf{x},\mathsf{y},\mathsf{z},\mathsf{c})$, we rely on the fact that these children encode n-bits numbers, with n given in unary. Then, instead of employing a quantifier $\exists^k$ to refer to one of these children, we can rely on $n+1$ local quantifiers $\exists^{k-1}$ to copy the values of $p_1,\ldots,p_n$ and b of a child directly on its parent. For instance, to check if w_{x} and w_{y} have children encoding the same numbers and equisatisfying b, one can follow the steps below, also sketched in Fig. 3:

1. using $\exists^{k-1}$, we quantify over fresh propositional symbols $r_1^{\mathsf{v}},\ldots,r_n^{\mathsf{v}}$ and q_{v}, with $\mathsf{v} \in \{\mathsf{x},\mathsf{y}\}$, to modify the truth of these symbols on w_{x} and w_{y};
2. using $@_{\mathsf{x}}^{k-1}$, we move the evaluation point to w_{x}. We check that the truth of the propositions $r_1^{\mathsf{x}},\ldots,r_n^{\mathsf{x}},q_{\mathsf{x}}$ on w_{x} is mirroring the truth of $p_1,\ldots,p_n,b$ on a child of w_{x}. For this, we rely on the formula $copy((r_1^{\mathsf{x}},\ldots,r_n^{\mathsf{x}}),q_{\mathsf{x}})$ that, for an n-tuple of atomic propositions $\mathbf{r} = (r_1,\ldots,r_n)$ and $q \in \mathsf{AP}$, is defined as: $copy(\mathbf{r},q) \overset{\text{def}}{=} \exists^1\mathsf{u} : \Diamond\mathsf{u} \wedge (q \Leftrightarrow @_{\mathsf{u}}^1b) \wedge \bigwedge_{i=1}^n (r_i \Leftrightarrow @_{\mathsf{u}}^1p_i)$. This step is also done (in parallel) for w_{y}, by relying on $copy((r_1^{\mathsf{y}},\ldots,r_n^{\mathsf{y}}),q_{\mathsf{y}})$;
3. with respect to the initial point of evaluation w, we check that the truth of the propositions $r_1^{\mathsf{x}},\ldots,r_n^{\mathsf{x}},q_{\mathsf{x}}$ on w_{x} corresponds to the truth of $r_1^{\mathsf{y}},\ldots,r_n^{\mathsf{y}},q_{\mathsf{y}}$ on w_{y}, i.e. $@_{\mathsf{x}}^{k-1}q_{\mathsf{x}} \Leftrightarrow @_{\mathsf{y}}^{k-1}q_{\mathsf{y}}$ and $@_{\mathsf{x}}^{k-1}r_i^{\mathsf{x}} \Leftrightarrow @_{\mathsf{y}}^{k-1}r_i^{\mathsf{y}}$, for all $i \in [1,n]$.

This idea of copying information about children of w_{x}, w_{y}, w_{z} and w_{c} directly in these four worlds is at the base of our definition of add_k^{k-1}, which we now formalise. Similarly to $\mathbf{n}_1(.)$, for an n-tuple of symbols $\mathbf{r} = (r_1,\ldots,r_n)$, $\mathbf{n}_{\mathbf{r}}(w) \overset{\text{def}}{=} \sum\{2^{i-1} : i \in [1,n], w \in \mathcal{V}(r_i)\}$ stands for the n-bits number encoded by the world w by looking at the truth values of $r_1,\ldots,r_n$. Given a second n-tuple of atomic
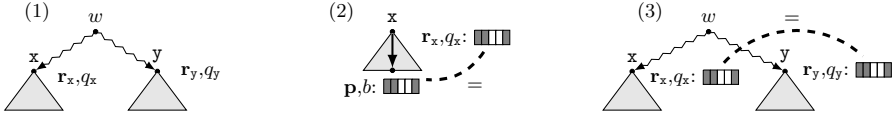
Fig. 3: Steps to check if two children of w_x and w_y encoding the same $\mathbf{n}_1(.)$ equisatisfy b.

propositions $\mathbf{s} = (s_1, \ldots, s_n)$, we introduce the formulae $succ(\mathbf{r}@\mathbf{x}, \mathbf{s}@\mathbf{y}) \stackrel{\text{def}}{=}$
$\bigvee_{i=1}^{n}\big(@_x^{k-1}r_i \wedge @_y^{k-1}\neg s_i \wedge \bigwedge_{j=1}^{i-1}(@_x^{k-1}\neg r_j \wedge @_y^{k-1}s_j) \wedge \bigwedge_{j=i+1}^{n}(@_x^{k-1}r_j \Leftrightarrow @_y^{k-1}s_j)\big)$
and $eq(\mathbf{r}@\mathbf{x}, \mathbf{s}@\mathbf{y}) \stackrel{\text{def}}{=} \bigwedge_{i=1}^{n}(@_x^{k-1}r_i \Leftrightarrow @_y^{k-1}s_i)$, having the following semantics:

- $\mathcal{K}, w \models eq(\mathbf{r}@\mathbf{x}, \mathbf{s}@\mathbf{y})$ if and only if $\mathbf{n_r}(w_x) = \mathbf{n_s}(w_y)$; and
- $\mathcal{K}, w \models succ(\mathbf{r}@\mathbf{x}, \mathbf{s}@\mathbf{y})$ if and only if $\mathbf{n_r}(w_x) = \mathbf{n_s}(w_y) + 1$.

The correctness of $succ(\mathbf{r}@\mathbf{x}, \mathbf{s}@\mathbf{y})$ follows from standard arithmetical properties: for two n-bits numbers $\mathbf{a}$ and $\mathbf{b}$ represented as binary bit vectors with most significant digit first, $\mathbf{a} = \mathbf{b} + 1$ holds iff $\mathbf{a} = \mathbf{c10}$ and $\mathbf{b} = \mathbf{c01}$ hold for a prefix $\mathbf{c} \in \{0, 1\}^*$ and bit vectors of same length $\mathbf{0} \in \{0\}^*$ and $\mathbf{1} \in \{1\}^*$.

The definition of $add_k^{k-1}(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{c})$ is given below, where $X \stackrel{\text{def}}{=} \{\mathbf{x}, \mathbf{y}, \mathbf{c}\}$ and for $\mathbf{v} \in \{\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{c}, \mathbf{g}\}$, $\mathbf{r_v} \stackrel{\text{def}}{=} (r_1^\mathbf{v}, \ldots, r_n^\mathbf{v})$ and $\forall^{k-1}\mathbf{r_v}$ is short for $\forall^{k-1}r_1^\mathbf{v} \ldots \forall^{k-1}r_n^\mathbf{v}$.

$\forall^{k-1}\mathbf{r_x}, q_x, \mathbf{r_y}, q_y, \mathbf{r_z}, q_z, \mathbf{r_c}, q_c, \mathbf{r_g}, q_g : \bigwedge_{\mathbf{v} \in \{\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{c}\}} @_\mathbf{v}^{k-1} copy(\mathbf{r_v}, q_\mathbf{v}) \wedge @_\mathbf{c}^{k-1} copy(\mathbf{r_g}, q_\mathbf{g}) \Rightarrow$

(A): $@_\mathbf{c}^{k-1}\square(0_1 \Rightarrow \neg b) \wedge \bigwedge_{q \in X} @_q^{k-1}\big(\Diamond(\mathcal{E}_1 \wedge b) \Rightarrow \bigwedge_{r \in X \setminus \{q\}} @_r^{k-1}\square(\mathcal{E}_1 \Rightarrow \neg b)\big)$

(B): $\wedge \big(eq(\mathbf{r_x}@\mathbf{x}, \mathbf{r_y}@\mathbf{y}) \wedge eq(\mathbf{r_y}@\mathbf{y}, \mathbf{r_c}@\mathbf{c}) \wedge succ(\mathbf{r_g}@\mathbf{c}, \mathbf{r_c}@\mathbf{c})$
$\Rightarrow \big(@_\mathbf{c}^{k-1}q_\mathbf{g} \Leftrightarrow maj(@_x^{k-1}q_x, @_y^{k-1}q_y, @_\mathbf{c}^{k-1}q_\mathbf{c})\big)\big)$

(C): $\wedge \big(eq(\mathbf{r_x}@\mathbf{x}, \mathbf{r_y}@\mathbf{y}) \wedge eq(\mathbf{r_y}@\mathbf{y}, \mathbf{r_z}@\mathbf{z}) \wedge eq(\mathbf{r_z}@\mathbf{z}, \mathbf{r_c}@\mathbf{c})$
$\Rightarrow \big(@_\mathbf{z}^{k-1}q_\mathbf{z} \Leftrightarrow \big((@_x^{k-1}q_x \oplus @_y^{k-1}q_y) \oplus @_\mathbf{c}^{k-1}q_\mathbf{c}\big)\big)\big).$

Notice that this formula first quantifies over fresh atomic propositions $\mathbf{r_v}$ and $q_\mathbf{v}$, with $\mathbf{v} \in \{\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{c}, \mathbf{g}\} \subseteq \mathsf{AP}$, so that the worlds w_x, w_y, w_z, w_c copy the truth of $p_1, \ldots, p_n$ and b of some of their children w.r.t. the fresh atomic propositions (see subformula $\bigwedge_{\mathbf{v} \in \{\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{c}\}} @_\mathbf{v}^{k-1} copy(\mathbf{r_v}, q_\mathbf{v}) \wedge @_\mathbf{c}^{k-1} copy(\mathbf{r_g}, q_\mathbf{g})$). Afterwards, the formula follows very closely the constraints (†) of $+_2$.

By induction on i, we show that add_k^i respects the specification from Fig. 2.

Lemma 2. *Let $(\mathcal{K}, w)$ be a pointed forest, and $\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{c}$ be four i-local nominals for $(\mathcal{K}, w)$, with corresponding worlds w_x, w_y, w_z and w_c. If $\mathcal{K}, w_p \models type(k-i, n)$ for every $p \in \{\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{c}\}$, then $\mathcal{K}, w \models add_k^i(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{c})$ iff $+_{k-i+1}(w_x, w_y, w_z, w_c)$.*

Making add_k^i polynomial. At this stage, add_k^i ($i < k - 1$) has size exponential in k, as it is recursively defined using multiple occurrences of add_k^{i+1} (appearing inside eq_k^{i+1} and $succ_k^{i+1}$). However, all these occurrences have the same polarity, i.e. they all appear positively in the antecedents of the implications for the conditions (B) or (C). This property allows us to rely on a *recursion trick* by Fisher and Rabin [20] to obtain a polynomial size formulation of add_k^i. In a nutshell, given a first-order formula $\varphi(\mathbf{x})$ free in the tuple of variables $\mathbf{x}$, the trick consists in rewriting $\psi \stackrel{\text{def}}{=} \varphi(\mathbf{y}) \wedge \varphi(\mathbf{z})$ as $\forall \mathbf{x} : (\mathbf{x} = \mathbf{y} \vee \mathbf{x} = \mathbf{z}) \Rightarrow \varphi(\mathbf{x})$, so

that the size of ψ becomes only $|\varphi(\mathbf{x})|$ plus a constant, instead of being roughly twice $|\varphi(\mathbf{x})|$. In a similar way, one can treat arbitrary formulae, as long as all occurrences of $\varphi(\mathbf{x})$ have the same polarity, as it is the case of add_k^{i+1}. The (simple) manipulation of the formula add_k^i using this trick directly leads to a definition of $type(k, n)$ of size polynomial in k and n.

Multi-tiling. The definition of $type(k, n)$ provides the key technical step required to show the lower bounds of Thms. 1 and 2. Using this formula, both theorems can be proved by suitable reductions from the k-exp alternating multi-tiling problem (kAMTP), as we now briefly discuss.

A *multi-tiling system* $\mathcal{P}$ is a tuple $(\mathcal{T}, \mathcal{T}_0, \mathcal{T}_{\mathrm{acc}}, \mathcal{H}, \mathcal{V}, \mathcal{M}, n)$ where $\mathcal{T}$ is a finite set of *tile types*, $\mathcal{T}_0, \mathcal{T}_{\mathrm{acc}} \subseteq \mathcal{T}$ are sets of *initial* and *accepting* tiles, respectively, $n \in \mathbb{N}_+$ (written in unary) is the *dimension* of the system, and $\mathcal{H}, \mathcal{V}, \mathcal{M} \subseteq \mathcal{T} \times \mathcal{T}$ are the horizontal, vertical and multi-tiling matching relations, respectively.

Fix $k \in \mathbb{N}_+$. We write $\widehat{\Sigma}$ for the set of words of length $\mathfrak{t}(k, n)$ over an alphabet Σ. The *initial row* $I(f)$ of a map $f \colon [0, \mathfrak{t}(k, n) - 1]^2 \to \mathcal{T}$ is the word $f(0,0), f(0,1), \ldots, f(0, \mathfrak{t}(k,n)-1)$ from $\widehat{\mathcal{T}}$. A *tiling* for the *grid* $[0, \mathfrak{t}(k,n)-1]^2$ is a tuple $(f_1, f_2, \ldots, f_n)$ such that, for all $\ell \in [1, n]$, the following conditions hold:

maps. $f_\ell \colon [0, \mathfrak{t}(k, n) - 1]^2 \to \mathcal{T}$ assigns a tile type to each position of the grid;
init & acc. $I(f_\ell) \in \widehat{\mathcal{T}_0}$, and $f_n(\mathfrak{t}(k,n) - 1, j) \in \mathcal{T}_{\mathrm{acc}}$ for some $0 \leq j < \mathfrak{t}(k, n)$;
hori. $(f_\ell(i, j), f_\ell(i+1, j)) \in \mathcal{H}$, for every $i \in [0, \mathfrak{t}(k, n) - 2]$ and $0 \leq j < \mathfrak{t}(k, n)$;
vert. $(f_\ell(i, j), f_\ell(i, j+1)) \in \mathcal{V}$, for every $j \in [0, \mathfrak{t}(k, n) - 2]$ and $0 \leq i < \mathfrak{t}(k, n)$;
multi. if $\ell < n$ then $(f_\ell(i,j), f_{\ell+1}(i,j)) \in \mathcal{M}$ for every $0 \leq i, j < \mathfrak{t}(k, n)$.

The kAMTP takes as input $\mathcal{P}$ and a *quantifier prefix* $\mathbf{Q} = (Q_1, \cdots, Q_n) \in \{\exists, \forall\}^n$, and accepts whenever the statement "$Q_1 w_1 \in \widehat{\mathcal{T}_0} \ldots Q_n w_n \in \widehat{\mathcal{T}_0}$: there is a tiling $(f_1, \ldots, f_n)$ of $[0, \mathfrak{t}(k, n) - 1]^2$ s.t. $I(f_\ell) = w_\ell$ for all $\ell \in [1, n]$" is true.

The AExp_{pol}-completeness of kAMTP for $k = 1$ can be traced back to [11]. The proof therein is independent from the size of the grid, and can be easily adapted to show $k\mathrm{AExp}_{pol}$-completeness for arbitrary k (see [24] for a self-contained presentation). The problem is $k\mathrm{NExp}$-complete if we fix $\mathbf{Q}$ to only contain existential quantifiers. For the lower bound of Thm. 1, we reduce kAMTP on instances with $\mathbf{Q} \in \{\exists\}^n$ to the sat. problem of $\mathsf{ML}(\exists_{FO}^k)$, so that the translation produces a formula of $\mathsf{ML}(\exists_{FO}^1)$ of modal depth 1 for the case $k = 1$, and otherwise a round-bounded formula from $\mathsf{ML}(\exists_{SO}^{k-1})$ of modal depth k. For Thm. 2 we get a similar reduction, from instances of the kAMTP with arbitrary $\mathbf{Q}$ to $\mathsf{ML}(\exists_{SO}^k)$.

The first step is to define an $\mathsf{ML}(\exists_{FO}^k)$ formula $grid(k, n)$ that, when satisfied by a pointed forest $(\mathcal{K}, w)$, forces the children of w to encode every position in the grid $[0, \mathfrak{t}(k, n) - 1]^2$, together with a formula $tiling(k, \mathcal{P})$ that characterises the various tiling conditions. Fortunately, both these formulae can be defined as in [7], modulo very minor changes. Briefly, each child w' of w shall encode a different pair of numbers $(\mathbf{n}_k^{\mathcal{H}}(w'), \mathbf{n}_k^{\mathcal{V}}(w'))$ representing a position in the grid. The number of bits required to represent $\mathbf{n}_k^{\mathcal{H}}(w')$ and $\mathbf{n}_k^{\mathcal{V}}(w')$ is the same as $\mathbf{n}_k(.)$, which allows us to define $grid(k, n)$ by slightly updating $type(k, n)$. In particular, $\mathbf{n}_k^{\mathcal{H}}(w')$ and $\mathbf{n}_k^{\mathcal{V}}(w')$ can be encoded requiring w' to satisfy $type(k - 1, n)$, and by

using fresh symbols $p_1^{\mathcal{H}}, \ldots, p_n^{\mathcal{H}}, b^{\mathcal{H}}$ and $p_1^{\mathcal{V}}, \ldots, p_n^{\mathcal{V}}, b^{\mathcal{V}}$ to encode $(\mathbf{n}_k^{\mathcal{H}}(w'), \mathbf{n}_k^{\mathcal{V}}(w'))$.
For $k = 1$, the horizontal position is $\mathbf{n}_1^{\mathcal{H}}(w') \stackrel{\text{def}}{=} \{2^{i-1} : i \in [1, n] \text{ and } w' \in \mathcal{V}(p_i^{\mathcal{H}})\}$.
For $k \geq 2$, $\mathbf{n}_k^{\mathcal{H}}(w') \stackrel{\text{def}}{=} \sum\{2^i : \exists w'' \in R(w') \text{ s.t. } \mathbf{n}_{k-1}(w'') = i \text{ and } w'' \in \mathcal{V}(b^{\mathcal{H}})\}$.
The vertical position $\mathbf{n}_k^{\mathcal{V}}(w')$ is defined in a similar way. Notice that, in the case
of $k \geq 2$, $\mathbf{n}_k^{\mathcal{H}}(w')$ and $\mathbf{n}_k^{\mathcal{V}}(w')$ are defined in terms of $\mathbf{n}_{k-1}(w'')$, and thus using
the $\mathbf{t}(k-1, n)$ children of w'. For $tiling(k, \mathcal{P})$, we see each tile type $t \in \mathcal{T}$ as
an atomic proposition, and consider n distinct copies $t^{(1)}, \ldots, t^{(n)} \in \mathsf{AP}$ of it,
so that the maps $f_1, \ldots, f_n$ can be encoded using just the set of worlds forced
by $grid(k, n)$. In particular, for every $i \in [1, n]$, each child w' shall satisfy exactly
one proposition in $\{t^{(i)} : t \in \mathcal{T}\}$, encoding the fact that $f_i(\mathbf{n}_k^{\mathcal{H}}(w'), \mathbf{n}_k^{\mathcal{V}}(w')) = t$.

Following the above specification, the toolkit of formulae in Fig. 2 can be eas-
ily adapted to express properties of the horizontal and vertical positions encoded
by a world, leading to the definition of $grid(k, n)$ and $tiling(k, \mathcal{P})$. For instance,
given $G \in \{\mathcal{H}, \mathcal{V}\}$ and $\varphi \in \{0_k, 1_k, \mathcal{E}_k\}$ we define the formula φ^G as follows: for
$k = 1$ we set $\varphi^G \stackrel{\text{def}}{=} \varphi[p_i \leftarrow_0 p_i^G : i \in [1, n]]$, and for $k \geq 2$ we set $\varphi^G \stackrel{\text{def}}{=} \varphi[b \leftarrow_1 b^G]$.
Then, w' satisfies the formula $1_k^{\mathcal{H}} \wedge 0_k^{\mathcal{V}}$ whenever $(\mathbf{n}_k^{\mathcal{H}}(w'), \mathbf{n}_k^{\mathcal{V}}(w')) = (1, 0)$.

Lemma 3. *The* $\mathsf{ML}(\exists_{FO}^k)$ *formula* $grid(k, n) \wedge tiling(k, \mathcal{P})$ *is satisfiable if and
only if* $k\mathsf{AMTP}$ *accepts on input* $(\mathcal{P}, \mathbf{Q})$, *with* $\mathbf{Q} \in \{\exists\}^n$.

For the lower bound of Thm. 2, it remains to show how to capture in $\mathsf{ML}(\exists_{SO}^k)$
the arbitrary prefixes of quantification $\mathbf{Q} = (Q_1, \ldots, Q_n)$ of $k\mathsf{AMTP}$. Compared
to [6,7], novel machinery is required to perform this step. As $\mathsf{ML}(\exists_{SO}^k)$ captures
$\mathsf{ML}(\exists_{FO}^k)$, we now see $grid(k, n)$ and $tiling(k, \mathcal{P})$ as formulae of $\mathsf{ML}(\exists_{SO}^k)$. For each
tile type $t \in \mathcal{T}$, we consider an additional set of copies $t^{(n+1)}, \ldots, t^{(2n)} \in \mathsf{AP}$. We
also define $\mathbf{t}^{(i)} \stackrel{\text{def}}{=} (t_1^{(i)}, \ldots, t_r^{(i)})$, where $\mathcal{T} = \{t_1, \ldots, t_r\}$. We use the propositions
in $\mathbf{t}^{(n+i)}$ to simulate the quantifier Q_i, which we recall quantifies over the possible
initial rows $I(f_i) \in \widehat{\mathcal{T}_0}$ of the map f_i. If $Q_i = \exists$, we simulate this form of
quantification with the following shortcut, parametric on φ:

$$E_i(\varphi) \stackrel{\text{def}}{=} \exists^1 \mathbf{t}^{(n+i)} : \varphi \wedge \Box(0_k^{\mathcal{H}} \Rightarrow \bigvee_{t \in \mathcal{T}_0}(t^{(n+i)} \wedge \bigwedge_{s \in \mathcal{T} \setminus \{t\}} \neg s^{(n+i)})).$$

Here, the last conjunct states that each world encoding a position $(0, j)$ of the
grid, for some $j \in [0, \mathbf{t}(k, n) - 1]$, satisfies exactly one proposition $t^{(n+i)}$ with
$t \in \mathcal{T}_0$. For $Q_i = \forall$, we just define $A_i(\varphi) \stackrel{\text{def}}{=} \neg E_i(\neg \varphi)$. Then, the prefix of
quantification $\mathbf{Q}$ is captured by $\mathbf{Q}(\varphi) \stackrel{\text{def}}{=} Q_1(Q_2(\ldots Q_n(\varphi)))$, where $Q_i(\varphi) \stackrel{\text{def}}{=}$
$E_i(\varphi)$ if $Q_i = \exists$, else $Q_i(\varphi) \stackrel{\text{def}}{=} A_i(\varphi)$. In deciding whether $\mathcal{K}, w \models \mathbf{Q}(\varphi)$ holds for
a pointed forest $(\mathcal{K}, w)$ satisfying $grid(k, n)$, the satisfaction of φ is checked w.r.t.
a model where each world encoding a position $(0, j)$ of the grid satisfies exactly
one $t^{(n+i)}$ with $t \in \mathcal{T}_0$, for all $i \in [1, n]$. In terms of tilings, this corresponds
to having set the initial row $I(f_i) \in \widehat{\mathcal{T}_0}$ of each of the maps f_i. We now want
to tile the remaining part of the grid by finding a suitable instantiation for φ.
To do so, we quantify over all $\mathbf{t}^{(1)}, \ldots \mathbf{t}^{(n)}$, searching for an arrangement of
these propositions that satisfies $tiling(k, \mathcal{P})$ and such that, on worlds encoding
a position $(0, j)$ of the grid, the satisfaction of propositions in $\mathbf{t}^{(i)}$ mirrors the
satisfaction of the corresponding propositions in $\mathbf{t}^{(n+i)}$. In formula:

$$\overline{tiling}(k, \mathcal{P}) \stackrel{\text{def}}{=} \exists^1 \mathbf{t}^{(1)}, \ldots, \mathbf{t}^{(n)} : tiling(k, \mathcal{P}) \wedge \Box(0_k^{\mathcal{H}} \Rightarrow \bigwedge_{i=1}^n \bigvee_{t \in \mathcal{T}}(t^{(i)} \Leftrightarrow t^{(n+i)})).$$

Lemma 4. *The* $\mathsf{ML}(\exists_{SO}^k)$ *formula* $grid(k,n) \wedge \mathbf{Q}(\overline{tiling}(k,\mathcal{P}))$ *is satisfiable if and only if* $k\mathrm{AMTP}$ *accepts on input* $(\mathcal{P}, \mathbf{Q})$.

Round-boundedness. In defining $type(k,n)$, we made sure to respect the following round-boundedness condition: $type(1,n)$ has modal depth 1 and belongs to $\mathsf{ML}(\exists_{FO}^1)$, whereas for every $k \geq 2$, $type(k,n)$ is a round-bounded formula of $\mathsf{ML}(\exists_{FO}^{k-1})$ of modal depth k. The same holds for $grid(k,n)$, $tiling(k,\mathcal{P})$ and $\mathbf{Q}(\overline{tiling}(k,\mathcal{P}))$. Then, Lemmas 3 and 4 imply the lower bounds of Thms. 1 and 2.

4   Upper bounds via a small-model property for $\mathsf{ML}(\exists_{SO}^k)$

In this section, we establish the following small model property.

Proposition 1. *Each satisfiable round-bounded formula* φ *in* $\mathsf{ML}(\exists_{SO}^k)$ *is satisfied by a pointed forest with* $\mathsf{t}(k+1, \mathcal{O}(|\varphi|))$ *worlds. Each satisfiable* φ *in* $\mathsf{ML}(\exists_{SO}^k)$ *with* $\mathsf{md}(\varphi) \leq k$ *is satisfied by a pointed forest with* $\mathsf{t}(k, \mathcal{O}(|\varphi|^3))$ *worlds.*

As the logic $\mathsf{ML}(\exists_{SO}^k)$ captures $\mathsf{ML}(\exists_{FO}^k)$, Prop. 1 transfers to the latter logic. With this result at hand, the upper bounds of Thm. 1 and Thm. 2 easily follow. Consider a round-bounded formula φ of either $\mathsf{ML}(\exists_{SO}^k)$ of $\mathsf{ML}(\exists_{FO}^k)$ (the arguments for a formula of modal depth k are similar). First, we guess a pointed forest $(\mathcal{K}, w)$ with bounds as in Prop. 1. This can be done in $(k+1)\mathrm{NExp}$. Then, we check whether $(\mathcal{K}, w)$ satisfies φ. For $\mathsf{ML}(\exists_{SO}^k)$, by seeing this logic as a fragment of *monadic second-order logic*, this can be done in polynomial time in the sizes of $(\mathcal{K}, w)$ and φ by using an alternating Turing machine that performs $|\varphi|$ many alternations. As $(\mathcal{K}, w)$ has $(k+1)$-exponential size with respect to $|\varphi|$, the whole algorithm runs in $(k+1)\mathrm{AExp}_{pol}$. For $\mathsf{ML}(\exists_{FO}^k)$, we rely on the fact that there is a deterministic algorithm for the model checking problem of *first-order logic* that runs in time $\mathcal{O}(|\varphi| \cdot M^{|\varphi|})$ where M is the size of the model. From the bounds on $(\mathcal{K}, w)$ we conclude that the procedure for $\mathsf{ML}(\exists_{FO}^k)$ is in $(k+1)\mathrm{NExp}$.

Prop. 1 is shown through a *quantifier elimination (QE) procedure* that translates every formula of $\mathsf{ML}(\exists_{SO}^k)$ into an equivalent formula from GML, establishing Cor. 2 as a by-product. Without loss of generality, in this section we extend $\mathsf{ML}(\exists_{SO}^k)$ with graded modalities $\Diamond_{\geq j}\varphi$, with $j \in \mathbb{N}$ given in unary, and see the modality $\Diamond$ as a shortcut for $\Diamond_{\geq 1}$. Recall that a GML formula $\Diamond_{\geq j}\varphi$ can be represented with an $\mathsf{ML}(\exists_{SO}^k)$ formula of size $\mathcal{O}(j + |\varphi|)$ (Sec. 2).

Parameters of a formula. Fig. 4 introduces a set of parameters for a $\mathsf{ML}(\exists_{SO}^k)$ formula φ, which we rely on to establish Prop. 1. For instance, for $\varphi = (p \vee \Diamond_{\geq 3} r) \wedge (q \vee \Diamond_{\geq 5}\Diamond_{\geq 2}q)$ we have $\mathsf{ap}(1, \varphi) = \{r\}$, $\mathsf{gsf}(0, \varphi) = \{\Diamond_{\geq 3}r, \Diamond_{\geq 5}\Diamond_{\geq 2}q\}$, $\mathsf{msf}(1, \varphi) = \{r, \Diamond_{\geq 2}q\}$, $\mathsf{gsf}(1, \varphi) = \{\Diamond_{\geq 2}q\}$, $\mathsf{gr}(0, \varphi) = 5$ and $\mathsf{bd}(0, \varphi) = 8$. Note that every GML formula φ is a Boolean combination of formulae from $\mathsf{ap}(0, \varphi) \cup \mathsf{gsf}(0, \varphi)$, and for every $d \in \mathbb{N}$, $\mathsf{bd}(d, \varphi) \leq \mathsf{gr}(d, \varphi) \cdot |\mathsf{msf}(d + 1, \varphi)|$.

For a set of formulae $\Phi = \{\varphi_1, \ldots, \varphi_n\}$, we define $\mathcal{C}(\Phi)$ to be the set of all complete conjunctions of possibly negated formulae of Φ. Formally, $\mathcal{C}(\Phi) \stackrel{\mathrm{def}}{=} \{\gamma_1 \wedge \cdots \wedge \gamma_n : \text{for all } i \in [1,n], \gamma_i \in \{\varphi_i, \neg\varphi_i\}\}$, and we fix $\mathcal{C}(\varnothing) = \{\top\}$. Given $\mathsf{P} \subseteq_{fin} \mathsf{AP}$ we refer to the formulae in $\mathcal{C}(\mathsf{P})$ as $\boldsymbol{\rho}_1, \boldsymbol{\rho}_2, \cdots$.

$\boxed{\begin{array}{ll}
\mathsf{ap}(d, \varphi) : & \text{set of atomic propositions of } \varphi \text{ in the scope of exactly } d \text{ graded modalities.} \\
\mathsf{gsf}(d, \varphi) : & \text{set of } \textit{subformulae } \lozenge_{\geq j}\psi \text{ of } \varphi, \text{ in the scope of exactly } d \text{ graded modalities.} \\
\mathsf{msf}(d, \varphi) : & \text{set of } \textit{maximal subformulae of } \varphi \text{ in the scope of } d \text{ graded modalities:} \\
& \quad \mathsf{msf}(0, \varphi) = \{\varphi\}, \text{ and } \psi \in \mathsf{msf}(d+1, \varphi) \text{ iff } \lozenge_{\geq j}\psi \in \mathsf{gsf}(d, \varphi) \text{ for some } j \in \mathbb{N}. \\
\mathsf{gr}(d, \varphi) : & \text{largest } j \in \mathbb{N} \text{ such that either } j = 0 \text{ or } \lozenge_{\geq j}\psi \in \mathsf{gsf}(d, \varphi), \text{ for some } \psi. \\
\mathsf{bd}(d, \varphi) : & \text{for } d = 0 \text{ and let } \mathsf{gsf}(0, \varphi) = \{\lozenge_{\geq j_1}\psi_1, \ldots, \lozenge_{\geq j_n}\psi_n\}, \mathsf{bd}(0, \varphi) \stackrel{\text{def}}{=} j_1 + \cdots + j_n. \\
& \text{For } d \geq 1, \mathsf{bd}(d, \varphi) \stackrel{\text{def}}{=} \max\{\mathsf{bd}(d-1, \psi) : \psi \in \mathsf{msf}(1, \varphi)\}.
\end{array}}$

Fig. 4: Parameters of an $\mathsf{ML}(\exists^k)$ formula φ $(d \in \mathbb{N})$.

Normal forms. We introduce a set of normal forms that are used by our QE procedure. An $\mathsf{ML}(\exists^k_{S0})$ formula φ is in *prenex normal form* if it is of the form $Q_1 p_1 Q_2 p_2 \ldots Q_n p_n \psi$ where $Q_i \in \{\exists^k, \forall^k\}$ and ψ is in GML. If ψ is instead in $\mathsf{ML}(\exists^k_{S0})$ but all quantifiers are under the scope of at least k modalities, we say that φ is in *prenex normal form up to k*. An $\mathsf{ML}(\exists^k_{S0})$ formula φ is in *prenex round-bounded (p.r.b.) form* if φ is round-bounded and, for all $i \in \mathbb{N}$, all formulae in $\mathsf{msf}(i \cdot k, \varphi)$ are in prenex normal form up to k. E.g., given a p.r.b. formula ψ in $\mathsf{ML}(\exists^2_{S0})$, $\exists^2 p \exists^2 q \lozenge \lozenge \exists^2 r \psi$ is in p.r.b. form, while $\exists^2 p \exists^1 q \lozenge \exists^2 r \psi$ is not. Thanks to the equivalences below one can translate each round-bounded formula φ of $\mathsf{ML}(\exists^k_{S0})$ into an equivalent well-quantified p.r.b. formula of size $\mathcal{O}(|\varphi|)$:

$$\lozenge \exists^{k-1} p \, \varphi \equiv \exists^k p \, \lozenge \varphi, \qquad\qquad \square \exists^{k-1} p \, \varphi \equiv_{S0} \exists^k p \, \square \varphi, \qquad \text{for } k \geq 2. \qquad (\ddagger)$$

Similarly, every φ in $\mathsf{ML}(\exists^k_{S0})$ of modal depth at most k can be translated to a well-quantified prenex formula of $\mathsf{ML}(\exists^k_{S0})$ having size $\mathcal{O}(|\varphi|)$. Notice that the second equivalence in ($\ddagger$) only holds on pointed forests and for the logic $\mathsf{ML}(\exists^k_{S0})$. It does not hold for arbitrary Kripke structures, nor for $\mathsf{ML}(\exists^k_{F0})$.

Our QE procedure translates each formula of $\mathsf{ML}(\exists^k_{S0})$ into a GML formula in *disjoint normal form* (called *good formulae* in [23, Def. 8.5]) for which it is easy to estimate bounds on the size of the smallest satisfying pointed forest, if any. We say that a set $\{\varphi_1, \ldots, \varphi_n\}$ of formulae in GML is a *disjoint set over* $\mathsf{P} \subseteq_{\text{fin}} \mathsf{AP}$ whenever for all $i, j \in [1, n]$, we have $\varphi_i = \boldsymbol{\rho}_i \wedge \gamma_i$ and $\varphi_j = \boldsymbol{\rho}_j \wedge \gamma_j$, where $\boldsymbol{\rho}_i, \boldsymbol{\rho}_j \in \mathcal{C}(\mathsf{P})$, $\mathsf{ap}(0, \gamma_i) \cap \mathsf{P} = \mathsf{ap}(0, \gamma_j) \cap \mathsf{P} = \varnothing$, and either $\gamma_i \equiv \gamma_j$ or $(\gamma_i \wedge \gamma_j) \equiv \bot$. By taking $\boldsymbol{\rho}_i$ and $\boldsymbol{\rho}_j$ up-to commutativity and associativity of $\wedge$, a disjoint set over P is also a disjoint set over any $\mathsf{P}' \subset \mathsf{P}$. We say that φ is in *disjoint normal form* (DisjNF) if for every $d \in [0, \mathsf{md}(\varphi)]$, $\mathsf{msf}(d, \varphi)$ is a disjoint set over $\varnothing$.

Proposition 2 ([23], Lemma 8.7). *Each satisfiable GML formula φ in DisjNF is satisfied by a pointed forest with at most $(\max_{d \in \mathbb{N}}(\mathsf{bd}(d, \varphi)) + 1)^{\mathsf{md}(\varphi)}$ worlds.*

To translate a well-quantified p.r.b. formula φ from $\mathsf{ML}(\exists^k_{S0})$ into a GML formula in DisjNF, we consider the largest $i \in \mathbb{N}$ for which $\mathsf{msf}(i \cdot k, \varphi)$ is non-empty, and inductively translate, for each $j = i, i-1, \cdots, 0$, all formulae in $\mathsf{msf}(j \cdot k, \varphi)$ into equivalent ones in GML. At each of these $i+1$ rounds, the following two steps are applied at most k times:

1. Let $\ell = \min\{r \in \mathbb{N}_+ : \text{all formulae of } \mathsf{msf}(j \cdot k, \varphi) \text{ are in } \mathsf{ML}(\exists^r_{S0})\}$. We update all $\psi \in \mathsf{msf}(j \cdot k, \varphi)$ so that $\mathsf{msf}(\ell, \psi)$ becomes a disjoint set over $\mathsf{bp}(\psi)$.

2. By manipulating all quantified propositions of the formulae in $\mathsf{msf}(\ell, \psi)$, we translate ψ into a formula of either GML (if $\ell = 1$) or $\mathsf{ML}(\exists^{\ell-1}_{S0})$ (if $\ell \geq 2$).

At the end of the round, $\mathsf{msf}(j \cdot k, \varphi)$ solely contains GML formulae in DisjNF, and the next round considers the set $\mathsf{msf}((j-1)\cdot k, \varphi)$, that now contains $\mathsf{ML}(\exists^k_{S0})$ formulae in prenex normal form. The QE procedure has thus three key steps, which we now formalise: (I) manipulating a formula φ so that $\mathsf{msf}(j, \varphi)$ becomes a disjoint set, (II) eliminating the quantifier $\exists^1$ obtaining a formula from GML, and (III) reducing the elimination of $\exists^\ell$ to the elimination of $\exists^{\ell-1}$ (for $\ell \geq 2$).

Step (I): making a single set disjoint. Let $j \in \mathbb{N}_+$ and $\mathsf{P} \subseteq_{fin} \mathsf{AP}$. We show how to transform a GML formula φ into an equivalent formula ψ such that $\mathsf{msf}(j, \psi)$ is a disjoint set over P. Two strategies are possible, which will be combined and carefully chosen in order to obtain the bounds required by Prop. 1.

The *first strategy* considers the set $\mathcal{S} \stackrel{\text{def}}{=} \mathcal{C}(\mathsf{P} \cup \mathsf{ap}(j, \varphi) \cup \mathsf{gsf}(j, \varphi))$, which is disjoint over P (and so over $\varnothing$), and rewrites φ into an equivalent formula ψ with $\mathsf{msf}(j, \psi) \subseteq \mathcal{S}$. Consider $\gamma \in \mathsf{msf}(j, \varphi)$. By definition of $\mathcal{C}(.)$, $\bigvee_{\chi \in \mathcal{S}} \chi$ is a tautology, and since γ is a Boolean combination of formulae in $\mathsf{ap}(j, \varphi) \cup \mathsf{gsf}(j, \varphi)$, for all $\chi \in \mathcal{S}$ the formula $\gamma \wedge \chi$ is equivalent to either $\bot$ or χ. Then, $\gamma \equiv \bigvee_{\chi \in T} \chi$ for some $T \subseteq \mathcal{S}$. Notice that $\gamma \in \mathsf{msf}(j, \varphi)$ holds if and only if $\Diamond_{\geq i}\gamma \in \mathsf{gsf}(j-1, \varphi)$, for some $i \in \mathbb{N}$. By relying on the equivalence of GML

$$\Diamond_{\geq i}(\chi_1 \vee \chi_2) \equiv \bigvee_{i=i_1+i_2}(\Diamond_{\geq i_1}\chi_1 \wedge \Diamond_{\geq i_2}\chi_2), \qquad \text{whenever } \chi_1 \wedge \chi_2 \equiv \bot,$$

we rewrite $\Diamond_{\geq i}\gamma$ into a Boolean combination of formulae $\Diamond_{\geq i'}\chi$ with $i' \leq i$ and $\chi \in T \subseteq \mathcal{S}$. These steps are applied to all the formulae in $\mathsf{msf}(j, \varphi)$.

The *second strategy* is as follows: for each $\gamma \in \mathsf{msf}(j, \varphi)$ and $\boldsymbol{\rho} \in \mathcal{C}(\mathsf{P})$, let $\gamma_{\boldsymbol{\rho}} \stackrel{\text{def}}{=} \gamma[p \leftarrow_0 v : v \in \{\top, \bot\}, p \in \mathsf{P}$, and $v = \top$ iff p occurs positively in $\boldsymbol{\rho}]$. Notice that $\mathsf{ap}(0, \gamma_{\boldsymbol{\rho}}) \cap \mathsf{P} = \varnothing$. As $\boldsymbol{\rho}$ gives a polarity to all propositions in P, we have $\boldsymbol{\rho} \wedge \gamma \equiv \boldsymbol{\rho} \wedge \gamma_{\boldsymbol{\rho}}$. Set $\mathcal{T} \stackrel{\text{def}}{=} \mathcal{C}(\{\gamma_{\boldsymbol{\rho}} : \gamma \in \mathsf{msf}(j, \varphi), \boldsymbol{\rho} \in \mathcal{C}(\mathsf{P})\})$. Consider $\mathcal{S}' \stackrel{\text{def}}{=} \mathcal{C}(\mathsf{P} \cup \mathcal{T})$, which is a disjoint set over P, and replay the arguments used for $\mathcal{S}$ in the first strategy to rewrite φ into an equivalent formula ψ with $\mathsf{msf}(j, \psi) \subseteq \mathcal{S}'$.

While both strategies keep most of the parameters of Fig. 4 unchanged (one exception being $\mathsf{ap}(j, \psi) \subseteq \mathsf{ap}(j, \varphi) \cup \mathsf{P}$), they yield profoundly different bounds on the size of $\mathsf{msf}(j, \psi)$. Because of the definition of $\mathcal{S}$, from the first strategy we obtain $|\mathsf{msf}(j, \psi)| \leq 2^{|\mathsf{P}|+|\mathsf{ap}(j,\varphi)|+|\mathsf{gsf}(j,\varphi)|}$, where we highlight the exponential dependence on $|\mathsf{gsf}(j, \varphi)|$, and thus on the number of outermost graded modalities appearing in formulae of $\mathsf{msf}(j, \varphi)$. From the definition of $\mathcal{S}'$, the second strategy yields $|\mathsf{msf}(j, \psi)| \leq 2^{|\mathsf{P}|+2^{|\mathsf{P}|} \cdot |\mathsf{msf}(j,\varphi)|}$. Here, $|\mathsf{msf}(j, \psi)|$ does not depend on $\mathsf{gsf}(j, \varphi)$, but it is doubly exponential in $|\mathsf{P}|$. Remarkably, in both strategies $\mathsf{gsf}(j, \psi) \subseteq \mathsf{gsf}(j, \varphi)$, thus if $\mathsf{msf}(j+1, \varphi)$ is a disjoint set over $\varnothing$, so is $\mathsf{msf}(j+1, \psi)$. This property is essential, as it allows us to bring the full formula in DisjNF.

Step (II): eliminating $\exists^1$. Given a well-quantified formula $\varphi = \exists^1 p \, \varphi'$, where φ' is in GML and $\mathsf{msf}(1, \varphi)$ is a disjoint set over P, and $p \in \mathsf{P}$, it is quite easy to eliminate the quantifier $\exists^1 p$ and produce a formula ψ in GML equivalent to φ and such that $\mathsf{msf}(1, \psi)$ is a disjoint set over $\mathsf{P} \setminus \{p\}$. We sketch here the main points. First, from standard axioms of propositional calculus and by distributing $\exists^1 p$ over $\vee$, we obtain a representation of φ as a disjunction of formulae of the form $\exists^1 p \, (\boldsymbol{\rho} \wedge \gamma)$ with $\boldsymbol{\rho} \in \mathcal{C}(\mathsf{ap}(0, \varphi))$ and $\gamma \in \mathcal{C}(\mathsf{gsf}(0, \varphi))$. We eliminate the

quantifier $\exists^1$ from every such disjunct $\exists^1 p\,(\boldsymbol{\rho} \wedge \gamma)$. Below, let χ be an arbitrary formula with $p \notin \mathsf{ap}(0, \chi)$. First, using the equivalences $\exists^1 p\,(p \wedge \chi) \equiv_{S0} \exists^1 p\,\chi$ and $\exists^1 p\,(\neg p \wedge \chi) \equiv_{S0} \exists^1 p\,\chi$, we get rid of the occurrences of p in $\boldsymbol{\rho}$, obtaining a formula $\boldsymbol{\rho}' \in \mathcal{C}(\mathsf{ap}(0, \varphi) \setminus \{p\})$. Next, we remove p from γ thanks to the equivalences:

$$\exists^1 p : \lozenge_{\geq i}(p \wedge \chi) \wedge \lozenge_{\geq j}(\neg p \wedge \chi) \equiv_{S0} \lozenge_{\geq i+j}\chi;$$
$$\exists^1 p : \neg\lozenge_{\geq i}(p \wedge \chi) \wedge \neg\lozenge_{\geq j}(\neg p \wedge \chi) \equiv_{S0} \neg\lozenge_{\geq i+j-1}\chi.$$

We obtain a GML formula γ' such that $\exists^1 p\,(\boldsymbol{\rho} \wedge \gamma) \equiv_{S0} \boldsymbol{\rho}' \wedge \gamma'$. Size-wise, Step (II) preserves all the parameters of Fig. 4 except $\mathsf{gr}(0, \psi) \leq 2 \cdot \mathsf{gr}(0, \varphi)$.

Step (III): from $\exists^{k+1}$ to $\exists^k$. Consider a well-quantified $\mathsf{ML}(\exists^k_{S0})$ formula φ' having all quantifiers appearing outside the scope of graded modalities, and with the set $\mathsf{msf}(k+1, \varphi')$ disjoint over P. Given $p \in \mathsf{P}$, we translate $\varphi \stackrel{\text{def}}{=} \exists^{k+1} p\,\varphi'$ into an equivalent well-quantified $\mathsf{ML}(\exists^k_{S0})$ formula ψ having all quantifiers outside the scope of graded modalities, and with the set $\mathsf{msf}(k+1, \psi)$ disjoint over $\mathsf{P} \setminus \{p\}$. This is done by replacing $\exists^{k+1} p$ with multiple $\exists^k$. The first step is to single out the occurrences of p under the scope of $k+1$ modalities by replacing them with a fresh symbol $\widetilde{p}$ and splitting $\exists^{k+1} p$ into $\exists^k p$ and $\exists^{k+1}\widetilde{p}$. We get $\varphi \equiv_{S0} \exists^k p\, \exists^{k+1}\widetilde{p}\,\varphi''$ where $\varphi'' = \varphi'[p \leftarrow_{k+1} \widetilde{p}]$. Let $\mathsf{gsf}(k, \varphi'') = \{\lozenge_{\geq k_1}\chi_1, \ldots, \lozenge_{\geq k_n}\chi_n\}$. From the properties of φ', no proposition from $\mathsf{bp}(\varphi'')$ appears in the GML formulae $\chi_1, \ldots, \chi_n$. Using fresh propositions $q_1, \ldots, q_n$, we rewrite φ as

$$\exists^k p\, \exists^{k+1}\widetilde{p}\, \exists^k q_1, \ldots, q_n : \varphi''[\lozenge_{\geq k_i}\chi_i \leftarrow_k q_i : 1 \leq i \leq n] \wedge \square^k \bigwedge_{i=1}^{n}(q_i \Leftrightarrow \lozenge_{\geq k_i}\chi_i).$$

Essentially, the subformula $\square^k \bigwedge_{i=1}^{n}(q_i \Leftrightarrow \lozenge_{\geq k_i}\chi_i)$ constraints each q_i to be true in exactly those worlds satisfying $\lozenge_{\geq k_i}\chi_i$. This allows us to replace with q_i all occurrences of $\lozenge_{\geq k_i}\chi_i$ appearing in φ'' under the scope of k modalities (first conjunct of the formula above), without changing the semantics of φ. By definition, $\varphi''[\lozenge_{\geq k_i}\chi_i \leftarrow_k q_i : 1 \leq i \leq n]$ has modal depth at most k, and thus the proposition $\widetilde{p}$ does not occur in it. We reorder the existential prefix of the formula and, by distributing $\exists^{k+1}\widetilde{p}$, conclude that φ is equivalent to:

$$\exists^k p, q_1, \ldots, q_n : \varphi''[\lozenge_{\geq k_i}\chi_i \leftarrow_k q_i : 1 \leq i \leq n] \wedge \exists^{k+1}\widetilde{p}\,\square^k \bigwedge_{i=1}^{n}(q_i \Leftrightarrow \lozenge_{\geq k_i}\chi_i).$$

Lastly, we eliminate $\exists^{k+1}\widetilde{p}$, obtaining the aforementioned $\mathsf{ML}(\exists^k_{S0})$ formula ψ. Using the second equivalence in ($\ddagger$), we rewrite $\exists^{k+1}\widetilde{p}\,\square^k \bigwedge_{i=1}^{n}(q_i \Leftrightarrow \lozenge_{\geq k_i}\chi_i)$ into $\square^k \exists^1 \widetilde{p} \bigwedge_{i=1}^{n}(q_i \Leftrightarrow \lozenge_{\geq k_i}\chi_i)$. Since $\{\chi_1, \ldots, \chi_n\}$ is a set of formulae form GML that is disjoint over $(\mathsf{P} \setminus \{p\}) \cup \{\widetilde{p}\}$, by applying Step (II) one computes a formula ψ' in GML equivalent to $\exists^1 \widetilde{p} \bigwedge_{i=1}^{n}(q_i \Leftrightarrow \lozenge_{\geq k_i}\chi_i)$ and such that $\mathsf{msf}(1, \psi')$ is a disjoint set over $\mathsf{P} \setminus \{p\}$. Then, the (output) formula ψ is defined as follows:

$$\psi \stackrel{\text{def}}{=} \exists^k p, q_1, \ldots, q_n : \varphi''[\lozenge_{\geq k_i}\chi_i \leftarrow_k q_i : 1 \leq i \leq n] \wedge \square^k \psi'.$$

Down to GML, inductively. The manipulation we just described yield the crucial inductive argument that allows us to translate any well-quantified prenex formula of $\mathsf{ML}(\exists^k_{S0})$ into a formula of GML. Inductively on k, consider a well-quantified formula $\varphi = Q_1 p_1 \ldots Q_n p_n \varphi'$ where each $Q_i \in \{\exists^k, \forall^k\}$, the formula φ' is in GML and $\mathsf{msf}(k, \varphi)$ is a disjoint set over $\{p_1, \ldots, p_n\}$. If $k = 1$, we repeatedly apply Step (II) to translate φ into a GML formula. If $k \geq 2$,

starting from p_n down to p_1, we apply Step (III) to translate φ into a well-quantified prenex formula χ from $\mathsf{ML}(\exists_{SO}^{k-1})$. Afterwards, we rely on the first strategy of Step (I) to make the set $\mathsf{msf}(k-1,\chi)$ disjoint over $\mathsf{bp}(\chi)$, and inductively obtain a GML formula ψ equivalent to φ. For a sake of conciseness, let $|\varphi|_k \stackrel{\text{def}}{=} \max(k, |\bigcup_{i\in[0,k]} \mathsf{ap}(i,\varphi)|, \max_{i<k} \mathsf{gr}(i,\varphi))$. Fundamentally, the formula ψ has the same modal depth as φ, and for every $i \in [0, k-1]$ it satisfies:

$$\mathsf{gr}(i,\psi) \leq \mathsf{t}(k-1, 2^{8\cdot|\varphi|_k} \cdot |\mathsf{msf}(k,\varphi)|); \quad \mathsf{msf}(i,\psi) \leq \mathsf{t}(k-1, 2^{8\cdot|\varphi|_k} \cdot |\mathsf{msf}(k,\varphi)|).$$

With these bounds at hand, Prop. 1 follows from Steps (I)–(III) and Prop. 2. First, consider the case of a well-quantified prenex formula φ in $\mathsf{ML}(\exists^k)$ of modal depth k. Using the first strategy from Step (I), we translate φ into an equivalent formula ψ such that the set $\mathsf{msf}(k,\psi)$ is disjoint over $\mathsf{bp}(\varphi)$ and has size exponential in $|\varphi|$. We apply the inductive argument discussed above, and translate ψ into a GML formula χ in DisjNF with $\mathsf{md}(\chi) \leq \mathsf{md}(\varphi)$ and $\mathsf{bd}(d,\chi) \leq \mathsf{gr}(d,\chi) \cdot |\mathsf{msf}(d+1,\chi)|) \leq \mathsf{t}(k, \mathcal{O}(|\varphi|^2))$ for all $d \in \mathbb{N}$. By Prop. 2, whenever satisfiable, φ is satisfied by a pointed forest with at most $\mathsf{t}(k, \mathcal{O}(|\varphi|^3))$ worlds. The case of general p.r.b. formulae of $\mathsf{ML}(\exists_{SO}^k)$ is similar, but we need to appeal to the second strategy of Step (I) to stop the chain of exponential blow-ups. For simplicity, let us consider the case of φ being a well-quantified p.r.b. formula of modal depth at most $2k$. The arguments used for this case can be adapted for formulae of arbitrary modal depth. First, we look at the formulae of $\mathsf{msf}(k,\varphi)$, whose modal depth is at most k, and eliminate all local quantifiers from each of these formulae, as described above. In doing so, $|\mathsf{gsf}(k,\varphi)|$ witnesses a k-exponential blow-up, but the size of $\mathsf{msf}(k,\varphi)$ is unchanged. We consider the quantification prefix of φ, and eliminate all its quantifiers over P to produce an equivalent formula from GML. The first step is to make the set $\mathsf{msf}(k,\varphi)$ a disjoint set over P. Because of the k-exponential blow-up on $\mathsf{gsf}(k,\varphi)$, the first strategy of Step (I) is of no use. We appeal to the second one, which modifies $\mathsf{msf}(k,\varphi)$ into a disjoint set of size only doubly-exponential in the size of the original formula φ. By relying on the inductive reasoning discussed above, we produce the equivalent GML formula in DisjNF. Because of the doubly-exponential bound on $\mathsf{msf}(k,\varphi)$, this elimination is exponentially worse than the one done for formulae of modal depth at most k. Then, appealing to Prop. 2 yields Prop. 1.

5   Further connections

In introducing $\mathsf{ML}(\exists_{FO}^k)$ and $\mathsf{ML}(\exists_{SO}^k)$, one of our goals is to provide a common framework to relate several modal logics featuring propositional quantification in disguise. Apart from the relations stated in Sec. 2, in an extended version of this work we aim at establishing connections between $\mathsf{ML}(\exists_{SO}^1)$ and *propositional team logics* [21], *propositional logic of dependence* [32] and *ambient logics* [13]; as well as connections bwteen $\mathsf{ML}(\exists_{FO}^\infty)$ and *sabotage logics* [8,4].

References

1. Andréka, H., Németi, I., van Benthem, J.: Modal languages and bounded fragments of predicate logic. Journal of Philosophical Logic **27**(3), 217–274 (1998)
2. Areces, C., Blackburn, P., Marx, M.: Hybrid logics: characterization, interpolation and complexity. The Journal of Symbolic Logic **66**(3), 977–1010 (2001)
3. Areces, C., ten Cate, B.: Hybrid logics. In: Handbook of Modal Logic, Studies in logic and practical reasoning, vol. 3, pp. 821–868. North-Holland (2007)
4. Areces, C., Fervari, R., Hoffmann, G.: Relation-changing modal operators. Logic Journal of the IGPL **23**(4), 601–627 (2015)
5. Barnaba, M.F., Caro, F.D.: Graded modalities. Studia Logica **44**(2), 197–221 (1985)
6. Bednarczyk, B., Demri, S.: Why propositional quantification makes modal logics on trees robustly hard? In: Logic in Computer Science. pp. 1–13. IEEE (2019)
7. Bednarczyk, B., Demri, S., Fervari, R., Mansutti, A.: Modal logics with composition on finite forests: Expressivity and complexity. In: Logic in Computer Science. pp. 167–180. ACM (2020)
8. van Benthem, J.: An essay on sabotage and obstruction. In: Mechanizing Mathematical Reasoning. LNCS, vol. 2605, pp. 268–276 (2005)
9. Blackburn, P., Braüner, T., Kofod, J.: Remarks on Hybrid Modal Logic with Propositional Quantifiers, pp. 401–426. No. 4 in Logic and Philosophy of Time (2020)
10. Blackburn, P., Wolter, F., van Benthem, J. (eds.): Handbook of Modal Logics, Studies in logic and practical reasoning, vol. 3. Elsevier (2006)
11. Bozzelli, L., Molinari, A., Montanari, A., Peron, A.: On the complexity of model checking for syntactically maximal fragments of the interval temporal logic HS with regular expressions. In: GandALF'17. EPTCS, vol. 256, pp. 31–45 (2017)
12. Bull, R.A.: On modal logic with propositional quantifiers. The Journal of Symbolic Logic **34**(2), 257–263 (1969)
13. Calcagno, C., Cardelli, L., Gordon, A.: Deciding validity in a spatial logic for trees. In: International Workshop on Types in Languages Design and Implementation. pp. 62–73. ACM (2003)
14. ten Cate, B., Franceschet, M.: On the complexity of hybrid logics with binders. In: Ong, L. (ed.) Computer Science Logic. pp. 339–354 (2005)
15. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. Journal of the ACM **28**(1), 114–133 (1981)
16. Demri, S., Fervari, R.: The power of modal separation logics. Journal of Logic and Computation **29**(8), 1139–1184 (2019)
17. Ding, Y.: On the logics with propositional quantifiers extending s5Π. In: Advances in Modal Logic. pp. 219–235. College Publications (2018)
18. Fine, K.: Propositional quantifiers in modal logic. Theoria **36**, 336–346 (1970)
19. Fischer, M.J., Ladner, R.E.: Propositional modal logic of programs. In: ACM Symposium on Theory of Computing. p. 286–294 (1977)
20. Fischer, M.J., Rabin, M.O.: Super-exponential complexity of presburger arithmetic. In: Complexity of Computation, SIAM–AMS Proceedings. pp. 27–41 (1974)
21. Hannula, M., Kontinen, J., Virtema, J., Vollmer, H.: Complexity of propositional logics in team semantic. ACM Transactions on Computational Logic **19**(1), 2:1–2:14 (2018)
22. Kaplan, D.: S5 with quantifiable propositional variables. The Journal of Symbolic Logic **35**(2), 355 (1970)

23. Mansutti, A.: Reasoning with Separation Logics: Complexity, Expressive Power, Proof Systems. Ph.D. thesis, Université Paris-Saclay (December 2020)
24. Mansutti, A.: Notes on kAExp(pol) problems for deterministic machines (2021)
25. Meier, A., Mundhenk, M., Thomas, M., Vollmer, H.: The complexity of satisfiability for fragments of CTL and CTL*. Electronic Notes in Theoretical Computer Science **223**, 201–213 (2008)
26. Prior, A.: Past, Present and Future. Oxford Books (1967)
27. Rabin, M.: Decidability of second-order theories and automata on infinite trees. Transactions of the American Mathematical Society **41**, 1–35 (1969)
28. de Rijke, M.: A note on graded modal logic. Studia Logica **64**(2), 271–283 (2000)
29. Schmitz, S.: Complexity hierarchies beyond elementary. ACM Transactions on Computation Theory **8**(1), 3:1–3:36 (2016)
30. Schneider, T.: The complexity of hybrid logics over restricted frame classes. Ph.D. thesis, Friedrich Schiller University of Jena (2007)
31. Sistla, A.P., Clarke, E.M.: The complexity of propositional linear temporal logics. Journal of the ACM **32**(3), 733–749 (1985)
32. Yang, F., Väänänen, J.: Propositional logics of dependence. Annals of Pure and Applied Logic **167**(7), 557–589 (2016)

Temporal Stream Logic modulo Theories*

Bernd Finkbeiner, Philippe Heim✉, and Noemi Passing

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
{finkbeiner, philippe.heim, noemi.passing}@cispa.de

Abstract. Temporal stream logic (TSL) extends LTL with updates and predicates over arbitrary function terms. This allows for specifying data-intensive systems for which LTL is not expressive enough. In the semantics of TSL, functions and predicates are left uninterpreted. In this paper, we extend TSL with first-order theories, enabling us to specify systems using interpreted functions and predicates such as incrementation or equality. We investigate the satisfiability problem of TSL modulo the standard underlying theory of uninterpreted functions as well as with respect to Presburger arithmetic and the theory of equality: For all three theories, TSL satisfiability is neither semi-decidable nor co-semi-decidable. Nevertheless, we identify three fragments of TSL for which the satisfiability problem is (semi-)decidable in the theory of uninterpreted functions. Despite the undecidability, we present an algorithm – which is not guaranteed to terminate – for checking the satisfiability of a TSL formula in the theory of uninterpreted functions and evaluate it: It scales well and is able to validate assumptions in a real-world system design.

1 Introduction

Linear-time temporal logic (LTL) [32] is one of the standard specification languages to describe properties of reactive systems. The success of LTL is largely due to its ability to abstract from the detailed data manipulations and to focus on the change of control over time. In data-intensive applications, such as smartphone apps, LTL is, however, often not expressive enough to capture the relevant properties. When specifying a music player app, for instance, we would like to state that if the user leaves the app, the track that is currently playing will be stored and will resume playing once the user returns to the app.

To specify data-intensive systems, extensions of LTL such as Constraint LTL (CLTL) [6] and, more recently, Temporal Stream Logic (TSL) [15] have been proposed. In CLTL, the atomic propositions of LTL are replaced with atomic constraints over a concrete domain D and an interpretation for relations. Relating variables with the equality relation, such as $x = y$, denoting that the value

---

of x is equal to the value of y, allows for specifying assignment-like statements. In this paper, however, we focus on the logic TSL to specify data-intensive systems.

TSL extends LTL with updates and predicates over arbitrary function terms. An update $[\![x \leftarrow f(y)]\!]$ denotes that the result of applying function f to variable y is assigned to variable x. For the music player app, for instance, the update $[\![paused \leftarrow track(current)]\!]$ specifies that the track that is currently playing, obtained by applying function $track$ to variable $current$, is stored in variable $paused$. Updates are the main characteristic of TSL that differentiates it from other first-order extensions of LTL: They allow for specifying the evolution of variables over time. Thus, programs can be represented in TSL and therefore, for instance, the model checking problem can be encoded.

In the semantics of TSL, functions and predicates are left uninterpreted, i.e., a system satisfies a TSL formula if the formula evaluates to *true* for all possible interpretations of the function and predicate symbols. This semantics has proven especially useful in the synthesis of reactive programs [15,17], where the synthesis algorithm builds a control structure, while the implementation of the functions and predicates is either done manually or provided by some library. One exemplary success story of TSL-based specification and synthesis of a reactive system is the arcade game Syntroids [17] realized on an FPGA.

In this paper, we define and investigate the satisfiability problem of TSL modulo the standard underlying theory of uninterpreted functions and with respect to other first-order theories such as the theory of equality and Presburger arithmetic. Intuitively, a TSL formula φ is satisfiable in a theory T if there is an execution satisfying φ that matches the function applications and predicate constraints of an interpretation in T. TSL validity in T is dual: A TSL formula φ is valid in a theory T if, and only if, $\neg\varphi$ is unsatisfiable in T.

For LTL, satisfiability is decidable [37] and efficient algorithms for checking the satisfiability of an LTL formula have been implemented in tools like Aalta [25]. Satisfiability checking has numerous applications in the specification and analysis of reactive systems, such as identifying inconsistent system requirements during the design process, comparing different formalizations of the same requirements, and various types of vacuity checking. TSL satisfiability checking extends these applications to data-intensive systems.

We present an algorithm for checking the satisfiability of a TSL formula in the theory of uninterpreted functions. It is based on *Büchi stream automata* (BSAs), a new kind of ω-automata that we introduce in this paper. BSAs can handle the predicates and updates occurring in TSL formulas. Similar to the relationship between LTL formulas and nondeterministic Büchi automata, BSAs are an automaton representation of TSL formulas, i.e., there exists an equivalent BSA for every TSL formula. Given a TSL formula φ, our algorithm constructs an equivalent BSA $\mathcal{B}_\varphi$ and then tries to prove satisfiability and unsatisfiability in parallel: For proving satisfiability, it searches for a lasso that ensures consistency of the function terms in an accepting run of $\mathcal{B}_\varphi$. If such a lasso is found, φ is satisfiable. For proving unsatisfiability, the algorithm discards inconsistent runs of $\mathcal{B}_\varphi$. If no accepting run is left, φ is unsatisfiable.

The algorithm does not always terminate. In fact, we show that TSL satisfiability is neither semi-decidable nor co-semi-decidable in the theory of uninterpreted functions. Thus, no complete algorithm exists. The undecidability result extends to the theory of equality and Presburger arithmetic. There exist, however, (semi-)decidable fragments of TSL in the theory of uninterpreted functions: For satisfiable formulas with a single variable as well as satisfiable reachability formulas, our algorithm is guaranteed to terminate. For slightly more restricted reachability formulas, satisfiability is decidable.

We have implemented the algorithm and evaluated it, clearly illustrating its applicability: It terminates within one second on many randomly generated formulas and scales particularly well for satisfiable formulas. Moreover, it is able to check realistic benchmarks for consistency and to (in-)validate their assumptions. Most notably, we successfully validate the assumptions of a Syntroids module.

A preliminary version of this paper has been published on arXiv [13]. This already lead to further research on TSL modulo theories: Maderbacher and Bloem show that the synthesis problem for TSL modulo theories is undecidable in general and present a synthesis procedure for TSL modulo theories based on a counter-example guided LTL synthesis loop [27].

Further details and proofs are available in the full version of this paper [14].

2   Preliminaries

We assume time to be discrete. A *value* can be of arbitrary type and we denote the set of all values by $\mathcal{V}$. The Boolean values are denoted by $\mathbb{B} \subseteq \mathcal{V}$. Given n values, an n-ary function $f : \mathcal{V}^n \to \mathcal{V}$ computes a new value. An n-ary predicate $p : \mathcal{V}^n \to \mathbb{B}$ determines whether a property over n values is satisfied. The sets of all functions and predicates are denoted by $\mathcal{F}$ and $\mathcal{P} \subseteq \mathcal{F}$, respectively. Constants are both functions of arity zero and values. Starting from 0, we denote the i-th position of an infinite word σ by σ_i and the i-th component of a tuple t by $\pi_i(t)$.

To argue about functions and predicates, we use a term based notation. *Function terms* τ_f are constructed from variables and functions, recursively applied to a set of function terms. *Predicate terms* τ_p are constructed by applying a predicate to function terms. The sets of all function and predicate terms are denoted by $\mathcal{T}_F$ and $\mathcal{T}_P \subseteq \mathcal{T}_F$, respectively. Given sets Σ_F, Σ_P of function and predicate symbols with $\Sigma_P \subseteq \Sigma_F$, a set V of variables, and a set $\mathcal{V}$ of values, let $\langle \cdot \rangle : V \cup \Sigma_F \to \mathcal{V} \cup \mathcal{F}$ be an *assignment function* assigning a concrete function (predicate) to each function (predicate) symbol and an initial value to each variable. We require $\langle v \rangle \in \mathcal{V}$, $\langle f \rangle \in \mathcal{F}$, and $\langle p \rangle \in \mathcal{P}$ for $v \in V$, $f \in \Sigma_F$, $p \in \Sigma_P$. The evaluation $\chi_{\langle \cdot \rangle} : \mathcal{T}_F \to \mathcal{V} \cup \mathbb{B}$ of function terms is defined by $\chi_{\langle \cdot \rangle}(v) := \langle v \rangle$ for $v \in V$, and by $\chi_{\langle \cdot \rangle}(f(\tau_0, \ldots, \tau_n)) := \langle f \rangle (\chi_{\langle \cdot \rangle}(\tau_0), \ldots, \chi_{\langle \cdot \rangle}(\tau_n))$ for $f \in \Sigma_F \cup \Sigma_P$.

Functions and predicates are not tied to a specific interpretation. To restrict the possible interpretations, we utilize *first-order theories*. A first-order theory T is a tuple $(\Sigma_F, \Sigma_P, \mathcal{A})$, where Σ_F and Σ_P are sets of function and predicate symbols, respectively, and $\mathcal{A}$ is a set of closed first-order logic formulas over Σ_F, Σ_P, and a set of variables V. For an introduction to first-order logic, we refer to the

full version [14]. The elements of $\mathcal{A}$ are called the *axioms* of T and $\Sigma_F \cup \Sigma_P$ is called the *signature* of T. A *model* $\mathcal{M}$ for a theory $T = (\Sigma_F, \Sigma_P, \mathcal{A})$ is a tuple $(\mathcal{V}, \langle \cdot \rangle)$, where $\mathcal{V}$ is a set of values and $\langle \cdot \rangle$ is an assignment function as introduced above. Furthermore, $(\mathcal{V}, \langle \cdot \rangle)$ is required to entail φ_A for each axiom $\varphi_A \in \mathcal{A}$. The set of all models of a theory T is denoted by $Models(T)$.

In the remainder of this paper, we focus on the following three theories: The *theory of uninterpreted functions* T_U is a theory without any axioms, i.e., every symbol is uninterpreted. It allows for arbitrarily many function and predicate symbols. The *theory of equality* T_E additionally includes equality, i.e., its axioms enforce the equality symbol $=$ to indeed represent equality. The *theory of Presburger arithmetic* $T_{\mathbb{N}}$ implements the idea of numbers. Its axioms define the constants 0 and 1 as well as equality and addition.

3   Temporal Stream Logic modulo Theories

In this section, we introduce *Temporal Stream Logic modulo theories*, an extension of the recently introduced logic Temporal Stream Logic (TSL) [15] with first-order theories. First, we recap the main idea of TSL as well as its syntax and semantics. Afterwards, we extend TSL with first-order theories and define the basic notions of satisfiability and validity for TSL formulas modulo theories.

3.1   Temporal Stream Logic

Temporal Stream Logic (TSL) [15] is a temporal logic that separates temporal control and pure data. Data is represented as infinite streams of arbitrary type. TSL allows for checks and manipulations of streams on an abstract level: It focuses on the control flow and abstracts away concrete implementation details. The temporal structure of the data is expressed by temporal operators as in LTL [32]. TSL is especially designed for reactive synthesis and thus distinguishes between uncontrollable input streams and controllable output streams, so-called *cells*. In this paper, this distinction is not necessary since we consider TSL independent of its usage in synthesis. Thus, we use the notions of streams and cells as synonyms. The finite set of all cells is denoted by $\mathbb{C}$.

In TSL, we use functions $f \in \mathcal{F}$ to modify cells and predicates $p \in \mathcal{P}$ to perform checks on cells. The cells $\mathsf{c} \in \mathbb{C}$ serve as variables for function terms. The sets of all function and predicate terms over Σ_F, Σ_P, and $\mathbb{C}$ are denoted by $\mathcal{T}_F$ and $\mathcal{T}_P$. TSL formulas are built according to the following grammar:

$$\varphi, \psi := \mathit{true} \mid \neg\varphi \mid \varphi \wedge \psi \mid \bigcirc\varphi \mid \varphi\,\mathcal{U}\,\psi \mid \tau_p \mid [\![\mathsf{c} \hookleftarrow \tau_f]\!]$$

where $\mathsf{c} \in \mathbb{C}$, $\tau_p \in \mathcal{T}_P$, and $\tau_f \in \mathcal{T}_F$. An *update* $[\![\mathsf{c} \hookleftarrow \tau_f]\!]$ denotes that the value of the function term τ_f is assigned to cell c. The value of τ_f may depend on the value of the cells occurring in τ_f. The temporal operators $\bigcirc\varphi$ and $\varphi\,\mathcal{U}\,\psi$ are similar to the ones in LTL. We define $\Diamond\varphi = \mathit{true}\,\mathcal{U}\,\varphi$ and $\Box\varphi = \neg\Diamond\neg\varphi$.

Since functions and predicates are represented symbolically, they are not tied to a specific implementation. To assign an interpretation to them, we use an

assignment function $\langle \cdot \rangle : \mathbb{C} \cup \Sigma_F \to \mathcal{V} \cup \mathcal{F}$, where $\mathcal{V}$ is a set of values. We require $\langle c \rangle \in \mathcal{V}$, $\langle f \rangle \in \mathcal{F}$ and $\langle p \rangle \in \mathcal{P}$ for $c \in \mathbb{C}$, $f \in \Sigma_F$, and $p \in \Sigma_P$ Note that $\langle \cdot \rangle$ also assigns an initial value to each cell. Terms can be compared syntactically with the equivalence relation $\equiv$. The set of all assignments of cells $c \in \mathbb{C}$ to function terms $\tau_f \in \mathcal{T}_F$ is denoted by $\mathcal{C}$. A *computation* $\varsigma \in \mathcal{C}^\omega$ is an infinite sequence of assignments of cells to function terms, capturing the behavior of cells over time. The satisfaction of a TSL formula φ with respect to ς, a set of values $\mathcal{V}$, an assignment function $\langle \cdot \rangle$, and a time step t is defined by:[1]

$$\varsigma, t \models_{\mathcal{V}, \langle \cdot \rangle} \neg\varphi \qquad :\Leftrightarrow \varsigma, t \not\models_{\mathcal{V}, \langle \cdot \rangle} \varphi$$

$$\varsigma, t \models_{\mathcal{V}, \langle \cdot \rangle} \varphi \wedge \psi \qquad :\Leftrightarrow \varsigma, t \models_{\mathcal{V}, \langle \cdot \rangle} \varphi \wedge \varsigma, t \models_{\mathcal{V}, \langle \cdot \rangle} \psi$$

$$\varsigma, t \models_{\mathcal{V}, \langle \cdot \rangle} \bigcirc\varphi \qquad :\Leftrightarrow \varsigma, t+1 \models_{\mathcal{V}, \langle \cdot \rangle} \varphi$$

$$\varsigma, t \models_{\mathcal{V}, \langle \cdot \rangle} \varphi\, \mathcal{U}\, \psi \qquad :\Leftrightarrow \exists t'' \geq t. \forall t \leq t' < t''. \varsigma, t' \models_{\mathcal{V}, \langle \cdot \rangle} \varphi \ \wedge\ \varsigma, t'' \models_{\mathcal{V}, \langle \cdot \rangle} \psi$$

$$\varsigma, t \models_{\mathcal{V}, \langle \cdot \rangle} \llbracket c \hookleftarrow \tau \rrbracket \qquad :\Leftrightarrow \varsigma_t(c) \equiv \tau$$

$$\varsigma, t \models_{\mathcal{V}, \langle \cdot \rangle} p(\tau_0, \ldots, \tau_m) :\Leftrightarrow \chi_{\langle \cdot \rangle}(\eta(\varsigma, t, p(\tau_0, \ldots, \tau_m))),$$

where $\eta : \mathcal{C}^\omega \times \mathbb{N} \times \mathcal{T}_F \to \mathcal{T}_F$ is a symbolic evaluation function defined by

$$\eta(\varsigma, t, c) = \begin{cases} c & \text{if } t = 0 \\ \eta(\varsigma, t-1, \varsigma_{t-1}(c)) & \text{if } t > 0 \end{cases}$$

$$\eta(\varsigma, t, f(\tau_0, \ldots, \tau_m)) = f(\eta(\varsigma, t, \tau_0), \ldots, \eta(\varsigma, t, \tau_m))$$

We call $(\varsigma, \mathcal{V}, \langle \cdot \rangle)$ an *execution*. The satisfaction of a predicate depends on the current and the past steps in the computation. For updates, the satisfaction depends solely on the current step. While updates are only checked syntactically, the satisfaction of predicates depends on the given assignment $\langle \cdot \rangle$. An execution $(\varsigma, \mathcal{V}, \langle \cdot \rangle)$ *satisfies* a TSL formula φ, denoted $\varsigma \models_{\mathcal{V}, \langle \cdot \rangle} \varphi$, if $\varsigma, 0 \models_{\mathcal{V}, \langle \cdot \rangle} \varphi$ holds.

Example 1. Suppose that we have a single cell x, i.e., $\mathbb{C} = \{x\}$. Consider the computation $\varsigma = (\{\lambda c.f(x)\})^\omega$, i.e., $f(x)$ is assigned to cell x in every time step. Let $\mathcal{V} = \mathbb{N}$ be the set of values and let $\langle \cdot \rangle$ be an assignment function such that the initial value of x is 1, function f corresponds to incrementation, and predicate p determines whether its argument is even (*true*) or odd (*false*). Consider the TSL formula $\varphi := \llbracket x \hookleftarrow f(x) \rrbracket \wedge \neg p(x) \wedge \bigcirc p(x)$. By the semantics of TSL, we have $\varsigma, 0 \models_{\mathcal{V}, \langle \cdot \rangle} \varphi$ if, and only if, $(\varsigma_0(x) = f(x)) \wedge (\neg\langle p \rangle(\langle x \rangle)) \wedge (\langle p \rangle(\langle f \rangle(\langle x \rangle)))$ holds. The first conjunct clearly holds by construction of ς. Since 1 is odd and $1 + 1 = 2$ is even, the other two conjuncts hold as well for the chosen assignment function. Hence, $(\varsigma, \mathcal{V}, \langle \cdot \rangle)$ satisfies φ for $\varsigma = (\{\lambda c.f(x)\})^\omega$, $\mathcal{V} = \mathbb{N}$ and $\langle \cdot \rangle$.

A computation ς is called *finitary* with respect to φ, denoted $\text{fin}_\varphi(\varsigma)$, if for all cells $c \in \mathbb{C}$ and for all points in time t, either $\varsigma_t(c) \equiv c$ holds, or there is an update $\llbracket c \hookleftarrow \tau \rrbracket$ in φ such that $\varsigma_t(c) \equiv \tau$, i.e., a finitary computation only contains updates occurring in φ and self-updates. For ς and φ from Example 1, for instance, ς is finitary with respect to φ.

---

[1]Note that we use a slightly different, but equivalent, definition than [15]: Instead of evaluating the function and predicate symbols on the fly, we construct the whole term first and then evaluate it recursively using the evaluation function $\chi_{\langle \cdot \rangle}$.

3.2 Extending TSL with Theories

In this paper, we extend TSL with first-order theories. That is, we restrict the possible interpretations of predicate and function symbols to a theory. Hence, we define the notions of satisfiability and validity of a TSL formula *modulo a theory T*. Intuitively, a TSL formula φ is satisfiable in a theory T if there exists an execution satisfying φ whose domain and assignment function represent a model in T, i.e., that entail all axioms of T. Formally:

Definition 1 (TSL Satisfiability). *Let $T = (\Sigma_F, \Sigma_P, \mathcal{A})$ be a theory and let φ be a TSL formula over Σ_F, Σ_P, and $\mathbb{C}$. We call φ satisfiable in T if, and only if, there exists an execution $(\varsigma, \mathcal{V}, \langle \cdot \rangle)$, such that $\varsigma \models_{\mathcal{V}, \langle \cdot \rangle} \varphi$ and $(\mathcal{V}, \langle \cdot \rangle) \in Models(T)$ hold. If additionally $\mathrm{fin}_\varphi(\varsigma)$ holds, then φ is called finitary satisfiable in T.*

Intuitively, a formula φ is valid in a theory T, if for all executions and all matching models of the theory the formula is satisfied. Formally:

Definition 2 (TSL Validity). *Let $T = (\Sigma_F, \Sigma_P, \mathcal{A})$ be a theory and let φ be a TSL formula over Σ_F, Σ_P, and $\mathbb{C}$. The formula φ is called valid in T if, and only if, for all executions $(\varsigma, \mathcal{V}, \langle \cdot \rangle)$ with $(\mathcal{V}, \langle \cdot \rangle) \in Models(T)$, we have $\varsigma \models_{\mathcal{V}, \langle \cdot \rangle} \varphi$. If $\varsigma \models_{\mathcal{V}, \langle \cdot \rangle} \varphi$ holds for all executions $(\varsigma, \mathcal{V}, \langle \cdot \rangle)$ with both $(\mathcal{V}, \langle \cdot \rangle) \in Models(T)$ and $\mathrm{fin}_\varphi(\varsigma)$, then φ is called finitary valid in T.*

It follows directly from their definitions that (finitary) TSL satisfiability and (finitary) TSL validity are dual. Thus, we focus on TSL satisfiability in the remainder of this paper as the results can easily be extended to TSL validity.

Theorem 1 (Duality of TSL Satisfiability and Validity). *Let φ be a TSL formula over Σ_F, Σ_P, and $\mathbb{C}$ and let $T = (\Sigma_F, \Sigma_P, \mathcal{A})$ be a theory. Then, φ is (finitary) satisfiable in T if, and only if, $\neg\varphi$ is not (finitary) valid in T.*

4   TSL modulo T_U Satisfiability Checking

In this section, we investigate the satisfiability of TSL modulo the theory of uninterpreted functions T_U. Since T_U has no axioms, there are no restrictions on how a model for T_U evaluates the function and predicate symbols. The only condition is that the evaluated symbols are indeed functions. Therefore, we have $(\varsigma, \mathcal{V}, \langle \cdot \rangle) \in Models(T_U)$ for all executions. Thus, finding some execution satisfying a TSL formula φ is sufficient for showing that φ is satisfied in T_U:

Lemma 1. *Let φ be a TSL formula over Σ_F, Σ_P, and $\mathbb{C}$. If there exists an execution $(\varsigma, \mathcal{V}, \langle \cdot \rangle)$ with $\varsigma \models_{\mathcal{V}, \langle \cdot \rangle} \varphi$, then φ is satisfiable in T_U. If additionally $\mathrm{fin}_\varphi(\varsigma)$ holds, then φ is finitary satisfiable in T_U.*

In the following, we introduce an (incomplete) algorithm for checking the satisfiability of a TSL formula φ in the theory of uninterpreted functions. By

Lemma 1, it suffices to find an execution satisfying φ to prove its satisfiability in T_U. To search for such an execution, we introduce *Büchi stream automata* (BSAs), a new kind of ω-automata that reads executions and allows for dealing with predicates and updates. BSAs are, similar to the connection between LTL and Büchi automata, an automaton representation for TSL. Then, we present the algorithm for checking satisfiability in T_U based on BSAs.

4.1 Büchi Stream Automata

Intuitively, a *Büchi stream automaton* (BSA) is an ω-automaton with Büchi acceptance condition that reads infinite executions instead of infinite words. Furthermore, it is able to deal with predicates and updates occurring in TSL formulas. To do so, the transitions of a BSA are labeled with *guards* and *update terms*. Intuitively, the former define which predicates need to hold when taking the transition. The latter define how the corresponding cells are updated when taking the transition. Formally, a BSA is defined as follows:

Definition 3 (Büchi Stream Automaton). *Let Σ_F, Σ_P be sets of function and predicate symbols, respectively, and let $\mathbb{C}$ be a finite set of cells. A* Büchi Stream automaton $\mathcal{B}$ *over Σ_F, Σ_P, and $\mathbb{C}$ is a tuple $(Q, Q_0, F, \bullet, \mathcal{G}, \mathcal{U}, \delta)$, where Q is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of accepting states, $\bullet$ is a fresh term symbol such that $\bullet \notin \mathbb{C} \cup \Sigma_F \cup \Sigma_P$, $\mathcal{G} \subseteq \mathcal{T}_P$ is a finite set of predicate terms over Σ_F, Σ_P, and $\mathbb{C}$, called* guards, $\mathcal{U} \subseteq \mathcal{T}_F \cup \{\bullet\}$ *is a finite set of function terms over Σ_F, Σ_P, and $\mathbb{C}$, called* update terms, *and $\delta \subseteq Q \times (\mathcal{G} \to \mathbb{B}) \times (\mathbb{C} \to \mathcal{U}) \times Q$ is a finite transition relation.*

Note that by requiring the update terms $\mathcal{U}$ to be a *finite* set of function terms, not all executions can be read by a BSA: Non-finitary executions contain updates with function terms that do not occur in the given TSL formula. Thus, they may require infinitely many update terms. Therefore, we introduce the fresh term symbol $\bullet \notin \mathbb{C} \cup \Sigma_F \cup \Sigma_P$. If a transition in a BSA assigns $\bullet$ to a cell $c \in \mathbb{C}$, then any function term can be assigned to c. This allows for reading non-finitary executions while maintaining finite representability of BSAs.

Example 2. Consider the three BSAs depicted in Figure 1. If $\mathcal{B}_1$ is in state q_0 and $p(x)$ holds, then cell x is updated with $f(x)$ and $\mathcal{B}_1$ chooses nondeterministically to either stay in q_0 or to move to the accepting state q_1. In contrast, $\mathcal{B}_2$ is deterministic. Yet, it is incomplete: In both q_0 and q_1, no guard is satisfied if $\neg p(x)$ holds. Hence, $\mathcal{B}_2$ gets stuck, preventing satisfaction of the Büchi winning condition for any execution containing $\neg p(x)$. The BSA $\mathcal{B}_3$ makes use of the fresh term symbol $\bullet$: If $p(x)$ holds, any function term can be assigned to x.

Given sets Σ_F, Σ_P, $\mathbb{C}$ and a BSA $\mathcal{B} = (Q, Q_0, F, \bullet, \mathcal{G}, \mathcal{U}, \delta)$ over Σ_F, Σ_P, $\mathbb{C}$, an infinite word $c \in (Q \times (\mathcal{G} \to \mathbb{B}) \times (\mathbb{C} \to \mathcal{U}) \times Q)^\omega$ is called *run of $\mathcal{B}$* if, and only if, the first state of c is an initial state, i.e., $\pi_1(c_0) \in Q_0$, and both $c_t \in \delta$ and $\pi_4(c_t) = \pi_1(c_{t+1})$ hold for all points in time $t \in \mathbb{N}_0$. Intuitively, a run c is an infinite sequence of tuples (q, g, u, q') encoding transitions in the BSA: q is the

(a) BSA $\mathcal{B}_1$ (b) BSA $\mathcal{B}_2$ (c) BSA $\mathcal{B}_3$
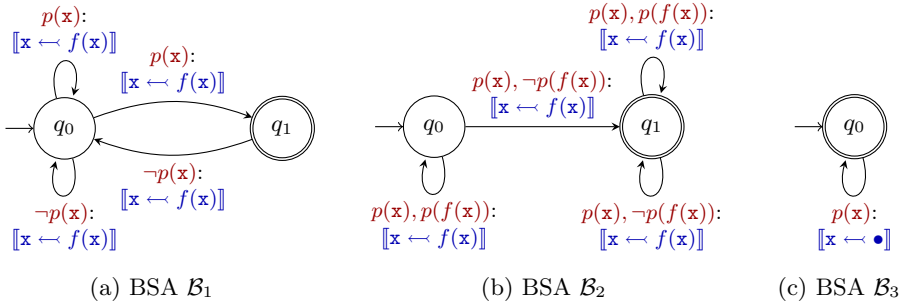
Fig. 1: Three exemplary Büchi stream automata. Accepting states are marked with double circles. Guards are highlighted in red, update terms in blue.

source state, q' is the target state, g determines which predicate terms hold, and u defines which updates are performed when taking the transition. A run c is called *accepting* if it contains infinitely many accepting states, i.e., for all points in time $t \in \mathbb{N}_0$, there exists a $t' > t$ such that $\pi_1(c_{t'}) \in F$ holds.

Example 3. Let $g_1(p(\mathbf{x})) = \textit{true}$, $g_2(p(\mathbf{x})) = \textit{false}$, and $u(\mathbf{x}) = f(\mathbf{x})$. The infinite word $c = (q_0, g_1, u, q_1)(q_1, g_2, u, q_0)(q_0, g_1, u, q_1)(q_1, g_2, u, q_0)\ldots$ is a run of BSA $\mathcal{B}_1$ from Figure 1a. It is accepting as it visits q_1 infinitely often.

    The characteristics of a BSA are its predicates and updates. Thus, it is not sufficient to solely consider accepting runs since the constraints produced by the predicates might be inconsistent. Therefore, we define the *execution of a BSA* that only permits consistent accepting runs. Intuitively, given a run c of a BSA $\mathcal{B}$, an *execution of c* consists of a computation $\varsigma \in \mathcal{C}^\omega$, a domain $\mathcal{V}$, and an assignment $\langle \cdot \rangle$ such that the updates in ς match the updates in c and such that the recursive evaluation of a predicate term using $\langle \cdot \rangle$ matches its truth value in ς. To capture the constraints accumulated in ς as well as their truth values, we define the *constraint trace* $\varrho : (\tau_p \times \mathbb{B})^\omega$ *of ς and c*: Formally, ϱ for ς and c is defined by $\varrho_t := \emptyset$ if $t = 0$, and $\varrho_t := \varrho_{t-1} \cup \{(\eta(\varsigma, t-1, \tau_p), \pi_2(c_{t-1})(\tau_p)) \mid \tau_p \in \mathcal{G}\}$ if $t > 0$. As an example, reconsider the computation ς from Example 1 and the run c of BSA $\mathcal{B}_1$ from Example 3. The constraint trace of ς and c is given by $\varrho = \emptyset\{(p(\mathbf{x}), \textit{true})\}\{(p(\mathbf{x}), \textit{true}), (p(f(\mathbf{x})), \textit{false})\}\ldots$. A constraint trace ϱ is called *consistent* if there is no predicate term $\tau_p \in \mathcal{T}_P$ such that both $(\tau_p, \textit{true})$ and $(\tau_p, \textit{false})$ occur in $\bigcup_{t \in \mathbb{N}_0} \varrho_t$. ϱ from the example above is consistent. Using constraint traces, we now formally define the execution of a BSA:

Definition 4 (Execution of a BSA). *Let Σ_F and Σ_P be sets of function and predicate symbols, respectively, and let $\mathbb{C}$ be a finite set of cells. Let $\mathcal{B}$ be a BSA over Σ_F, Σ_P, and $\mathbb{C}$ and let c be a run of $\mathcal{B}$. Let $\varsigma \in \mathcal{C}^\omega$ be an infinite computation and let $\langle \cdot \rangle : \mathbb{C} \cup \Sigma_F \to \mathcal{V} \cup \mathcal{F}$ be an assignment function. Let ϱ be the constraint trace of ς and c. We call $(\varsigma, \mathcal{V}, \langle \cdot \rangle)$ execution for c if (1) for all points in time $t \in \mathbb{N}_0$ and all cells $\mathbf{c} \in \mathbb{C}$, we have either $\pi_3(c_t)(\mathbf{c}) = \varsigma_t(\mathbf{c})$ or $\pi_3(c_t)(\mathbf{c}) = \bullet$, and (2) for all $(\tau_p, b) \in \bigcup_{t \in \mathbb{N}_0} \varrho_t$, we have $\chi_{\langle \cdot \rangle}(\tau_p) = b$.*

Note that the second requirement can only be fulfilled if the constraint trace is consistent. Consider the computation ς and the assignment function $\langle\cdot\rangle$ from Example 1, the run c of $\mathcal{B}_1$ from Example 3, and the constraint trace ϱ of ς and c given above. Then, $(\varsigma, \mathbb{N}, \langle\cdot\rangle)$ is an execution for c: Since in both ς and c, cell $\mathbf{x}$ is always updated with $f(\mathbf{x})$, the updates in ς and c coincide at every point in time. Furthermore, by construction of $\langle\cdot\rangle$, the constraints of ϱ match the truth values obtained by recursively evaluating $\langle\cdot\rangle$ for all predicate terms.

We define two languages of a BSA $\mathcal{B}$: The *symbolic language* $\mathcal{L}(\mathcal{B})$ is the set of all executions that have a respective accepting run, i.e., $(\varsigma, \mathcal{V}, \langle\cdot\rangle) \in \mathcal{L}(\mathcal{B})$ if, and only if, there exists an accepting run c such that $(\varsigma, \mathcal{V}, \langle\cdot\rangle)$ is an execution for c. The *language* $\mathcal{L}_T(\mathcal{B})$ *in a theory* T is the set of all executions whose domain and assignment function additionally form a model in T, i.e., $(\varsigma, \mathcal{V}, \langle\cdot\rangle) \in \mathcal{L}_T(\mathcal{B})$ if, and only if, $(\varsigma, \mathcal{V}, \langle\cdot\rangle) \in \mathcal{L}(\mathcal{B})$ and $(\mathcal{V}, \langle\cdot\rangle) \in Models(T)$.

We call a BSA $\mathcal{B} = (Q, Q_0, F, \bullet, \mathcal{G}, \mathcal{U}, \delta)$ *finitary* if $\bullet \notin \mathcal{U}$ holds. Hence, every run c of a finitary BSA, has a unique computation ς and thus a unique constraint trace ϱ. Therefore, for a finite prefix c_p of c, we can compute its *execution effect* $\mathrm{effect}(c_p) := (\lambda\mathsf{c}.\, \eta(\varsigma, |c_p|, \mathsf{c}), \varrho_{|c_p|})$ from c_p itself, i.e., without considering ς and ϱ explicitly. Intuitively, c_p's execution effect consists of the function terms assigned to the cells during the execution of c_p as well as the constraints and their truth values on the transitions taken with c_p in the BSA. The BSAs $\mathcal{B}_1$ and $\mathcal{B}_2$, depicted in Figure 1, are finitary while $\mathcal{B}_3$ is not. Since $\mathcal{B}_1$ is finitary, consider the prefix $c_p = (q_0, g_1, u, q_1)(q_1, g_2, u, q_0)$ of the run c of $\mathcal{B}_1$ presented in Example 3. Its execution effect is given by $\mathrm{effect}(c_p) = (\lambda\mathsf{c}.f(f(\mathbf{x})), \{(p(\mathbf{x}), true), (p(f(\mathbf{x})), false)\})$.

An LTL formula φ can be translated into a nondeterministic Büchi automaton (NBA) $\mathcal{A}_\varphi$ with $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A}_\varphi)$ [38]. An analogous relation exists between TSL formulas and BSAs: A TSL formula φ can be translated into an equivalent BSA $\mathcal{B}_\varphi$: First, we approximate φ by an LTL formula φ_{LTL}, similarly to the approximation described in [15]. The main idea of the approximation is to represent every function and predicate term as well as every update occurring in φ_{LTL} by an atomic proposition and to add conjuncts that ensure that exactly one update is performed for every cell in every time step. Second, we build an equivalent NBA $\mathcal{A}_{\varphi_{LTL}}$ from φ_{LTL}. Third, we construct a BSA $\mathcal{B}_\varphi$ from $\mathcal{A}_{\varphi_{LTL}}$ by, intuitively, translating the atomic propositions back into predicate terms and updates and by dividing them into guards and update terms, while maintaining the structure of the NBA $\mathcal{A}_{\varphi_{LTL}}$. The full construction of an equivalent BSA $\mathcal{B}_\varphi$ from a TSL formula φ is given in the full version [14].

Theorem 2 (TSL to BSA Translation). *Given a TSL formula φ, there exists an equivalent (finitary) Büchi stream automaton $\mathcal{B}$ such that for all theories T, $\mathcal{L}_T(\mathcal{B}) \neq \emptyset$ holds if, and only if, φ is (finitary) satisfiable in T.*

For instance, the TSL formula $\varphi_1 := \Box[\![\mathbf{x} \leftarrowtail f(\mathbf{x})]\!] \wedge \Box\Diamond(p(\mathbf{x}) \wedge \bigcirc \neg p(\mathbf{x}))$ is finitary satisfiable in a theory T if, and only if, $\mathcal{L}_T(\mathcal{B}_1) \neq \emptyset$ holds for the BSA $\mathcal{B}_1$ from Figure 1a. Analogously, $\varphi_2 := \Box([\![\mathbf{x} \leftarrowtail f(\mathbf{x})]\!] \wedge p(\mathbf{x})) \wedge \Diamond\neg p(f(\mathbf{x}))$, and $\varphi_3 := \Box p(\mathbf{x})$ correspond to the BSAs $\mathcal{B}_2$ and $\mathcal{B}_3$ from Figure 1b and Figure 1c.

Algorithm 1: Algorithm for Checking TSL modulo T_U Satisfiability

---

    Input: φ: TSL Formula
    Output: SAT, UNSAT

1 $\mathcal{B} :=$ Finitary BSA for φ as defined in Theorem 2;
2 $\mathcal{R} :=$ Set of runs of $\mathcal{B}$;
3 **Function** *SatSearch*
4   **for** $pref.rec^\omega \in \{c \mid c \in \mathcal{R} \wedge \text{accepting}(c)\}$ **do**
5 $(v_p, \_):=\text{effect}(pref)$;
6 $(v_r, P):=\text{effect}(pref.rec)$;
7   **if** $\text{SMT}\left(\left(\bigwedge_{(t_p,v)\in P} \begin{cases} t_p & \text{if } v = true \\ \neg t_p & \text{if } v = false \end{cases}\right) \wedge \bigwedge_{\mathsf{c}\in\mathbb{C}} v_p(\mathsf{c}) = v_r(\mathsf{c})\right) = \text{SAT}$ **then**
8   **return** SAT

9 **Function** *UnsatSearch*
10   **for** $n \in \mathbb{N}_0$ **do**
11   **for** $c \in \{c \mid c \in \text{finiteSubwords}(\mathcal{R}) \wedge |c| = n\}$ **do**
12 $(\_, P):=\text{effect}(c)$;
13   **if** $\exists t_p. \, (t_p, true), (t_p, false) \in P$ **then**
14 $\mathcal{R} := \mathcal{R} \setminus \{c' \mid \exists m \in \mathbb{N}_0. \, \forall 0 \leq i < n. \, c'_{i+m} = c_i\}$
15   **if** $\{c \mid c \in \mathcal{R} \wedge \text{accepting}(c)\} = \emptyset$ **then**
16   **return** UNSAT

17 **return** *parallel(SatSearch, UnsatSearch)*

4.2   An Algorithm for TSL modulo T_U Satisfiability Checking

Utilizing BSAs, we present an algorithm for checking the satisfiability of a TSL formula in the theory of uninterpreted functions T_U in the following. First, recall that finitary computations only perform self-updates or updates that occur in the given TSL formula. Since there are only finitely many cells, the behavior of finitary computations is thus restricted to a finite set of possibilities in each step. Hence, reasoning with finitary computations is easier than reasoning with non-finitary ones. In the algorithm, we make use of the fact that satisfiability can be reduced to finitary satisfiability in the theory of uninterpreted functions, enabling us to focus on finitary computations. The main idea of the reduction is to introduce a new cell for each cell of a given TSL formula. The new cells then capture the values that are constructed by the non-finitary parts of a computation. The proof is given in the full version [14].

Lemma 2. *Let φ be a TSL formula. Then, there is a TSL formula φ_{fin} such that φ is satisfiable in T_U if, and only if, $\varphi \wedge \varphi_{fin}$ is finitary satisfiable in T_U.*

Algorithm 1 shows the algorithm for checking TSL modulo T_U satisfiability. It directly works on Büchi stream automata. First, an equivalent BSA $\mathcal{B}$ is generated for the input formula φ. Then, in parallel, *SatSearch* tries to prove that φ is satisfiable in T_U while *UnsatSearch* tries to prove unsatisfiability of φ.

SatSearch enumerates all lasso-shaped accepting runs *pref.rec^ω* of $\mathcal{B}$, i.e., accepting runs consisting of a finite prefix *pref* and a finite recurring part *rec* that is repeated infinitely often. Both *pref* and *rec* need to end in the same state of $\mathcal{B}$. Then, the execution effects of *pref* and *pref.rec* are computed. *SatSearch* checks if it is possible to satisfy all predicate constraints induced by *pref.rec* under the condition that, for each cell, *pref* and *pref.rec* construct equal function terms. For this, it utilizes an SMT solver to check the satisfiability of a quantifier-free first-order logic formula, encoding the consistency requirement, in the theory of equality. If the check succeeds, adding *rec* to *pref* des not create an inconsistency and hence repeating *rec* infinitely often is consistent. Therefore, there exists an execution for *pref.rec^ω* and thus φ is finitary satisfiable in T_U by Lemma 1.

UnsatSearch computes the execution effect of finite subwords of runs of $\mathcal{B}$ and checks whether they are consistent. If a subword is inconsistent, then every run that contains this subword is inconsistent. Hence, there do not exist executions for these runs and therefore they are removed from the set of candidate runs. If there is no accepting candidate run left, then $\mathcal{B}$ has an empty symbolic language and thus, by Theorem 2, φ is unsatisfiable in T_U.

Example 4. Consider the finitary BSAs $\mathcal{B}_1$ and $\mathcal{B}_2$ from Figures 1a and 1b as well as their respective TSL formulas $\varphi_1 := \Box[\![\mathtt{x} \hookleftarrow f(\mathtt{x})]\!] \land \Box \Diamond (p(\mathtt{x}) \land \bigcirc \neg p(\mathtt{x}))$ and $\varphi_2 := (\Box([\![\mathtt{x} \hookleftarrow f(\mathtt{x})]\!] \land p(\mathtt{x})) \land \Diamond \neg p(f(\mathtt{x}))$. If we execute Algorithm 1 on φ_1, *SatSearch* considers the accepting lasso $q_0 \rightarrow q_1 \rightarrow q_0$ in $\mathcal{B}_1$ at some point. Then, *pref* $= \varepsilon$ and *rec* $= (q_0, g_1, u, q_1)(q_1, g_2, u, q_0)$. Note that *pref.rec* is the finite prefix c_p of a run of $\mathcal{B}_1$ from Example 3. Thus, effect(*pref.rec*) is given by $(\lambda \mathtt{c}.f(f(\mathtt{x})), \{(p(\mathtt{x}), true), (p(f(\mathtt{x})), false)\})$. Since effect(*pref*) $= (\lambda \mathtt{c}.c, \emptyset)$ holds, *SatSearch* generates the query $p(\mathtt{x}) \land \neg p(f(\mathtt{x})) \land \mathtt{x} = f(f(\mathtt{x}))$ which is satisfiable in T_E. Hence, we can repeat the lasso $q_0 \rightarrow q_1 \rightarrow q_0$ infinitely often without getting any inconsistent constraints and thus φ_1 is satisfiable.

If we execute Algorithm 1 on φ_2, *UnsatSearch* checks at some point wether in $\mathcal{B}_2$ the transition sequence $q_0 \rightarrow q_1$ followed by the upper self-loop is consistent. This is not the case as it requires $p(f(\mathtt{x}))$ to be true (first transition) and false (second transition): We have $\varrho_1 = \{(p(\mathtt{x}), true), (p(f(\mathtt{x})), false)\}$ and $\varrho_2 = \varrho_1 \cup \{(p(f(\mathtt{x})), true), (p(f(f(\mathtt{x}))), true)\}$ by definition of the constraint trace. *UnsatSearch* also checks the transition sequence $q_0 \rightarrow q_1$ followed by the lower self-loop which is also inconsistent. Hence, there is no consistent transition after $q_0 \rightarrow q_1$ and thus there is no valid accepting run. Hence, φ_2 is unsatisfiable.

Note that the presentation of Algorithm 1 omits implementation details such as the enumeration of accepting loops and the implementation of the infinite set $\mathcal{R}$. A more detailed description addressing these issues is given in [14].

Algorithm 1 is correct. Intuitively, it terminates with SAT if the constraint trace ϱ of the unique computation ς of *pref.rec^ω* is consistent. Hence, ϱ defines an assignment $\langle \cdot \rangle$ such that $(\varsigma, \mathcal{V}, \langle \cdot \rangle)$ is an execution of *pref.rec^ω*, implying satisfiability of φ in T_U. If the algorithm terminates with UNSAT, then all accepting runs of the BSA are inconsistent and thus no finitary execution satisfying φ exists. For the proof, we refer the reader to the full version [14].

Theorem 3 (Correctness of Algorithm 1). *Let φ be a TSL formula. If Algorithm 1 terminates on φ with* SAT, *then there exists an execution $(\varsigma, \mathcal{V}, \langle \cdot \rangle)$ such that both $\varsigma \models_{\mathcal{V}, \langle \cdot \rangle} \varphi$ and $\mathrm{fin}_\varphi(\varsigma)$ hold. If Algorithm 1 terminates with* UN-SAT, *then for all executions $(\varsigma, \mathcal{V}, \langle \cdot \rangle)$ with $\mathrm{fin}_\varphi(ctr)$, $\varsigma \not\models_{\mathcal{V}, \langle \cdot \rangle} \varphi$ holds.*

5 Undecidability of TSL modulo T_U Satisfiability

The algorithm for TSL satisfiability checking in T_U presented in the previous section does not necessarily terminate. In this section, we show that no complete algorithm exists: The satisfiability of a TSL formula in the theory of uninterpreted functions T_U (TSL-T_U-SAT) is neither semi-decidable nor co-semi-decidable:

Theorem 4 (Undecidability of TSL-T_U-SAT). *The satisfiability (validity) problem of TSL in T_U is neither semi-decidable nor co-semi-decidable.*

The main intuition behind the undecidability result is that we can encode numbers with TSL in the theory of uninterpreted functions. That is, we are able to encode incrementation, resetting some variable to zero, and equality. We only give the encoding here, for the proof of its correctness we refer to [14].

Lemma 3. *Let f be a unary function, let $\widehat{=}$ be a binary predicate, and let z be a constant. Let $f^x(z)$ correspond to applying f x-times to z. There exists a TSL formula φ_{num} such that every execution entailing φ_{num} requires from its models that for all $a, b \in \mathbb{N}_0$, $a = b$ holds if, and only if, $f^a(z) \widehat{=} f^b(z)$ holds.*

Proof (Sketch). We construct $\varphi_{num} = \varphi_1 \wedge \varphi_2$ as follows: The first conjunct is defined by $\varphi_1 := [\![e \leftarrow z]\!] \wedge \bigcirc\square([\![e \leftarrow f(e)]\!] \wedge e \widehat{=} e)$. Let

$$\varphi_{eq} := (x \widehat{=} b) \rightarrow ([\![x \leftarrow z]\!] \wedge [\![b \leftarrow f(b)]\!] \wedge \neg (b \widehat{=} f(b)) \wedge \neg (f(b) \widehat{=} b))$$
$$\varphi_{neq} := \neg (x \widehat{=} b) \rightarrow ([\![x \leftarrow f(x)]\!] \wedge [\![b \leftarrow b]\!] \wedge \neg (x \widehat{=} f(b)) \wedge \neg (f(b) \widehat{=} x)).$$

Then, φ_2 is defined by $\varphi_2 := [\![x \leftarrow z]\!] \wedge [\![b \leftarrow z]\!] \wedge \bigcirc\square(\varphi_{eq} \wedge \varphi_{neq})$.

Intuitively, f corresponds to incrementation, z to resetting a variable to zero, and $\widehat{=}$ to equality: φ_1 ensures that $f^n(z) \widehat{=} f^n(z)$ holds for all $n \in \mathbb{N}_0$. In contrast, φ_2 ensures that if $a \neq b$ holds, then $\neg(f^a(z) \widehat{=} f^b(z))$: Starting with $x = b = z$, φ_1 ensures that $x \widehat{=} b$ holds initially. Then, φ_{eq} resets x to z and "increments" b, while ensuring that $\neg(f^k(z) \widehat{=} f^{k+1}(z))$ holds, where $b = f^k(z)$. Then, $\neg(x \widehat{=} b)$ holds and thus φ_{neq} "increments" x until it reaches $b = f^{k+1}(z)$, while ensuring that $\neg(f^{k+1}(z) \widehat{=} f^\ell(z))$ holds for all $\ell < k + 1$.

Using this encoding in TSL modulo T_U, we can construct a TSL formula $\varphi_{\mathcal{G}}$ for every GOTO-program $\mathcal{G}$ such that $\varphi_{\mathcal{G}} \wedge \varphi_{num}$ is satisfiable in T_U if, and only if, $\mathcal{G}$ terminates on every input. Intuitively, $\varphi_{\mathcal{G}}$ "simulates" $\mathcal{G}$ on different inputs by starting with input zero and incrementing the input if the halting location was reached. The temporal operators of TSL allow for requiring that $\mathcal{G}$ terminates infinitely often. The construction of $\varphi_{\mathcal{G}}$ is given in the full version [14]. Since the universal halting problem for GOTO programs is neither semi-decidable nor

co-semi-decidable, the same undecidability result follows for the satisfiability of a TSL formula modulo T_U, proving Theorem 4.

Since the theory of Presburger arithmetic $T_\mathbb{N}$ allows for incrementation, resetting a variable to zero, and equality, we can reuse the TSL formula $\varphi_\mathcal{G}$ from above to reduce the universal halting problem for GOTO programs to TSL satisfiability modulo $T_\mathbb{N}$ (TSL-$T_\mathbb{N}$-SAT), proving undecidability of TSL-$T_\mathbb{N}$-SAT. Note that this result holds for other theories that can express incrementation, reset, and equality, for instance Peano Arithmetic, as well.

Theorem 5 (Undecidability of TSL-$T_\mathbb{N}$-SAT).   *The satisfiability (validity) problem of TSL in $T_\mathbb{N}$ is neither semi-decidable nor co-semi-decidable.*

Furthermore, equality allows for encoding incrementation and resetting a variable to zero. Hence, similarly to T_U, there exists a TSL formula φ_{enc} that, if entailed, enforces a binary function and a constant to behave as incrementation and a reset, respectively. The construction of φ_{enc} is given in the full version [14]. Thus, the TSL formula $\varphi_\mathcal{G}$ constructed as above for a GOTO program $\mathcal{G}$ ensures that $\varphi_\mathcal{G} \wedge \varphi_{enc}$ is satisfiable in the theory of equality T_E if, and only if, $\mathcal{G}$ terminates on every input. Hence, undecidability of TSL-T_E-SAT follows:

Theorem 6 (Undecidability of TSL-T_E-SAT).   *The satisfiability (validity) problem of TSL in T_E is neither semi-decidable nor co-semi-decidable.*

6   (Semi-)Decidable Fragments

In Section 5, we showed that TSL satisfiability is undecidable in T_U. In this section, however, we identify fragments of TSL on which Algorithm 1 terminates for certain inputs. In fact, we present one fragment for which TSL-T_U-SAT is decidable and two fragments for which TSL-T_U-SAT is semi-decidable.

First, we consider the TSL reachability fragment, i.e., the fragment of TSL that only permits the next operator and the eventually operator as temporal operators. In our applications, this fragment corresponds to finding counterexamples to safety properties. For satisfiable reachability formulas, Algorithm 1 terminates. The main idea behind the termination is that the BSA of a reachability formula has an accepting lasso-shaped run and since φ is satisfiable, this run is consistent. For the proof, we refer to the full version [14].

Lemma 4.   *Let φ be a TSL formula in the reachability fragment. If φ is finitary satisfiable in T_U, then Algorithm 1 terminates on φ.*

Restricting the reachability fragment further, we consider TSL formulas with updates, predicates, logical operators, next operators, and at most one top-level eventually operator. Such formulas are either completely time-bounded or they are of the form $\varphi = \Diamond \varphi'$, where φ' is time-bounded. In the dual validity problem, such formulas correspond to invariants on a fixed time window, a useful property for many applications. Algorithm 1 is guaranteed to terminate for satisfiable and unsatisfiable formulas of the above form if a *suitable* BSA is constructed. Such a

suitable BSA has a single accepting state q indicating that the time-bounded part has been satisfied. Intuitively, a suitable BSA ensures that all runs reaching q are accepting and that only finitely many transition sequences lead to q. Then, if φ is unsatisfiable, Algorithm 1 is able to exclude all transition sequences leading to q and thus to terminate. A BSA with infinitely many transition sequences leading to q, in contrast, may cause the algorithm to diverge as it may consider infinitely many consistent subsequences before finding the inconsistent one yielding the exclusion of the sequences leading to q. A suitable BSA exists for every TSL formula in the considered fragment. For the proof, including a more detailed description of suitable BSAs, we refer to the full version [14].

Lemma 5. *Let φ be a TSL formula with only logical operators, predicates, updates, next operators, and at most one top-level eventually operator. Algorithm 1 terminates on φ if it picks a suitable respective BSA.*

Note that Algorithm 1 is only a formal decider for this fragment if we ensure that a suitable BSA is always generated. In practice, we experienced that this is usually the case even without posing restrictions on the BSA construction. Lastly, we consider a fragment of TSL that does not restrict the temporal structure of the formula but the number of used cells. For TSL formulas with a single cell, Algorithm 1 always terminates on satisfiable inputs:

Lemma 6. *Let φ be a TSL formula such that $|\mathbb{C}| = 1$. If φ is finitary satisfiable in the theory of uninterpreted functions, then Algorithm 1 terminates on φ.*

Intuitively, restricting the TSL formula to use only a single cell prevents us from simulating arbitrary computations and thus from reducing from the universal halting problem of GOTO programs as in the general undecidability proof. The formal proof, given in the full version [14], however, is unrelated to the above intuition. Combining the three observations, we obtain the following (semi-)decidability results for the satisfiability of fragments of TSL modulo T_U:

Theorem 7. *The satisfiability problem of TSL formulas in T_U is (1) semi-decidable for the reachability fragment of TSL, (2) decidable for formulas consisting of only logical operators, predicates, updates, next operators, and at most one top-level eventually operator, and (3) semi-decidable for formulas with one cell.*

7   Evaluation

We implemented the algorithm for checking TSL modulo T_U satisfiability[2]. We used *TSL tools*[3] to handle TSL, *spot* [11] to transform the approximated LTL formulas into NBAs, *SyFCo* [20] for LTL transformations, and *z3* [31] to solve SMT queries. The implementation follows the extended algorithm described in [14]. Since in some cases the default optimizations of *spot* produce a large overhead in
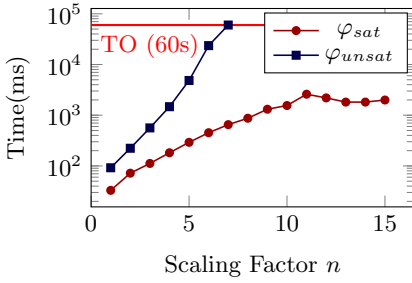

Fig. 2: Execution times in milliseconds of the scalability benchmark series.
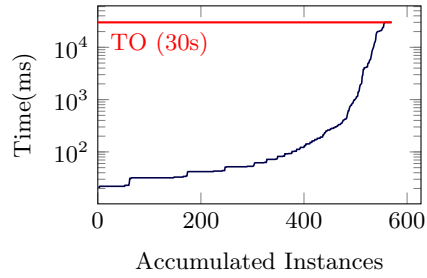

Fig. 3: Execution times in milliseconds of the random benchmark series.

computation time, we first execute it with these and if this does not terminate within 20s, we execute it without optimizations. We evaluated the implementation on three benchmark classes and a machine with an AMD Ryzen 7 processor, using a virtual machine with two logical cores and 6 GB of RAM.

Scalability Benchmark Series. We test the scalability of the algorithm with parameterized decidable benchmarks. The timeout is one minute. Note that *spot* can always perform its optimizations. The satisfiable benchmarks are defined by $\varphi_{sat}(n) := (\Box[\![\mathtt{x} \leftarrow f(\mathtt{x})]\!]) \wedge (\Diamond \neg p(\mathtt{x})) \wedge (\bigwedge_{i=0}^{n} p(f^i(\mathtt{x})))$. The parameter n corresponds to the number of updates that have to be performed to find a satisfiable lasso. By Lemma 6, the algorithm always terminates. The TSL formula $\varphi_{unsat}(n) := (\Box(q(\mathtt{x}) \leftrightarrow \neg q(f^n(\mathtt{x})))) \wedge (\Box[\![\mathtt{x} \leftarrow f(\mathtt{x})]\!]) \wedge \Diamond(q(\mathtt{x}) \wedge \bigcirc^n q(\mathtt{x}))$ defines the unsatisfiable benchmarks. The parameter n corresponds to the "distance" in time and number of updates of the conflict causing unsatisfiability. The algorithm always terminates. The results are shown in Figure 2. The algorithm scales particularly well for the satisfiable formulas. However, the experiments indicate an exponential complexity of the algorithm for the unsatisfiable formulas.

Random Benchmark Series. We implemented a random TSL formula generator that uses *spot*'s `ltlrand` to generate random LTL formulas and then substitutes the atomic propositions with random updates and predicates. The generated TSL formulas have one to three cells, one to three different updates and one to three different predicates. For the LTL formulas generated by `ltlrand` we use tree sizes from 5 to 95 in steps of five. For each of the tree sizes, we generate 30 formulas; 570 in total. The execution times are shown in Figure 3. On many formulas, the algorithm terminates within one second. The implementation returns SAT for 513 of the 570 formulas. It times out after 30s on 29 formulas. However, the timeouts already occur in the automaton construction, both with and without *spot*'s optimizations. Only 28 formulas are unsatisfiable. For 25 of these unsatisfiable formulas, the intermediate LTL approximation formula is already unsatisfiable, i.e., only for three formulas there is some conflict due to updates and predicate evaluation.

Table 1: Execution times in seconds of the application benchmark series.

Benchmark	Result	Time	Benchmark	Result	Time
Chain	SAT	7.06	Inductive Ass.	UNSAT	0.25
Filter	UNSAT	0.33	One Of Two	UNSAT	1.20
Gamemodechooser Ass.	UNSAT	35.55	One Of Three	UNSAT	4.25
Holding Arbiter	SAT	11.75	Injector	UNSAT	1.52
Small Holding Arbiter	SAT	36.69	Invariant Holding	UNSAT	2.33
P. T. Arbiter	UNSAT	56.03	Scheduler	UNSAT	3.87
Approx. P. T. Arbiter	UNSAT	940.03			

Applications Benchmark Series. These benchmarks correspond to checking consistency of a specification and validating assumptions of a system. Hence, they illustrate how satisfiability results can aid the system design. The results are presented in Table 1. We introduce two of the benchmarks in more detail here. The other, slightly larger, ones, including different kinds of arbiters, a scheduler, and modules of the Syntroids [17] arcade game, are described in [14].

The *Chain* benchmark considers a compound system of two chained modules m_1 and m_2 that receive an input value, store it, and forward it to the next system: $\varphi_i := \Box \Diamond(\llbracket \mathtt{mem}_i \hookleftarrow \mathtt{in}_i \rrbracket \wedge \bigcirc \llbracket \mathtt{in}_{i+1} \hookleftarrow \mathtt{mem}_i \rrbracket)$ for $i \in \{1, 2\}$. To simulate the input of the first module, we use an update with an uninterpreted function: $\varphi_{inp} := \Box \llbracket \mathtt{in}_1 \hookleftarrow f(\mathtt{in}_1) \rrbracket$. We require that if some property p holds on m_1's input, p also needs to hold hold on m_2's output: $\varphi_{spec} := \Box(p(\mathtt{in}_1) \to \Diamond p(\mathtt{in}_3))$. Our algorithm determines within 8s that $(\varphi_{inp} \wedge \varphi_1 \wedge \varphi_2) \wedge \neg\varphi_{spec}$ is satisfiable, detecting an inconsistency: If m_1 stores some value on which p holds, it may overwrite it before m_2 copies it, preventing the value to reach m_2's output.

The *Filter* benchmark studies a system that "passes through" an input value to a cell if it fulfills a certain property p and holds the previous value otherwise: $\varphi_{filter} := \llbracket \mathtt{out} \hookleftarrow d() \rrbracket \wedge \bigcirc \Box((p(\mathtt{in}) \to \llbracket \mathtt{out} \hookleftarrow \mathtt{in} \rrbracket) \wedge (\neg p(\mathtt{in}) \to \llbracket \mathtt{out} \hookleftarrow \mathtt{out} \rrbracket))$, where d is a constant representing an initial default value. The default value fulfills p, i.e., $\varphi_{fact} := \Box p(d())$. As for the chain, $\varphi_{inp} := \Box \llbracket \mathtt{in} \hookleftarrow f(\mathtt{in}) \rrbracket$ simulates the input. The filter is valid if p holds on all outputs after the initialization: $\varphi_{spec} := \bigcirc \Box p(\mathtt{out})$. Within 400ms, the algorithm confirms that $(\varphi_{inp} \wedge \varphi_{fact} \wedge \varphi_{filter}) \wedge \neg\varphi_{spec}$ is unsatisfiable, validating the filter.

8   Related Work

Linear-time temporal logic (LTL) [32] is one of the most popular specification languages for reactive systems. It is based on an underlying assertion logic, such as propositional logic, which is extended with temporal modalities. Satisfiability of propositional LTL has long known to be decidable [37] and there are efficient tools for LTL satisfiability checking [36,25].

While propositional LTL is very common, especially in hardware verification, LTL with richer assertion logics, such as first-order logic and various theories, have long been used in verification (cf. [28]). Temporal Stream Logic (TSL) [15]

was introduced as a new temporal logic for reactive synthesis. In the original TSL semantics, all functions and predicates are uninterpreted. TSL synthesis is undecidable in general, even without inputs or equality, but can be under-approximated by the decidable LTL synthesis problem [15]. TSL has been used to specify and synthesize an arcade game realized on an FPGA [17].

Constraint LTL (CLTL) [6] extends LTL with the possibility of expressing constraints between variables at bounded distance. A constraint system $\mathcal{D}$ consists of a concrete domain and an interpretation of relations on the domain. In Constraint LTL over $\mathcal{D}$ (CLTL($\mathcal{D}$)), one can relate variables with relations defined in $\mathcal{D}$. Similar to updates in TSL, CLTL can specify assignment-like statements by utilizing the equality relation. Like for all constraints allowing for a counting mechanism, LTL with Presburger constraints, i.e., CLTL($\mathbb{Z}, =, +$), is undecidable [6]. However, there exist decidable fragments such as LTL with finite constraint systems [4] and LTL with integer periodicity constraints [5]. Permitting constraints between variables at an unbounded distance leads to undecidability even for constraint systems that only allow equality checks on natural numbers. Restricting such systems to a finite number of constraints yields decidability again [9]. In TSL modulo theories, a theory is given from which a model can be chosen. In CLTL, in contrast, the concrete model is fixed. Therefore, TSL modulo theories cannot be encoded into CLTL in general.

LTL has been extended with the *freeze operator* [8,7], allowing for storing an input in a register. Then, the stored value can be compared with a current value for equality. Freeze LTL with two registers is undecidable [26,10] . For flat formulas, i.e., formulas where the possible occurrences of the freeze operator are restricted, decidability is regained [10]. Similar to the freeze operator, updates in TSL allow for storing values in cells and in TSL modulo the theory of equality the equality check can be performed. In TSL, we can perform computations on the stored values which is not possible in freeze LTL. Hence, freeze LTL can be seen as a special case of TSL. Constraint LTL has been augmented with the freeze operator as well [10]. For an infinite domain equipped with the equality relation, it is undecidable. For flat formulas, decidability is regained [10].

The temporal logic of actions (TLA) [24] is designed to model computer systems. States are assignments of values to variables and actions relate states. Actions can, similar to updates in TSL, describe assignments of variables. A TLA formula may contain state functions and predicates. Actions and state functions are combined with the temporal operators $\square$ and $\lozenge$. In contrast to TSL, $\bigcirc$ and $\mathcal{U}$ are not permitted. The validity problem for the propositional fragment of TLA, i.e., with uninterpreted functions and predicates, is PSPACE complete [35].

Similar to temporal logics, dynamic logic [33,19] is an extension of modal logic to reason about computer programs. Dynamic logic allows for stating that after action a, it is necessarily the case that p holds, or it is possible that p holds. Compound actions can be build up from smaller actions. In propositional dynamic logic (PDL) [16], data is omitted, i.e., its terms are actions and propositions. PDL satisfiability is decidable in EXPTIME [34]. First-order dynamic logic (FODL) [18] allows for including data: First-order quantification over a

first-order structure, the so-called domain of computation, is allowed. Dynamic logic does not contain temporal operators such as $\square$ or $\diamondsuit$. Since we consider reactive systems, i.e., systems that continually interact with their environment, temporal logics are better suited than dynamic logics for our setting.

Symbolic automata (see e.g. [2,3]) and *register automata* [21] are extensions of finite automata that are capable of handling large or infinite alphabets. Register automata have additionally been considered over infinite words in some works (see e.g. [8,22,12]). Similar to BSAs, transitions of symbolic automata are labeled with predicates over a domain of alphabet symbols. Register automata are equipped with a finite amount of registers that, similar to cells in BSAs, can store input values. *Symbolic register automata* (SRAs) [1] combine the features of both automata models. BSAs have the additional ability to modify the stored values and thus to perform actual computations on them. Moreover, they read infinite instead of finite words. Thus, SRAs can be seen as a special case of BSAs.

More recently, the verification of uninterpreted programs has been investigated [29]. Uninterpreted programs are similar to WHILE-programs with equality and uninterpreted functions and predicates. They are annotated with assumptions. The verification of uninterpreted programs is undecidable in general; for the subclass of coherent uninterpreted programs, however, it is decidable [29]. The verification problem has been extended with theories, i.e., with axioms over the functions and predicates [30]. Adding axioms to coherent uninterpreted programs preserves decidability for some axioms, e.g., idempotence, while it yields undecidability for others, e.g., associativity. The synthesis problem for uninterpreted programs is undecidable in general, but decidable for coherent ones [23].

9   Conclusion

We have extended Temporal Stream Logic (TSL) with first-order theories and formalized the satisfiability and validity of a TSL formula in a theory. While we show that TSL satisfiability is neither semi-decidable nor co-semi-decidable in the theory of uninterpreted functions T_U, the theory of equality T_E, and Presburger arithmetic $T_\mathbb{N}$, we identify three fragments for which satisfiability in T_U is (semi-)decidable: For reachability formulas as well as for formulas with a single cell, TSL satisfiability in T_U is semi-decidable. For slightly more restricted reachability formulas, it is decidable. Moreover, we have presented an algorithm for checking the satisfiability of a TSL formula in the theory of uninterpreted functions that is based on Büchi stream automata, an automaton representation of TSL formulas introduced in this paper. Satisfiability checking has various applications in the specification and analysis of reactive systems such as identifying inconsistent requirements during the design process. We have implemented the algorithm and evaluated it on three different benchmark series, including consistency checks and assumption validations: The algorithm terminates on many randomly generated formulas within one second and scales particularly well for satisfiable formulas. Moreover, it is able to prove or disprove consistency of realistic benchmarks and to validate or invalidate their assumptions.

References

1. D'Antoni, L., Ferreira, T., Sammartino, M., Silva, A.: Symbolic Register Automata. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11561, pp. 3–21. Springer (2019), https://doi.org/10.1007/978-3-030-25540-4_1

2. D'Antoni, L., Veanes, M.: The Power of Symbolic Automata and Transducers. In: Majumdar, R., Kuncak, V. (eds.) Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10426, pp. 47–67. Springer (2017), https://doi.org/10.1007/978-3-319-63387-9_3

3. D'Antoni, L., Veanes, M.: Automata modulo Theories. Commun. ACM **64**(5), 86–95 (2021), https://doi.org/10.1145/3419404

4. Demri, S.: Linear-time Temporal Logics with Presburger Constraints: An Overview. J. Appl. Non Class. Logics **16**(3-4), 311–348 (2006), https://doi.org/10.3166/jancl.16.311-347

5. Demri, S.: LTL Over Integer Periodicity Constraints. Theor. Comput. Sci. **360**(1-3), 96–123 (2006), https://doi.org/10.1016/j.tcs.2006.02.019

6. Demri, S., D'Souza, D.: An Automata-Theoretic Approach to Constraint LTL. Inf. Comput. **205**(3), 380–415 (2007), https://doi.org/10.1016/j.ic.2006.09.006

7. Demri, S., D'Souza, D., Gascon, R.: A Decidable Temporal Logic of Repeating Values. In: Artëmov, S.N., Nerode, A. (eds.) Logical Foundations of Computer Science, International Symposium, LFCS 2007, New York, NY, USA, June 4-7, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4514, pp. 180–194. Springer (2007), https://doi.org/10.1007/978-3-540-72734-7_13

8. Demri, S., Lazic, R.: LTL with the Freeze Quantifier and Register Automata. In: 21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings. pp. 17–26. IEEE Computer Society (2006), https://doi.org/10.1109/LICS.2006.31

9. Demri, S., Lazic, R., Nowak, D.: On the Freeze Quantifier in Constraint LTL: Decidability and Complexity. In: 12th International Symposium on Temporal Representation and Reasoning (TIME 2005), 23-25 June 2005, Burlington, Vermont, USA. pp. 113–121. IEEE Computer Society (2005), https://doi.org/10.1109/TIME.2005.28

10. Demri, S., Lazic, R., Nowak, D.: On the Freeze Quantifier in Constraint LTL: Decidability and Complexity. In: 12th International Symposium on Temporal Representation and Reasoning (TIME 2005), 23-25 June 2005, Burlington, Vermont, USA. pp. 113–121. IEEE Computer Society (2005), https://doi.org/10.1109/TIME.2005.28

11. Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, E., Xu, L.: Spot 2.0 - A Framework for LTL and ω-Automata Manipulation. In: Artho, C., Legay, A., Peled, D. (eds.) Automated Technology for Verification and Analysis - 14th International Symposium, ATVA 2016, Chiba, Japan, October 17-20, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9938, pp. 122–129 (2016), https://doi.org/10.1007/978-3-319-46520-3_8

12. Exibard, L., Filiot, E., Reynier, P.: Synthesis of Data Word Transducers. Log. Methods Comput. Sci. **17**(1) (2021), https://lmcs.episciences.org/7279

13. Finkbeiner, B., Heim, P., Passing, N.: Temporal Stream Logic modulo Theories. CoRR **abs/2104.14988v1** (2021), https://arxiv.org/abs/2104.14988v1

14. Finkbeiner, B., Heim, P., Passing, N.: Temporal Stream Logic modulo Theories (Full Version). CoRR **abs/2104.14988v2** (2021), https://arxiv.org/abs/2104.14988v2

15. Finkbeiner, B., Klein, F., Piskac, R., Santolucito, M.: Temporal Stream Logic: Synthesis Beyond the Bools. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11561, pp. 609–629. Springer (2019), https://doi.org/10.1007/978-3-030-25540-4_35

16. Fischer, M.J., Ladner, R.E.: Propositional Dynamic Logic of Regular Programs. J. Comput. Syst. Sci. **18**(2), 194–211 (1979), https://doi.org/10.1016/0022-0000(79)90046-1

17. Geier, G., Heim, P., Klein, F., Finkbeiner, B.: Syntroids: Synthesizing a Game for FPGAs using Temporal Logic Specifications. In: 2019 Formal Methods in Computer Aided Design, FMCAD 2019, San Jose, CA, USA, October 22-25, 2019. pp. 138–146. IEEE (2019), https://doi.org/10.23919/FMCAD.2019.8894261

18. Harel, D.: First-Order Dynamic Logic, Lecture Notes in Computer Science, vol. 68. Springer (1979), https://doi.org/10.1007/3-540-09237-4

19. Harel, D., Meyer, A.R., Pratt, V.R.: Computability and Completeness in Logics of Programs (Preliminary Report). In: Hopcroft, J.E., Friedman, E.P., Harrison, M.A. (eds.) Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA. pp. 261–268. ACM (1977), https://doi.org/10.1145/800105.803416

20. Jacobs, S., Klein, F., Schirmer, S.: A High-level LTL Synthesis Format: TLSF v1.1. In: Piskac, R., Dimitrova, R. (eds.) Proceedings Fifth Workshop on Synthesis, SYNT@CAV 2016, Toronto, Canada, July 17-18, 2016. EPTCS, vol. 229, pp. 112–132 (2016), https://doi.org/10.4204/EPTCS.229.10

21. Kaminski, M., Francez, N.: Finite-Memory Automata. Theor. Comput. Sci. **134**(2), 329–363 (1994), https://doi.org/10.1016/0304-3975(94)90242-9

22. Khalimov, A., Maderbacher, B., Bloem, R.: Bounded Synthesis of Register Transducers. In: Lahiri, S.K., Wang, C. (eds.) Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings. Lecture Notes in Computer Science, vol. 11138, pp. 494–510. Springer (2018), https://doi.org/10.1007/978-3-030-01090-4_29

23. Krogmeier, P., Mathur, U., Murali, A., Madhusudan, P., Viswanathan, M.: Decidable Synthesis of Programs with Uninterpreted Functions. In: Lahiri, S.K., Wang, C. (eds.) Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12225, pp. 634–657. Springer (2020), https://doi.org/10.1007/978-3-030-53291-8_32

24. Lamport, L.: The Temporal Logic of Actions. ACM Trans. Program. Lang. Syst. **16**(3), 872–923 (1994), https://doi.org/10.1145/177492.177726

25. Li, J., Zhang, L., Pu, G., Vardi, M.Y., He, J.: LTL Satisfiability Checking Revisited. In: Sánchez, C., Venable, K.B., Zimányi, E. (eds.) 2013 20th International Symposium on Temporal Representation and Reasoning, Pensacola, FL, USA, September 26-28, 2013. pp. 91–98. IEEE Computer Society (2013), https://doi.org/10.1109/TIME.2013.19

26. Lisitsa, A., Potapov, I.: Temporal Logic with Predicate λ-Abstraction. In: 12th International Symposium on Temporal Representation and Reasoning (TIME 2005), 23-25 June 2005, Burlington, Vermont, USA. pp. 147–155. IEEE Computer Society (2005), https://doi.org/10.1109/TIME.2005.34

27. Maderbacher, B., Bloem, R.: Reactive Synthesis Modulo Theories Using Abstraction Refinement. CoRR **abs/2108.00090** (2021), https://arxiv.org/abs/2108.00090

28. Manna, Z., Pnueli, A.: Verification of Concurrent Programs: The Temporal Framework. In: Boyer, R.S., Moore, J.S. (eds.) The Correctness Problem in Computer Science. Academic Press, London (1981)

29. Mathur, U., Madhusudan, P., Viswanathan, M.: Decidable Verification of Uninterpreted Programs. Proc. ACM Program. Lang. **3**(POPL), 46:1–46:29 (2019), https://doi.org/10.1145/3290359

30. Mathur, U., Madhusudan, P., Viswanathan, M.: What's Decidable About Program Verification Modulo Axioms? In: Biere, A., Parker, D. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12079, pp. 158–177. Springer (2020), https://doi.org/10.1007/978-3-030-45237-7_10

31. de Moura, L.M., Bjørner, N.: Z3: An Efficient SMT Solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings. Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer (2008), https://doi.org/10.1007/978-3-540-78800-3_24

32. Pnueli, A.: The Temporal Logic of Programs. In: Annual Symposium on Foundations of Computer Science, 1977. pp. 46–57. IEEE Computer Society (1977)

33. Pratt, V.R.: Semantical Considerations on Floyd-Hoare Logic. In: 17th Annual Symposium on Foundations of Computer Science, Houston, Texas, USA, 25-27 October 1976. pp. 109–121. IEEE Computer Society (1976), https://doi.org/10.1109/SFCS.1976.27

34. Pratt, V.R.: A Practical Decision Method for Propositional Dynamic Logic: Preliminary Report. In: Lipton, R.J., Burkhard, W.A., Savitch, W.J., Friedman, E.P., Aho, A.V. (eds.) Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA. pp. 326–337. ACM (1978), https://doi.org/10.1145/800133.804362

35. Ramakrishna, Y.S.: On the Satisfiability Problem for Lamport's Propositional Temporal Logic of Actions and Some of Its Extensions. Fundam. Informaticae **24**(4), 387–405 (1995), https://doi.org/10.3233/FI-1995-2444

36. Rozier, K.Y., Vardi, M.Y.: LTL Satisfiability Checking. In: Bosnacki, D., Edelkamp, S. (eds.) Model Checking Software, 14th International SPIN Workshop, Berlin, Germany, July 1-3, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4595, pp. 149–167. Springer (2007), https://doi.org/10.1007/978-3-540-73370-6_11

37. Sistla, A.P., Clarke, E.M.: The Complexity of Propositional Linear Temporal Logics. J. ACM **32**(3), 733–749 (1985), https://doi.org/10.1145/3828.3837

38. Vardi, M.Y., Wolper, P.: Reasoning About Infinite Computations. Inf. Comput. **115**(1), 1–37 (1994), https://doi.org/10.1006/inco.1994.1092

The Different Shades of Infinite Session Types [*]

Simon J. Gay[1] , Diogo Poças[2](✉) , and Vasco T. Vasconcelos[2]

[1] School of Computing Science, University of Glasgow, UK
simon.gay@glasgow.ac.uk
[2] LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal
{dmpocas,vmvasconcelos}@ciencias.ulisboa.pt

Abstract. Many type systems include infinite types. In session type systems, infinite types are important because they specify communication protocols that are unbounded in time. Usually infinite session types are introduced as simple finite-state expressions rec $X.T$ or by non-parametric equational definitions $X \doteq T$. Alternatively, some systems of label- or value-dependent session types go beyond simple recursive types. However, leaving dependent types aside, there is a much richer world of infinite session types, ranging through various forms of parametric equational definitions, to arbitrary infinite types in a coinductively defined space. We study infinite session types across a spectrum of shades of grey on the way to the bright light of general infinite types. We identify four points on the spectrum, characterised by different styles of equational definitions, and show that they form a strict hierarchy by establishing bidirectional correspondences with classes of automata: finite-state, 1-counter, pushdown and 2-counter. This allows us to establish decidability and undecidability results for type formation, type equivalence and duality in each class of types. We also consider previous work on context-free session types (and extend it to higher-order) and nested session types, and locate them on our spectrum of infinite types.

Keywords: Infinite types · Recursive types · Session types · Automata and formal language theory

1 Introduction

Session types [19,20,23,38] are an established approach to specifying communication protocols, so that protocol implementations can be verified by static typechecking or dynamic monitoring. The simplest protocols are finite: for example, ?int.!bool.end describes a protocol in which an integer is received, then a boolean is sent, and that's all. Most systems of session types, however, include

equi-recursive types for greater expressivity. A type that endlessly repeats the simple send-receive protocol is X such that $X \doteq\ ?\mathsf{int}.!\mathsf{bool}.X$, which can also be specified by $\mathsf{rec}\,X.?\mathsf{int}.!\mathsf{bool}.X$. More realistic examples usually combine recursion and choice, as in Y such that $Y \doteq \&\{\mathsf{go}\colon ?\mathsf{int}.!\mathsf{bool}.Y, \mathsf{quit}\colon \mathsf{end}\}$ which offers a choice between go and quit operations, each with its own protocol. A natural observation is that session types look like finite-state automata, but some systems from the literature go beyond the finite-state format: for example, context-free session types [39] and nested session types [9,10], as well as label-dependent session types [40] and value-dependent session types [41].

Even without introducing dependent types, a range of definitional formats can be considered for session types, presumably with varying degrees of expressivity, but they have never been systematically studied. That is the aim of the present paper. We consider various forms of parameterised equational definitions, illustrated by six running examples. Because our formal system only has one base type, the terminated channel type end, the running examples simply use end (or skip for context-free session types) as a representative basic message type that could otherwise be bool or int.

Our study of classes of infinite types should be generally applicable; we make it concrete by concentrating on session types where (potential) infinite types occur naturally. For the sake of uniformity, all our non-finite session types are introduced by equations, rather than, say, rec-types. Equations may be further parameterized, thus accounting for types that go beyond recursive types. The examples below illustrate the different kinds of parameterized equations we use.

Example 1 (No parameters). Type T_{loop} is X with equation $X \doteq\ !\mathsf{end}.X$. Intuitively $T_{\mathrm{loop}} =\ !\mathsf{end}.!\mathsf{end}\ldots$ continuously outputs values of type end.

Example 2 (One natural number parameter). Assuming z and s as the natural number constructors and N as a variable over natural numbers, type T_{counter} is $X\langle\mathsf{z}\rangle$ with equations

$$X\langle\mathsf{z}\rangle \doteq \&\{\mathrm{inc}\colon X\langle\mathsf{s}\,\mathsf{z}\rangle, \mathrm{dump}\colon Y\langle\mathsf{z}\rangle\} \qquad\qquad Y\langle\mathsf{z}\rangle \doteq \mathsf{end}$$
$$X\langle\mathsf{s}\,N\rangle \doteq \&\{\mathrm{inc}\colon X\langle\mathsf{s}\,\mathsf{s}\,N\rangle, \mathrm{dump}\colon Y\langle\mathsf{s}\,N\rangle\} \qquad Y\langle\mathsf{s}\,N\rangle \doteq\ !\mathsf{end}.Y\langle N\rangle$$

A sequence of n inc operations followed by a dump triggers a reply of n end output messages.

Example 3 (Context-free types). With type skip used either to finish a session or to move to the next operation, type T_{tree} is X with equation

$$X \doteq \&\{\mathsf{leaf}\colon \mathsf{skip}, \mathsf{node}\colon X; ?\mathsf{skip}; X\}$$

The leaf choice terminates the reception of a binary tree of skip values and the node choice triggers the reception of a (left) tree, followed by ?skip (root), followed by a (right) tree. Even though the development in the rest of the paper considers higher-order types (where messages may convey arbitrary types rather than skip alone), for simplicity our example is first-order.

Example 4 (One list parameter). Assuming σ and τ as symbols and S as a variable over sequences of symbols (with ε the empty sequence), type T_{meta} is $X\langle\varepsilon\rangle$ with equations

$$X\langle\varepsilon\rangle \doteq \&\{\mathrm{addOut}\colon X\langle\sigma\rangle, \mathrm{addIn}\colon X\langle\tau\rangle\}$$
$$X\langle\sigma S\rangle \doteq \&\{\mathrm{addOut}\colon X\langle\sigma\sigma S\rangle, \mathrm{addIn}\colon X\langle\tau\sigma S\rangle, \mathrm{pop}\colon \text{!end}.X\langle S\rangle\}$$
$$X\langle\tau S\rangle \doteq \&\{\mathrm{addOut}\colon X\langle\sigma\tau S\rangle, \mathrm{addIn}\colon X\langle\tau\tau S\rangle, \mathrm{pop}\colon \text{?end}.X\langle S\rangle\}$$

Type T_{meta} records simple protocols composed of !end and ?end messages. Symbol σ in a parameter to a type identifier X denotes an output message and symbol τ an input message. The protocol behaves as a stack with two distinct push operations (addOut and addIn). The symbol (σ or τ) at top of the stack determines whether a pop operation triggers !end or ?end, respectively.

Example 5 (Nested types). Taking α as a variable over types, type T_{nest} is X_ε with equations

$$X_\varepsilon \doteq \&\{\mathrm{addOut}\colon X_{\mathrm{out}}\langle X_\varepsilon\rangle, \mathrm{addIn}\colon X_{\mathrm{in}}\langle X_\varepsilon\rangle\}$$
$$X_{\mathrm{out}}\langle\alpha\rangle \doteq \&\{\mathrm{addOut}\colon X_{\mathrm{out}}\langle X_{\mathrm{out}}\langle\alpha\rangle\rangle, \mathrm{addIn}\colon X_{\mathrm{in}}\langle X_{\mathrm{out}}\langle\alpha\rangle\rangle, \mathrm{pop}\colon \text{!end}.\alpha\}$$
$$X_{\mathrm{in}}\langle\alpha\rangle \doteq \&\{\mathrm{addOut}\colon X_{\mathrm{out}}\langle X_{\mathrm{in}}\langle\alpha\rangle\rangle, \mathrm{addIn}\colon X_{\mathrm{in}}\langle X_{\mathrm{in}}\langle\alpha\rangle\rangle, \mathrm{pop}\colon \text{?end}.\alpha\}$$

Type identifiers such as $X_\varepsilon, X_{\mathrm{out}}, X_{\mathrm{in}}$ take an arbitrary but fixed number of arguments. Type T_{nest} behaves as T_{meta} in Example 4. The alignment should be clear if we take, e.g. $X_{\mathrm{out}}\langle X_{\mathrm{in}}\langle\alpha\rangle\rangle$ for $X\langle\sigma\tau S\rangle$, with σ denoting output and τ denoting input. Type identifiers X_{out} and X_{in} play the roles of stack symbols (symbols at the top of the stack, σ or τ); type variable α denotes the lower part of the stack (S in Example 4).

Example 6 (Two natural number parameters). Type T_{iter} is $X\langle z,z\rangle$ with

$$X\langle z,N'\rangle \doteq \text{?end}.Y\langle z, \text{s }N'\rangle \qquad\qquad Y\langle N,z\rangle \doteq X\langle N,z\rangle$$
$$X\langle \text{s }N,N'\rangle \doteq \text{!end}.X\langle N, \text{s }N'\rangle \qquad\qquad Y\langle N, \text{s }N'\rangle \doteq Y\langle \text{s }N,N'\rangle$$

Informally, writing !end^n for a sequence of n output end messages, these definitions give $T_{\mathrm{iter}} = \text{?end}.\text{!end}^1.\text{?end}.\text{!end}^2.\text{?end}.\text{!end}^3\ldots$

It is intuitively clear that Examples 2 and 4 to 6 cannot be expressed without parameters. It is perhaps less clear that each definitional style in Examples 1, 2, 4 and 6 is strictly more expressive than the previous one. This is the main result of the paper. We establish a hierarchy from finite session types all the way up to non-computable types that have no representation at all. The latter certainly exist, because for every infinite binary expansion of a real number between zero and one there is a session type derived by mapping 0 to send and 1 to receive — for cardinality reasons, almost all of these types are non-computable.

Our methodology is to develop the connection between session types and automata, in particular between progressively more expressive definitional styles of types and progressively more powerful classes of automata. We also consider

the formal language class corresponding to each class of automata, and the decidability of important properties such as contractiveness, type formation, type equivalence and type duality. Our results are summarised in the table below, establishing a hierarchy of session types in parallel to the Chomsky hierarchy of languages, where by a 1-counter language, we mean a language accepted by a (deterministic) 1-counter automaton and where DCFL abbreviates deterministic context-free languages. The final row of the table emphasises that it is impossible to give an explicit example of a non-computable type or to even state the decision problems.

Context-free and 1-counter types are incomparable. Essentially, both models lie between levels 2 and 3 of the Chomsky hierarchy and correspond to different restrictions of deterministic pushdown automata. Context-free types correspond to constraining automata with a single state, whereas 1-counter types correspond to constraining the stack to have a single symbol.

Type class	Example	Contractiveness	Type duality / equivalence	Language model
Finite	!end.end	Polytime	Polytime	Finite languages
Recursive	T_{loop}	Polytime	Polytime	Regular languages
1-counter	$T_{counter}$	Polytime	Polytime	1-counter languages
HO context-free	T_{tree}	Polytime	Decidable	Open[3]
Pushdown	T_{meta}	Polytime	Decidable	DCFL
Nested	T_{nest}	Polytime	Decidable	DCFL
2-counter	T_{iter}	Undecidable	Undecidable	Decidable languages
Non-computable	—	—	—	General languages

Our main contributions can be summarized as follows.

- We propose three novel formal systems for representing session types (1-counter, pushdown, 2-counter), show that they are strictly more expressive than recursive session types, and that each system is strictly more expressive than the previous one (Theorem 1).
- We show that nested session types [9] are equivalent to pushdown session types (Theorem 1).
- We introduce higher-order context-free session types and show that they stand between recursive and pushdown types, strictly (Theorem 1).
- We characterize each of the novel session types in our paper by a corresponding class in the Chomsky hierarchy of languages. Notably, we show that each model captures precisely the power of the corresponding class of automata (Theorem 2). This is in contrast with the results of Das et al. [9], who only show (in one direction) that nested session types can be simulated by deterministic pushdown automata.
- We prove that type formation, type equivalence and type duality are decidable up to pushdown session types (Theorem 3), but undecidable for 2-

---

[3]Possibly languages accepted by a single-state pushdown automata with empty stack acceptance.

Polarity and view

$$\sharp ::= \; ? \; | \; ! \qquad \star ::= \; \& \; | \; \oplus$$

Type formation $\boxed{T \; \text{type}}$

$$\text{end type} \qquad\qquad (\text{T-End})$$

$$\frac{T \; \text{type} \quad U \; \text{type}}{\sharp T.U \; \text{type}} \qquad (\text{T-Msg})$$

$$\frac{T_\ell \; \text{type} \quad (\forall \ell \in L)}{\star\{\ell\colon T_\ell\}_{\ell \in L} \; \text{type}} \qquad (\text{T-Choice})$$

Type equivalence $\boxed{T \simeq T}$

$$\text{end} \simeq \text{end} \qquad\qquad (\text{E-End})$$

$$\frac{T \simeq U \quad V \simeq W}{\sharp T.V \simeq \sharp U.W} \qquad (\text{E-Msg})$$

$$\frac{T_l \simeq U_l \quad (\forall \ell \in L)}{\star\{\ell\colon T_\ell\}_{\ell \in L} \simeq \star\{\ell\colon U_\ell\}_{\ell \in L}} \qquad (\text{E-Choice})$$

Duality $\boxed{S \perp S}$

$$\overline{?} = \; ! \qquad \overline{!} = \; ? \qquad \overline{\&} = \oplus \qquad \overline{\oplus} = \&$$

$$\text{end} \perp \text{end} \qquad\qquad (\text{D-End})$$

$$\frac{T \simeq U \quad V \perp W}{\sharp T.V \perp \overline{\sharp} U.W} \qquad (\text{D-Msg})$$

$$\frac{T_\ell \perp U_\ell \quad (\forall \ell \in L)}{\star\{\ell\colon T_\ell\}_{\ell \in L} \perp \overline{\star}\{\ell\colon U_\ell\}_{\ell \in L}} \qquad (\text{D-Choice})$$

Fig. 1. Finite and infinite types.

counter session types (Theorem 4). This implies that equivalence for higher-order context-free session types is decidable. The decidability results are not entirely unexpected, given that type equivalence for nested session types was recently shown to be decidable [9], and that these are equivalent to pushdown types. However, our proofs are independent of Das et al. [9].

Organization of the paper In Section 2 we introduce the various classes of session types. In Section 3 we explain how to associate to each given type a labelled infinite tree, as well as a set which we call the language of traces of that type. We also present our results on the strict hierarchy of types and state how previously studied classes of types fit into this hierarchy (Theorem 1). In Section 4 we describe how to convert a type into an automaton accepting its traces. In Section 5 we travel in the converse direction, i.e., from an automata into the corresponding type, and present a characterisation theorem of the different types in our hierarchy (Theorem 2). We then present our main algorithmic results: type formation, type equivalence and type duality are all decidable up to pushdown types (Theorem 3), and undecidable for 2-counter types (Theorem 4). Due to space constraints, all proofs and additional details can be found in the extended version of our paper at arXiv [16].

2 Shades of types

The finite world Finite types are in Fig. 1. The syntax of types is introduced via formation rules, paving the way for infinite types. Session types comprise the terminated type end, the input type $?T.U$ (input a value of type T and

| Type contractivity (*ind.*) | $\boxed{T \text{ contr}}$ | New equivalence rules (*coind.*) | $\boxed{T \simeq T}$ |

$$\text{end contr} \qquad \text{(C-End)}$$

$$\sharp T.U \text{ contr} \qquad \text{(C-Msg)}$$

$$\star\{\ell: T_\ell\}_{\ell \in L} \text{ contr} \qquad \text{(C-Choice)}$$

$$\frac{X \doteq T \quad T \text{ contr}}{X \text{ contr}} \qquad \text{(C-Id)}$$

$$\frac{X \doteq U \quad U \text{ contr} \quad U \simeq T}{X \simeq T} \qquad \text{(E-ConsL)}$$

$$\frac{X \doteq U \quad U \text{ contr} \quad T \simeq U}{T \simeq X} \qquad \text{(E-ConsR)}$$

New duality rules (*coind.*) $\boxed{T \perp T}$

New formation rules (*coind.*) $\boxed{T \text{ type}}$

$$\frac{X \doteq T \quad T \text{ contr} \quad T \text{ type}}{X \text{ type}} \qquad \text{(T-Id)}$$

$$\frac{X \doteq U \quad U \text{ contr} \quad U \perp T}{X \perp T} \qquad \text{(D-IdL)}$$

$$\frac{X \doteq U \quad U \text{ contr} \quad T \perp U}{T \perp X} \qquad \text{(D-IdR)}$$

Fig. 2. Recursive types. Extends Fig. 1.

continue as U), the output type $!T.U$ (output a value of type T and continue as U), external choice $\&\{\ell: T_\ell\}_{\ell \in L}$ (receive a label $k \in L$ and continue as T_k) and internal choice $\oplus\{\ell: T_\ell\}_{\ell \in L}$ (select a label $k \in L$ and continue as T_k). To avoid repeating similar rules, we use the symbol $\sharp$ to denote either $?$ or $!$, and the symbol $\star$ to denote either $\&$ or $\oplus$. At this point type equivalence is essentially syntactic equality, but the rule format allows for seamless extensions to infinite settings. Types, type equivalence and duality are all standard [15,20,44]. Note that rule D-Msg defines duality with respect to type equivalence: $!T.V$ and $?U.W$ are dual types iff the type being exchanged is the same ($T \simeq U$) and the continuations are dual ($V \perp W$).

For finite types all judgements in Fig. 1 are interpreted *inductively*. For example, we can show that $!(?\text{end}.\text{end}).!\text{end}.\text{end}$ is a type by exhibiting a finite derivation ending with this judgement.

The recursive world Recursive types suggest the first glimpse of infinity. The details are in Fig. 2. Recursion is given via equations, rather than μ-types for example, for easier extension. Towards this end, we introduce type identifiers X and equations of the form $X \doteq T$. The set of type identifiers is finite. We further assume at most one equation for each type, so that there are finitely many type equations. Every valid type T is required to be contractive, that is T contr. Contractiveness ensures that types reveal a type constructor after finitely many unfolds, and excludes undesirable cycles that don't describe any behaviour, e.g. cycles of the form $\{X \doteq Y, Y \doteq Z, Z \doteq X\}$. Contractiveness is inductive: we look for finite derivations for T contr judgements. A coinductive interpretation of the rules would allow to conclude X contr given an equation $X \doteq X$. In contrast, type formation, type equivalence and duality are now interpreted *coinductively*.

For example, no finite derivation would allow showing that T_{loop} type. Instead we proceed by showing that set $\{\text{end}, !\text{end}.X, X\}$ is *backward closed* [34] for the rules for T type in Fig. 2, given that $!\text{end}.X$, the right-hand side of the equation for X, is contractive.

Natural numbers

$$n ::= z \mid s\,n$$

New contractivity rules (*ind.*) $\boxed{T \text{ contr}}$

$$\frac{X\langle z\rangle \doteq T \quad T \text{ contr}}{X\langle z\rangle \text{ contr}} \quad \text{(C-z)}$$

$$\frac{X\langle s\,N\rangle \doteq T \quad T[n/N] \text{ contr}}{X\langle s\,n\rangle \text{ contr}} \quad \text{(C-s)}$$

New formation rule (*coind.*) $\boxed{T \text{ type}}$

$$\frac{X\langle z\rangle \doteq T \quad T \text{ contr} \quad T \text{ type}}{X\langle z\rangle \text{ type}} \quad \text{(T-z)}$$

$$\frac{X\langle s\,N\rangle \doteq T \quad T[n/N] \text{ contr} \quad T[n/N] \text{ type}}{X\langle s\,n\rangle \text{ type}} \quad \text{(T-s)}$$

New equivalence rules (*coind.*) $\boxed{T \simeq T}$

$$\frac{X\langle z\rangle \doteq U \quad U \text{ contr} \quad U \simeq T}{X\langle z\rangle \simeq T} \quad \text{(E-zL)}$$

$$\frac{X\langle s\,N\rangle \doteq U \quad U[n/N] \text{ contr} \quad U[n/N] \simeq T}{X\langle s\,n\rangle \simeq T} \quad \text{(E-sL)}$$

Fig. 3. 1-counter types. Extends Fig. 2; removes X; adds $X\langle n\rangle$. Right versions of rules E-zL and E-sL omitted. New rules for duality obtained from those for equivalence by replacing $\simeq$ by $\bot$.

The 1-counter world The next step takes us to equations parameterised on natural numbers. The details are in Fig. 3. Natural numbers are built from the nullary constructor z and the unary constructor s. We discuss the changes from the recursive world in Fig. 2. Given a variable N on natural numbers, to each type identifier X we associate at most two equations, $X\langle z\rangle \doteq T$ and $X\langle s\,N\rangle \doteq U$. The rules for recursive types are naturally adapted to 1-counter types. Here again, type formation requires a suitable notion of contractiveness to exclude cycles of equations that never reach a type identifier, e.g. cycles of the form $\{X\langle s\,N\rangle \doteq Y\langle s\,s\,N\rangle, Y\langle s\,N\rangle \doteq X\langle N\rangle\}$. The right-hand-side of an equation $X\langle s\,N\rangle \doteq T$ is not necessarily a type for it may contain natural number variables (N in particular). However, if n is a natural number, then $T[n/N]$ (that is, T with occurrences of N replaced by n) should be a type (cf. rule T-s). Again, to prove that T_{counter} type, we show backward closure of the set $\{X\langle n\rangle, Y\langle n\rangle, \text{end}, !\text{end}.Y\langle n\rangle, \&\{\text{inc}: X\langle s\,n\rangle, \text{dump}: Y\langle n\rangle\} \mid n \text{ nat}\}$ for the type formation rules.

Higher-order context-free session types A little detour takes us to context-free session types, proposed by Thiemann and Vasconcelos [39] (see also Almeida et al. [1]). Here we follow the distilled presentation of Almeida et al. [2], extending types to the higher-order setting (that is, allowing $?T$ and $!T$ for an arbitrary type T instead of just basic type skip).

The pushdown world The next extension replaces natural numbers by finite sequences s of symbols σ taken from a given stack alphabet. The details are in Fig. 4. We use ε to denote the empty sequence. The extension from 1-counter is straightforward. Parameters to type identifiers are now sequences of symbols, rather than natural numbers; all the rest remains the same. Once again, to show that T_{meta} type, we proceed coinductively.

Strings

$$s ::= \varepsilon \mid \sigma s$$

New contractive rules (*ind.*) $\boxed{T \text{ contr}}$

$$\frac{X\langle\varepsilon\rangle \doteq T \quad T \text{ contr}}{X\langle\varepsilon\rangle \text{ contr}} \quad \text{(C-z)}$$

$$\frac{X\langle\sigma S\rangle \doteq T \quad T[s/S] \text{ contr}}{X\langle\sigma s\rangle \text{ contr}} \quad \text{(C-s)}$$

New formation rules (*coind.*) $\boxed{T \text{ type}}$

$$\frac{X\langle\varepsilon\rangle \doteq T \quad T \text{ contr} \quad T \text{ type}}{X\langle\varepsilon\rangle \text{ type}} \quad \text{(T-z)}$$

$$\frac{X\langle\sigma S\rangle \doteq T \quad T[s/S] \text{ contr} \quad T[s/S] \text{ type}}{X\langle\sigma s\rangle \text{ type}} \quad \text{(T-s)}$$

New equivalence rules (*coind.*) $\boxed{T \simeq T}$

$$\frac{X\langle\varepsilon\rangle \doteq U \quad U \text{ contr} \quad U \simeq T}{X\langle\varepsilon\rangle \simeq T} \quad \text{(E-zL)}$$

$$\frac{X\langle\sigma S\rangle \doteq U \quad U[s/S] \text{ contr} \quad U[s/S] \simeq T}{X\langle\sigma s\rangle \simeq T} \quad \text{(E-sL)}$$

Fig. 4. Pushdown types. Extends Fig. 2; removes X; adds $X\langle s\rangle$. Right versions of rules E-zL and E-sL omitted. For duality, proceed as in Fig. 3.

Nested session types A class of types that turns out to be equivalent to pushdown types was recently proposed by Das et al. [9]. The main idea is to have type identifiers that are applied not to natural numbers or to sequences of symbols but to types themselves, and to let type identifiers take a variable (but fixed) number of parameters.

The 2-counter world 2-counter types extend the 1-counter types by introducing equations parameterised on two natural numbers, rather than one. The new rules are a straightforward adaptation of those in Fig. 3 for 1-counter types and are thus omitted. To show that T_{iter} type, we proceed coinductively.

The infinite world The final destination takes us to arbitrary, coinductive, infinite types. The details are in Fig. 1, except that all judgements not explicitly marked are taken coinductively. No equations (of any sort) are needed, just plain infinite types. We also allow choices with an infinite number of branches.

Infinite types arise by interpreting the syntax rules coinductively, which gives rise to potentially infinite chains of interactions. The structure of these arbitrary, coinductively defined, infinite types does not need to follow any pattern (e.g. it does not need to repeat itself), and arguably, the best way to think about these objects are as labelled infinite trees (Section 3). Such objects do not have in general a finite representation (or finite encoding), which can be shown by a simple cardinality argument. Hence the need for finding suitable subclasses of infinite types that can be represented and can be used in practice.

We can think of a type in two possible ways: as (one of) its representation(s), which is great for practical purposes as we can reason about types by reasoning about their representations; or as the underlying, possibly infinite, coinductive object which is being represented, which is suitable for developing a theory of types, in particular for comparing different models with one another.

3   Types, trees and traces

It should be clear that the constructions defined in Section 2 form some sort of type hierarchy; this section studies the hierarchy. In any case, every type lives in the largest universe; that of arbitrary, coinductively defined, infinite types.

To each type one can associate a labelled infinite tree [14,32]. This tree can in turn be expressed by the language of words encoding its paths. Let $\mathbb{L}$ be the set of labels used in choice types. Following Pierce [32, Definition 21.2.1], a tree is a partial function $t \in (\{\mathsf{d},\mathsf{c}\} \cup \mathbb{L})^* \to \{\mathsf{end}, ?, !, \&_L, \oplus_L \mid L \subseteq \mathbb{L}\}$ subject to the following constraints (σ ranges over symbols and π over strings of symbols):

- $t(\varepsilon)$ is defined;
- if $t(\pi\sigma)$ is defined, then $t(\pi)$ is defined;
- if $t(\pi) = ?$ or $t(\pi) = !$, then $t(\pi\sigma)$ is defined for $\sigma \in \{\mathsf{d},\mathsf{c}\}$ and undefined for all other σ;
- if $t(\pi) = \&_L$ or $t(\pi) = \oplus_L$, then $t(\pi\sigma)$ is defined for $\sigma \in L$ and undefined for all other σ;
- if $t(\pi) = \mathsf{end}$, then $t(\pi\sigma)$ is undefined for all σ.

The labels d and c are abbreviations for *data* and *continuation*, corresponding to the two components of session types for messages.

If all sets L in a tree are finite, the tree is finitely branching. The tree generated by a (finite or infinite) type is coinductively defined as follows.

$$\mathrm{treeof}(\sharp T_\mathsf{d}.T_\mathsf{c})(\varepsilon) = \sharp \qquad\qquad \mathrm{treeof}(\star\{\ell : T_\ell\}_{\ell \in L})(\varepsilon) = \star_L$$
$$\mathrm{treeof}(\sharp T_\mathsf{d}.T_\mathsf{c})(\mathsf{d}\pi) = \mathrm{treeof}(T_\mathsf{d})(\pi) \qquad \mathrm{treeof}(\star\{\ell : T_\ell\}_{\ell \in L})(\ell\pi) = \mathrm{treeof}(T_\ell)(\pi)$$
$$\mathrm{treeof}(\sharp T_\mathsf{d}.T_\mathsf{c})(\mathsf{c}\pi) = \mathrm{treeof}(T_\mathsf{c})(\pi) \qquad\qquad \mathrm{treeof}(\mathsf{end})(\varepsilon) = \mathsf{end}$$

A *path* in a tree t is a word obtained by combining the symbols in the domain and the range of t. Given a symbol $\sigma \in \{?, !, \&_L, \oplus_L \mid L \subseteq \mathbb{L}\}$ in the codomain of t (but different from end), and a symbol $\tau \in \{\mathsf{d},\mathsf{c}\} \cup \mathbb{L}$, let $\langle \sigma, \tau \rangle$ denote the combination of both symbols, viewed as a letter over the alphabet $\{?, !, \&_L, \oplus_L \mid L \subseteq \mathbb{L}\} \times (\{\mathsf{d},\mathsf{c}\} \cup \mathbb{L})$. For simplicity in exposition, we often drop the angular brackets and the subscript L on the label set, and write, for example, $?\mathsf{c}$ instead of $\langle ?, \mathsf{c} \rangle$, $\oplus l$ instead of $\langle \oplus_L, l \rangle$, etc.

Given a string π in the domain of a tree t, we can define the word $\mathrm{path}_t(\pi)$ recursively as $\mathrm{path}_t(\varepsilon) = \varepsilon$ and $\mathrm{path}_t(\pi\tau) = \mathrm{path}_t(\pi) \cdot \langle t(\pi), \tau \rangle$. We say that a string π *is terminal wrt to* t if $t(\pi) = \mathsf{end}$. For terminal strings, we can further define $\mathrm{dpath}_t(\pi) = \mathrm{path}_t(\pi) \cdot \mathsf{end}$.

Finally, we can define the language of (the paths in) a tree t as the set $\{\mathrm{path}_t(\pi) \mid \pi \in \mathrm{dom}(t)\} \cup \{\mathrm{dpath}_t(\pi) \mid \pi \in \mathrm{dom}(t), \pi \text{ is terminal wrt } t\}$. The language of (the traces of) a type T, denoted by $\mathcal{L}(T)$, is the language of $\mathrm{treeof}(T)$. Note that the traces of types are defined over the following alphabet.

$$\Sigma = \{?, !, \&_L, \oplus_L \mid L \subseteq \mathbb{L}\} \times (\{\mathsf{d},\mathsf{c}\} \cup \mathbb{L}) \cup \{\mathsf{end}\} \tag{1}$$
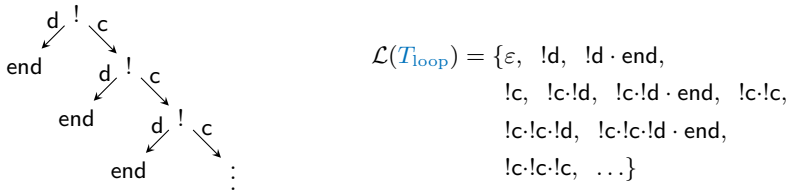
$$\mathcal{L}(T_{\text{loop}}) = \{\varepsilon, \ !\mathsf{d}, \ !\mathsf{d} \cdot \mathsf{end},$$
$$!\mathsf{c}, \ !\mathsf{c}\cdot!\mathsf{d}, \ !\mathsf{c}\cdot!\mathsf{d} \cdot \mathsf{end}, \ !\mathsf{c}\cdot!\mathsf{c},$$
$$!\mathsf{c}\cdot!\mathsf{c}\cdot!\mathsf{d}, \ !\mathsf{c}\cdot!\mathsf{c}\cdot!\mathsf{d} \cdot \mathsf{end},$$
$$!\mathsf{c}\cdot!\mathsf{c}\cdot!\mathsf{c}, \ \ldots\}$$

Fig. 5. The tree and the language of type T_{loop}.

$$\mathcal{L}(T_{\text{counter}}) = \{\varepsilon, \ \&\mathsf{inc}, \ \&\mathsf{dump}, \ \&\mathsf{dump} \cdot \mathsf{end},$$
$$\&\mathsf{inc} \cdot \&\mathsf{inc}, \ \&\mathsf{inc} \cdot \&\mathsf{dump},$$
$$\&\mathsf{inc} \cdot \&\mathsf{dump}\cdot!\mathsf{d},$$
$$\&\mathsf{inc} \cdot \&\mathsf{dump}\cdot!\mathsf{d} \cdot \mathsf{end},$$
$$\&\mathsf{inc} \cdot \&\mathsf{dump}\cdot!\mathsf{c},$$
$$\&\mathsf{inc} \cdot \&\mathsf{dump}\cdot!\mathsf{c} \cdot \mathsf{end},$$
$$\&\mathsf{inc} \cdot \&\mathsf{inc} \cdot \&\mathsf{inc},$$
$$\&\mathsf{inc} \cdot \&\mathsf{inc} \cdot \&\mathsf{dump},$$
$$\&\mathsf{inc} \cdot \&\mathsf{inc} \cdot \&\mathsf{dump}\cdot!\mathsf{d}, \ \ldots\}$$
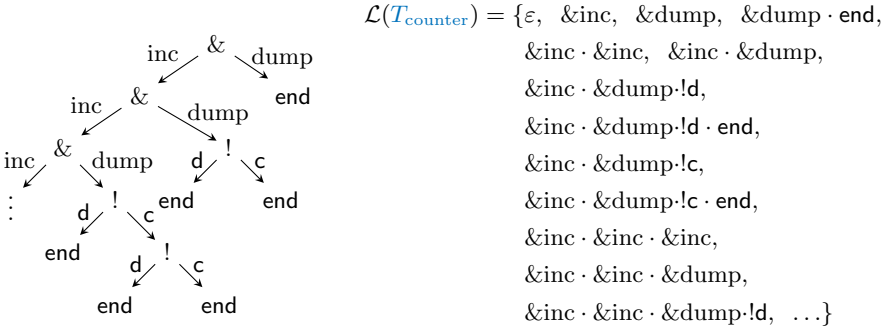
Fig. 6. The tree and the language of type T_{counter}.

$$\mathcal{L}(T_{\text{tree}}) = \{\varepsilon, \ \&\mathsf{leaf}, \ \&\mathsf{leaf} \cdot \mathsf{end}, \ \&\mathsf{node},$$
$$\&\mathsf{node} \cdot \&\mathsf{leaf}, \ \&\mathsf{node} \cdot \&\mathsf{node},$$
$$\&\mathsf{node} \cdot \&\mathsf{leaf}\cdot?\mathsf{d},$$
$$\&\mathsf{node} \cdot \&\mathsf{leaf}\cdot?\mathsf{d} \cdot \mathsf{end},$$
$$\&\mathsf{node} \cdot \&\mathsf{leaf}\cdot?\mathsf{c},$$
$$\&\mathsf{node} \cdot \&\mathsf{node} \cdot \&\mathsf{leaf},$$
$$\&\mathsf{node} \cdot \&\mathsf{node} \cdot \&\mathsf{node}, \ \ldots\}$$
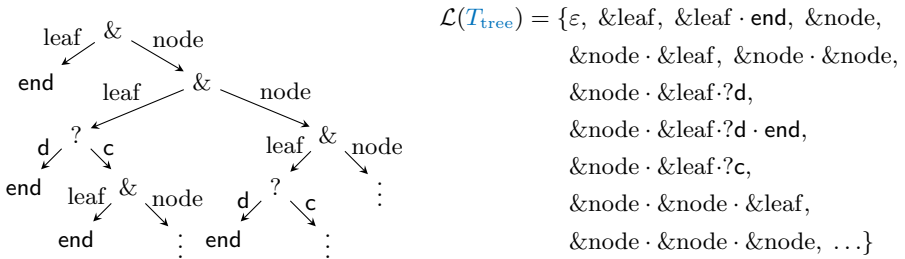
Fig. 7. The tree and the language of type T_{tree}.

Figure 5 depicts (a finite fragment of) the tree corresponding to treeof(T_{loop}) (Example 1) and (some of the words in) its language $\mathcal{L}(T_{\text{loop}})$. Type T_{counter} (Example 2) describes an interaction that keeps track of a counter. Finite fragments of the corresponding tree and language are depicted in Fig. 6. Type T_{tree} (Example 3) describes the reception of a binary tree of end values. Finite fragments of the corresponding tree and language are depicted in Fig. 7.

In the above examples, the language $\mathcal{L}(T)$ is closed under prefixes. This holds for a general type T, since elements of $\mathcal{L}(T)$ correspond to paths in treeof(T).

Proposition 1. *If $w \in \mathcal{L}(T)$ and u is a prefix of w then $u \in \mathcal{L}(T)$.*

Another immediate observation is that treeof (resp. $\mathcal{L}$) is an embedding from the class of all types to the class of all trees (resp. all languages).

Proposition 2. *Let T and U be two types. The following are equivalent: a) $T \simeq U$; b) treeof$(T) =$ treeof(U); c) $\mathcal{L}(T) = \mathcal{L}(U)$.*

Proposition 2 tells us that two types are equivalent iff they have the same traces. In general, trace equivalence is a notion weaker than bisimulation [34]. However, both notions coincide for deterministic transition systems. The syntax of (infinite) session types is in fact deterministic (e.g. given a label ℓ for a choice, there can only be one type that continues from $\&\ell$), which explains our result.

Section 2 introduces eight classes of types. We now distinguish them by means of subscripts: finite types (T type$_f$, Fig. 1), recursive types (T type$_r$, Fig. 2), 1-counter types (T type$_1$, Fig. 3), context-free types (T type$_c$), pushdown types (T type$_p$, Fig. 4), nested types (T type$_n$), 2-counter types (T type$_2$) and coinductive, infinite types (T type$_\infty$, Fig. 1 with rules interpreted coinductively). To each class of types we introduce the corresponding class of languages. For example, $\mathbb{T}_r$ is the set $\{\mathcal{L}(T) \mid T \text{ type}_r\}$. The strict hierarchy result is as follows:

$$\mathbb{T}_f \subsetneq \mathbb{T}_r \subsetneq \mathbb{T}_1 \subsetneq \mathbb{T}_p \subsetneq \mathbb{T}_2 \subsetneq \mathbb{T}_\infty$$

We remark that the last step in the chain of strict inclusions is obtained by a cardinality argument, since the set $\mathbb{T}_\infty$ is uncountable. This shows an even stronger statement: for any finite representation system (including the systems $\mathbb{T}_f$ to $\mathbb{T}_2$, as well as $\mathbb{T}_c$ and $\mathbb{T}_n$), there is an *infinite, uncountable* set of types that cannot be represented by that system.

We now turn our attention to nested types (T type$_n$), which turn out to be equivalent to pushdown types, and further establish equivalent sub-hierarchies inside both classes, parameterised by the 'complexity' of the corresponding representations. For pushdown session types, a natural measure of complexity is the number of type identifiers required to represent a given type. This number can be arbitrarily large, but always finite. For a given $n \in \mathbb{N}$, we let $\mathbb{T}_p^n$ denote the subset corresponding to those types that can be represented with at most n type identifiers. When $n = 0$, there are no identifiers, and we can only represent finite types. As n increases, so does the expressivity of our constructions, and we have the infinite chain of inclusions[4]

$$\mathbb{T}_f = \mathbb{T}_p^0 \subsetneq \mathbb{T}_p^1 \subseteq \mathbb{T}_p^2 \subseteq \cdots \subseteq \mathbb{T}_p.$$

Similarly, for nested session types we can define a hierarchy by looking at the arities of the type identifiers used. For a given $n \in \mathbb{N}$, we let $\mathbb{T}_n^n$ denote the subset corresponding to the nested session types whose type identifiers have arity at most n. When $n = 0$ all type identifiers are constant, and we recover the class of recursive types. As n increases, so does the expressivity, and we also have an infinite chain of inclusions[4]

$$\mathbb{T}_r = \mathbb{T}_n^0 \subsetneq \mathbb{T}_n^1 \subseteq \mathbb{T}_n^2 \subseteq \cdots \subseteq \mathbb{T}_n.$$

---

[4]Although not proven, we conjecture that all inclusions are strict.

It turns out that these hierarchies are one and the same (with the exception of the bottom level), so that we have [4]

$$\mathbb{T}_f = \mathbb{T}_p^0 \subsetneq \mathbb{T}_r = \mathbb{T}_n^0 \subsetneq \mathbb{T}_p^1 = \mathbb{T}_n^1 \subseteq \mathbb{T}_p^2 = \mathbb{T}_n^2 \subseteq \cdots \subseteq \mathbb{T}_p = \mathbb{T}_n.$$

Higher-order context-free types (denoted by $\mathbb{T}_c$) lie between levels 0 and 1 in the sub-hierarchies above, i.e., they can represent recursive types, and can be represented by pushdown session types using at most one type identifier, or equivalently, by nested session types with either constant or unary type identifiers, so that we have

$$\mathbb{T}_r \subsetneq \mathbb{T}_c \subsetneq \mathbb{T}_p^1 = \mathbb{T}_n^1.$$

We have a stronger observation than the inclusion $\mathbb{T}_c \subsetneq \mathbb{T}_p^1$. Context-free session types are included in pushdown session types which have only one type identifier X, and where the equation $X\langle\varepsilon\rangle \doteq \mathsf{end}$ accounts for the only occurrence of end. The latter means that the type ends iff the state $X\langle\varepsilon\rangle$ is reached, that is, iff the stack is empty. Thus, we can intuitively think of context-free session types as pushdown types with a single identifier and an empty stack acceptance criterion. This observation points to the fact that the qualifier 'context-free' in the so called context-free session types is a misnomer [9].

The result below summarises the entire hierarchy.

Theorem 1 (Inclusions).

$$\mathbb{T}_f = \mathbb{T}_p^0 \subsetneq \mathbb{T}_r = \mathbb{T}_n^0 \qquad \subsetneq \qquad \mathbb{T}_1 \qquad \subsetneq \qquad \mathbb{T}_p = \mathbb{T}_n \subsetneq \mathbb{T}_2 \subsetneq \mathbb{T}_\infty$$
$$\mathbb{T}_c \subsetneq \mathbb{T}_p^1 = \mathbb{T}_n^1 \subseteq \mathbb{T}_p^2 = \mathbb{T}_n^2 \subseteq \cdots$$

4 From types to automata

This section describes procedures to convert types in different levels of the hierarchy (recursive systems, 1-counter, pushdown and 2-counter) into automata at the same level. All constructions follow the same guiding principles, so we focus on the bottom level of the hierarchy (recursive systems) and then highlight the main differences as we advance in the hierarchy.

All automata that we consider are *deterministic* and *total*, i.e., the transition functions are such that any input word has a well-defined, unique computation path. We use the alphabet Σ defined in (1). Standard references in automata theory are Hopcroft and Ullman's book [22] and Valiant's PhD thesis [42].

Recursive types and finite-state automata Following the usual notation, a (deterministic) *finite-state automaton* is given by a set Q of states, with a specified initial state $q_0 \in Q$, a transition function $\delta : Q \times \Sigma \to Q$, and a set $A \subseteq Q$ of accepting states. Given a finite word $a_1 a_2 \cdots a_n$, its execution by the automaton yields the sequence of states $s_0, s_1, \ldots, s_n$ where $s_0 = q_0$ and $s_{i+1} = \delta(s_i, a_{i+1})$. A word is *accepted* by the automaton if its execution ends in an accepting state.

The definition of finite-state automata can be augmented into other types of automata. Essentially: in a 1-counter automata we have access to a counter (with operations for incrementing, decrementing, and checking whether the counter is non-zero), in addition to the current state; in a pushdown automata we have access to a stack (with operations for pushing a symbol, popping a symbol, and observing the top symbol of the stack); in a 2-counter automata, we have access to two counters.

Suppose we are given a system of recursive equations $\{X_i \doteq T_i\}_{i \in I}$ over a set $\mathcal{X} = \{X_i\}_{i \in I}$ (which may or may not be contractive, i.e., define a type). Our first step is to convert this system into a normal form in which every right-hand side is either a identifier X, or a single application of one of the type constructors end, $?X.Y$, $!X.Y$, $\&\{\ell \colon X_\ell\}_{\ell \in L}$ or $\oplus\{\ell \colon X_\ell\}_{\ell \in L}$. We can do this by introducing fresh, intermediate identifiers as needed. Essentially, whenever we have an equation $X \doteq ?T_1.T_2$ where T_1, T_2 are not identifiers, we add two new identifiers X', X'', replace the above equation by $X \doteq ?X'.X''$, and add two new equations $X' \doteq T_1$ and $X'' \doteq T_2$. The process is similar for the other type constructors. By doing this repeatedly, we "break down" a long equation into many small equations. The number of new identifiers is linear in the size of the original system of equations.

Given such a system, we construct a finite-state automaton (over the alphabet Σ) as follows. The automaton has a state q_X for every type identifier X, and two additional states: an 'end' state q_{end} and an 'error' state q_{error}. The transitions from q_{error} are described by $q_{\mathsf{error}} \xrightarrow{a} q_{\mathsf{error}}$ for every symbol a. Similarly, the transitions at q_{end} are described by $q_{\mathsf{end}} \xrightarrow{a} q_{\mathsf{error}}$ for every symbol a. The transitions at state q_X are given by the corresponding equation for identifier X, in the obvious way. Some examples:

- If our system contains equation $X \doteq Y$, we have the ε-transition $q_X \xrightarrow{\varepsilon} q_Y$.
- If our system contains $X \doteq !Y.Z$, we have the reading transitions $q_X \xrightarrow{!\mathsf{d}} q_Y$, $q_X \xrightarrow{!\mathsf{c}} q_Z$, and $q_X \xrightarrow{a} q_{\mathsf{error}}$ for any $a \neq !\mathsf{d}, !\mathsf{c}$.
- If our system contains $X \doteq \oplus\{l \colon X, m \colon Y\}$, we have the reading transitions $q_X \xrightarrow{\oplus l} q_X$, $q_X \xrightarrow{\oplus m} q_Y$ and $q_X \xrightarrow{a} q_{\mathsf{error}}$ for any $a \neq \oplus l, \oplus m$.
- If our system contains $X \doteq \mathsf{end}$, we have the reading transitions $q_X \xrightarrow{\mathsf{end}} q_{\mathsf{end}}$ and $q_X \xrightarrow{a} q_{\mathsf{error}}$ for any $a \neq \mathsf{end}$.

We define all states other than q_{error} to be accepting states.[5] Notice that the finite-state automaton described above is an automaton with possible ε-moves. Although, by definition, deterministic finite-state automata do not permit ε-moves, in our case paths of ε-moves are uniquely determined and always reach a state without outgoing ε-transitions (they cannot become stuck in a loop, assuming type contractivity). We can convert the given automaton into an equivalent automaton without ε-moves by 'shortcutting' such moves. Formally, suppose a

---

[5] We need all states to be accepting, since we might need to look at finite traces to distinguish between two types. For example, $X \doteq \&\{\mathsf{a} \colon X\}$ and $Y \doteq \&\{\mathsf{b} \colon Y\}$ define non-equivalent types that have no finite terminating paths.

$$X \doteq \ !\mathrm{end}.X$$
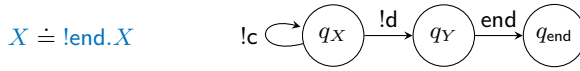
Fig. 8. An automaton for T_{loop} with initial state q_X. All depicted states are accepting.

state X has an outgoing ε-transition to Y; by construction, it is X's only outgoing transition. Assuming X and Y are different states, we can change every transition entering X and make it enter Y instead; finally, we can remove state X (hence removing the ε-transition from X).

We show in Fig. 8 the automaton that corresponds to type T_{loop} (Example 1). Every missing transition points to q_{error} which is not shown. In our examples, all depicted states are accepting, so we omit the usual double circle notation.

1-counter types For 1-counter systems, the only difference in the above construction is that instead of non-parameterised identifiers our equations now involve terms of the form $X\langle \mathsf{z}\rangle$, $X\langle \mathsf{s\,z}\rangle$, $X\langle N\rangle$, $X\langle \mathsf{s}\,N\rangle$, etc. We assume for simplicity that the identifiers appearing in these equations are restricted as follows: if the left-hand side of an equation is of the form $X\langle \mathsf{z}\rangle$, then the identifiers appearing in the right-hand side must be of the form $X'\langle \mathsf{z}\rangle$ or $X'\langle \mathsf{s\,z}\rangle$ (with X' possibly different from X); and if the left-hand side of an equation is of the form $X\langle \mathsf{s}\,N\rangle$, then the identifiers appearing in the right-hand side must be of the form $X'\langle N\rangle$, $X'\langle \mathsf{s}\,N\rangle$ or $X'\langle \mathsf{s\,s}\,N\rangle$. Any system can be converted into this form by adding finitely many new equations, e.g. $X\langle \mathsf{z}\rangle \doteq Y\langle \mathsf{s\,s\,s\,z}\rangle$ can be rewritten as

$$X\langle \mathsf{z}\rangle \doteq X'\langle \mathsf{s\,z}\rangle \qquad X'\langle \mathsf{s}\,N\rangle \doteq X''\langle \mathsf{s\,s}\,N\rangle \qquad X''\langle \mathsf{s}\,N\rangle \doteq Y\langle \mathsf{s\,s}\,N\rangle$$

and $X\langle \mathsf{s}\,N\rangle \doteq Y\langle \mathsf{z}\rangle$ can be rewritten as

$$X\langle \mathsf{s}\,N\rangle \doteq X'\langle N\rangle \qquad X'\langle \mathsf{s}\,N\rangle \doteq X'\langle N\rangle \qquad X'\langle \mathsf{z}\rangle \doteq Y\langle \mathsf{z}\rangle.$$

We can convert a 1-counter type into a (deterministic) 1-counter automaton, so that the transition function depends on whether the counter value is zero (corresponding to a left-hand side of the form $X\langle \mathsf{z}\rangle$) or positive (corresponding to a left-hand side of the form $X\langle \mathsf{s}\,N\rangle$). Furthermore, the changes in the counter value along the identifiers are incorporated by changes in the counter value along the automaton. For example, take equation $X\langle \mathsf{s}\,N\rangle \doteq Y\langle N\rangle$. The corresponding transition from $(q_X, \mathsf{s}, \varepsilon)$ to q_Y decrements the counter.

For illustration purposes, we show how to construct a 1-counter automaton accepting $\mathcal{L}(T_{\mathrm{counter}})$ from Example 2. First, we need to convert the equation for $Y\langle \mathsf{s}\,N\rangle$ into normal form. We add an extra identifier Z and write

$$X\langle \mathsf{z}\rangle \doteq \&\{\mathrm{inc}\colon X\langle \mathsf{s\,z}\rangle, \mathrm{dump}\colon Y\langle \mathsf{z}\rangle\} \quad X\langle \mathsf{s}\,N\rangle \doteq \&\{\mathrm{inc}\colon X\langle \mathsf{s\,s}\,N\rangle, \mathrm{dump}\colon Y\langle \mathsf{s}\,N\rangle\}$$
$$Y\langle \mathsf{z}\rangle \doteq \mathrm{end} \qquad\qquad\qquad\qquad Y\langle \mathsf{s}\,N\rangle \doteq \ !Z\langle \mathsf{s}\,N\rangle.Y\langle N\rangle$$
$$Z\langle \mathsf{z}\rangle \doteq \mathrm{end} \qquad\qquad\qquad\qquad Z\langle \mathsf{s}\,N\rangle \doteq \mathrm{end}$$

The corresponding automaton has states q_X, q_Y, q_Z, one for each type identifier X, Y, Z, as well as an additional state q_{end}. The outgoing transitions for state q_X
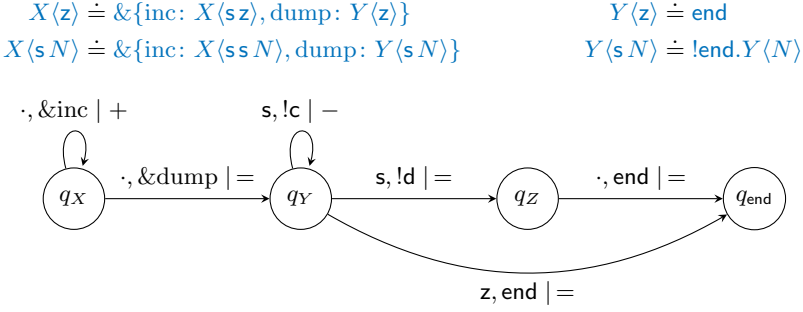
$$X\langle z\rangle \doteq \&\{\text{inc}\colon X\langle s\, z\rangle, \text{dump}\colon Y\langle z\rangle\} \qquad\qquad Y\langle z\rangle \doteq \text{end}$$
$$X\langle s\, N\rangle \doteq \&\{\text{inc}\colon X\langle s\, s\, N\rangle, \text{dump}\colon Y\langle s\, N\rangle\} \qquad Y\langle s\, N\rangle \doteq !\text{end}.Y\langle N\rangle$$

Fig. 9. A 1-counter automaton for type $T_{\text{counter}} = X\langle z\rangle$. The initial configuration is $(q_X, 0)$. Here a transition $\delta(q, g, a) = (o, q')$ is denoted by an arc from q to q' with label $g, a \mid o$, where $g \in \{z, s\}$, $a \in \{\varepsilon\} \cup \Sigma$, and $o \in \{=, +, -\}$. If both $g = z$ and $g = s$ lead to the same transition, then we use the symbol $\cdot$ to refer to both transitions. All depicted states are accepting, and non-depicted transitions lead to a non-accepting sink state.

are the same regardless of the counter value: either read &inc, incrementing the counter and staying in q_X; or read &dump, keeping the counter value and moving to q_Y. For state q_Y, if the counter is zero, we can read end while moving to state q_{end}. On the other hand, if the counter is non-zero, we can read !d, keeping the counter value and moving to q_Z; or read !c, decrementing the counter value and staying in q_Y. Finally, for state q_Z we can only read end and move to state q_{end}. Whatever we write in the equation for $Z\langle z\rangle$ is irrelevant, as this configuration is unreachable. All of this gives the automaton in Fig. 9.

Pushdown types Pushdown systems are similar, but now the behaviour of a identifier is specified by $|\Delta| + 1$ equations, where Δ is the stack alphabet; one equation for each possible symbol at the top of the stack, and one equation for the case that the stack is empty. Accordingly, we use a (deterministic) pushdown automaton to simulate the stack contents by means of push and pop operations. The transitions from a state q_X and a given stack indicator in $\{\varepsilon\} \cup \Delta$ are once more given by the corresponding equation with X as the type identifier on the left-hand side. Fig. 10 shows a pushdown automaton accepting $\mathcal{L}(T_{\text{meta}})$.

2-counter types The translation to 2-counter automata is as for the 1-counter case, but now the behaviour is specified by one of four different cases, depending on which of the two counters is zero or non-zero. Accordingly, we use a (deterministic) 2-counter automaton with the appropriate transition function.

5   From automata to types

The construction in Section 4 explains how we can build an automaton from a system of equations at some level in the hierarchy. If $X\langle\sigma\rangle$ type$_{\text{p}}$, then the

$$X\langle\varepsilon\rangle \doteq \&\{\mathrm{addOut}\colon X\langle\sigma\rangle, \mathrm{addIn}\colon X\langle\tau\rangle\}$$
$$X\langle\sigma S\rangle \doteq \&\{\mathrm{addOut}\colon X\langle\sigma\sigma S\rangle, \mathrm{addIn}\colon X\langle\tau\sigma S\rangle, \mathrm{pop}\colon \mathsf{!end}.X\langle S\rangle\}$$
$$X\langle\tau S\rangle \doteq \&\{\mathrm{addOut}\colon X\langle\sigma\tau S\rangle, \mathrm{addIn}\colon X\langle\tau\tau S\rangle, \mathrm{pop}\colon \mathsf{?end}.X\langle S\rangle\}$$
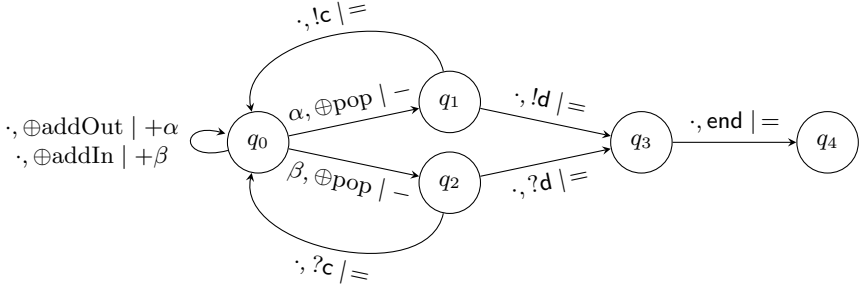
Fig. 10. A pushdown automaton for type $T_{\mathrm{meta}} = X\langle\varepsilon\rangle$. The initial configuration is (q_0, ε). A transition $\delta(q, g, a) = (o, q')$ is denoted by an arc from q to q' with label $g, a \mid o$, where $g \in \{\varepsilon\} \cup \Delta$, $a \in \{\varepsilon\} \cup \Sigma$, and $o \in \mathrm{Op}$. If all choices of g lead to the same transition, we use $\cdot$ to stand for all transitions. All depicted states are accepting.

language of the type given by $X\langle\sigma\rangle$ is the language accepted by the automaton with initial configuration (q_X, σ) (and similarly for recursive, 1-counter, and 2-counter types). Conversely, given an automaton which accepts the language of traces of a type, we can construct the corresponding system of equations that specifies that type. This allow us to obtain a complete correspondence between classes of types and different models of computation based on automata theory. The following result is stronger than previous similar results which only show a forward implication [9]. Recall that a language is said to be *regular* if it is the set of words accepted by some finite-state automaton. We also say that a tree is *regular* if it has a finite number of distinct subtrees.

Theorem 2 (Types, traces and automata).

1. T type$_{\mathsf{r}}$ *iff* $\mathcal{L}(T)$ *is regular iff* treeof(T) *is regular.*
2. T type$_1$ *iff* $\mathcal{L}(T)$ *is accepted by a 1-counter automaton.*
3. T type$_{\mathsf{p}}$ *iff* $\mathcal{L}(T)$ *is a deterministic context-free language.*
4. T type$_2$ *iff* $\mathcal{L}(T)$ *is decidable.*

We can now address the decidability of the key problems of type formation, type equivalence and type duality for our various classes of type languages.

Theorem 3 (Decidability results).

1. *Problems* T type$_{\mathsf{r}}$, T type$_1$ *and* T type$_{\mathsf{p}}$ *are all decidable in polynomial time.*
2. *Problems* $T \simeq_{\mathsf{r}} U$, $T \simeq_1 U$ *and* $T \simeq_{\mathsf{p}} U$ *are all decidable.*
3. *Problems* $T \perp_{\mathsf{r}} U$, $T \perp_1 U$ *and* $T \perp_{\mathsf{p}} U$ *are all decidable.*

We are also able to prove that these problems are undecidable for 2-counter types, since Theorem 2 also provides a construction from automata to systems of equations, and the corresponding problems for automata are undecidable.

Theorem 4 (Undecidability results).
Problems T type$_2$, $T \simeq_2 U$ and $T \perp_2 U$ are all undecidable.

6 Related work

The first papers on session types by Honda [19] and Takeuchi et al. [38] feature finite types only. Recursive types were introduced later [20] using μ-notation. Gay and Hole [15] introduce algorithms for deciding duality and subtyping of finite-state session types, based on bisimulation. Much of the literature on session types, surveyed by Hüttel et al. [23], uses the same approach. The natural decision algorithms for duality and subtyping presented by Gay and Hole were shown to be exponential in the size of the types by Lange and Yoshida [27], due to reliance on syntactic unfolding. Our polytime complexity for recursive type equivalence follows from the equivalence algorithm for finite-state automata by Hopcroft and Karp [21], and thus has quadratic complexity in the description size, improving on Gay and Hole. Lange and Yoshida use an automata-based algorithm to also achieve quadratic complexity for checking subtyping.

We use a coinductive formulation of infinite session types. This approach has some connections with the work of Keizer et al. [25] who present session types as states of coalgebras. Their types are restricted to finite-state recursive types, but they do address subtyping and non-linear types, two notions that we do not take into consideration. Our coinductive presentation avoids explicitly building coalgebras, and follows Gay et al. [17], solving problems with duality in the presence of recursive types [5,17,28].

We have not addressed the problem of deciding subtyping, but the panorama is not promising. Subtyping is known to be decidable for recursive types $\mathbb{T}_r$ [15] and undecidable for context-free types $\mathbb{T}_c$ [31] or nested types with arity at most one $\mathbb{T}_n^1$ [10], hence for pushdown types with one type constructor $\mathbb{T}_p^1$ (Theorem 1). The undecidability proof of the subtyping problem for context-free session types reduces from the inclusion problem for simple deterministic languages, which was shown to be undecidable by Friedman [13]. That for nested session types reduces from the inclusion problem for Basic Process Algebra [4], which was shown to be undecidable by Groote and Hüttel [18]. Given that 1-counter types $\mathbb{T}_1$ and pushdown types with one type constructor $\mathbb{T}_p^1$ are incomparable (Theorem 1), the problem of subtyping for 1-counter types remains open.

Dependent session types have been studied for binary session types [40,41], for multi-party session types [12,29,45] and for polymorphic, nested session types [9]. Although our parameterised type definitions have some similarities with definitions in some dependently typed systems, we do not support the connection between values in messages and parameters in types, and we have not yet studied how the types that can be expressed in dependent systems fit into our hierarchy.

Connections between multiparty session types and communicating finite-state automata have been explored by Deniélou and Yoshida [11] but the investigation has not been extended to other classes of automata.

Solomon [37] studies the connection between inductive type equality for nested types and language equality for DPDAs and shows that the equivalence problem for nested types is as hard as the equivalence problem for DPDAs, an open problem at the time. We follow a similar approach but define type equivalence as a bisimulation rather than as language equivalence.

Many of the main results in this paper borrow from the theory of automata, developed in the mid-20th century. Here our standard reference is the book by Hopcroft and Ullman [22], where the notions of finite-state, pushdown, and counter automata can be found. 1-counter automata were studied in detail in Valiant's PhD thesis [42]. To prove the equivalence between types and automata, we need to convert automata to satisfying certain properties; similar techniques have appeared in Kao et al. [24] and Valiant and Paterson [43]. Our proofs of decidability of type equivalence make use of the corresponding results for automata [8,21,33,35,36,43]; we specifically mention Sénizergues' impressive result on equivalence of deterministic pushdown automata [36], a work which granted him the Gödel Prize in 2002. Finally, the strict hierarchy results use textbook pumping lemmas for regular languages (due to Rabin and Scott [33]) and context-free languages (due to Bar-Hillel et al. [3] and Kreowski [26]), as well as a somewhat less known result for 1-counter automata (due to Boasson [7]).

7   Conclusion

We introduce different classes of session types, some new, others from the literature, under a uniform framework and place them in n hierarchy. We further study different type-related problems—formation, equivalence and duality—and show that these relations are all decidable up to and including pushdown types.

Much remains to be done. From the point of view of programming languages, one should investigate whether decidability results translate into algorithms that may be incorporated in compilers. Even if subtyping is known to be undecidable for most systems "above" that of recursive types, the problem remains open for 1-counter types, an interesting avenue for further investigation. Our study of classes of infinite types may have applications beyond session types. One promising direction is that of non regular datatypes for functional programming (or polymorphic recursion schemes [30]), such as nested datatypes [6].

We have not addressed the decidability of the type checking problem. Type checking is known to be decidable for finite types, recursive, context-free and nested session types. Given that type checking for nested session types is incorporated in the RAST language [9], a natural first step would be to investigate how to translate 1-counter and pushdown processes into that language.

References

1. Almeida, B., Mordido, A., Thiemann, P., Vasconcelos, V.T.: Polymorphic context-free session types. CoRR **abs/2106.06658** (2021), https://arxiv.org/abs/2106.06658

2. Almeida, B., Mordido, A., Vasconcelos, V.T.: Deciding the bisimilarity of context-free session types. In: TACAS. LNCS, vol. 12079, pp. 39–56. Springer (2020). https://doi.org/10.1007/978-3-030-45237-7_3

3. Bar-Hillel, Y., Perles, M., Shamir, E.: On formal properties of simple phrase structure grammars. Sprachtypologie und Universalienforschung **14**, 143–172 (1961)

4. Bergstra, J.A., Klop, J.W.: Process theory based on bisimulation semantics. In: Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency. LNCS, vol. 354, pp. 50–122. Springer (1988). https://doi.org/10.1007/BFb0013021

5. Bernardi, G., Hennessy, M.: Using higher-order contracts to model session types. Logical Methods in Computer Science **12**(2) (2016). https://doi.org/10.2168/LMCS-12(2:10)2016

6. Bird, R.S., Meertens, L.G.L.T.: Nested datatypes. In: MPC. LNCS, vol. 1422, pp. 52–67. Springer (1998). https://doi.org/10.1007/BFb0054285

7. Boasson, L.: Two iteration theorems for some families of languages. Journal of Computer and System Sciences **7**(6), 583–596 (1973)

8. Böhm, S., Göller, S., Jancar, P.: Equivalence of deterministic one-counter automata is nl-complete. In: STOC. pp. 131–140. ACM (2013). https://doi.org/10.1145/2488608.2488626

9. Das, A., DeYoung, H., Mordido, A., Pfenning, F.: Nested session types. In: ESOP. LNCS, vol. 12648, pp. 178–206. Springer (2021). https://doi.org/10.1007/978-3-030-72019-3_7

10. Das, A., DeYoung, H., Mordido, A., Pfenning, F.: Subtyping on nested polymorphic session types. CoRR **abs/2103.15193** (2021), https://arxiv.org/abs/2103.15193

11. Deniélou, P., Yoshida, N.: Multiparty session types meet communicating automata. In: ESOP. LNCS, vol. 7211, pp. 194–213. Springer (2012). https://doi.org/10.1007/978-3-642-28869-2_10

12. Deniélou, P., Yoshida, N., Bejleri, A., Hu, R.: Parameterised multiparty session types. Log. Methods Comput. Sci. **8**(4) (2012). https://doi.org/10.2168/LMCS-8(4:6)2012

13. Friedman, E.P.: The inclusion problem for simple languages. Theor. Comput. Sci. **1**(4), 297–316 (1976). https://doi.org/10.1016/0304-3975(76)90074-8

14. Gapeyev, V., Levin, M.Y., Pierce, B.C.: Recursive subtyping revealed. J. Funct. Program. **12**(6), 511–548 (2002). https://doi.org/10.1017/S0956796802004318

15. Gay, S.J., Hole, M.: Subtyping for session types in the pi calculus. Acta Inf. **42**(2-3), 191–225 (2005). https://doi.org/10.1007/s00236-005-0177-z

16. Gay, S.J., Poças, D., Vasconcelos, V.T.: The different shades of infinite session types. CoRR **abs/2201.08275** (2022), https://arxiv.org/abs/2201.08275

17. Gay, S.J., Thiemann, P., Vasconcelos, V.T.: Duality of session types: The final cut. In: PLACES. EPTCS, vol. 314, pp. 23–33 (2020). https://doi.org/10.4204/EPTCS.314.3

18. Groote, J.F., Hüttel, H.: Undecidable equivalences for basic process algebra. Inf. Comput. **115**(2), 354–371 (1994). https://doi.org/10.1006/inco.1994.1101

19. Honda, K.: Types for dyadic interaction. In: CONCUR. LNCS, vol. 715, pp. 509–523. Springer (1993). https://doi.org/10.1007/3-540-57208-2_35

20. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: ESOP. LNCS, vol. 1381, pp. 122–138. Springer (1998). https://doi.org/10.1007/BFb0053567
21. Hopcroft, J.E., Karp, R.M.: A linear algorithm for testing equivalence of finite automata. Tech. rep., Cornell University (1971)
22. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley Publishing Company (1979)
23. Hüttel, H., Lanese, I., Vasconcelos, V.T., Caires, L., Carbone, M., Deniélou, P., Mostrous, D., Padovani, L., Ravara, A., Tuosto, E., Vieira, H.T., Zavattaro, G.: Foundations of session types and behavioural contracts. ACM Comput. Surv. **49**(1), 3:1–3:36 (2016). https://doi.org/10.1145/2873052
24. Kao, J.Y., Rampersad, N., Shallit, J.: On NFAs where all states are final, initial, or both. Theoretical Computer Science **410**(47-49), 5010–5021 (2009)
25. Keizer, A.C., Basold, H., Pérez, J.A.: Session coalgebras: A coalgebraic view on session types and communication protocols. In: ESOP. LNCS, vol. 12648, pp. 375–403. Springer (2021). https://doi.org/10.1007/978-3-030-72019-3_14
26. Kreowski, H.J.: A pumping lemma for context-free graph languages. In: International Workshop on Graph Grammars and Their Application to Computer Science. pp. 270–283. Springer (1978)
27. Lange, J., Yoshida, N.: Characteristic formulae for session types. In: TACAS. LNCS, vol. 9636, pp. 833–850. Springer (2016). https://doi.org/10.1007/978-3-662-49674-9_52
28. Lindley, S., Morris, J.G.: Talking bananas: structural recursion for session types. In: ICFP. pp. 434–447. ACM (2016). https://doi.org/10.1145/2951913.2951921
29. de Muijnck-Hughes, J., Brady, E.C., Vanderbauwhede, W.: Value-dependent session design in a dependently typed language. In: PLACES. EPTCS, vol. 291, pp. 47–59 (2019). https://doi.org/10.4204/EPTCS.291.5
30. Mycroft, A.: Polymorphic type schemes and recursive definitions. In: International Symposium on Programming. LNCS, vol. 167, pp. 217–228. Springer (1984). https://doi.org/10.1007/3-540-12925-1_41
31. Padovani, L.: Context-free session type inference. ACM Trans. Program. Lang. Syst. **41**(2), 9:1–9:37 (2019). https://doi.org/10.1145/3229062
32. Pierce, B.C.: Types and programming languages. MIT Press (2002)
33. Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM journal of research and development **3**(2), 114–125 (1959)
34. Sangiorgi, D.: Introduction to Bisimulation and Coinduction. Cambridge University Press (2012). https://doi.org/10.1017/CBO9780511777110
35. Sénizergues, G.: The equivalence problem for deterministic pushdown automata is decidable. In: ICALP'97. LNCS, vol. 1256, pp. 671–681. Springer (1997). https://doi.org/10.1007/3-540-63165-8_221
36. Sénizergues, G.: L (a)= l(b)? decidability results from complete formal systems. Theoretical Computer Science **251**(1-2), 1–166 (2001)
37. Solomon, M.H.: Type definitions with parameters. In: POPL. pp. 31–38. ACM Press (1978). https://doi.org/10.1145/512760.512765
38. Takeuchi, K., Honda, K., Kubo, M.: An interaction-based language and its typing system. In: PARLE. LNCS, vol. 817, pp. 398–413. Springer (1994). https://doi.org/10.1007/3-540-58184-7_118
39. Thiemann, P., Vasconcelos, V.T.: Context-free session types. In: ICFP. pp. 462–475 (2016). https://doi.org/10.1145/2951913.2951926
40. Thiemann, P., Vasconcelos, V.T.: Label-dependent session types. Proc. ACM Program. Lang. **4**(POPL), 67:1–67:29 (2020). https://doi.org/10.1145/3371135

41. Toninho, B., Caires, L., Pfenning, F.: Dependent session types via intuitionistic linear type theory. In: PPDP. pp. 161–172. ACM (2011). https://doi.org/10.1145/2003476.2003499
42. Valiant, L.G.: Decision procedures for families of deterministic pushdown automata. Ph.D. thesis, University of Warwick (1973)
43. Valiant, L.G., Paterson, M.S.: Deterministic one-counter automata. Journal of Computer and System Sciences **10**(3), 340–350 (1975)
44. Vasconcelos, V.T.: Fundamentals of session types. Inf. Comput. **217**, 52–70 (2012). https://doi.org/10.1016/j.ic.2012.05.002
45. Yoshida, N., Deniélou, P., Bejleri, A., Hu, R.: Parameterised multiparty session types. In: FOSSACS. LNCS, vol. 6014, pp. 128–145. Springer (2010). https://doi.org/10.1007/978-3-642-12032-9_10

Complete and tractable machine-independent characterizations of second-order polytime

Emmanuel Hainry[1] , Bruce M. Kapron[2] , Jean-Yves Marion[1], and
Romain Péchoux[1✉]

[1] Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
[2] University of Victoria, Victoria, BC, Canada
{hainry,marion,pechoux}@inria.fr   bmkapron@uvic.ca

Abstract. The class of Basic Feasible Functionals BFF is the second-order counterpart of the class of first-order functions computable in polynomial time. We present several implicit characterizations of BFF based on a typed programming language of terms. These terms may perform calls to imperative procedures, which are not recursive. The type discipline has two layers: the terms follow a standard simply-typed discipline and the procedures follow a standard tier-based type discipline. BFF consists exactly of the second-order functionals that are computed by typable and terminating programs. The completeness of this characterization surprisingly still holds in the absence of lambda-abstraction. Moreover, the termination requirement can be specified as a completeness-preserving instance, which can be decided in time quadratic in the size of the program. As typing is decidable in polynomial time, we obtain the first tractable (*i.e.*, decidable in polynomial time), sound, complete, and implicit characterization of BFF, thus solving a problem opened for more than 20 years.

Keywords: Basic feasible functionals · Type 2 · Second-order · Polynomial time · Tiering · Safe recursion

1 Introduction

Motivations. The class of second-order functions computable in polynomial time was introduced and studied by Mehlhorn [27], building on an earlier proposal by Constable [10]. Kapron and Cook characterized this class using oracle Turing machines, giving it the name Basic Feasible Functionals (BFF):

Definition 1 ([19]). *A functional F is in* BFF*, if there are an oracle Turing machine M and a second-order polynomial[3] P such that M computes F in time bounded by $P(|f|, |x|)$, for any oracle f and any input x.[4]*

Since then, BFF was consensually considered as the natural extension to second-order of the well-known class of (first-order) polynomial time computable functions, FP. Notions of second-order polynomial time, while of intrinsic interest,

---

[3] Second-order polynomials are a type-2 analogue of ordinary polynomials.

[4] The size of an oracle f is a first-order function defined by $|f|(n) = \max_{|y| \le n} |f(y)|$.

have also been applied in a range of areas, including structural complexity theory [27], resource-bounded topology [29], complexity of total search problems [5], feasible real analysis [21], and verification [14].

Starting with Cobham's seminal work [9], there have been several attempts to provide *machine-independent* characterizations of complexity classes such as (P and) FP, that is, characterizations based on programming languages rather than on machines. Beyond the purely theoretical aspects, the practical interest of such characterizations is to be able to automatically guarantee that a program can be executed efficiently and in a secure environment. For these characterizations to hold, some restrictions are placed on a given programming language. They ensure that a program can be simulated by a Turing machine in polynomial time and, therefore, corresponds to a function in FP. This property is called *soundness*. Conversely, we would like any function in FP to be computable by a program satisfying the restrictions. This property is called (extensional) *completeness*. For automation to be possible, it is necessary that the characterizations studied be *tractable*; that is, decidable in polynomial time. Moreover, they should preferably not require a prior knowledge of the program complexity. One speaks then of *implicit* characterization insofar as the programmer does not have to know an explicit bound on the complexity of the analyzed programs.

In the first-order setting, different restrictions and techniques have been developed to characterize the complexity class FP. One can think, among others, of the safe recursion and ramified recursion techniques for function algebras [6,24], of interpretation methods for term rewrite systems [8], or of light and soft linear logics typing-discipline for lambda-calculi [15,4,3].

In the second-order setting, a machine-independent characterization of BFF was provided in [16]. This characterization uses the tier-based (*i.e.*, safe/ramified recursion-based) type discipline introduced in [26] on imperative programs for characterizing FP and can be restated as follows:

$$\text{BFF} = \lambda(\llbracket \text{ST} \rrbracket)_2,$$

$\llbracket \text{ST} \rrbracket$ denotes the set of functions computed by typable and terminating programs; λ denotes the lambda closure, that is, for a given set of functionals X, $\lambda(X)$ is the set of functionals denoted by simply-typed lambda-terms using constants in X; X_2 is the restriction of X to second-order functionals. Type inference for $\llbracket \text{ST} \rrbracket$ is fully automatic and can be performed in time cubic in the size of the analyzed program. However the above characterization has two main weaknesses:

– It is not complete: As $\llbracket \text{ST} \rrbracket \subsetneq \text{BFF}$, the typed language alone is not complete for BFF and a lambda closure (*i.e.*, $\lambda(X)$) of functionals computed by typable and terminating programs is required to ensure completeness.
– It is not tractable: the set $\llbracket \text{ST} \rrbracket$ relies on a termination assumption and it is unclear whether the characterization sill holds for a decidable or, for that matter, tractable termination technique.

Thus, providing a tractable, implicit, sound, and complete programming language for characterizing second-order polynomial time is still an open problem.

Contributions. Our paper provides the first solution to this problem, open for more than 20 years. To this end, we introduce a higher-order programming language and design a suitable typing discipline that address the two weaknesses described above. The lambda closure requirement for completeness is removed by designing a suitable programming language that consists of a layer of simply-typed terms that can perform calls to a layer of imperative and non-recursive procedures following a tier-based type discipline. This language allows for some restricted forms of procedure composition that are handled by the simply-typed terms and also allows for some restricted forms of oracle composition that are managed through the use of *closures*, syntactic elements playing the role of first-order abstractions with free variables. The termination criterion is specified as a completeness-preserving instance, called SCP_S, of a variant of Size Change Termination [23] introduced in [7] that can be checked in time quadratic in the size of the analyzed program. The main contributions of this paper are:

- A programming language in which typable (SAFE) and terminating (SN) programs capture exactly BFF (Theorem 2).
- A restriction to lambda-free programs, called rank-0 programs, such that typable ($SAFE_0$) and terminating (SN) programs still capture exactly BFF (Theorem 3); hence showing that lambda-abstraction only provides a syntactic relaxation, and corresponds to a conservative extension in terms of computable functions.
- A proof that type inference for SAFE is P-complete, and a type inference procedure running in time cubic in the program size for $SAFE_0$ (Theorem 4).
- A simple termination criterion, called SCP_S, preserving soundness and completeness of the characterizations both for SAFE and for $SAFE_0$ (Theorem 5) that can be checked in quadratic time.
- A complete characterization of BFF in terms of typable (SAFE) and terminating (SCP_S) programs (Theorem 6) that captures strictly more programs (Example 1) than [16], and is decidable in P-time.

The contributions of the paper are a non-trivial extension of existing works:

- The critical Programming Language design decisions rely mostly on the notion of *continuation*, that fixes a given oracle (closure) for once in the imperative layer. If the oracle were allowed to be updated inside a while loop, depending on some local value, then the language would yield a class beyond BFF, by computing exponential functions.
- It is a surprising result that the characterization of BFF still holds in the absence of lambda-abstraction as a basic construct of the proposed programming language, in particular that completeness does not rely on lambda-abstractions. This is an important improvement over [16] and [20], both of which required external lambda-closure.
- The type system is designed so that each procedure is typed exactly once. Types are not unique, but this does not prevent type inference from being polytime, as exhibiting one type is sufficient. The tractability of type inference is obtained by combining the tractabilities of type inference in the tier-based layer and in the simply-typed layer [25].

– The particular choice of the termination criterion SCP_S was made to show that termination can be specified as a tractable/feasible criterion while preserving completeness. This is also a new result. SCP_S may include nested loops (as described in [7]) and can be replaced by any termination criterion capturing the programs of our completeness proof. SCP_S was chosen for its tractability, but not only: the SCP criterion of [7] ensures termination by using an error state which breaks the control flow. This control-flow break damages the non-interference property needed for tier-based typing to guarantee time complexity bounds.

Leading example. The program ce of Example 1 will be our leading example, as it computes a function known to be in $BFF - [\![ST]\!]$ (*i.e.*, it computes a function in BFF and not in $[\![ST]\!]$, see [20]). This program will be shown to be in $SAFE_0$ and, consequently, in SAFE and to terminate with SCP_S.

Example 1 (Program ce). Let $\mathbb{W}$ be the set of words. Let the operator ε of arity 0 represent the empty word constant, let the operator != test whether or not its arguments are distinct, and let the operator pred remove the first letter of a word. The binary operator $\upharpoonright$ truncates and pads the size of its first operand to the size of its second operand plus 1. When the boxed variables X and y are fed with the inputs $f \in \mathbb{W} \to \mathbb{W}$ and $w \in \mathbb{W}$, respectively, program ce calls procedure KS in the term t. Program ce computes $|w|$ (*i.e.*, the size of the word w) bounded iterations of $f \circ f$ through the iteration of the assignment $z := X_2(z \upharpoonright w)$ in procedure KS. The bound on the output size of each iteration is computed by the first assignment $w := X_1(\varepsilon \upharpoonright \varepsilon)$ of KS and is equal to $f(1)$ (that is, $f([\![\varepsilon \upharpoonright \varepsilon]\!])$), with $[\![\varepsilon \upharpoonright \varepsilon]\!]=1$; $[\![e]\!]$ being the result of evaluating the expression e.

```
box [X,y] in
    declare
        KS(X₁, X₂, v) {
            var w,z;

            w   :=  X₁(ε ↾ ε);          } Statement st

            z   :=  ε;
            while (v != ε) {
                v   :=  pred(v);        } Statement st'
                z   :=  X₂(z ↾ w)
            }
            return z
        }

    in call KS({x → X @ x}, {x → X @ (X @ x)}, y) } Term t
```

Procedure p / ce

Related work. Several tools providing machine-independent characterizations of distinct complexity classes have been developed in the field of Implicit Com-

putational Complexity (ICC). Most of these tools are restricted to the study of first-order complexity classes. Whereas light logic approaches can deal with programs at higher types, their applications are restricted to first-order complexity classes such as FP [15,4,3]. Interpretation methods were extended to higher-order polynomials in [2] to study FP and adapted in [13] and [17] to characterize BFF. However, these characterizations are not decidable as they require checking of second-order polynomial inequalities. [12] and [18] study characterizations of BFF in terms of a simple imperative programming language that enforces an explicit external bound on the size of oracle outputs within loops. The corresponding restriction is not implicit by nature and is impractical from a programming perspective as the size of oracle outputs cannot be predicted. In this paper, the bound is programmer friendly because it is implicit and it only constrains the size of the oracle input.

2   A second-order language with imperative procedures

The syntax and semantics of the programming language designed to capture the complexity class BFF are introduced in this section. Programs of this language consist in second-order terms in which imperative procedures are declared and called. These procedures have no global variables, are not recursive, and their parameters can be of order 1 (oracles) or 0 (local variables). Oracles are in read-only mode: they cannot be declared and, hence, modified inside a procedure. Oracles can only be composed at the term level through the use of closures, first-order abstractions that can be passed as parameters in a procedure call.

Syntax. When we refer to a *type-i* syntactic element e (a variable, an expression, a statement, ...), for $i \in \mathbb{N}$, we implicitly assume that the element e denotes some function of order i over words as basic type. We will sometimes write e^i in order to make the order explicit. For example, e^0 denotes a word. This notion will be formally defined in Section 3. Let $\bar{e}$ denote a (possibly empty) tuple of n elements $e_1, \ldots, e_n$, where n is given by the context. Let $|\bar{e}|$ denote the length of tuple $\bar{e}$, *i.e.*, $|\bar{e}| \triangleq n$. Let π_i, $i \le |\bar{e}|$, denote the projectors on tuples, *i.e.*, $\pi_i(\bar{e}) \triangleq e_i$.

Let $\mathbb{V}$ be a set of variables that can be split into three disjoint sets $\mathbb{V} = \mathbb{V}_0 \uplus \mathbb{V}_1 \uplus \mathbb{V}_{\ge 2}$. The type-0 variables in $\mathbb{V}_0$ will be denoted by lower case letters $\mathtt{x}, \mathtt{y}, \ldots$ and the type-1 variables in $\mathbb{V}_1$ will be denoted by upper case letters $\mathtt{X}, \mathtt{Y}, \ldots$ Variables in $\mathbb{V}$ of arbitrary type will be denoted by letters $\mathtt{a}, \mathtt{b}, \mathtt{a}_1, \mathtt{a}_2, \ldots$.

Let $\mathbb{O}$ be a set of (type-1) operators $\mathtt{op}$ of fixed arity $ar(\mathtt{op})$ that will be used both in infix and prefix notations for notational convenience and that are always fully applied, *i.e.*, applied to a number $ar(\mathtt{op})$ of operands.

The programs are defined by the grammar of Figure 1. A program is either a term $\mathtt{t}^0$, a *procedure declaration* declare p in prog, or the declaration of a *boxed variable* a, called *box*, followed by a program: box [a] in prog. Boxed variables will represent the program inputs.

In Figure 1, there are three constructor/destructor pairs for abstraction and application; each of them playing a distinct rôle:

- λa.t and $t_1 @ t_2$ are the standard abstraction and application on terms.
- The application of a type-1 variable X within a statement is called an *oracle call*, written $X(e_1 \upharpoonright e_2)$, where e_1 is called the *input data*, e_2 is called the *input bound*, and $e_1 \upharpoonright e_2$ is called the *input*. The corresponding abstraction is called a *closure*, a type-1 map of the shape $\{x \to t^0\}$, where the type-0 term t may contain free variables.
- A procedure declaration $P(\overline{X}, \overline{x})\{[\text{var } \overline{y};] \text{ st return } x\}$ is an abstraction that computes type-2 functions mapping type-1 and type-0 inputs ($\overline{X}$ and $\overline{x}$, respectively) to a type-0 output (x). The *procedure calls* of the shape $\text{call } P(\overline{c}, \overline{t^0})$ are the corresponding applications and take closures as type-1 inputs and terms as type-0 inputs.

Type-0 var.	$x, y, u, v, w \dots \in$	$\mathbb{V}_0$	
Type-1 var.	$X, Y, X_1, X_2, \dots \in$	$\mathbb{V}_1$	
Variables	$a, b, a_1, a_2, \dots \in$	$\mathbb{V} = \mathbb{V}_0 \uplus \mathbb{V}_1 \uplus \mathbb{V}_{\geq 2}$	
Operators	$\text{op}, \upharpoonright$	$\in$	$\mathbb{O}$
Expressions	$e, e_1, e_2, \dots$	$::= x \mid \text{op}(\overline{e}) \mid X(e_1 \upharpoonright e_2)$	
Statements	$st, st_1, \dots$	$::= \text{skip} \mid x := e \mid st_1; st_2 \mid \text{if}(e)\{st_1\} \text{ else } \{st_2\}$	
		$\mid \text{while}(e)\{st\}$	
Procedures	$p, p_1, p_2, \dots$	$::= P(\overline{X}, \overline{x})\{[\text{var } \overline{y};] \text{ st return } x\}$	
Terms	$t, t_1, t_2, \dots$	$::= a \mid \lambda a.t \mid t_1 @ t_2 \mid \text{call } P(\overline{c}, \overline{t^0})$	
Closures	$c, c_1, c_2, \dots$	$::= \{x \to t^0\}$	
Programs	prog	$::= t^0 \mid \text{declare } \overline{p} \text{ in prog} \mid \text{box } [\overline{a}] \text{ in prog}$	

Fig. 1: Syntax of type-2 programs

For some syntactic element e of the language, let $\mathbb{V}(e) \subseteq \mathbb{V}$ be the set of all variables occurring in e. A variable is free if it is not under the scope of an abstraction and it is not boxed. A program is *closed* if it has no free variable.

For a given procedure declaration $p = P(\overline{X}, \overline{x})\{[\text{var } \overline{y};] \text{ st return } x\}$, define the *procedure name* of p by $n(p) \triangleq P$. Define also $\text{body}(P) \triangleq st$, $\text{local}(P) \triangleq \{\overline{y}\}$, and $\text{param}(P) \triangleq \{\overline{X}, \overline{x}\}$. $\text{body}(P)$ is called the *body* of procedure P. The variables in $\text{local}(P)$ are called *local variables* and the variables in $\text{param}(P)$ are called *parameters*. Finally, define $\text{Proc}(t)$ (and $\text{Proc}(\text{prog})$) to be the set of procedure names that are called within the term t (respectively program prog).

Throughout the paper, we will restrict our study to closed programs in *normal form*. These consist of programs with no free variable that can be written as follows $\text{box } [\overline{X}, \overline{x}] \text{ in declare } \overline{p} \text{ in t}$, for some term t such that the following well-formedness conditions hold: (i) There are no name clashes. (ii) There are no free variables in a given procedure. (iii) Any procedure call has a corresponding procedure declaration. A closed program in normal form of the shape $\text{box } [\overline{X}, \overline{x}] \text{ in declare } \overline{p} \text{ in t}^0$, for some type-0 term t, will compute a type-2

functional. The typing discipline presented in Section 3 will restrict the analysis to such programs.

Operational semantics. Let $\mathbb{W} = \Sigma^*$ be the set of words over a finite alphabet Σ such that $\{0,1\} \subseteq \Sigma$. The symbol ϵ denotes the empty word. The length of a word w is denoted $|w|$. Given two words w and v in $\mathbb{W}$ let $v.w$ denote the concatenation of v and w. For a given symbol $a \in \Sigma$, let a^n be defined inductively by $a^0 = \epsilon$ and $a^{n+1} = a.a^n$. Let $\trianglelefteq$ be the sub-word relation over $\mathbb{W}$, which is defined by $v \trianglelefteq w$, if $\exists u, u' \in \mathbb{W}$, $w = u.v.u'$.

For a given word $w \in \mathbb{W}$ and an integer n, let $w_{\upharpoonright n}$ be the word obtained by truncating w to its first $\min(n, |w|)$ symbols and then padding with a word of the form 10^k to obtain a word of size exactly $n + 1$. For example, $1001_{\upharpoonright 0} = 1$, $1001_{\upharpoonright 1} = 11$, $1001_{\upharpoonright 2} = 101$, and $1001_{\upharpoonright 6} = 1001100$. Define $\forall v, w \in \mathbb{W}$, $[\![\uparrow]\!](v, w) = v_{\upharpoonright |w|}$. Padding ensures that $|[\![\uparrow]\!](v, w)| = |w| + 1$. The syntax of programs enforces that oracle calls are always performed on input data padded by the input bound and, consequently, oracle calls are always performed on input data whose size does not exceed the size of the input bound plus one.

A total function $[\![\mathtt{op}]\!] : \mathbb{W}^{ar(\mathtt{op})} \to \mathbb{W}$ is associated with each operator $\mathtt{op}$ of arity $ar(\mathtt{op})$. Constants may be viewed as operators of arity zero. We define two classes of operators called neutral and positive depending on the total function they compute. This categorization of operators will be used by our type system as the admissible types for operators will depend on their category.

An operator $\mathtt{op}$, computing the total function $[\![\mathtt{op}]\!] : \mathbb{W}^{ar(\mathtt{op})} \to \mathbb{W}$, is:

- *neutral* if:
    1. either $[\![\mathtt{op}]\!]$ is constant, *i.e.*, $ar(\mathtt{op}) = 0$;
    2. $[\![\mathtt{op}]\!] : \mathbb{W}^{ar(\mathtt{op})} \to \{0,1\}$ is a predicate;
    3. or $\forall \overline{w} \in \mathbb{W}^{ar(\mathtt{op})}$, $\exists i \leq ar(\mathtt{op})$, $[\![\mathtt{op}]\!](\overline{w}) \trianglelefteq w_i$;
- *positive* if $\exists c_{\mathtt{op}} \in \mathbb{N}$ s.t.: $\forall \overline{w} \in \mathbb{W}^{ar(\mathtt{op})}$, $|[\![\mathtt{op}]\!](\overline{w})| \leq \max_{1 \leq i \leq ar(\mathtt{op})} |w_i| + c_{\mathtt{op}}$.

As neutral operators are always positive, in the sequel, we reserve the name positive for those operators that are positive but not neutral.

In what follows, let $f, g, \ldots$ denote total functions in $\mathbb{W} \to \mathbb{W}$. A *store* μ consists of the disjoint union of a map μ_0 from $\mathbb{V}_0$ to $\mathbb{W}$ and a map μ_1 from $\mathbb{V}_1$ to total functions in $\mathbb{W} \to \mathbb{W}$. For $i \in \{0,1\}$, μ_i is called a *type-i store*. Let $dom(\mu)$ be the domain of the store μ. Let $\mu[\mathtt{x} \leftarrow w]$ denote the store μ' satisfying $\mu'(b) = \mu(b)$, for all $b \neq \mathtt{x}$, and $\mu'(\mathtt{x}) = w$. This notation is extended naturally to type-1 variables $\mu[\mathtt{X} \leftarrow f]$ and to sequences of variables $\mu[\overline{\mathtt{x}} \leftarrow \overline{w}, \overline{\mathtt{X}} \leftarrow \overline{f}]$. Finally, let $\mu_\emptyset$ denote the empty store.

Let $\downarrow$ denote the standard big-step call-by-name reduction relation on terms defined by: if $\mathtt{t}_1 \downarrow \lambda\mathtt{a}.\mathtt{t}$ and $\mathtt{t}\{\mathtt{t}_2/\mathtt{a}\} \downarrow v$ then $\mathtt{t}_1 @ \mathtt{t}_2 \downarrow v$, where $\{\mathtt{t}_2/\mathtt{a}\}$ is the standard substitution and where v can be a type-0 variable $\mathtt{x}$, a lambda-abstraction $\lambda\mathtt{a}.\mathtt{t}$, a type-1 variable application $\mathtt{X}@\mathtt{t}$, or a procedure call $\mathtt{call\ P}(\overline{\mathtt{c}}, \overline{\mathtt{t}^0})$.

A *continuation* is a map ϕ from $\mathbb{V}_1$ to $\mathtt{Closures}$, *i.e.*, $\phi(\mathtt{X}) = \{\mathtt{x} \to \mathtt{t}^0\}$ for some type-1 variable $\mathtt{X}$, some type-0 variable $\mathtt{x}$, and some type-0 term $\mathtt{t}^0$. Let $\overline{\mathtt{X}} \mapsto \overline{\mathtt{c}}$ with $|\overline{\mathtt{X}}| = |\overline{\mathtt{c}}|$, be a notation for the continuation mapping each $\mathtt{X}_i \in \mathbb{V}_1$ to the closure $\mathtt{c}_i$.

Given a set of procedures σ, a store μ, and a continuation ϕ, we define three distinct kinds of judgments: $(\sigma, \mu, \phi, \mathrm{e}) \rightarrow_{\mathrm{exp}} w$ for expressions, $(\sigma, \mu, \phi, \mathrm{st}) \rightarrow_{\mathrm{st}} \mu'$ for statements, and $(\sigma, \mu, \mathrm{prog}) \rightarrow_{\mathrm{env}} w$ for programs. The big-step operational semantics of the language is described in Figure 2.

A program $\mathrm{prog} = \mathrm{box} \ [\overline{\mathrm{X}}, \overline{\mathrm{x}}] \ \mathrm{in} \ \mathrm{declare} \ \overline{\mathrm{p}} \ \mathrm{in} \ \mathrm{t}^0$ computes the second-order partial functional $[\![\mathrm{prog}]\!] \in (\mathbb{W} \rightarrow \mathbb{W})^{|\overline{\mathrm{X}}|} \rightarrow \mathbb{W}^{|\overline{\mathrm{x}}|} \rightarrow \mathbb{W}$, defined by:

$$[\![\mathrm{prog}]\!](\overline{f}, \overline{w}) = w \ \text{iff} \ (\emptyset, \mu_\emptyset[\overline{\mathrm{x}} \leftarrow \overline{w}, \overline{\mathrm{X}} \leftarrow \overline{f}], \mathrm{prog}) \rightarrow_{\mathrm{env}} w.$$

In the special case where $[\![\mathrm{prog}]\!]$ is a total function, the program prog is said to be terminating (strongly normalizing). We will denote by SN the set of terminating programs. For a given set of programs S, let $[\![S]\!]$ denote the set of functions computed by programs in S. Formally, $[\![S]\!] = \{[\![\mathrm{prog}]\!] \mid \mathrm{prog} \in S\}$.

Example 2. Consider the program ce provided in Example 1, where:

$$[\![\varepsilon]\!]() = \epsilon \in \mathbb{W}, \quad [\![!=]\!](w, v) = \begin{cases} 1 & \text{if } v = w \\ 0 & \text{otherwise,} \end{cases} \quad [\![\mathrm{pred}]\!](v) = \begin{cases} \epsilon & \text{if } v = \epsilon \\ u & \text{if } v = a.u \end{cases}$$

Program ce is in normal form and computes the second-order functional $F \colon (\mathbb{W} \rightarrow \mathbb{W}) \rightarrow \mathbb{W} \rightarrow \mathbb{W}$ defined by: $\forall f \in \mathbb{W} \rightarrow \mathbb{W}, \forall w \in \mathbb{W}, \ F(f)(w) = F_{|w|}(f)$, where F_n is defined recursively as $F_0(f) = \epsilon$ and $F_{n+1}(f) = (f \circ f)([\![\restriction]\!](F_n(f), f(1)) = (f \circ f)(F_n(f)_{\restriction|f(1)|})$. That is a function that composes the input function $2|w|$ times f while restricting its input to a fixed size $|f(1)|$ every other iteration. Indeed, $[\![\varepsilon]\!]() = \epsilon$ and $[\![\restriction]\!](\epsilon, \epsilon) = \epsilon_{\restriction|\epsilon|} = 1$. Consequently, the oracle bound w in the oracle call $\mathrm{X}_2(\mathrm{z} \restriction \mathrm{w})$ is bound to value $f(1)$ in the store by the statement $\mathrm{w} := \mathrm{X}_1(\varepsilon \restriction \varepsilon)$.

Observe that the operators ε, $!=$ and pred are all neutral. An example of positive operator can be given by the successor operators defined by $[\![\mathrm{suc}_i]\!](v) = i.v$, for $i \in \{0, 1\}$. These operators are positive since $|[\![\mathrm{suc}_i]\!](v)| = |i.v| = |v| + 1$.

3 Type system

Tiers and typing environments. Let $\mathbb{W}$ be the type of words in $\mathbb{W}$. *Simple types* over $\mathbb{W}$ are defined inductively by $\mathrm{T}, \mathrm{T}', \ldots ::= \mathbb{W} \mid \mathrm{T} \rightarrow \mathrm{T}$. Let $\mathcal{T}_{\mathbb{W}}$ be the set of simple types over $\mathbb{W}$. The order of a simple type in $\mathcal{T}_{\mathbb{W}}$ is defined inductively by: $\mathrm{ord}(\mathrm{T}) = 0$, if $\mathrm{T} = \mathbb{W}$, and $\mathrm{ord}(\mathrm{T}) = \max(1 + \mathrm{ord}(\mathrm{T}_1), \mathrm{ord}(\mathrm{T}_2))$, if $\mathrm{T} = \mathrm{T}_1 \rightarrow \mathrm{T}_2$.

Tiers are elements of the totally ordered set $(\mathbf{N}, \preceq, \mathbf{0}, \vee, \wedge)$, where $\mathbf{N} = \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \ldots\}$ is the set of natural numbers, $\preceq$ is the standard ordering on integers, and $\vee$ and $\wedge$ are the max and min operators over integers. Let $\prec$ be defined by $\prec := \preceq \cap \neq$. We use the symbols $\mathbf{k}, \mathbf{k}', \ldots, \mathbf{k}_1, \mathbf{k}_2, \ldots$ to denote tier variables. For a finite set of tiers, $\{\mathbf{k}_1, \ldots, \mathbf{k}_n\}$, let $\vee_{i=1}^n \mathbf{k}_i$ ($\wedge_{i=1}^n \mathbf{k}_i$, respectively) denote $\mathbf{k}_1 \vee \ldots \vee \mathbf{k}_n$ ($\mathbf{k}_1 \wedge \ldots \wedge \mathbf{k}_n$, respectively). A *first-order tier* is of the shape $\mathbf{k}_1 \rightarrow \ldots \rightarrow \mathbf{k}_n \rightarrow \mathbf{k}'$, with $\mathbf{k}_i, \mathbf{k}' \in \mathbf{N}$.

A *simple typing environment* $\Gamma_{\mathbb{W}}$ is a finite partial map from $\mathbb{V}$ to $\mathcal{T}_{\mathbb{W}}$, which assigns simple types to variables.

$$\frac{}{(\sigma, \mu, \phi, \mathtt{x}) \to_{\mathrm{exp}} \mu(\mathtt{x})} \ (\mathrm{Var}) \qquad \frac{(\sigma, \mu, \phi, \overline{\mathtt{e}}) \to_{\mathrm{exp}} \overline{w}}{(\sigma, \mu, \phi, \mathtt{op}(\overline{\mathtt{e}})) \to_{\mathrm{exp}} [\![\mathtt{op}]\!](\overline{w})} \ (\mathrm{Op})$$

$$\frac{(\sigma, \mu, \phi, \mathtt{e}_1) \to_{\mathrm{exp}} v \qquad (\sigma, \mu, \phi, \mathtt{e}_2) \to_{\mathrm{exp}} u \qquad \phi(\mathtt{X}) = \{\mathtt{x} \to \mathtt{t}^0\} \qquad (\sigma, \mu[\mathtt{x} \leftarrow [\![]\!](v, u)], \mathtt{t}^0) \to_{\mathrm{env}} w}{(\sigma, \mu, \phi, \mathtt{X}(\mathtt{e}_1 \upharpoonright \mathtt{e}_2)) \to_{\mathrm{exp}} w} \ (\mathrm{Orc})$$

<center>(a) Expressions</center>

$$\frac{}{(\sigma, \mu, \phi, \mathtt{skip}) \to_{\mathrm{st}} \mu} \ (\mathrm{Skip}) \qquad \frac{(\sigma, \mu, \phi, \mathtt{st}_1) \to_{\mathrm{st}} \mu' \qquad (\sigma, \mu', \phi, \mathtt{st}_2) \to_{\mathrm{st}} \mu''}{(\sigma, \mu, \phi, \mathtt{st}_1;\ \mathtt{st}_2) \to_{\mathrm{st}} \mu''} \ (\mathrm{Seq})$$

$$\frac{(\sigma, \mu, \phi, \mathtt{e}) \to_{\mathrm{exp}} w}{(\sigma, \mu, \phi, \mathtt{x} := \mathtt{e}) \to_{\mathrm{st}} \mu[\mathtt{x} \leftarrow w]} \ (\mathrm{Asg})$$

$$\frac{(\sigma, \mu, \phi, \mathtt{e}) \to_{\mathrm{exp}} w \qquad (\sigma, \mu, \phi, \mathtt{st}_w) \to_{\mathrm{st}} \mu' \qquad w \in \{0, 1\}}{(\sigma, \mu, \phi, \mathtt{if}(\mathtt{e})\{\mathtt{st}_1\}\ \mathtt{else}\ \{\mathtt{st}_0\}) \to_{\mathrm{st}} \mu'} \ (\mathrm{Cond})$$

$$\frac{(\sigma, \mu, \phi, \mathtt{e}) \to_{\mathrm{exp}} 0}{(\sigma, \mu, \phi, \mathtt{while}(\mathtt{e})\{\mathtt{st}\}) \to_{\mathrm{st}} \mu} \ (\mathrm{Wh}_0)$$

$$\frac{(\sigma, \mu, \phi, \mathtt{e}) \to_{\mathrm{exp}} 1 \qquad (\sigma, \mu, \phi, \mathtt{st};\ \mathtt{while}(\mathtt{e})\{\mathtt{st}\}) \to_{\mathrm{st}} \mu'}{(\sigma, \mu, \phi, \mathtt{while}(\mathtt{e})\{\mathtt{st}\}) \to_{\mathrm{st}} \mu'} \ (\mathrm{Wh}_1)$$

<center>(b) Statements</center>

$$\frac{\mathtt{t}^0 \downarrow \mathtt{x}}{(\sigma, \mu, \mathtt{t}^0) \to_{\mathrm{env}} \mu(\mathtt{x})} \ (\mathrm{TVar}) \qquad \frac{\mathtt{t}^0 \downarrow \mathtt{X@t}_1^0 \qquad (\sigma, \mu, \mathtt{t}_1^0) \to_{\mathrm{env}} w}{(\sigma, \mu, \mathtt{t}^0) \to_{\mathrm{env}} \mu(\mathtt{X})(w)} \ (\mathrm{OA})$$

$$\frac{\mathtt{t}^0 \downarrow \mathtt{call}\ \mathtt{P}(\overline{\mathtt{c}}, \overline{\mathtt{t}^0}) \qquad (\sigma, \mu, \overline{\mathtt{t}^0}) \to_{\mathrm{env}} \overline{w} \qquad (\sigma, \mu[\overline{\mathtt{x}} \leftarrow \overline{w}, \overline{\mathtt{y}} \leftarrow \overline{\epsilon}], \overline{\mathtt{X}} \mapsto \overline{\mathtt{c}}, \mathtt{st}) \to_{\mathrm{st}} \mu'}{(\sigma \cup \{\mathtt{P}(\overline{\mathtt{X}}, \overline{\mathtt{x}})\{\mathtt{var}\ \overline{\mathtt{y}};\ \mathtt{st}\ \mathtt{return}\ \mathtt{z}\}\}, \mu, \overline{\mathtt{t}^0}) \to_{\mathrm{env}} \mu'(\mathtt{z})} \ (\mathrm{Call})$$

<center>(c) Type-0 terms</center>

$$\frac{(\sigma \cup \{\mathtt{p}\}, \mu, \mathtt{prog}) \to_{\mathrm{env}} w}{(\sigma, \mu, \mathtt{declare}\ \mathtt{p}\ \mathtt{in}\ \mathtt{prog}) \to_{\mathrm{env}} w} \ (\mathrm{Dec}) \qquad \frac{(\sigma, \mu, \mathtt{prog}) \to_{\mathrm{env}} w \qquad \mathtt{a} \in dom(\mu)}{(\sigma, \mu, \mathtt{box}\ [\mathtt{a}]\ \mathtt{in}\ \mathtt{prog}) \to_{\mathrm{env}} w} \ (\mathrm{Box})$$

<center>(d) Programs</center>

<center>Fig. 2: Big step operational semantics</center>

A *variable typing environment* Γ is a finite partial map from $\mathbb{V}_0$ to $\mathbf{N}$, which assigns single tiers to type-0 variables.

An *operator typing environment* Δ is a mapping that associates to some operator op and some tier $\mathbf{k} \in \mathbf{N}$ a set of admissible first-order tiers $\Delta(\text{op})(\mathbf{k})$ of the shape $\mathbf{k}_1 \to \ldots \to \mathbf{k}_{ar(\text{op})} \to \mathbf{k}'$.

A *procedure typing environment* Ω is a mapping that associates to each procedure name P a pair $\langle \Gamma, \overline{\mathbf{k}} \rangle$ consisting of a variable typing environment Γ and a triplet of tiers $\overline{\mathbf{k}}$. Let $\Omega_i \triangleq \pi_i(\Omega)$, $i \in \{1, 2\}$.

Let $dom(\Gamma)$, $dom(\Gamma_{\mathtt{W}})$, $dom(\Delta)$, and $dom(\Omega)$ denote the sets of variables typed by Γ and $\Gamma_{\mathtt{W}}$, the set of operators typed by Δ, and the set of procedures typed by Ω, respectively.

For a procedure typing environment Ω, it will be assumed that for every $\mathtt{P} \in dom(\Omega)$, $\mathtt{param}(\mathtt{P}) \cup \mathtt{local}(\mathtt{P}) \subseteq dom(\Omega_1(\mathtt{P}))$.

While operator and procedure typing environments are global, *i.e.*, defined for the whole program, variable typing environments are local, *i.e.*, relative to the procedure under analysis. In a program typing judgment, the simple typing environment can be viewed as the typing environment for the main program.

Typing judgments and type system. The typing discipline includes two distinct kinds of typing judgments: *Procedure typing judgments* $\Gamma, \Delta \vdash o : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})$ and *Term typing judgments* $\Gamma_{\mathtt{W}}, \Omega, \Delta \vdash \mathtt{prog} : \mathtt{T}$, with $\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out} \in \mathbf{N}$, $o \in \mathtt{Expressions} \cup \mathtt{Statements}$, and $\mathtt{T} \in \mathcal{T}_{\mathtt{W}}$.

The meaning of the procedure typing judgment is that the *expression tier* (or *statement tier*) is $\mathbf{k}$, the *innermost tier* is $\mathbf{k}_{in}$, and the *outermost tier* is $\mathbf{k}_{out}$. The innermost (resp. outermost) tier is the tier of the innermost (resp. outermost) while loop guard where the expression or statement is located. The meaning of term typing judgments is that the program $\mathtt{prog}$ is of simple type $\mathtt{T}$ under the operator typing environment Δ, the procedure typing environment Ω and the simple typing environment $\Gamma_{\mathtt{W}}$.

A program $\mathtt{prog}$ (or term $\mathtt{t}$) is of *type-i*, if $\Gamma_{\mathtt{W}}, \Omega, \Delta \vdash \mathtt{prog} : \mathtt{T}$ ($\Gamma_{\mathtt{W}}, \Omega, \Delta \vdash \mathtt{t} : \mathtt{T}$) can be derived for some typing environments and type $\mathtt{T}$ s.t. $\mathtt{ord}(\mathtt{T}) = i$.

The type system for the considered programming language is provided in Figure 3. A *well-typed program* is a program that can be given the type $(\overline{\mathtt{W}} \to \mathtt{W}) \to \overline{\mathtt{W}} \to \mathtt{W}$, *i.e.*, the judgment $\Gamma_{\mathtt{W}}, \Omega, \Delta \vdash \mathtt{prog} : (\overline{\mathtt{W}} \to \mathtt{W}) \to \overline{\mathtt{W}} \to \mathtt{W}$ can be derived for the environments $\Gamma_{\mathtt{W}}, \Omega, \Delta$. Consequently, a well-typed program is a type-*i* program, for some $i \leq 2$, computing a functional.

For a given typing judgment j, a *typing derivation* $\pi \rhd j$ is a tree whose root is the (procedure or term) typing judgment j and whose children are obtained by applications of the typing rules of Figure 3. The name π will be used alone whenever mentioning the root of a typing derivation is not explicitly needed. A typing sub-derivation of a typing derivation π is a subtree of π.

Intuitions. We now give some brief intuition to the reader on the type discipline in the particular case where exactly two tiers, $\mathbf{0}$ and $\mathbf{1}$, are involved. The type system splits program variables, expressions, and statements between the two disjoint *tiers*:

$$\frac{\Gamma(\mathtt{x}) = \mathbf{k}}{\Gamma, \Delta \vdash \mathtt{x} : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})} \ \ (\text{E-VAR})$$

$$\frac{\mathbf{k}_1 \to \cdots \to \mathbf{k}_{|\bar{\mathbf{e}}|} \to \mathbf{k} \in \Delta(\mathtt{op})(\mathbf{k}_{in}) \qquad \forall i \leq |\bar{\mathbf{e}}|, \ \Gamma, \Delta \vdash \mathbf{e}_i : (\mathbf{k}_i, \mathbf{k}_{in}, \mathbf{k}_{out})}{\Gamma, \Delta \vdash \mathtt{op}(\bar{\mathbf{e}}) : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})} \ \ (\text{E-OP})$$

$$\frac{\Gamma, \Delta \vdash \mathbf{e}_1 : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out}) \qquad \Gamma, \Delta \vdash \mathbf{e}_2 : (\mathbf{k}_{out}, \mathbf{k}_{in}, \mathbf{k}_{out}) \qquad \mathbf{k} \prec \mathbf{k}_{in} \wedge \mathbf{k} \preceq \mathbf{k}_{out}}{\Gamma, \Delta \vdash \mathtt{X}(\mathbf{e}_1 \upharpoonright \mathbf{e}_2) : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})} \ \ (\text{E-OR})$$

$$\frac{}{\Gamma, \Delta \vdash \mathtt{skip} : (\mathbf{0}, \mathbf{k}_{in}, \mathbf{k}_{out})} \ \ (\text{S-SK}) \qquad\qquad \frac{\Gamma, \Delta \vdash \mathtt{st} : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})}{\Gamma, \Delta \vdash \mathtt{st} : (\mathbf{k+1}, \mathbf{k}_{in}, \mathbf{k}_{out})} \ \ (\text{S-SUB})$$

$$\frac{\Gamma, \Delta \vdash \mathtt{st}_1 : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out}) \qquad \Gamma, \Delta \vdash \mathtt{st}_2 : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})}{\Gamma, \Delta \vdash \mathtt{st}_1; \ \mathtt{st}_2 : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})} \ \ (\text{S-SEQ})$$

$$\frac{\Gamma, \Delta \vdash \mathtt{x} : (\mathbf{k}_1, \mathbf{k}_{in}, \mathbf{k}_{out}) \qquad \Gamma, \Delta \vdash \mathbf{e} : (\mathbf{k}_2, \mathbf{k}_{in}, \mathbf{k}_{out}) \qquad \mathbf{k}_1 \preceq \mathbf{k}_2}{\Gamma, \Delta \vdash \mathtt{x} := \mathbf{e} : (\mathbf{k}_1, \mathbf{k}_{in}, \mathbf{k}_{out})} \ \ (\text{S-ASG})$$

$$\frac{\Gamma, \Delta \vdash \mathbf{e} : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out}) \qquad \Gamma, \Delta \vdash \mathtt{st}_1 : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out}) \qquad \Gamma, \Delta \vdash \mathtt{st}_0 : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})}{\Gamma, \Delta \vdash \mathtt{if}(\mathbf{e})\{\mathtt{st}_1\} \ \mathtt{else} \ \{\mathtt{st}_0\} : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})} \ \ (\text{S-CND})$$

$$\frac{\Gamma, \Delta \vdash \mathbf{e} : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}) \qquad \Gamma, \Delta \vdash \mathtt{st} : (\mathbf{k}, \mathbf{k}, \mathbf{k}) \qquad \mathbf{1} \preceq \mathbf{k}}{\Gamma, \Delta \vdash \mathtt{while}(\mathbf{e})\{\mathtt{st}\} : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{0})} \ \ (\text{S-WINIT})$$

$$\frac{\Gamma, \Delta \vdash \mathbf{e} : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out}) \qquad \Gamma, \Delta \vdash \mathtt{st} : (\mathbf{k}, \mathbf{k}, \mathbf{k}_{out}) \qquad \mathbf{1} \preceq \mathbf{k} \preceq \mathbf{k}_{out}}{\Gamma, \Delta \vdash \mathtt{while}(\mathbf{e})\{\mathtt{st}\} : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})} \ \ (\text{S-WH})$$

(a) Tier-based typing rules for expressions and statements

$$\frac{\Gamma_{\mathtt{W}}, \Omega, \Delta \vdash \overline{\mathtt{X}} : \overline{\mathtt{W} \to \mathtt{W}} \qquad \Gamma_{\mathtt{W}}, \Omega, \Delta \vdash \overline{\mathtt{x}}, \overline{\mathtt{y}}, \mathtt{x} : \overline{\mathtt{W}}}{\Gamma_{\mathtt{W}}, \Omega, \Delta \vdash \mathtt{P}(\overline{\mathtt{X}}, \overline{\mathtt{x}})\{[\mathtt{var} \ \overline{\mathtt{y}};] \ \mathtt{st} \ \mathtt{return} \ \mathtt{x}\} : (\overline{\mathtt{W} \to \mathtt{W}}) \to \overline{\mathtt{W}} \to \mathtt{W}} \ \ (\text{PR-DEC})$$

$$\frac{\Gamma_{\mathtt{W}}, \Omega, \Delta \vdash \mathtt{P}(\overline{\mathtt{X}}, \overline{\mathtt{x}})\{\ldots\} : (\overline{\mathtt{W} \to \mathtt{W}}) \to \overline{\mathtt{W}} \to \mathtt{W} \qquad \Gamma_{\mathtt{W}}, \Omega, \Delta \vdash \overline{\mathtt{c}} : \overline{\mathtt{W} \to \mathtt{W}} \qquad \Gamma_{\mathtt{W}}, \Omega, \Delta \vdash \overline{\mathtt{t}} : \overline{\mathtt{W}}}{\Gamma_{\mathtt{W}}, \Omega, \Delta \vdash \mathtt{call} \ \mathtt{P}(\overline{\mathtt{c}}, \overline{\mathtt{t}}) : \mathtt{W}} \ \ (\text{P-CALL})$$

$$\frac{\Gamma_{\mathtt{W}}(\mathtt{a}) = \mathtt{T}}{\Gamma_{\mathtt{W}}, \Omega, \Delta \vdash \mathtt{a} : \mathtt{T}} \ \ (\text{P-VAR}) \qquad \frac{\Gamma_{\mathtt{W}} \uplus \{\mathtt{a} : \mathtt{T}\}, \Omega, \Delta \vdash \mathtt{t} : \mathtt{T}'}{\Gamma_{\mathtt{W}}, \Omega, \Delta \vdash \lambda \mathtt{a}.\mathtt{t} : \mathtt{T} \to \mathtt{T}'} \ \ (\text{P-ABS})$$

$$\frac{\Gamma_{\mathtt{W}}, \Omega, \Delta \vdash \mathtt{t}_1 : \mathtt{T} \to \mathtt{T}' \qquad \Gamma_{\mathtt{W}}, \Omega, \Delta \vdash \mathtt{t}_2 : \mathtt{T}}{\Gamma_{\mathtt{W}}, \Omega, \Delta \vdash \mathtt{t}_1 @ \mathtt{t}_2 : \mathtt{T}'} \ \ (\text{P-APP})$$

$$\frac{\Gamma_{\mathtt{W}}, \Omega, \Delta \vdash \mathtt{prog} : \mathtt{T} \qquad \Gamma, \Delta \vdash \mathtt{body}(\mathtt{n}(\mathtt{p})) : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out}) \qquad \Omega(\mathtt{n}(\mathtt{p})) = \langle \Gamma, (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out}) \rangle}{\Gamma_{\mathtt{W}}, \Omega, \Delta \vdash \mathtt{declare} \ \mathtt{p} \ \mathtt{in} \ \mathtt{prog} : \mathtt{T}} \ \ (\text{P-DEC})$$

$$\frac{\Gamma_{\mathtt{W}} \uplus \{\mathtt{x} : \mathtt{W}\}, \Omega, \Delta \vdash \mathtt{t} : \mathtt{W}}{\Gamma_{\mathtt{W}}, \Omega, \Delta \vdash \{\mathtt{x} \to \mathtt{t}\} : \mathtt{W} \to \mathtt{W}} \ \ (\text{P-CLOS})$$

$$\frac{\Gamma_{\mathtt{W}} \uplus \{\mathtt{a} : \mathtt{T}\}, \Omega, \Delta \vdash \mathtt{prog} : \mathtt{T}'}{\Gamma_{\mathtt{W}}, \Omega, \Delta \vdash \mathtt{box} \ [\mathtt{a}] \ \mathtt{in} \ \mathtt{prog} : \mathtt{T} \to \mathtt{T}'} \ \ (\text{P-BOX})$$

(b) Simple typing rules for procedures, terms, closures and programs

Fig. 3: Tier-based type system

- **0** corresponds to a program component whose execution may result in a memory increase (in size) and that cannot control the program flow.
- **1** corresponds to a program component whose execution cannot result in a memory increase and that may control the program flow.

The type system of Figure 3 is composed of two sub-systems. The typing rules provided in Figure 3b enforce that terms follow a standard simply-typed discipline. The typing rules of Figure 3a will implement a standard non-interference type discipline à la Volpano et al. [30] on the expression (and statement) tier, preventing data flows from tier **0** to tier **1**. The transition between the two sub-type-systems is performed in the rule (P-DEC) of Figure 3b that checks that the procedure body follows the tier-based type discipline once and for all in a procedure declaration.

In Figure 3a, as tier **1** data cannot grow (but can decrease) and are the only data driving the program flow, the number of distinct memory configurations on such data for a terminating procedure is polynomial in the size of the program input (*i.e.*, number of symbols). Hence a typable and terminating procedure has a *polynomial step count* (in the sense of [11]), *i.e.*, on any input, the execution time of a procedure is bounded by a first-order polynomial in the size of their input and the maximal size of any answer returned by an oracle call.

The innermost tier is used to implement a declassification mechanism on operators improving the type-system's expressive power: an operator may be typed differently depending on its calling context (the statement where it is applied). This is the reason why more than 2 tiers can be used in general.

The outermost tier is used to ensure that oracles are only called on inputs of bounded size. This latter restriction on oracle calls enforces a semantic restriction, called *finite lookahead revision*, introduced in [22,20] and requiring that, during each computation, the number of calls performed by the oracle on an input of increasing size is bounded by a constant.

Let MPT be the class of second-order functionals computable by an oracle Turing machine with a polynomial step count and a finite lookahead revision. [20] shows that $\texttt{BFF} = \lambda(\text{MPT})_2$. The type system of Figure 3 ensures that each terminating procedure of a well-typed program computes a function in MPT.

Safe programs. In this section, we restrict the set of admissible operators to prevent programs admitting exponential growth from being typable. A program satisfying such a restriction will be called *safe*.

An operator typing environment Δ is *safe* if for each $\texttt{op} \in dom(\Delta)$ such that $ar(\texttt{op}) > 0$, $\texttt{op}$ is neutral or positive, $[\![\texttt{op}]\!]$ is a polynomial time computable function, and for each $\mathbf{k} \in \mathbf{N}$, and for each $\mathbf{k}_1 \to \ldots \mathbf{k}_{ar(\texttt{op})} \to \mathbf{k}' \in \Delta(\texttt{op})(\mathbf{k})$, the two conditions below hold:

1. $\mathbf{k}' \preceq \wedge_{i=1}^{ar(\texttt{op})} \mathbf{k}_i \preceq \vee_{i=1}^{ar(\texttt{op})} \mathbf{k}_i \preceq \mathbf{k}$,
2. if $\texttt{op}$ is a positive operator then $\mathbf{k}' \prec \mathbf{k}$.

Example 3. Consider the operators $!=$, $\texttt{pred}$, and $\texttt{suc}_i$ discussed in Example 1 and an operator typing environment Δ that is safe and such that $!=$, $\texttt{pred}$, $\texttt{suc}_i$

$\in dom(\Delta)$. We can set $\Delta(!\!=)(\mathbf{1}) \triangleq \{\mathbf{1} \to \mathbf{1} \to \mathbf{1}\} \cup \{\mathbf{k} \to \mathbf{k'} \to \mathbf{0} \mid \mathbf{k}, \mathbf{k'} \preceq \mathbf{1}\}$, as $!\!=$ is neutral. However $\mathbf{1} \to \mathbf{0} \to \mathbf{1} \notin \Delta(!\!=)(\mathbf{1})$ as it breaks Condition 1) above (*i.e.*, $\mathbf{1} \npreceq \mathbf{1} \wedge \mathbf{0}$).

We can also set $\Delta(\texttt{pred})(\mathbf{2}) \triangleq \{\mathbf{2} \to \mathbf{k} \mid \mathbf{k} \preceq \mathbf{2}\} \cup \{\mathbf{1} \to \mathbf{k} \mid \mathbf{k} \preceq \mathbf{1}\} \cup \{\mathbf{0} \to \mathbf{0}\}$. We also have $\Delta(\texttt{suc}_i)(\mathbf{1}) = \{\mathbf{1} \to \mathbf{0}, \mathbf{0} \to \mathbf{0}\}$. $\mathbf{1} \to \mathbf{1} \notin \Delta(\texttt{suc}_i)(\mathbf{1})$ as $\texttt{suc}_i$ is a positive operator and, due to Condition 2) above, the operator output tier has to be strictly smaller than $\mathbf{1}$.

Given a simple typing environment $\Gamma_{\texttt{W}}$, a procedure typing environment Ω, and a safe operator typing environment Δ, a program $\texttt{prog}$ is a *safe program* if it is well-typed for these environments, *i.e.*, $\Gamma_{\texttt{W}}, \Omega, \Delta \vdash \texttt{prog} : (\overline{\texttt{W} \to \texttt{W}}) \to \overline{\texttt{W}} \to \texttt{W}$ can be derived. Let SAFE be the set of safe programs.

Example 4. We consider the program $\texttt{ce}$ of Example 1. We define the operator typing environment Δ by $\Delta(!\!=)(\mathbf{2}) \triangleq \{\mathbf{1} \to \mathbf{1} \to \mathbf{1}\}$, $\Delta(\texttt{pred})(\mathbf{1}) \triangleq \{\mathbf{1} \to \mathbf{1}\}$, and $\Delta(\varepsilon)(\mathbf{2}) \triangleq \{\mathbf{0}, \mathbf{1}\}$. As the three operators $!\!=$, $\texttt{pred}$, and ε are neutral, the environment Δ is safe. We define the simple typing environment $\Gamma_{\texttt{W}}$ by $\Gamma_{\texttt{W}}(\texttt{w}) \triangleq \texttt{W}, \Delta(\texttt{v}) \triangleq \texttt{W}, \Delta(\texttt{z}) \triangleq \texttt{W}, \Gamma_{\texttt{W}}(\texttt{X}_1) \triangleq \texttt{W} \to \texttt{W}$, and $\Gamma_{\texttt{W}}(\texttt{X}_2) \triangleq \texttt{W} \to \texttt{W}$. We define the variable typing environment Γ by $\Gamma(\texttt{w}) \triangleq \mathbf{1}, \Delta(\texttt{v}) \triangleq \mathbf{1}, \Delta(\texttt{z}) \triangleq \mathbf{0}$. Finally, define the procedure typing environment Ω by $\Omega(\texttt{KS}) \triangleq \langle \Gamma, (\mathbf{1}, \mathbf{2}, \mathbf{1}) \rangle$. Using the rules of Figure 3, the following typing judgement can be derived $\Gamma_{\texttt{W}}, \Omega, \Delta \vdash \texttt{ce} :$ $(\texttt{W} \to \texttt{W}) \to \texttt{W} \to \texttt{W}$. Hence $\texttt{ce} \in$ SAFE.

4   Characterizations of the class of Basic Feasible Functionals

Safe and terminating programs. In this section, we show that typable (safe) and terminating programs capture exactly the class of basic feasible functionals.

For a given set of functionals $\mathcal{S}$, let $\mathcal{S}_2$ be the restriction of $\mathcal{S}$ to second-order functionals and let $\lambda(\mathcal{S})$ be the set of functions computed by closed simply-typed lambda terms using functions in $\mathcal{S}$ as constants. Formally, let $\lambda(\mathcal{S})$ be the set of functions denoted by the set of closed simply-typed lambda terms generated inductively as follows:

- for each type τ, variables $x^\tau, y^\tau, \ldots$ are terms,
- each functional $F \in \mathcal{S}$ of type τ, F^τ is a term,
- for any term $t^{\tau'}$ and variable x^τ, $\lambda x^\tau.t^{\tau'}$ is a term of type $\tau \to \tau'$,
- for any terms $t^{\tau \to \tau'}$ and s^τ, $t^{\tau \to \tau'}s^\tau$ is a term of type τ'.

Each lambda term of type τ represents a function of type τ and terms are considered up to β and η equivalences. $\lambda(\mathcal{S})_2$ is called the *second-order simply-typed lambda closure* of $\mathcal{S}$.

Given a simple typing environment $\Gamma_{\texttt{W}}$, a safe operator typing environment Δ, and a triplet of tiers $(\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})$, a procedure $\texttt{p} \triangleq \texttt{P}(\overline{\texttt{X}}, \overline{\texttt{x}})\{[\texttt{var } \overline{\texttt{y}};] \texttt{ st return x}\}$ is *safe* if it is well-typed for these environments, i.e $\Gamma_{\texttt{W}}, \Delta \vdash \texttt{st} : (\mathbf{k}, \mathbf{k}_{in}, \mathbf{k}_{out})$ can be derived using the rules of Figure 3. $\texttt{P}$ computes a second-order partial functional $[\![\texttt{P}]\!] \in (\mathbb{W} \to \mathbb{W})^{|\overline{\texttt{X}}|} \to \mathbb{W}^{|\overline{\texttt{x}}|} \to \mathbb{W}$, defined by $[\![\texttt{P}]\!](\overline{f}, \overline{w}) = w$ iff $(\{\texttt{p}\}, \mu_\emptyset[\overline{\texttt{x}} \leftarrow$

$\overline{w}, \overline{X} \leftarrow \overline{f}]$, call $P(\overline{X}, \overline{x})) \rightarrow_{\texttt{env}} w$ (see Figure 2). If $[\![P]\!]$ is a total function, then the procedure terminates. Let ST be the set of safe and terminating procedures.

The characterization of BFF in terms of safe and terminating procedures discussed in the introduction can be stated as follows.

Theorem 1 ([16]). $\lambda([\![\text{ST}]\!])_2 = \texttt{BFF}$.

We are now ready to state a first characterization of BFF in terms of safe (SAFE) and terminating (SN) programs, showing that the external simply-typed lambda-closure of Theorem 1 can be removed.

Theorem 2. $[\![\text{SN} \cap \text{SAFE}]\!]_2 = \texttt{BFF}$.

We want to highlight that the characterization of Theorem 2 is not just "moving" the simply-typed lambda-closure inside the programming language by adding a construct for lambda-abstraction. Indeed, the soundness of this result crucially depends on some choices on the language design that we have enforced: the restricted ability to compose oracles using closures, and the read-only mode of oracles inside a procedure call, implemented through continuations.

Safe and terminating rank-r programs. More importantly, we also show that this characterization is still valid in the absence of lambda-abstraction.

A safe program prog w.r.t. to a typing derivation π is a *rank-r* program, if for any typing sub-derivation $\pi' \rhd \ \varGamma_{\mathtt{W}}, \varOmega, \varDelta \vdash \lambda \mathtt{a}.\mathtt{t} : \mathtt{T}$ of π, it holds that $\texttt{ord}(\mathtt{T}) \leq r$. In other words, all lambda-abstractions are at most type-k terms, for $k \leq r$. In particular, a rank-$(r+1)$ program, for $r \geq 1$, has variables that are at most type-r variables. Rank-0 and rank-1 programs may have both type-0 and type-1 variables as these variables can still be captured by closures, procedure declarations, or boxes.

For a given set S of well-typed programs, let S_r be the subset of rank-r programs in S, *i.e.*, $S_r \triangleq \{\texttt{prog} \in S \mid \texttt{prog} \text{ is a rank-}r \text{ program}\}$. For example, SAFE_r denotes the set of safe rank-r programs. It trivially holds that $\text{SAFE} = \cup_{r \in \mathbb{N}}\text{SAFE}_r$. The rank is clearly not uniquely determined for a given program. In particular, any rank-r program is also a rank-$(r+1)$ program. Consequently, for any set S of well-typed programs and any $i \leq j$, it trivially holds that $S_i \subseteq S_j$.

Example 5. Program ce of Example 1 is in SAFE_0. Indeed, $\texttt{ce} \in \text{SAFE}$, *cf.* Example 4, and ce is a rank-0 program, as it does not use any lambda-abstraction.

Now we revisit the syntax and semantics of safe rank-0 programs in SAFE_0. The programs are generated by the syntax of Figure 1, where the terms are all of type-0 and redefined by:

$$\texttt{Terms} \qquad \mathtt{t}^0, \mathtt{t}^0_1, \mathtt{t}^0_2, \ldots \ ::= \mathtt{x} \mid \mathtt{X@t}^0 \mid \texttt{call } \mathtt{P}(\overline{\mathtt{c}}, \overline{\mathtt{t}^0})$$

Moreover, there is no longer a need for call-by-name reduction in the big step operational semantics. As a consequence, the rules (TVar), (OA), and (Call) of Figure 2c can be replaced by the following simplified rules:

$$\frac{}{(\sigma, \mu, \mathtt{x}) \rightarrow_{\texttt{env}} \mu(\mathtt{x})} \ (\text{TVar}^0) \qquad \frac{(\sigma, \mu, \mathtt{t}^0_1) \rightarrow_{\texttt{env}} w}{(\sigma, \mu, \mathtt{X@t}^0_1) \rightarrow_{\texttt{env}} \mu(\mathtt{X})(w)} \ (\text{OA}^0)$$

$$\frac{(\sigma, \mu, \overline{\mathtt{t}^0}) \to_{\mathtt{env}} \overline{w} \quad (\sigma, \mu[\overline{\mathtt{x}} \leftarrow \overline{w}, \overline{\mathtt{y}} \leftarrow \overline{\epsilon}], \overline{\mathtt{X}} \mapsto \overline{\mathtt{c}}, \mathtt{st}) \to_{\mathtt{st}} \mu'}{(\sigma \cup \{\mathtt{P}(\overline{\mathtt{X}}, \overline{\mathtt{x}})\{\mathtt{var}\ \overline{\mathtt{y}};\ \mathtt{st\ return\ z}\}\}, \mu, \mathtt{call\ P}(\overline{\mathtt{c}}, \overline{\mathtt{t}^0})) \to_{\mathtt{env}} \mu'(\mathtt{z})} \ (\mathrm{Call}^0)$$

We are now ready to characterize BFF in terms of safe and terminating rank-0 programs.

Theorem 3. $[\![\mathrm{SN} \cap \mathrm{SAFE}_0]\!] = \mathtt{BFF}$.

Hence the characterization of Theorem 2 is just a conservative extension of Theorem 3: lambda-abstractions, viewed as a construct of the programming language, allow for more expressive power in the programming discipline but do not capture more functions. As lambda-abstraction is fully removed from the programming language, this also shows that the simply-typed lambda closure of Theorem 1 can be simulated through restricted oracle compositions in our programming language (using closures and continuations). Moreover, the full hierarchy of safe and terminating rank-r programs collapses.

Corollary 1. $\forall r \in \mathbb{N},\ [\![\mathrm{SN} \cap \mathrm{SAFE}_r]\!] = \mathtt{BFF}$.

Tractable type inference. Let the size $|\mathtt{prog}|$ of the program $\mathtt{prog}$ be the total number of symbols in $\mathtt{prog}$. Type inference is tractable for safe programs.

Theorem 4. *Given a program* $\mathtt{prog}$ *and a safe operator typing environment* Δ,

- *deciding whether* $\mathtt{prog} \in \mathrm{SAFE}$ *holds is a P-complete problem.*
- *deciding whether* $\mathtt{prog} \in \mathrm{SAFE}_0$ *holds can be done in time* $\mathcal{O}(|\mathtt{prog}|^3)$.

Tractability of type inference is a nice property of the type system. Showing $\mathtt{prog} \in \mathrm{SN}$ is at least as hard as showing the termination of a first-order program, hence $\varPi_2^0$-hard in the arithmetical hierarchy. Therefore, the characterizations of Theorems 1, 2, and 3 are unlikely to be decidable, let alone tractable.

5   A completeness-preserving termination criterion

In this section, we show that the undecidable termination assumption (SN) can be replaced with a criterion, called SCP_S, adapted from the Size-Change Termination (SCT) techniques of [23], that is decidable in polynomial time and that preserves the completeness of the characterizations. We first show that studying safe program termination can be reduced to the study of procedure termination.

Lemma 1. *For a given* $\mathtt{prog} \in \mathrm{SAFE}$, *if there exists* $\mathtt{P} \in \mathtt{Proc}(\mathtt{prog})$ *that terminates, then* $\mathtt{prog}$ *is terminating.*

Hence, ensuring the termination of any procedure of a given safe program is a sufficient condition for the program to terminate. The converse trivially does not hold as, for example, a procedure with an infinite loop may be declared and not be called within a given safe program.

Size-Change Termination. SCT relies on the fact that if all infinite executions imply an infinite descent in a well-founded order, then no infinite execution exists. To apply this fact for proving termination, [23] defines Size-Change Graphs (SCGs) that exhibit decreases in the parameters of function calls and then studies the infinite paths in all possible infinite sequences of calls. If all those infinite sequences have at least one strictly decreasing path, then the program must terminate for all inputs. While SCT is `PSpace`-complete, [7] develops a more effective technique, called SCP, that is in `P`. The SCP technique is strong enough for our use case. In the literature, SCT and SCP are applied to pure functional languages. As we shall enforce termination of procedures, we will follow the approach of [1] adapting SCT to imperative programs.

First, we distinguish two kinds of operators that will enforce some (strict) decrease. An operator `op` is *(strictly) decreasing in i*, for $i \leq ar(\mathtt{op})$, if $\forall \overline{w} \in \overline{\mathbb{W}}$, $\overline{w} \neq \overline{\epsilon}$, $|[\![\mathtt{op}]\!](\overline{w})| \leq |w_i|$ ($|[\![\mathtt{op}]\!](\overline{w})| < |w_i|$, respectively) and $[\![\mathtt{op}]\!](\overline{\epsilon}) = \epsilon$. For operators of arity greater than 2, i may not be unique but will be fixed for each operator in what follows.

For simplicity, we will assume that assignments of the considered programs are *flattened*, that is for any assignment $\mathtt{x} := \mathtt{e}$, either $\mathtt{e} = \mathtt{y} \in \mathbb{V}_0$, or $\mathtt{e} = \mathtt{op}(\overline{\mathtt{x}})$, with $\overline{\mathtt{x}} \in \overline{\mathbb{V}_0}$, or $\mathtt{e} = \mathtt{X}(\mathtt{y} \upharpoonright \mathtt{z})$, with $\mathtt{y}, \mathtt{z} \in \mathbb{V}_0$ and $\mathtt{X} \in \mathbb{V}_1$. Notice that, by using extra type-0 variables, any program can be easily transformed into a program with flattened assignments, while preserving semantics and safety properties.

For each assignment of a procedure `P`, we design a bipartite graph, called a SCG, whose nodes are type-0 variables in $(\mathtt{local}(\mathtt{P}) \cup \mathtt{param}(\mathtt{P})) \cap \mathbb{V}_0$ and arrows indicates decreases or stagnation from the old variable to the new. If a variable may increase, then the new variable will not have an in-arrow.

The bipartite graph is generated for any flattened assignment $\mathtt{x} := \mathtt{e}$ by:

- for each y, $\mathtt{y} \neq \mathtt{x}$, we draw arrows from left y to right y.
- If $\mathtt{e} = \mathtt{y}$, we draw an arrow from left y to right x.
- If $\mathtt{e} = \mathtt{op}(\overline{\mathtt{x}})$, with `op` a:
  - decreasing operator in i, we draw an arrow from $\mathtt{x}_i$ to x.
  - strictly decreasing operator in i, we draw a "down-arrow" from $\mathtt{x}_i$ to x.

In all other cases (neutral and non-decreasing operators, positive operators, oracle calls), we do not draw arrows. We will name this SCG graph $G(\mathtt{x} := \mathtt{e})$. Finally, for a set V of variables, G^V will denote the SCG obtained as a subgraph of G restricted to the variables of V.

Example 6. Here are the SCGs associated to simple assignments of a procedure with three type-0 variables x, y, z using a strictly decreasing operator in 1 (`pred`), a decreasing operator (`min`) in 1, a positive operator ($+1$), and an oracle call.

The language $\mathcal{L}(\mathtt{st})$ of (potentially infinite) *sequences of SCG* associated with the statement $\mathtt{st}$ is defined inductively as an ∞-regular expression.

$$\mathcal{L}(\mathtt{x := e}) \triangleq G(\mathtt{x := e}) \qquad \mathcal{L}(\mathtt{if(e)\{st_1\}else\{st_2\}}) \triangleq \mathcal{L}(\mathtt{st_1}) + \mathcal{L}(\mathtt{st_2})$$

$$\mathcal{L}(\mathtt{st_1; st_2}) \triangleq \mathcal{L}(\mathtt{st_1}).\mathcal{L}(\mathtt{st_2}) \qquad \mathcal{L}(\mathtt{while(e)\{st_1\}}) \triangleq \mathcal{L}(\mathtt{st_1})^{\infty}$$

where, following the standard terminology for automata [28], $\mathcal{L}(\mathtt{st})^{\infty}$ is defined by $\mathcal{L}(\mathtt{st})^{\infty} \triangleq \mathcal{L}(\mathtt{st})^* + \mathcal{L}(\mathtt{st})^{\omega}$. In the composition of SCGs, we are interested in paths that advance through the whole concatenated graph. Such a path implies that the final value of the destination variable is of size at most equal to the initial value of the source variable. If the path contains a down-arrow, then the size of the corresponding words decreases strictly.

Following the terminology of [7], a (potentially infinite) sequence of SCGs has a *down-thread* if the associated concatenated graph contains a path spanning every SCG in the sequence and this path includes a down-arrow.

Example 7. Consider the statement $\mathtt{st} \triangleq \mathtt{y := pred(x); y := min(x,y); x := x+}$ $\mathtt{1; x := X(y \upharpoonright z)}$, whose SCGs are described in Example 6. The concatenated graph obtained from the (unique and finite) sequence of SCGs in $\mathcal{L}(\mathtt{st})$ is provided below. It contains a down-thread (the path from $\mathtt{x}$ to $\mathtt{y}$).
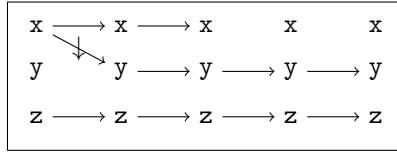
A (potentially infinite) sequence of SCGs is *fan-in free* if the in-degree of nodes is at most 1. By construction, all the considered SCGs are fan-in free.

Safety and Polynomial Size-Change. Unfortunately, programs with down-threads can loop infinitely in the ϵ state. To prevent this, we restrict the analysis to cases where while loops explicitly break out when the decreasing variable reaches ϵ, that is procedures with while loops of the shape $\mathtt{while(x != \varepsilon)\{st\}}$.

For a given set V of variables, we will say that $\mathtt{st}$ *satisfies the simple graph property for V* if for any while loop $\mathtt{while(x != \varepsilon)\{st'\}}$ in $\mathtt{st}$ all sequences of SCGs $G_1^V G_2^V \ldots$ such that $G_1 G_2 \ldots \in \mathcal{L}(\mathtt{st'})$ are fan-in free and contain a down-thread from $\mathtt{x}$ to $\mathtt{x}$. A procedure is in $\mathrm{SCP_S}$ if its statement satisfies the simple graph property for the set of variables in while guards. A program is in $\mathrm{SCP_S}$ if all its procedures are in $\mathrm{SCP_S}$.

Example 8. The program $\mathtt{ce}$ of Example 1 is in $\mathrm{SCP_S}$. The language $\mathcal{L}(\mathtt{body(KS)})$ corresponding to the body of procedure $\mathtt{KS}$ is equal to $G_1.G_2.(G_3.G_4)^{\infty}$, where the SCGs G_i are defined as follows:

G_1	G_2	G_3	G_4
$\mathtt{w} := \mathtt{X_1}(\varepsilon \restriction \varepsilon)$	$\mathtt{z} := \varepsilon$	$\mathtt{v} := \mathtt{pred(v)}$	$\mathtt{z} := \mathtt{X_2}(\mathtt{z} \restriction \mathtt{w})$
$\mathtt{v} \longrightarrow \mathtt{v}$	$\mathtt{v} \longrightarrow \mathtt{v}$	$\mathtt{v} \longrightarrow\hspace{-1.6em}\downarrow\hspace{0.8em} \mathtt{v}$	$\mathtt{v} \longrightarrow \mathtt{v}$
$\mathtt{w} \qquad \mathtt{w}$	$\mathtt{w} \longrightarrow \mathtt{w}$	$\mathtt{w} \longrightarrow \mathtt{w}$	$\mathtt{w} \longrightarrow \mathtt{w}$
$\mathtt{z} \longrightarrow \mathtt{z}$	$\mathtt{z} \qquad \mathtt{z}$	$\mathtt{z} \longrightarrow \mathtt{z}$	$\mathtt{z} \qquad \mathtt{z}$

First, the procedure body satisfies the syntactic restrictions on programs (flattened expressions and restricted while guards). Moreover, the procedure body satisfies the simple graph property for $\{\mathtt{v}\}$ as there is always a down-thread on the path from $\mathtt{v}$ to $\mathtt{v}$ in $(G_3.G_4)^\infty$ and any corresponding sequence is fan-in free. Consequently, the program $\mathtt{ce}$ is in $\mathrm{SCP_S} \cap \mathrm{SAFE_0}$, by Example 5.

$\mathrm{SCP_S}$ preserves completeness on safe programs for $\mathtt{BFF}$.

Theorem 5. $[\![\mathrm{SCP_S} \cap \mathrm{SAFE_0}]\!] = [\![\mathrm{SCP_S} \cap \mathrm{SAFE}]\!] = \mathtt{BFF}.$

While in general deciding if a program satisfies the size-change principle is $\mathtt{PSpace}$-complete, $\mathrm{SCP_S}$ can be checked in quadratic time and, consequently, we obtain the following results.

Theorem 6. *Given a program* $\mathtt{prog}$ *and a safe operator typing environment,*

- *deciding whether* $\mathtt{prog} \in \mathrm{SCP_S} \cap \mathrm{SAFE}$ *is a* P*-complete problem.*
- *deciding whether* $\mathtt{prog} \in \mathrm{SCP_S} \cap \mathrm{SAFE_0}$ *can be done in time* $\mathcal{O}(|\mathtt{prog}|^3)$.

6   Conclusion and future work

We have presented a typing discipline and a termination criterion for a programming language that is sound and complete for the class of second-order polytime computable functionals, $\mathtt{BFF}$. This characterization has three main advantages: 1) it is based on a natural higher-order programming language with imperative procedures; 2) it is pure as it does not rely on an extra semantic requirements (such as taking the lambda closure); 3) belonging to the set $\mathrm{SCP_S} \cap \mathrm{SAFE}$ can be decided in polynomial time. The benefits of tractability is that our method can be automated. However the expressive power of the captured programs is restricted. This drawback is the price to pay for tractability and we claim that the full SCT method, known to be $\mathtt{PSpace}$-complete, could be adapted in a more general way to our programming language in order to capture more programs at the price of a worse complexity. Moreover, any termination criterion based on the absence of infinite data flows with respect to some well-founded order could work and preserve completeness of our characterizations. Another issue of interest is to study whether the presented approach could be extended to characterize $\mathtt{BFF}$ in a purely functional language. We leave these open issues as future work.

References

1. Avery, J.: Size-change termination and bound analysis. In: Hagiya, M., Wadler, P. (eds.) FLOPS 2006. Lecture Notes in Computer Science, vol. 3945, pp. 192–207. Springer (2006). https://doi.org/10.1007/11737414_14

2. Baillot, P., Dal Lago, U.: Higher-order interpretations and program complexity. Inf. Comput. **248**, 56–81 (2016). https://doi.org/10.1016/j.ic.2015.12.008

3. Baillot, P., Mazza, D.: Linear logic by levels and bounded time complexity. Theor. Comput. Sci. **411**(2), 470–503 (2010). https://doi.org/10.1016/j.tcs.2009.09.015

4. Baillot, P., Terui, K.: Light types for polynomial time computation in lambda-calculus. In: Logic in Computer Science, LICS 2004. pp. 266–275. IEEE (2004). https://doi.org/10.1109/LICS.2004.1319621

5. Beame, P., Cook, S.A., Edmonds, J., Impagliazzo, R., Pitassi, T.: The relative complexity of NP search problems. J. Comput. Syst. Sci. **57**(1), 3–19 (1998). https://doi.org/10.1006/jcss.1998.1575

6. Bellantoni, S., Cook, S.: A new recursion-theoretic characterization of the polytime functions. Computational Complexity **2**, 97–110 (1992). https://doi.org/10.1007/BF01201998

7. Ben-Amram, A.M., Lee, C.S.: Program termination analysis in polynomial time. ACM Trans. Program. Lang. Syst. **29**(1), 5:1–5:37 (2007). https://doi.org/10.1145/1180475.1180480

8. Bonfante, G., Marion, J., Moyen, J.: Quasi-interpretations a way to control resources. Theor. Comput. Sci. **412**(25), 2776–2796 (2011). https://doi.org/10.1016/j.tcs.2011.02.007

9. Cobham, A.: The intrinsic computational difficulty of functions. In: Bar-Hillel, Y. (ed.) Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science, pp. 24–30. North-Holland, Amsterdam (1965)

10. Constable, R.L.: Type two computational complexity. In: Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA. pp. 108–121. ACM (1973). https://doi.org/10.1145/800125.804041

11. Cook, S.A.: Computability and complexity of higher type functions. In: Logic from Computer Science. pp. 51–72. Springer (1992). https://doi.org/10.1007/978-1-4612-2822-6_3

12. Cook, S.A., Kapron, B.M.: Characterizations of the basic feasible functionals of finite type. In: 30th Annual Symposium on Foundations of Computer Science (FOCS 1989). pp. 154–159. IEEE (1989). https://doi.org/10.1109/SFCS.1989.63471

13. Férée, H., Hainry, E., Hoyrup, M., Péchoux, R.: Characterizing polynomial time complexity of stream programs using interpretations. Theor. Comput. Sci. **585**, 41–54 (2015). https://doi.org/10.1016/j.tcs.2015.03.008

14. Gao, S., Avigad, J., Clarke, E.M.: δ-complete decision procedures for satisfiability over the reals. In: Gramlich, B., Miller, D., Sattler, U. (eds.) Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7364, pp. 286–300. Springer (2012). https://doi.org/10.1007/978-3-642-31365-3_23

15. Girard, J.Y.: Light linear logic. Inf. Comput. **143**(2), 175–204 (1998). https://doi.org/10.1006/inco.1998.2700

16. Hainry, E., Kapron, B.M., Marion, J., Péchoux, R.: A tier-based typed programming language characterizing feasible functionals. In: LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020. pp. 535–549 (2020). https://doi.org/10.1145/3373718.3394768

17. Hainry, E., Péchoux, R.: Theory of higher order interpretations and application to basic feasible functions. Log. Methods Comput. Sci. **16**(4) (2020), https://lmcs.episciences.org/6973

18. Irwin, R.J., Royer, J.S., Kapron, B.M.: On characterizations of the basic feasible functionals (part I). J. Funct. Program. **11**(1), 117–153 (2001). https://doi.org/10.1017/S0956796800003841

19. Kapron, B.M., Cook, S.A.: A new characterization of mehlhorn's polynomial time functionals (extended abstract). In: 32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991. pp. 342–347. IEEE (1991). https://doi.org/10.1109/SFCS.1991.185389

20. Kapron, B.M., Steinberg, F.: Type-two polynomial-time and restricted lookahead. In: Logic in Computer Science, LICS 2018. pp. 579–588. ACM (2018). https://doi.org/10.1145/3209108.3209124

21. Kawamura, A., Cook, S.A.: Complexity theory for operators in analysis. ACM Trans. Comput. Theory **4**(2), 5:1–5:24 (2012). https://doi.org/10.1145/2189778.2189780

22. Kawamura, A., Steinberg, F.: Polynomial running times for polynomial-time oracle machines. In: 2nd International Conference on Formal Structures for Computation and Deduction, FSCD 2017. pp. 23:1–23:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2017). https://doi.org/10.4230/LIPIcs.FSCD.2017.23

23. Lee, C.S., Jones, N.D., Ben-Amram, A.M.: The size-change principle for program termination. In: Hankin, C., Schmidt, D. (eds.) POPL 2001. pp. 81–92. ACM (2001). https://doi.org/10.1145/360204.360210

24. Leivant, D., Marion, J.Y.: Lambda calculus characterizations of poly-time. Fundam. Inform. **19**(1/2), 167–184 (1993)

25. Mairson, H.G.: Linear lambda calculus and ptime-completeness. J. Funct. Program. **14**(6), 623–633 (2004). https://doi.org/10.1017/S0956796804005131

26. Marion, J.: A type system for complexity flow analysis. In: Logic in Computer Science, LICS 2011. pp. 123–132. IEEE Computer Society (2011). https://doi.org/10.1109/LICS.2011.41

27. Mehlhorn, K.: Polynomial and abstract subrecursive classes. J. Comp. Sys. Sci. **12**(2), 147–178 (1976). https://doi.org/10.1016/S0022-0000(76)80035-9

28. Nivat, M., Perrin, D.: Automata on infinite words, vol. 192. Springer Science & Business Media (1985)

29. Townsend, M.: Complexity for type-2 relations. Notre Dame J. Formal Log. **31**(2), 241–262 (1990). https://doi.org/10.1305/ndjfl/1093635419

30. Volpano, D., Irvine, C., Smith, G.: A sound type system for secure flow analysis. Journal of computer security **4**(2-3), 167–187 (1996). https://doi.org/10.3233/JCS-1996-42-304

Variable binding and substitution for (nameless) dummies

André Hirschowitz[1], Tom Hirschowitz[2], Ambroise Lafont[3], and Marco Maggesi[4]

[1] Univ. Côte d'Azur, CNRS, LJAD, 06103, Nice, France
[2] Univ. Grenoble Alpes, Univ. Savoie Mont Blanc, CNRS, LAMA, 73000, Chambéry, France
[3] University of New South Wales, Sydney, Australia
[4] Università degli Studi di Firenze, Italy

Abstract. By abstracting over well-known properties of De Bruijn's representation with nameless dummies, we design a new theory of syntax with variable binding and capture-avoiding substitution. We propose it as a simpler alternative to Fiore, Plotkin, and Turi's approach, with which we establish a strong formal link. We also show that our theory easily incorporates simple types and equations between terms.

Keywords: syntax · variable binding · substitution · category theory

1 Introduction

There is a standard notion of signature for syntax with variable binding called *binding signature*. Such a signature consists of a set of operation symbols, together with, for each of them, a *binding arity*. A binding arity is a list $(n_1, \ldots, n_p)$ of natural numbers, whose meaning is that the considered operation has p arguments, with n_i variables bound in the ith argument, for all $i \in \{1, \ldots, p\}$.

Example 1.

- λ-abstraction has binding arity (1) (one argument, with one bound variable);
- application has binding arity $(0, 0)$ (two arguments, with no bound variable);
- unary explicit substitution $e[x \mapsto f]$ has binding arity $(1, 0)$ (two arguments, with one variable bound in the first and none in the second).

There are several possible representations of the syntax specified by a binding signature, most of them benefiting from good semantical understanding. The traditional, nominal representation has been nicely framed within nominal sets [12]. The representation by De Bruijn levels, a.k.a. nested datatypes [5,1], is well-understood thanks to presheaf models [11], as is higher-order abstract syntax [19]. However, one of the oldest representations, using De Bruijn's idea of modelling variables with nameless dummies, does not benefit from any semantical framework. This may be related to the fact that it is often perceived

as low-level and error-prone [4]. Our goal in this paper is to equip De Bruijn's representation with a suitable semantical framework.

Let us start by stressing some of the features of this representation, for some fixed binding signature S.

Inductive definition The set DB_S of terms is the least fixed point of a suitable endofunctor on sets, derived from S. In particular, there is a **variables** map $v\colon \mathbb{N} \to \mathrm{DB}_S$ and, for each operation o in S with binding arity $(n_1, \ldots, n_p)$, a map $o_{\mathrm{DB}_S}\colon \mathrm{DB}_S^p \to \mathrm{DB}_S$.

Substitution DB_S is equipped with a **(parallel) substitution** map

$$-[-]\colon \mathrm{DB}_S \times \mathrm{DB}_S^{\mathbb{N}} \to \mathrm{DB}_S,$$

which satisfies three standard substitution lemmas (**associativity**, **left** and **right unitality**).

Furthermore, substitution is compatible with operations, in the sense that it satisfies the following crucial **binding conditions**: for each operation o with binding arity $(n_1, \ldots, n_p)$, $e_1, \ldots, e_p \in \mathrm{DB}_S$, and $f\colon \mathbb{N} \to \mathrm{DB}_S$,

$$o_{\mathrm{DB}_S}(e_1, \ldots, e_p)[f] = o_{\mathrm{DB}_S}(e_1[\Uparrow^{n_1} f], \ldots, e_p[\Uparrow^{n_p} f]), \tag{1}$$

where $\Uparrow$ is a unary operation defined on $\mathrm{DB}_S^{\mathbb{N}}$ by

$$(\Uparrow \sigma)(0) = v(0)$$
$$(\Uparrow \sigma)(n + 1) = \sigma(n)[p \mapsto v(p + 1)].$$

In the present work, by abstracting over these properties, we propose a simple theory for syntax with variable binding, which we summarise as follows.

De Bruijn monad (§2) A De Bruijn monad consists of a set X, equipped with **variables** and **substitution** maps, say $v\colon \mathbb{N} \to X$ and $-[-]\colon X \times X^{\mathbb{N}} \to X$, satisfying the abstract counterparts of the above substitution lemmas.

De Bruijn S-algebra A De Bruijn S-algebra[5] is a De Bruijn monad $(X, -[-], v)$ equipped with operations from the signature S, satisfying the abstract counterpart of the above binding condition.

The term De Bruijn S-algebra We define the set DB_S by an abstract counterpart of the above inductive definition. The substitution map $-[-]\colon \mathrm{DB}_S \times \mathrm{DB}_S^{\mathbb{N}} \to \mathrm{DB}_S$ is then the unique map satisfying left unitality and the binding conditions. Furthermore, it satisfies both other substitution lemmas, hence upgrades DB_S into a De Bruijn S-algebra.

Category of De Bruijn S-algebras (§3) De Bruijn S-algebras are the objects of a category S - **DBAlg**, whose morphisms are all maps between underlying sets that commute with variables, substitution, and operations.

Initial-algebra Semantics Finally, DB_S is initial in S - **DBAlg**, which provides a relevant induction/recursion principle.

---

[5] There is a slightly different notion of De Bruijn algebra in the literature, see the related work section.

We thus propose a theory for syntax with substitution, which is an alternative to the mainstream initial-algebra semantics of Fiore et al.'s [11]. We have experienced the simplicity of our theory by formalising it not only in Coq, but also in HOL Light, which does not support dependent types.

Our theory is similar to the mainstream theory [11], in the following aspects.

– Our and their basic definitions of syntax can be recast using relative monads: De Bruijn monads are monads relative to the functor $1 \to$ **Set** selecting $\mathbb{N}$, while Fiore et al.'s substitution monoids are monads relative to the full and faithful embedding into **Set** of the category of finite ordinals and arbitrary maps between them.
– We find (Theorem 3, §4) that both approaches, in their own ways, include exotic models, and that when freed from them, our category of De Bruijn S-algebras and their category of S-models become equivalent. In this sense, both semantics differ only marginally.
– In §5, we show how De Bruijn S-algebras can be defined by resorting to (a slight generalisation [6] of) *pointed strong endofunctors*, in the spirit of [11].
– Their framework accomodates simple types and equations [7]; we also provide such extensions of our theory in §6 and §7.

Related work

Abstract frameworks for variable binding One of the mainstream such frameworks is [11]. This has been our main reference and in §5 we establish a strong link between this framework and our proposal. This link could probably be extended to variants such as [17,18,3].

In a more recent work, Allais et al. [1] introduce a universe of syntaxes, which essentially corresponds to a simply-typed version of binding signatures. Their framework is designed to facilitate the definition of so-called **traversals**, i.e., functions defined by structural induction, "traversing" their argument. We leave for future work the task of adapting our approach to such traversals.

In a similar spirit, let us mention the recent work of Gheri and Popescu [13], which presents a theory of syntax with binding, mechanised in Isabelle/HOL. Potential links with our approach remain unclear to us at the time of writing.

Finally, the categories of well-behaved objects obtained in §4 are technically very close to nominal sets [12]: finite supports appear in the action-based presentation of nominal sets, while pullback preservation appears in their sheaf-based presentation. And indeed, any well-behaved presheaf yields a nominal set, and so does any well-behaved De Bruijn monad. However, these links are not entirely satisfactory, because they do not account for substitution. The reason is that the only categorical theory of substitution that we know of for nominal sets, by Power [24], is operadic rather than monadic, so we do not immediately see how to extend the correspondence.

Proof assistant libraries Allais et al. [1] mechanise their approach in Agda. In the same spirit, the presheaf-based approach was recently formalised [9].

De Bruijn representation benefits from well-developed proof assistant libraries, in particular Autosubst [26,27]. They introduce a notion of De Bruijn algebra, and design a sound and complete decision procedure for their equational theory, which they furthermore implement for Coq.

Our notion of De Bruijn algebra differs from theirs, notably in that their substitutions are finitely generated. Our approach makes the theoretical development significantly simpler, but of course finite generation is crucial for their main purpose, namely decidability.

General notation

We denote by $A^* = \sum_{n \in \mathbb{N}} A^n$ the set of finite sequences of elements of A, for any set A. In any category $\mathbf{C}$, we tend to write $[C, D]$ for the hom-set $\mathbf{C}(C, D)$ between any two objects C and D. Finally, for any endofunctor F, $F\text{-}\mathbf{alg}$ denotes the usual category of F-algebras and morphisms between them.

2   De Bruijn monads

In this section, we start by introducing De Bruijn monads. Then, we define lifting of assignments, the binding conditions, and the models of a binding signature S in De Bruijn monads, De Bruijn S-algebras. Finally, we construct the term De Bruijn S-algebra.

2.1   Definition of De Bruijn monads

We start by fixing some terminology and notation, and then give the definition.

Definition 1. *Given a set X, an **X-assignment** is a map $\mathbb{N} \to X$. We sometimes merely use "assignment" when X is clear from context.*

Notation 21. *Consider any map $s \colon X \times Y^{\mathbb{N}} \to Z$.*

- *For all $x \in X$ and $g \colon \mathbb{N} \to Y$, we write $x[g]_s$ for $s(x, g)$, or even $x[g]$ when s is clear from context.*
- *Furthermore, s gives rise to the map*

$$X^{\mathbb{N}} \times Y^{\mathbb{N}} \to Z^{\mathbb{N}}$$
$$(f, g) \mapsto n \mapsto s(f(n), g).$$

We use similar notation for this map, i.e., $f[g](n) := f(n)[g]_s$.

Definition 2. *A **De Bruijn monad** is a set X, equipped with*

- *a **substitution** map $s \colon X \times X^{\mathbb{N}} \to X$, which takes an element $x \in X$ and an assignment $f \colon \mathbb{N} \to X$, and returns an element $x[f]$, and*
- *a **variables** map $v \colon \mathbb{N} \to X$,*

satisfying, for all $x \in X$, and $f, g \colon \mathbb{N} \to X$:

- **associativity:** $x[f][g] = x[f[g]]$,
- **left unitality:** $v(n)[f] = f(n)$, and
- **right unitality:** $x[v] = x$.

Example 2. The set $\mathbb{N}$ itself is clearly a De Bruijn monad, with variables given by the identity and substitution $\mathbb{N} \times \mathbb{N}^{\mathbb{N}} \to \mathbb{N}$ given by evaluation. This is in fact the initial De Bruijn monad, as should be clear from the development below.

Example 3. The set $\Lambda := \mu X.\mathbb{N} + X + X^2$ of λ-terms forms a De Bruijn monad. The variables map $\mathbb{N} \to \Lambda$ is the obvious one, while the substitution map $\Lambda \times \Lambda^{\mathbb{N}} \to \Lambda$ is less obvious but standard. In Example 5, as an application of Theorem 2, we will characterise this De Bruijn monad by a universal property.

2.2 Lifting assignments

Given a De Bruijn monad M, we define an operation called **lifting** on its set of assignments $\mathbb{N} \to M$. It is convenient to stress that only part of the structure of De Bruijn monad is needed for this definition.

Definition 3. *Consider any set M, equipped with maps $s\colon M \times M^{\mathbb{N}} \to M$ and $v\colon \mathbb{N} \to M$. For any assignment $\sigma\colon \mathbb{N} \to M$, we define the assignment $\Uparrow \sigma\colon \mathbb{N} \to M$ by*
$$(\Uparrow \sigma)(0) = v(0)$$
$$(\Uparrow \sigma)(n+1) = \sigma(n)[\uparrow],$$
where $\uparrow\colon \mathbb{N} \to X$ maps any n to $v(n+1)$.

Remark 1. Both $\Uparrow$ and $\uparrow$ depend on M and (part of) (s, v). Here, and in other similar situations below, we abuse notation and omit such dependencies for readability.

Of course we may iterate lifting:

Definition 4. *Let $\Uparrow^0 A = A$, and $\Uparrow^{n+1} A = \Uparrow (\Uparrow^n A)$.*

2.3 Binding arities and binding conditions

Our treatment of binding arities reflects the separation between the first-order part of the arity, namely its length, which concerns the syntax, and the binding information, namely the binding numbers, which concerns the compatibility with substitution.

Definition 5.

- *A **first-order arity** is a natural number.*
- *A **binding arity** is a sequence $(n_1, \ldots, n_p)$ of natural numbers, i.e., an element of $\mathbb{N}^*$.*
- *The first-order arity $|a|$ associated with a binding arity $a = (n_1, \ldots, n_p)$ is its length p.*

Let us now axiomatise what we call an operation of a given binding arity.

Definition 6. *Let $a = (n_1, \ldots, n_p)$ be any binding arity, M be any set, $s \colon M \times M^{\mathbb{N}} \to M$, and $v \colon \mathbb{N} \to M$ be any maps. An operation **of binding arity** a is a map $o \colon M^p \to M$ satisfying the following a-**binding condition** w.r.t. (s, v):*

$$\forall \sigma \colon \mathbb{N} \to M, x_1, \ldots, x_p \in M, \quad o(x_1, \ldots, x_p)[\sigma] = o(x_1[\Uparrow^{n_1} \sigma], \ldots, x_p[\Uparrow^{n_p} \sigma]). \tag{2}$$

Remark 2. Let us emphasise the dependency of this definition on v and s – which is hidden in the notations for substitution and lifting.

2.4   Binding signatures and algebras

In this section, we recall the standard notions of first-order (resp. binding) signatures, and adapt the definition of algebras to our De Bruijn context. Let us first briefly recall the former.

Definition 7. *A **first-order signature** consists of a set O of **operations**, equipped with an **arity** map $ar \colon O \to \mathbb{N}$.*

Definition 8. *For any first-order signature $S := (O, ar)$, an **S-algebra** is a set X, together with, for each operation $o \in O$, a map $o_X \colon X^{ar(o)} \to X$.*

Let us now generalise this to binding signatures.

Definition 9.

- *A **binding signature** [23] consists of a set O of **operations**, equipped with an **arity** map $ar \colon O \to \mathbb{N}^*$. Intuitively, the arity of an operation specifies the number of bound variables in each argument.*
- *The first-order signature $|S|$ associated with a binding signature $S := (O, ar)$ is $|S| := (O, |ar|)$, where $|ar| \colon O \to \mathbb{N}$ maps any $o \in O$ to $|ar(o)|$.*

Example 4. The binding signature for λ-calculus has two operations lam and app, of respective arities (1) and $(0, 0)$. The associated first-order signature has two operations lam and app, of respective arities 1 and 2.

Let us now present the notion of De Bruijn S-algebra:

Definition 10. *For any binding signature $S := (O, ar)$, a **De Bruijn S-algebra** is a De Bruijn monad (X, s, v) equipped with an operation o_X of binding arity $ar(o)$, for all $o \in O$.*

In order to state our characterisation of the term model, we associate to any binding signature an endofunctor on sets, as follows.

Definition 11. *The endofunctor Σ_S associated to a binding signature (O, ar) is defined by $\Sigma_S(X) = \sum_{o \in O} X^{|ar(o)|}$.*

Remark 3. The induced endofunctor just depends on the underlying first-order signature.

Remark 4. As is well known, for any binding signature, the initial $(\mathbb{N} + \Sigma_S)$-algebra has as carrier the least fixed point $\mu A.\mathbb{N} + \Sigma_S(A)$.

The following theorem defines the term model of a binding signature.

Theorem 1. *Consider any binding signature $S = (O, ar)$, and let DB_S denote the initial $(\mathbb{N}+\Sigma_S)$-algebra, with structure maps $v : \mathbb{N} \to \mathrm{DB}_S$ and $a : \Sigma_S(\mathrm{DB}_S) \to \mathrm{DB}_S$. Then,*

(i) *There exists a unique map $s : \mathrm{DB}_S \times \mathrm{DB}_S{}^{\mathbb{N}} \to \mathrm{DB}_S$ such that*
   - *for all $n \in \mathbb{N}$ and $f : \mathbb{N} \to \mathrm{DB}_S$, $s(v(n), f) = f(n)$, and*
   - *for all $o \in O$, the map o_{DB_S} satisfies the $ar(o)$-binding condition w.r.t. (s, v).*
(ii) *This map turns (DB_S, v, s, a) into a De Bruijn S-algebra.*

Proof. We have proved the result in both HOL Light [22] and Coq [20].

Remark 5. Point *(i)* may be viewed as an abstract form of recursive definition for substitution in the term model. The theorem thus allows us to construct the term model of a signature in two steps: first the underlying set, constructed as the inductive datatype $\mu Z.\mathbb{N} + \Sigma_S(Z)$, and then substitution, defined by the binding conditions viewed as recursive equations.

Remark 6. We hope that our mechanisations [22,20] may be useful for future developments based on De Bruijn representation, to automatically generate the correct syntax and substitution from a suitable signature. This will have the advantage of reducing what needs to be read to make sure that the development actually does what is claimed. Normally, this part includes the whole definition of syntax and substitution, while our framework reduces it to only the binding signature. Our mechanisations may in fact be used for this purpose on existing developments, to certify the syntax and substitution, leaving only the binding signature for the reader to check.

Example 5. For the binding signature of λ-calculus (Example 4), the carrier of the initial model is $\mu Z.\mathbb{N} + Z + Z^2$, and substitution is defined inductively by:

$$v(n)[\sigma] = \sigma(n)$$
$$\lambda(e)[\sigma] = \lambda(e[\Uparrow \sigma])$$
$$(e_1\, e_2)[\sigma] = e_1[\sigma]\, e_2[\sigma].$$

3   Initial-algebra semantics of binding signatures in De Bruijn monads

In this section, for any binding signature S, we organise De Bruijn S-algebras into a category, S-**DBAlg**, and prove that the term De Bruijn S-algebra is initial therein.

3.1   A category of De Bruijn monads

Let us start by organising general De Bruijn monads into a category:

Definition 12. *A morphism $(X, s, v) \to (Y, t, w)$ between De Bruijn monads is a set-map $f : X \to Y$ commuting with substitution and variables, in the sense that for all $x \in X$ and $\sigma : \mathbb{N} \to X$ we have $f(x[\sigma]) = f(x)[f \circ \sigma]$ and $f \circ v = w$.*

Remark 7. More explicitly, the first axiom says: $f(s(x, \sigma)) = t(f(x), f \circ \sigma)$.

Notation 31. *De Bruijn monads and morphisms between them form a category, which we denote by* **DBMnd**.

Let us conclude this subsection by briefly mentioning a categorical point of view on the category of De Bruijn monads for the categorically-minded reader, in terms of **relative** monads [2].

Proposition 1. *The category* **DBMnd** *is canonically isomorphic to the category of monads relative to the functor $1 \to$ **Set** picking $\mathbb{N}$.*

Remark 8. Canonicity here means that the isomorphism lies over the canonical isomorphism $[1, \mathbf{Set}] \cong \mathbf{Set}$.

According to the theory of [2], this yields:

Corollary 1. *The tensor product $X \otimes Y := X \times Y^{\mathbb{N}}$ induces a skew monoidal [28] structure on* **Set***, and* **DBMnd** *is precisely the category of monoids therein.*

Proof. To see this, let us observe that, by viewing any set X, in particular $\mathbb{N}$, as a functor $1 \to \mathbf{Set}$, one may compute the left Kan extension of X along $\mathbb{N}$, which is a functor $\mathrm{Lan}_{\mathbb{N}}(X) : \mathbf{Set} \to \mathbf{Set}$. By the standard formula for left Kan extensions [21], we have $\mathrm{Lan}_{\mathbb{N}}(X)(Y) \cong X \times Y^{\mathbb{N}} = X \otimes Y$. The result thus follows by [2, Theorems 4 and 5].   □

3.2   Categories of De Bruijn algebras

In this section, for any binding signature S, we organise De Bruijn S-algebras into a category S-**DBAlg**.

Let us start by recalling the category of S-algebras for a first-order S:

Definition 13. *For any first-order signature S, a morphism $X \to Y$ of S-algebras is a map between underlying sets commuting with operations, in the sense that for each $o \in O$, letting $p := ar(o)$, we have $f(o_X(x_1, \ldots, x_p)) = o_Y(f(x_1), \ldots, f(x_p))$.*

*We denote by S-**alg** the category of S-algebras and morphisms between them.*

We now exploit this to define De Bruijn S-algebras:

Definition 14. *For any binding signature S, a morphism of De Bruijn S-algebras is a map $f : X \to Y$ between underlying sets, which is a morphism both of De Bruijn monads and of $|S|$-algebras. We denote by S-**DBAlg** the category of De Bruijn S-algebras and morphisms between them.*

Theorem 2. *Consider any binding signature $S = (O, ar)$, and let DB_S denote the initial $(\mathbb{N} + \Sigma_S)$-algebra. Then, the De Bruijn S-algebra structure of Theorem 1 on DB_S makes it initial in S-$\mathbf{DBAlg}$.*

Proof. We have proved the result in both HOL Light [22] and Coq [20]. □

4   Relation to presheaf-based models

The classical initial-algebra semantics introduced in [11] associates in particular to each binding signature S a category, say Φ_S-$\mathbf{Mon}$ of models, while we have proposed in §3 an alternative category of models S-$\mathbf{DBAlg}$. In this section, we are interested in comparing both categories of models.

In fact, we find that both include exotic models, in the sense that we do not see any loss in ruling them out. And when we do so, we obtain equivalent categories.

4.1   Trimming down presheaf-based models

First of all, in this subsection, let us recall the mainstream approach we want to relate to, and exclude some exotic objects from it.

Presheaf-based models   We start by recalling the presheaf-based approach. The ambient category is the category of functors $[\mathbb{F}, \mathbf{Set}]$, where $\mathbb{F}$ denotes the category of finite ordinals, and all maps between them. As is well-known, this category is equivalent to the category $[\mathbf{Set}, \mathbf{Set}]_f$ of finitary endofunctors on sets, and inherits from it a **substitution** monoidal structure. By construction, monoids for this monoidal structure are equivalent to finitary monads on sets.

The idea is then to interpret binding signatures S as endofunctors Φ_S on $[\mathbb{F}, \mathbf{Set}]$, and to define models as monoids equipped with Φ_S-algebra structure, satisfying a suitable compatibility condition.

The definition of Φ_S relies on an operation called derivation:

Definition 15 (Endofunctor associated to a binding signature).

- *Let the **derivative** X' of any functor $X\colon \mathbb{F} \to \mathbf{Set}$ be defined by $X'(n) = X(n+1)$.*
- *Furthermore, let $X^{(0)} = X$, and $X^{(n+1)} = (X^{(n)})'$.*
- *For any binding arity $a = (n_1, \ldots, n_p)$, let $\Phi_a(X) = X^{(n_1)} \times \ldots \times X^{(n_p)}$.*
- *For any binding signature $S = (O, ar)$, let $\Phi_S = \sum_{o \in O} \Phi_{ar(o)}$.*

Proposition 2. *Through the equivalence with finitary functors, derivation becomes $F'(A) = F(A+1)$, for any finitary $F\colon \mathbf{Set} \to \mathbf{Set}$ and $A \in \mathbf{Set}$.*

Example 6. For the binding signature S_λ of Example 4 for λ-calculus we get $\Phi_{S_\lambda}(X)(n) = X(n)^2 + X(n+1)$.

Next, we want to express the relevant compatibility condition between algebra and monoid structure. For this, let us briefly recall the notion of pointed strength, see [11,10] for details.

Definition 16. *A **pointed strength** on an endofunctor $F\colon \mathbf{C} \to \mathbf{C}$ on a monoidal category $(\mathbf{C}, \otimes, I, \alpha, \lambda, \rho)$ is a family of morphisms $st_{C,(D,v)}\colon F(C) \otimes D \to F(C \otimes D)$, natural in $C \in \mathbf{C}$ and $(D, v\colon I \to D) \in I/\mathbf{C}$, the coslice category below I, satisfying two coherence conditions.*

The next step is to observe that binding signatures generate pointed strong endofunctors.

Definition 17. *The derivation endofunctor $X \mapsto X'$ on $[\mathbb{F}, \mathbf{Set}]$ has a pointed strength, defined through the equivalence with finitary functors by*

$$G(F(X) + 1) \xrightarrow{G(F(X)+v_1)} G(F(X) + F(1)) \xrightarrow{G[F(in_1),F(in_2)]} G(F(X + 1)).$$

Product, coproduct, and composition of endofunctors lift to pointed strong endofunctors, which yields:

Corollary 2 ([11,10]). *For all binding signatures S, Φ_S is pointed strong.*

At last, we arrive at the definition of models.

Definition 18. *For any pointed strong endofunctor F on $\mathbf{C}$, an F-**monoid** is an object X equipped with F-algebra and monoid structure, say $a\colon F(X) \to X$, $s\colon X \otimes X \to X$, and $v\colon I \to X$, such that the following pentagon commutes.*

$$
\begin{array}{ccccc}
F(X) \otimes X & \xrightarrow{st_{X,(X,v)}} & F(X \otimes X) & \xrightarrow{F(s)} & F(X) \\
{\scriptstyle a \otimes X} \downarrow & & & & \downarrow {\scriptstyle a} \\
X \otimes X & & \xrightarrow{\hspace{3cm} s \hspace{3cm}} & & X
\end{array}
$$

A morphism of F-monoids is a morphism in $\mathbf{C}$ which is a morphism both of F-algebras and of monoids. We let F-$\mathbf{Mon}$ denote the category of F-monoids and morphisms between them.

Example 7. For the binding signature S_λ of Example 4, a Φ_{S_λ}-monoid is an object X, equipped with maps $X' \to X$ and $X^2 \to X$, and compatible monoid structure. Compatibility describes how substitution should be pushed down through abstractions and applications.

Well-behaved presheaves The exoticness we want to rule out only concerns the underlying functor of a model, so we just have to define well-behaved functors in $[\mathbb{F}, \mathbf{Set}]$.

Well-behavedness for a functor $T\colon \mathbb{F} \to \mathbf{Set}$ is about getting closed terms right. More precisely, for some finite sets m and n, an element of $T(m + n)$ which both exists in $T(m)$ and $T(n)$ should also exist in $T(\emptyset)$, and uniquely so. This says exactly that T should preserve the pullback

$$\begin{array}{ccc} \emptyset & \longrightarrow & n \\ \downarrow & & \downarrow \\ m & \longrightarrow & m + n. \end{array}$$

Remark 9. The reader might wonder about other, i.e., non-empty pullbacks. But these are automatically preserved, by [29, Proposition 2.1].

Definition 19.

- *A functor* $\mathbb{F} \to \mathbf{Set}$ *is **well-behaved** iff it preserves binary intersections, or equivalently empty binary intersections. Let* $[\mathbb{F}, \mathbf{Set}]_{wb}$ *denote the full subcategory spanned by well-behaved functors.*
- *For any binding signature S, an object of* $\Phi_S \text{-} \mathbf{Mon}$ *is **well-behaved** iff the underlying functor is. Let* $\Phi_S \text{-} \mathbf{Mon}_{wb}$ *denote the full subcategory spanned by well-behaved objects.*

Example 8. As an example of a non well-behaved finitary monad, consider the monad L of λ-calculus but edited so that $L(\emptyset) = \emptyset$.

The important result for comparing the presheaf-based approach with ours is the following.

Proposition 3. *The subcategory* $\Phi_S \text{-} \mathbf{Mon}_{wb}$ *includes the initial object.*

Proof. Roughly, closed terms are isomorphic to terms in two free variables that use neither the first, nor the second.

Remark 10. In most natural situations, all models are in fact well-behaved [16, Proposition 5.17].

4.2 Trimming down De Bruijn monads

Let us now turn to well-behaved De Bruijn algebras. Here well-behavedness is about finitariness. However, it may not be immediately clear how to define finitariness of a De Bruijn monad.

Definition 20. *A De Bruijn monad (X, s, v) is finitary iff each of its elements $x \in X$ has a (finite) support $N_x \in \mathbb{N}$, in the sense that for all $f \colon \mathbb{N} \to \mathbb{N}$ fixing the first N_x numbers, the corresponding renaming $v \circ f$ fixes x.*

Example 9. By Proposition 4 below, the initial S-algebra is finitary, for any binding signature S. For a counterexample, consider the greatest fixed point $\nu A.\mathbb{N}+\Sigma_S(A)$, for any S with at least one operation with more than one argument. E.g., if S has an operation of binding arity $(0,0)$, like application in λ-calculus, then the term $v(0) \ (v(1) \ (v(2) \ \ldots))$ does not have finite support.

Definition 21. *For any binding signature S, let $S \text{-} \mathbf{DBAlg}_{wb}$ denote the full subcategory spanning De Bruijn S-algebras whose underlying De Bruijn monad is finitary.*

Proposition 4. *The subcategory $S \text{-} \mathbf{DBAlg}_{wb}$ includes the initial object.*

4.3 Bridging the gap

We may at last state the relationship between initial-algebra semantics of binding signatures in presheaves and in De Bruijn monads:

Theorem 3. *Consider any binding signature S. The subcategories Φ_S - $\mathbf{Mon}_{wb}$ and S - $\mathbf{DBAlg}_{wb}$ are equivalent.*

Proof. See [16, Appendix A].

Remark 11. The moral of this is that, if one removes exotic objects from both Φ_S - $\mathbf{Mon}$ and S - $\mathbf{DBAlg}$, then one obtains equivalent categories, which both retain the initial object. Thus, the two approaches to initial-algebra semantics of binding signatures differ only marginally.

Restricting attention to well-behaved objects, we may thus benefit from the strengths of both approaches. Typically, in De Bruijn monads, free variables need to be computed explicitly, while presheaves come with intrinsic scoping, as terms are indexed by sets of potential free variables. Conversely, in some settings, observational equivalence may relate programs with different sets of free variables [25]. In such cases, it is useful to have all terms collected in one single set. This needs to be computed (and involves non-trivial quotienting) in presheaves, while it is direct in De Bruijn monads.

5 Strength-based interpretation of the binding conditions

In the previous section, we have compared the category S - $\mathbf{DBAlg}$ of models of a binding signature in De Bruijn monads with the standard category of Φ_S-monoids [11]. In this section, we establish a different kind of link, by showing that, for any binding signature S, both categories S - $\mathbf{DBAlg}$ and Φ_S - $\mathbf{Mon}$ are instances of a common categorical construction. We have seen that the standard category Φ_S - $\mathbf{Mon}$ is constructed from the pointed strong endofunctor Φ_S, so we would like a similar construction of S - $\mathbf{DBAlg}$. However, pointed strong endofunctors live on monoidal categories [11,10], while we have seen in Corollary §1 that $\mathbb{N}$ and the tensor product only equip $\mathbf{Set}$ with skew monoidal structure. In order to bridge this gap, we resort to a generalisation of pointed strengths to skew monoidal categories proposed by Borthelle et al. [6].

We give a condensed account: the interested reader is referred to [16, §6].

The starting point is that the endofunctor Σ_S associated to any given binding signature S may be equipped with a family of maps

$$\mathbf{dbs}_S \colon \Sigma_S(X) \otimes Y \to \Sigma_S(X \otimes Y).$$

However, in order for such a map to be well-defined, we need to assume that Y features variables and renaming, i.e., that it is a **pointed $\mathbb{N}$-module**, as we now introduce:

Definition 22.

– An $\mathbb{N}$-***module*** is a set X equipped with an action $X \times \mathbb{N}^{\mathbb{N}} = X \otimes \mathbb{N} \to X$. of the monoid $\mathbb{N}^{\mathbb{N}}$.

– For such an action $r : X \times \mathbb{N}^{\mathbb{N}} = X \otimes \mathbb{N} \to X$, we generally denote $r(x, f)$ by $x[f]_r$, or merely $x[f]$ when clear from context.

– A morphism of $\mathbb{N}$-modules is a map between underlying sets, commuting with action in the obvious sense.

– A ***pointed*** $\mathbb{N}$-***module*** is an $\mathbb{N}$-module (X, r), equipped with a map $v : \mathbb{N} \to X$ which is a morphism of $\mathbb{N}$-modules.

– A morphism of pointed $\mathbb{N}$-modules is a map commuting with action and point, in the obvious sense.

– Let $\mathbb{N}\text{-}\mathbf{Mod}_{\mathbb{N}}$ denote the category of pointed $\mathbb{N}$-modules.

Example 10. Any De Bruijn monad (X, s, v) (in particular $\mathbb{N}$ itself) has a canonical structure of pointed $\mathbb{N}$-module given by v and $r(x, f) = x[v \circ f]$.

We may now define the map $\mathbf{dbs}_S$. Lifting of assignments (Definition 3) straightforwardly generalises to pointed $\mathbb{N}$-modules. Recalling the definition

$$\Sigma_S(X) = \sum_{o \in O} X^{p_o},$$

where $ar(o) = (n_1^o, \ldots, n_{p_o}^o)$ for all $o \in O$, we thus simply have:

Definition 23. *For any binding signature $S = (O, ar)$, the **De Bruijn strength** $\mathbf{dbs}_S$ of the induced endofunctor Σ_S is defined by*

$$\Sigma_S(X) \otimes Y \to \Sigma_S(X \otimes Y)$$
$$((o, (x_1, \ldots, x_{p_o})), \sigma) \mapsto (o, ((x_1, \Uparrow^{n_1} \sigma), \ldots, (x_{p_o}, \Uparrow^{n_{p_o}} \sigma))),$$

for all sets X and pointed $\mathbb{N}$-modules Y, with again $ar(o) = (n_1^o, \ldots, n_{p_o}^o)$.

The fact that any De Bruijn monad is in particular a pointed $\mathbb{N}$-module by Example 10 enables the definition of models in the strength-based approach:

Definition 24. *For any binding signature S, a Σ_S-**monoid** is an object X, equipped with monoid and Σ_S-algebra structure, say $s \colon X \otimes X \to X$, $v \colon \mathbb{N} \to X$, and $a \colon \Sigma_S(X) \to X$, making the following pentagon commute.*

$$
\begin{array}{ccc}
\Sigma_S(X) \otimes X \xrightarrow{\;\mathbf{dbs}_{S,X,X}\;} \Sigma_S(X \otimes X) \xrightarrow{\;\Sigma_S(s)\;} \Sigma_S(X) \\
{\scriptstyle a \otimes X} \downarrow \qquad\qquad\qquad\qquad\qquad\qquad \downarrow {\scriptstyle a} \\
X \otimes X \xrightarrow{\hspace{5cm} s \hspace{0.5cm}} X
\end{array}
\qquad (3)
$$

A morphism of Σ_S-monoids is a map which is both a monoid and a Σ_S-algebra morphism.

Let $\Sigma_S\text{-}\mathbf{Mon}$ denote the category of Σ_S-monoids and morphisms between them.

Remark 12. In [16], this definition is framed in a more general context, notably emphasising the fact that $\mathbf{dbs}_S$ is in fact a **structural strength** on the endofunctor Σ_S.

We may at last relate the initial-algebra semantics of §3 with the strength-based approach:

Proposition 5. *For any binding signature $S = (O, ar)$ and De Bruijn monad (M, s, v) equipped with a map $o_M : M^p \to M$ for all $o \in O$ with $ar(o) = (n_1, \ldots, n_p)$, the following are equivalent:*

(i) *each map $o_M : M^p \to M$ satisfies the **a-binding condition** w.r.t. (s, v);*
(ii) *the corresponding map $\Sigma_S M \to M$ renders the pentagon (3) commutative.*

Corollary 3. *For any binding signature S, we have an isomorphism $\Sigma_S\text{-}\mathbf{Mon} \cong S\text{-}\mathbf{DBAlg}$ of categories over $\mathbf{Set}$.*

This readily entails the following (bundled) reformulation of Theorems 1 and 2.

Corollary 4. *Consider any binding signature $S = (O, ar)$, and let DB_S denote the initial $(\mathbb{N}+\Sigma_S)$-algebra, with structure maps $v \colon \mathbb{N} \to \mathrm{DB}_S$ and $a \colon \Sigma_S(\mathrm{DB}_S) \to \mathrm{DB}_S$. Then:*

(i) *There exists a unique substitution map $s \colon \mathrm{DB}_S \otimes \mathrm{DB}_S \to \mathrm{DB}_S$ such that*
   - *the map $\mathbb{N} \otimes \mathrm{DB}_S \xrightarrow{v \otimes \mathrm{DB}_S} \mathrm{DB}_S \otimes \mathrm{DB}_S \xrightarrow{s} \mathrm{DB}_S$ coincides with the left unit of the skew monoidal structure $(n, f) \mapsto f(n)$, and*
   - *the pentagon (3) (with $\Sigma := \Sigma_S$) commutes.*
(ii) *This substitution map turns (DB_S, v, s, a) into a Σ_S-monoid.*
(iii) *This Σ_S-monoid is initial in $\Sigma_S\text{-}\mathbf{Mon}$.*

Proof. Let $\mathbf{Mon}(\mathbf{Set})$ denote the category of monoids in $\mathbf{Set}$ for the skew monoidal structure. We have an equality $\mathbf{Mon}(\mathbf{Set}) = \mathbf{DBMnd}$ of categories, and the algebra structure $\Sigma_S(\mathrm{DB}_S) \to \mathrm{DB}_S$ is merely the cotupling of the maps o_{DB_S} of Theorem 1. This correspondence translates one statement into the other.

Remark 13. This result hints at a potential push-button proof of Theorems 1 and 2 (and Corollary 4). Indeed, it is almost an instance of [6, Theorem 2.15]: the latter is stated for general skew monoidal categories instead of merely $\mathbf{Set}$, but does not directly apply in the present setting, because it assumes that the tensor product is finitary in the second argument.

6   Simply-typed extension

In this section, we extend the framework of §2–3, which is untyped, to the simply-typed case. The development essentially follows the same pattern, replacing sets with families.

We fix in the whole section a set $\mathbb{T}$ of **types**, and call $\mathbb{T}$**-sets** the objects of $\mathbf{Set}^{\mathbb{T}}$. A morphism $X \to Y$ is a family $(X(\tau) \to Y(\tau))_{\tau \in \mathbb{T}}$ of maps.

6.1   De Bruijn $\mathbb{T}$-monads

In this subsection, we define the typed analogue of De Bruijn monads.

The role of $\mathbb{N}$ will be played in the typed context by the following $\mathbb{T}$-set.

Definition 25. *Let* $\mathbf{N} \in \mathbf{Set}^{\mathbb{T}}$ *be defined by* $\mathbf{N}(\tau) = \mathbb{N}$.

Remark 14. This provides a countable set of variables at each type, which may not quite be what the reader would have called "typed De Bruijn representation". An inconvenience of this representation is that an "erasure" map from typed to untyped terms appears to need to rely on a bijection $\mathbb{T} \times \mathbb{N} \cong \mathbb{N}$ for "renaming" variables. In particular, not all indices can be preserved by such a map.

Definition 26. *Given a* $\mathbb{T}$*-set* X, *an* X-***assignment*** *is a morphism* $\mathbf{N} \to X$. *We sometimes merely use "assignment" when* X *is clear from context.*

The analogue of the tensor product $X \otimes Y = X \times Y^{\mathbb{N}}$ will be played by $[\mathbf{N}, Y] \cdot X$, i.e., the iterated self-coproduct of X, with one copy per Y-assignment.

Notation 61. *For coherence with the untyped case, we tend to write an element of* $([\mathbf{N}, Y] \cdot X)(\tau)$ *as* (x, f), *with* $x \in X(\tau)$ *and* $f \colon \mathbf{N} \to Y$.

Furthermore, Notation 21 straightforwardly adapts to the typed case.

The definition of De Bruijn monads generalises almost *mutatis mutandis*:

Definition 27. *A* ***De Bruijn*** $\mathbb{T}$***-monad*** *is a* $\mathbb{T}$*-set* X, *equipped with*

  – *a* ***substitution*** *morphism* $s \colon [\mathbf{N}, X] \cdot X \to X$, *which takes an element* $x \in X$ *and an assignment* $f \colon \mathbf{N} \to X$, *and returns an element* $x[f]$, *and*
  – *a* ***variables*** *morphism* $v \colon \mathbf{N} \to X$,

such that for all $x \in X$, *and* $f, g \colon \mathbf{N} \to X$, *we have*

$$x[f][g] = x[f[g]] \qquad\qquad v(n)[f] = f(n) \qquad\qquad x[v] = x.$$

Example 11. The set Λ_{ST} of simply-typed λ-terms with free variables of type τ in $\mathbb{N} \times \{\tau\}$, considered equivalent modulo α-renaming, forms a De Bruijn monad. Variables $\mathbf{N} \to \Lambda_{\mathrm{ST}}$ are given by mapping, at any τ, any $n \in \mathbb{N}$ to the variable (n, τ). Substitution $[\mathbf{N}, \Lambda_{\mathrm{ST}}] \cdot \Lambda_{\mathrm{ST}} \to \Lambda_{\mathrm{ST}}$ is standard, capture-avoiding substitution. One main purpose of this section is to characterise Λ_{ST} by a universal property, and reconstruct it categorically.

Morphisms generalise straightforwardly, and we get:

Proposition 6. *De Bruijn* $\mathbb{T}$*-monads and morphisms between them form a category* $\mathbf{DBMnd}(\mathbb{T})$.

6.2   Initial-algebra semantics

We now adapt the initial-algebra semantics of §3 to the typed case. Let us start by generalising lifting to the typed case. This relies on a typed form of lifting, which acts on all variables of a given type, leaving all other variables untouched.

Definition 28. *Let (X, s, v) denote any De Bruijn $\mathbb{T}$-monad. We first define a typed analogue $\uparrow^\tau$ of the $\uparrow$ of Definition 3, as below left, and then the **lifting** of any assignment $\sigma \colon \mathbf{N} \to X$ as below right.*

$$(\uparrow^\tau)_\tau(n) = v_\tau(n+1) \qquad\qquad (\Uparrow^\tau \sigma)_\tau(0) = v_\tau(0)$$
$$(\uparrow^\tau)_{\tau'}(n) = v_{\tau'}(n) \quad (\text{if } \tau \neq \tau') \qquad (\Uparrow^\tau \sigma)_\tau(n+1) = \sigma_\tau(n)[\uparrow^\tau]$$
$$(\Uparrow^\tau \sigma)_{\tau'}(n) = \sigma_{\tau'}(n)[\uparrow^\tau] \ (\text{if } \tau \neq \tau').$$

Finally, for any sequence $\gamma = (\tau_1, \ldots, \tau_n)$ of types, we define $\Uparrow^\gamma \sigma$ inductively, by $\Uparrow^\varepsilon \sigma = \sigma$ and $\Uparrow^{\gamma, \tau} \sigma = \Uparrow^\tau (\Uparrow^\gamma \sigma)$, where ε denotes the empty sequence.

We then generalise first-order and binding arities. The main point is:

Definition 29. *A **binding arity** is an element of $(\mathbb{T}^* \times \mathbb{T})^* \times \mathbb{T}$, i.e., a tuple $(((\gamma_1, \tau_1), \ldots, (\gamma_p, \tau_p)), \tau)$, where each $\gamma_i \in \mathbb{T}^*$ is a list of types, and each τ_i, as well as τ, are types, thought of as an inference rule* $\dfrac{\gamma_1 \vdash \tau_1 \quad \ldots \quad \gamma_p \vdash \tau_p}{\vdash \tau}$.

Example 12. The binding signature for simply-typed λ-calculus has two operations $\mathrm{lam}_{\tau, \tau'}$ and $\mathrm{app}_{\tau, \tau'}$ for all types τ and τ', of respective arities

$$\frac{\tau \vdash \tau'}{\vdash \tau \to \tau'} \qquad \text{and} \qquad \frac{\vdash \tau \to \tau' \quad \vdash \tau}{\vdash \tau'} \ .$$

This allows us to generalise binding conditions, as follows.

Definition 30. *Let $a = (((\gamma_1, \tau_1), \ldots, (\gamma_p, \tau_p)), \tau)$ be any binding arity, and M be any set equipped with morphisms $s \colon [\mathbf{N}, M] \cdot M \to M$ and $v \colon \mathbf{N} \to M$. An **operation of binding arity** a is a map $o \colon M(\tau_1) \times \ldots \times M(\tau_p) \to M(\tau)$ satisfying the following a-**binding condition** w.r.t. (s, v):*

$$\forall \sigma \colon \mathbf{N} \to M, x_1, \ldots, x_p \in M(\tau_1) \times \ldots \times M(\tau_p),$$
$$o(x_1, \ldots, x_p)[\sigma] = o(x_1[\Uparrow^{\gamma_1} \sigma], \ldots, x_p[\Uparrow^{\gamma_p} \sigma]). \tag{4}$$

We may now generalise signatures and their models.

Definition 31. *A $\mathbb{T}$-**binding signature** consists of a set O of **operations**, equipped with an arity map $O \to (\mathbb{T}^* \times \mathbb{T})^* \times \mathbb{T}$.*

Definition 32. *Consider any $\mathbb{T}$-binding signature $S := (O, ar)$. A **De Bruijn S-algebra** consists of a De Bruijn $\mathbb{T}$-monad (X, s, v), together with algebra structure on X for the underlying first-order signature $|S|$, in the obvious sense, such that for all $o \in O$ with arity $ar(o) = (((\gamma_1, \tau_1), \ldots, (\gamma_p, \tau_p)), \tau)$, the structural map $o_X \colon X(\tau_1) \times \ldots \times X(\tau_p) \to X(\tau)$ satisfies the $ar(o)$-binding condition w.r.t. (s, v).*

*We denote by S-**DBAlg** the category of De Bruijn S-algebras and (the obvious notion of) morphisms between them.*

Finally, following the untyped case, we may associate to each signature an endo-functor Σ_S, and we have the following typed extension of the initiality theorem.

Theorem 4. *For any $\mathbb{T}$-binding signature S, let DB_S denote the initial $(\mathbf{N}+\Sigma_S)$-algebra, with structure maps $v\colon \mathbf{N} \to \mathrm{DB}_S$ and $a\colon \Sigma_S(\mathrm{DB}_S) \to \mathrm{DB}_S$, inducing maps $o_{\mathrm{DB}_S}\colon \mathrm{DB}_S(\tau_1) \times \ldots \times \mathrm{DB}_S(\tau_p) \to \mathrm{DB}_S(\tau)$ for all $o \in O$ with $ar(o) = (((\gamma_1, \tau_1), \ldots, (\gamma_p, \tau_p)), \tau)$. Then:*

(i) *There exists a unique map $s\colon [\mathbf{N}, \mathrm{DB}_S] \cdot \mathrm{DB}_S \to \mathrm{DB}_S$ such that*
   - *for all $\tau \in \mathbb{T}$, $n \in \mathbb{N}$, and $f : \mathbf{N} \to \mathrm{DB}_S$, $s_\tau(v_\tau(n), f) = f_\tau(n)$, and*
   - *for all $o \in O$, the map o_{DB_S} satisfies the $ar(o)$-binding condition w.r.t. (s, v).*
(ii) *This map turns (DB_S, v, s, a) into a De Bruijn S-algebra.*
(iii) *This De Bruijn S-algebra is initial in S-$\mathbf{DBAlg}$.*

Example 13. While we saw in Example 12 that the De Bruijn monad of simply-typed λ-calculus terms admits a simple signature, there is another relevant, related monad, whose elements at any type are **values** of that type. (Indeed, values are closed under value substitution.) It is relatively straightforward to design a binding signature for this De Bruijn monad, following [15].

7   Equations

In this section, we introduce a notion of equational theory for specifying (typed) De Bruijn monads, following ideas from [8].

Definition 33. *A **De Bruijn equational theory** consists of*

- *two binding signatures S and T, and*
- *two functors $L, R\colon S$-$\mathbf{DBAlg} \to T$-$\mathbf{DBAlg}$ over $\mathbf{DBMnd}(\mathbb{T})$, i.e., making the following diagram commute serially, where U^S and U^T denote the forgetful functors.*

Example 14. Recalling the binding signature S_Λ for λ-calculus from Example 4, let us define a De Bruijn equational theory for β-equivalence. We take $T_\beta = (1, 0)$, and for any De Bruijn S_Λ-algebra X,

- $L(X)$ has as structure map $(e_1, e_2) \mapsto \mathrm{app}(\mathrm{lam}(e_1), e_2)$ while
- $R(X)$ has as structure map $(e_1, e_2) \mapsto e_1[e_2 \cdot \mathrm{id}]$.
  (Here $e_2 \cdot \mathrm{id}$ denotes the assignment $0 \mapsto e_2$, $n + 1 \mapsto v(n)$.)

Definition 34. *Given an equational theory $E = (S, T, L, R)$, a De Bruijn E-algebra is a De Bruijn S-algebra X such that $L(X) = R(X)$.*

Let E-$\mathbf{DBAlg}$ denote the category of E-algebras, with morphisms of De Bruijn S-algebras between them.

Remark 15. The category E - **DBAlg** is an equaliser of L and R in **CAT**.

Let us now turn to characterising the initial De Bruijn E-algebra, for any De Bruijn equational theory E. For this, we introduce the following relation.

Definition 35. *For any De Bruijn equational theory $E = (S, T, L, R)$, with $S = (O, ar)$ and $T = (O', ar')$, let DB_S denote the initial $(\mathbf{N} + \Sigma_S)$-algebra. We define $\sim_E$ to be the smallest equivalence relation on DB_S satisfying the following rules,*

$$\frac{\phantom{o'_{L(\mathrm{DB}_S)}(e_1,\dots,e_p) \sim_E o'_{R(\mathrm{DB}_S)}(e_1,\dots,e_p)}}{o'_{L(\mathrm{DB}_S)}(e_1,\dots,e_p) \sim_E o'_{R(\mathrm{DB}_S)}(e_1,\dots,e_p)}$$

$$\frac{e_1 \sim_E e'_1 \quad \dots \quad e_q \sim_E e'_q}{o_{\mathrm{DB}_S}(e_1,\dots,e_q) \sim_E o_{\mathrm{DB}_S}(e'_1,\dots,e'_q)}$$

for all $e, e_1, \dots$ in DB_S, $o' \in O'$ with $|ar'(o')| = p$, and $o \in O$ with $|ar(o)| = q$.

Example 15. For the equational theory of Example 14, the first rule instantiates precisely to the β-rule, while the second enforces congruence.

Theorem 5. *For any equational theory $E = (S, T, L, R)$, E - **DBAlg** admits an initial object, whose carrier set is the quotient $\mathrm{DB}_S/\sim_E$.*

Proof. This has been mechanised in Coq [20] and HOL [22].

Example 16. The initial model for the equational theory of Example 14 is the quotient of λ-terms in De Bruijn representation by β-equivalence.

Remark 16. In [16, §9], we mention an equivalent way of defining De Bruijn equational theories in terms of modules.

8   Conclusion

We have proposed a simple, set-based theory of syntax with variable binding, which associates a notion of model (or algebra) to each binding signature, and constructs a term model following De Bruijn representation. The notion of model features a substitution operation. We have experienced the simplicity of this theory by implementing it in both Coq and HOL Light.

We have furthermore equipped the construction with an initial-algebra semantics, organising the models of any binding signature into a category, and proving that the term model is initial therein.

We have then studied this initial-algebra semantics in a bit more depth, in two directions. We have first established a formal link with the mainstream, presheaf-based approach [11], proving that well-behaved models (in a suitable sense on each side of the correspondence) agree up to an equivalence of categories. We have then recast the whole initial-algebra semantics into the mainstream, abstract framework of [11,10]. Finally, we have shown that our theory extends easily to a simply-typed setting, and smoothly incorporates equations.

References

1. Allais, G., Atkey, R., Chapman, J., McBride, C., McKinna, J.: A type and scope safe universe of syntaxes with binding: their semantics and proofs. Proceedings of the ACM on Programming Languages **2**(ICFP), 90:1–90:30 (2018). https://doi.org/10.1145/3236785

2. Altenkirch, T., Chapman, J., Uustalu, T.: Monads need not be endofunctors. Logical Methods in Computer Science **11**(1) (2015). https://doi.org/10.2168/LMCS-11(1:3)2015

3. Arkor, N., McDermott, D.: Abstract clones for abstract syntax. In: Kobayashi, N. (ed.) Proc. 6th International Conference on Formal Structures for Computation and Deduction. Leibniz International Proceedings in Informatics (LIPIcs), vol. 195, pp. 30:1–30:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPIcs.FSCD.2021.30

4. Berghofer, S., Urban, C.: A head-to-head comparison of de bruijn indices and names. Electronic Notes in Theoretical Computer Science **174**(5), 53–67 (2007). https://doi.org/10.1016/j.entcs.2007.01.018

5. Bird, R.S., Paterson, R.: De bruijn notation as a nested datatype. Journal of Functional Programming **9**(1), 77–91 (1999). https://doi.org/10.1017/S0956796899003366

6. Borthelle, P., Hirschowitz, T., Lafont, A.: A cellular Howe theorem. In: Hermanns, H., Zhang, L., Kobayashi, N., Miller, D. (eds.) Proc. 35th ACM/IEEE Symposium on Logic in Computer Science ACM (2020). https://doi.org/10.1145/3373718.3394738

7. Fiore, M.P., Hur, C.K.: Second-order equational logic. In: Proceedings of the 19th EACSL Annual Conference on Computer Science Logic (CSL 2010) (2010)

8. Fiore, M., Hur, C.K.: On the construction of free algebras for equational systems. Theoretical Computer Science **410**, 1704–1729 (2009)

9. Fiore, M., Szamozvancev, D.: Formal metatheory of second-order abstract syntax. Proceedings of the ACM on Programming Languages **6**(POPL) (2022). https://doi.org/10.1145/3498715

10. Fiore, M.P.: Second-order and dependently-sorted abstract syntax. In: LICS. pp. 57–68. IEEE (2008). https://doi.org/10.1109/LICS.2008.38

11. Fiore, M.P., Plotkin, G.D., Turi, D.: Abstract syntax and variable binding. In: Proc. 14th Symposium on Logic in Computer Science IEEE (1999)

12. Gabbay, M.J., Pitts, A.M.: A new approach to abstract syntax involving binders. In: Proc. 14th Symposium on Logic in Computer Science IEEE (1999)

13. Gheri, L., Popescu, A.: A formalized general theory of syntax with bindings: Extended version. Journal of Automated Reasoning **64**(4), 641–675 (2020). https://doi.org/10.1007/s10817-019-09522-2

14. Hirschowitz, A., Hirschowitz, T., Lafont, A.: Modules over monads and operational semantics. In: Ariola, Z.M. (ed.) Proc. 5th International Conference on Formal Structures for Computation and Deduction. Leibniz International Proceedings in Informatics (LIPIcs), vol. 167, pp. 12:1–12:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020). https://doi.org/10.4230/LIPIcs.FSCD.2020.12

15. Hirschowitz, A., Hirschowitz, T., Lafont, A.: Modules over monads and operational semantics (2021), submitted expanded version of [14]

16. Hirschowitz, A., Hirschowitz, T., Lafont, A., Maggesi, M.: Variable binding and substitution for (nameless) dummies (2021), https://amblafont.github.io/articles/debruijn-extended.pdf, preprint

17. Hirschowitz, A., Maggesi, M.: Modules over monads and linearity. In: Proc. 14th International Workshop on Logic, Language, Information and Computation LNCS, vol. 4576, pp. 218–237. Springer (2007). https://doi.org/10.1007/3-540-44802-0_3

18. Hirschowitz, A., Maggesi, M.: Modules over monads and initial semantics. Information and Computation **208**(5), 545–564 (2010). https://doi.org/10.1016/j.ic.2009.07.003

19. Hofmann, M.: Semantical analysis of higher-order abstract syntax. In: Proc. 14th Symposium on Logic in Computer Science IEEE (1999)

20. Lafont, A.: Initial algebra semantics for de Bruijn monads in Coq. https://github.com/amblafont/binding-debruijn (2021)

21. Mac Lane, S.: Categories for the Working Mathematician. No. 5 in Graduate Texts in Mathematics, Springer, 2nd edn. (1998)

22. Maggesi, M.: Initial algebra semantics for de Bruijn monads in HOL Light. https://github.com/maggesi/dbmonad/tree/master/De_Bruijn (2021)

23. Plotkin, G.: An illative theory of relations. In: Cooper, R., et al. (eds.) Situation Theory and its Applications. p. 133–146. No. 22 in CSLI Lecture Notes, Stanford University (1990)

24. Power, J.: Abstract syntax: Substitution and binders: Invited address. Electronic Notes in Theoretical Computer Science **173**, 3–16 (04 2007). https://doi.org/10.1016/j.entcs.2007.02.024

25. Sangiorgi, D., Walker, D.: The π-calculus – A Theory of Mobile Processes. Cambridge University Press (2001)

26. Schäfer, S., Tebbi, T., Smolka, G.: Autosubst: Reasoning with de Bruijn terms and parallel substitutions. In: Urban, C., Zhang, X. (eds.) Proc. 6th International Conference on Interactive Theorem Proving. LNCS, vol. 9236, pp. 359–374. Springer (2015). https://doi.org/10.1007/978-3-319-22102-1_24

27. Stark, K., Schäfer, S., Kaiser, J.: Autosubst 2: reasoning with multi-sorted de bruijn terms and vector substitutions. In: Mahboubi, A., Myreen, M.O. (eds.) Proc. 8th International Conference on Certified Programs and Proofs. pp. 166–180. ACM (2019). https://doi.org/10.1145/3293880.3294101

28. Szlachányi, K.: Skew-monoidal categories and bialgebroids. Advances in Mathematics **231**, 1694–1730 (2012). https://doi.org/10.1016/j.aim.2012.06.027

29. Trnková, V.: Some properties of set functors. Commentationes Mathematicæ Universitatis Carolinæ **10**(2), 323–352 (1969)

Uniform Guarded Fragments

Reijo Jaakkola(✉)

Tampere University, Tampere, Finland
`reijo.jaakkola@tuni.fi`
`https://reijojaakkola.github.io`

Abstract. In this paper we prove that the uniform one-dimensional guarded fragment, which is a natural polyadic generalization of guarded two-variable logic, has the Craig interpolation property. We will also prove that the satisfiability problem of uniform guarded fragment is NExpTime-complete.

Keywords: Guarded fragment · Interpolation · Satisfiability problem.

1 Introduction

The guarded fragment $\mathcal{GF}$ is a well studied fragment of first-order logic $\mathcal{FO}$, which was introduced by Andréka, van Benthem and Németi [1] as a generalization of modal logic. Informally speaking, $\mathcal{GF}$ is obtained from $\mathcal{FO}$ by requiring that all quantification must be relativised by $\mathcal{FO}$-atoms, which is motivated by the observation that "quantificaction" in modal logics is relativised by accessability relations. Like modal logic, $\mathcal{GF}$ behaves well both computationally and model-theoretically. In particular, it is decidable, it has a (generalized) tree-model property and it satisfies various preservation theorems [1,7].

We say that a logic $\mathcal{L}$ has Craig interpolation property (CIP), if for every two formulas φ and ψ of $\mathcal{L}$ we have that if $\varphi \models \psi$, then there exists a third formula — the interpolant — χ of $\mathcal{L}$, so that $\varphi \models \chi$, $\chi \models \psi$ and χ contains only relation symbols which occur in both φ and χ. CIP is widely regarded as a property that a "nice" logic should have and for (reasonable logics with compactness) it implies several other desirable model-theoretic properties such as Projective Beth Definability and Robinson's consistency theorem [1,4,13,19].

It is well-known that various modal logics have CIP [1,6,19], while $\mathcal{GF}$ fails to have it [11]. This is somewhat surprising, given that $\mathcal{GF}$ is a very natural generalisation of modal logic, and certainly raises the question of how the syntax of $\mathcal{GF}$ should be modified so as to obtain a logic which does have CIP, and which also behaves well both computationally and model-theoretically. One option would be to extend further the expressive power of $\mathcal{GF}$, and in this direction we have the *guarded negation fragment*, which has CIP, is decidable and shares with $\mathcal{GF}$ various desirable model-theoretic properties [2].

The other option (and the one which is more relevant for this paper) is to investigate fragments of $\mathcal{GF}$. In this direction we also have a positive result,

namely that $\mathcal{GF}^2$ — the two-variable fragment of $\mathcal{GF}$ — has CIP [11]. Given this result, it is natural to ask whether there exists a polyadic extension of $\mathcal{GF}^2$ which would also have CIP, where by a polyadic extension we mean intuitively a logic which contains $\mathcal{GF}^2$ and can express non-trivial properties of polyadic relations. Indeed, it seems rather unlikely that there would not be such an extension, since it is well-known that there are polyadic modal logics which have CIP [1].

In [9] the *uniform one-dimensional fragment* $\mathcal{UF}_1$ was introduced, which is a very natural polyadic extension of the two-variable fragment $\mathcal{FO}^2$ of $\mathcal{FO}$. Roughly speaking, $\mathcal{UF}_1$ is obtained from $\mathcal{FO}$ by requiring that each maximal existential (or universal) block of quantifiers leaves at most one variable free and that when forming boolean combinations of formulas with more than one free variable, the formulas need to have exactly the same set of variables. Formulas satisfying the first restriction are called *one-dimensional*, while formulas satisfying the second restriction are called *uniform*. In [16] it was proved that $\mathcal{UF}_1$ has the finite model property and the complexity of its satisfiability problem is NExpTime-complete, which is the same as for $\mathcal{FO}^2$ [8]. The research around $\mathcal{UF}_1$ and its variants has been quite active, see for instance [12,14,15,17,18].

Given that $\mathcal{UF}_1$ is a polyadic extension of $\mathcal{FO}^2$, the guarded $\mathcal{UF}_1$ is a natural candidate for being a polyadic extension of $\mathcal{GF}^2$ with CIP. As the first main result of this paper we will prove that guarded $\mathcal{UF}_1$ does, in fact, have CIP. Our proof follows closely the argument given in [11] for proving that $\mathcal{GF}^2$ has CIP, the main technical difference being that the proof presented in [11] uses crucially the fact that in the case of $\mathcal{GF}^2$ we can assume live sets to have size at most two, while in our case we have to deal with live sets of arbitrary size.

Since the research around modal-like fragments of $\mathcal{FO}$ is largely motivated by the fact that their satisfiability problems are often decidable, it is natural to also study the complexity of the satisfiability problem of the guarded $\mathcal{UF}_1$, which was in fact already done in [15]. More precisely it was proved in [15] that the satisfiability problem of one-dimensional $\mathcal{GF}$ is in NExpTime, while it is already NExpTime-hard for guarded $\mathcal{UF}_1$. These results left open the problem of determining the complexity of uniform $\mathcal{GF}$ and as the second main result of this paper we will prove that the satisfiability problem of uniform $\mathcal{GF}$ is also in NExpTime (and hence it is NExpTime-complete).

We also emphasize that as a necessary by-product of this second technical result, we isolate the uniformity restriction imposed to formulas of $\mathcal{UF}_1$ as an independent syntactical restriction and provide a formal definition for it (which so far has been missing from the literature). [1] We believe that uniformity is an important and a natural syntactical restriction (at least) in the context of fragments of $\mathcal{FO}$. Indeed, in addition to $\mathcal{UF}_1$ there are several known decidable fragments of $\mathcal{FO}$ which satisfy this restriction up to some degree, such as the one-binding fragments introduced in [20] and the ordered logic introduced in [10]. We hope that the results presented in this paper provide further motivation for the study of various uniform fragments of $\mathcal{FO}$.

---

[1] To be precise, we only define what it means for a formula to be uniform in the context of $\mathcal{GF}$; however, it is easy to extend this definition for other logics.

The structure of this paper is as follows. After the preliminaries in Section 2, we define a notion of bisimulation for $\mathcal{UGF}_1$ and establish its basic properties in Section 3. After this we will prove that $\mathcal{UGF}_1$ has CIP in Section 4. In Section 5 we will establish that the complexity of the satisfiability problem of uniform $\mathcal{GF}$ is NExpTime-complete. The final Section will list some new problems that the research conducted in this paper raises.

2 Preliminaries

2.1 Notation

In this paper we will work with vocabularies which do not contain constants and function symbols. We will also assume that there are no relation symbols of arity 0. We will use the Fraktul capital letters to denote structures, and the corresponding Roman letters to denote their domains. Given a model $\mathfrak{A}$ and $C \subseteq A$, we will use $\mathfrak{A} \upharpoonright C$ to denote the restriction of $\mathfrak{A}$ to the set C. Given two structures $\mathfrak{A}$ and $\mathfrak{B}$, we will use $\mathfrak{A} \leq \mathfrak{B}$ to denote that $\mathfrak{A}$ is a substructure of $\mathfrak{B}$.

Occasionally we will identify tuples $\bar{a} = (a_1, \ldots, a_n)$ with sets $\{a_1, \ldots, a_n\}$, which allows us to use notations such as $b \in \bar{a}$ and $\bar{a} = X$, where X is a set. Given two tuples $\bar{a}$ and $\bar{b}$ of the same length, we will use $\bar{a} \mapsto \bar{b}$ and $p : \bar{a} \to \bar{b}$ to denote the mapping induced by the relation $a_i \mapsto b_i$. Given a tuple $\bar{a} = (a_1, \ldots, a_n)$ and a unary function f, we will use $f(\bar{a})$ to denote the tuple $(f(a_1), \ldots, f(a_n))$. Given a positive integer n we will denote $[n] = \{1, \ldots, n\}$. Finally, if $\bar{a} = (a_1, \ldots, a_n)$ and $k \geq n$ and $\mu : [k] \to [n]$ is a surjection, we will use $\bar{a}_\mu$ to denote the tuple $(a_{\mu(1)}, \ldots, a_{\mu(k)})$.

2.2 Types and Tables

The following definitions are standard in the context of $\mathcal{UF}_1$ and were first introduced in [16]. Let σ be a vocabulary. Given a set $X = \{x_1, \ldots, x_n\}$ of distinct variables and a k-ary relation $R \in \sigma$, we say that an atomic formula $R(x_{i_1}, \ldots, x_{i_k})$ is an X-atom over σ, if $X = \{x_{i_1}, \ldots, x_{i_k}\}$. If α is an X-atom, then α and $\neg\alpha$ are both X-literals over σ. A 1-type over σ is a maximal satisfiable set of $\{x\}$-literals over σ. We identify 1-types π with conjunctions of their elements

$$\bigwedge \pi(x)$$

A k-table is a tuple $\langle \rho, \pi_1, \ldots, \pi_k \rangle$, where each π_ℓ is a 1-type over σ, while ρ is a maximal satisfiable set of $\{x_1, \ldots, x_k\}$-literals over σ. We identify k-tables $\langle \rho, \pi_1, \ldots, \pi_k \rangle$ with conjunctions

$$\bigwedge \rho(x_1, \ldots, x_k) \wedge \bigwedge_{1 \leq \ell \leq k} \pi_\ell(x_\ell).$$

Let $\mathfrak{A}$ be a σ-model. Given a 1-type π over σ, we say that $a \in A$ realizes π if π is the unique 1-type so that $\mathfrak{A} \models \pi[a]$; we denote by $\mathrm{tp}_{\mathfrak{A}}^{\sigma}[a]$ the (unique) 1-type

π over σ which is realized by a in $\mathfrak{A}$. For *distinct* elements $a_1, \ldots, a_k \in A$ we will use $\mathrm{tp}_{\mathfrak{A}}^{\sigma}[a_1, \ldots, a_k]$ to denote the (unique) k-table over σ which is realized by the tuple $(a_1, \ldots, a_k)$.

2.3   Syntax of Uniform Fragments of $\mathcal{GF}$

Given a vocabulary σ, we define $\mathcal{GF}[\sigma]$ to be the smallest set $\mathcal{F}$ which satisfies the following requirements.

- $\mathcal{F}$ contains all the atomic formulas over σ, which includes also equalities between variables.
- If $\varphi, \psi \in \mathcal{F}$, then $\neg\varphi \in \mathcal{F}$ and $(\varphi \wedge \psi) \in \mathcal{F}$.
- If $\psi(\overline{x}) \in \mathcal{F}$, where each free variable of ψ occurs in the tuple $\overline{x}$, then

$$\exists \overline{y}(\alpha(\overline{x}) \wedge \psi(\overline{x})) \in \mathcal{F},$$

  where $\overline{y} \subseteq \overline{x}$ and α is an atomic formula over σ.

If the vocabulary σ is irrelevant or known from the context, then we will simply use $\mathcal{GF}$ to denote $\mathcal{GF}[\sigma]$.

Next we will give a formal definitions for the syntactical notions of one-dimensionality and uniformity. We will start by making the technical remark that we will define recursively the set of subformulas $\mathrm{Sf}(\varphi)$ of $\varphi \in \mathcal{GF}$ otherwise in a standard way, except that for formulas of the form $\varphi := \exists \overline{y}(\alpha(\overline{x}) \wedge \psi(\overline{x}))$, we define $\mathrm{Sf}(\varphi)$ to be

$$\{\exists \overline{y}(\alpha(\overline{x}) \wedge \psi(\overline{x}))\} \cup \mathrm{Sf}((\alpha(\overline{x}) \wedge \psi(\overline{x}))).$$

In other words, we treat each maximal sequence of existential quantification as a single logical operator.

Definition 1. *Let $\varphi \in \mathcal{GF}$ be a formula. We say that φ is one-dimensional, if every subformula of φ of the form*

$$\exists \overline{y}(\alpha(\overline{x}) \wedge \psi(\overline{x}))$$

has at most one free variable. In other words each maximal sequence of (guarded) existential quantification leaves at most one variable free.

Next we will define what it means for a formula of $\mathcal{GF}$ to be uniform. The precise definition turns out to be somewhat technical, and we will start with the following auxiliary definition.

Definition 2. *Let X be a (possibly empty) set of variables and let σ be a vocabulary. A relative X-atom over σ is a formula ψ of $\mathcal{GF}[\sigma]$ which satisfies one of the following conditions.*

1. *ψ is a sentence.*
2. *ψ has a one free variable which belongs to X.*

3. ψ is of the form $x = y$, where $x, y \in X$.
4. ψ is an X-atom over σ.
5. ψ is of the form $\exists \overline{z}(\alpha(\overline{x}) \wedge \psi(\overline{x}))$ and the set of its free variables is precisely X.

With the aid of this definition we are in a position where we can define the notion of uniformity formally.

Definition 3. *Let $\varphi \in \mathcal{GF}[\sigma]$ be a formula. We say that φ is uniform, if every subformula ψ of φ is a boolean combination of relative X-atoms, where X is the set of free variables of ψ.*

Remark 1. Consider a uniform quantifier-free formula $\psi(x_1, \ldots, x_k)$ of $\mathcal{GF}[\sigma]$. Let $\mathfrak{A}$ be a σ-model and let $(a_1, \ldots, a_k)$ be a tuple of not necessarily distinct elements. Then whether or not

$$\mathfrak{A} \models \psi(a_1, \ldots, a_k)$$

holds depends only on the table of $(c_1, \ldots, c_\ell)$, where $(c_1, \ldots, c_\ell)$ is an arbitrary enumeration of the set of distinct elements of $(a_1, \ldots, a_k)$.

The definition of uniformity is somewhat technical, but the following examples should clarify the intuition behind it.

Example 1. Let $\sigma = \{S, R, P\}$, where S is a ternary relation symbol, R is a binary relation symbol and P is a unary relation symbol. The formula

$$\exists x \exists y (P(x) \wedge R(x, y) \wedge S(x, y, y) \wedge R(y, x) \wedge P(y)))$$

is both uniform and one-dimensional. On the other hand the formula

$$\exists x \exists y (\exists z (S(x, y, z) \wedge P(z)) \wedge R(x, y) \wedge S(x, y, x))$$

is uniform but not one-dimensional. Finally, the formula

$$\exists x \exists y \exists w (R(x, y) \wedge \exists z S(x, w, z))$$

is neither one-dimensional nor uniform.

Example 2. The standard translation of polyadic modal logic into $\mathcal{FO}$ results in formulas of the form

$$\exists x_1 \ldots \exists x_k (R(x_0, x_1, \ldots, x_k) \wedge \bigwedge_{1 \leq \ell \leq k} \psi_\ell(x_\ell))$$

which are uniform and one-dimensional [5].

We will use $\mathcal{UGF}$ to denote the set of formulas of $\mathcal{GF}$ which are uniform and $\mathcal{UGF}_1$ to denote the set of formulas of $\mathcal{GF}$ which are both uniform and one-dimensional. Throughout this paper we will use $\varphi(x_1, \ldots, x_n)$, where all the variables in the tuple $(x_1, \ldots, x_n)$ are distinct, to denote a formula of either $\mathcal{UGF}_1$ or $\mathcal{UGF}$ such that either $\{x_1, \ldots, x_n\}$ is precisely the set of free variables of φ or φ has at most one free variable which belongs to $\{x_1, \ldots, x_n\}$ or φ is of the form $x_i = x_j$, where $1 \leq i, j \leq n$.

2.4   Interpolation

We start by recalling the definition of the Craig interpolation property.

Definition 4. *Given a logic $\mathcal{L}$, we say that $\mathcal{L}$ has the Craig interpolation property (CIP), if for every $\varphi \in \mathcal{L}[\sigma]$ and $\psi \in \mathcal{L}[\tau]$ we have that $\varphi \models \psi$ implies that there exists an interpolant $\chi \in \mathcal{L}[\sigma \cap \tau]$ for this entailment, i.e., a sentence for which $\varphi \models \chi$ and $\chi \models \psi$ hold.*

It is well-known that the full $\mathcal{GF}$ fails to have CIP. The known examples of sentences which demonstrate this can be used to make the following observation.

Proposition 1. *The one-dimensional $\mathcal{GF}$ does not have CIP.*

Proof. Consider the following sentences, which are simple variants of the formulas used in [13].

$$\varphi := \exists x \exists y \exists z (G(x, y, z) \wedge R(x, y) \wedge R(y, z) \wedge R(z, x))$$

$$\psi := \forall x \forall y (R(x, y) \to (A(x) \leftrightarrow \neg A(y)))$$

Notice that both of these sentence are one-dimensional. Now one can show, using essentially the same argument as the one used in Example 1 in [13], that there is no interpolant for the implication $\varphi \models \neg\psi$.

We remark that, in the context of fragments of $\mathcal{FO}$, CIP is usually defined for *formulas* instead of sentences (as we have defined it). We could have also formulated it for formulas, but we decided to work with sentences for simplicity.

3   Bisimulation for $\mathcal{UGF}_1$

Given two models $\mathfrak{A}$ and $\mathfrak{B}$, and tuples $\bar{c} \in A^n$ and $\bar{d} \in B^n$ we will use

$$(\mathfrak{A}, \bar{c}) \equiv_\sigma (\mathfrak{B}, \bar{d})$$

to denote the fact that for every $\varphi(x_1, \ldots, x_n) \in \mathcal{UGF}_1$ we have that

$$\mathfrak{A} \models \varphi(c_1, \ldots, c_n) \iff \mathfrak{B} \models \varphi(d_1, \ldots, d_n).$$

The purpose of this section is to define a corresponding notion of bisimulation for $\mathcal{UGF}_1$ which captures the above equivalence relation. We will start by defining a suitable notion of partial isomorphism.

Definition 5. *Let $\mathfrak{A}$ and $\mathfrak{B}$ be models, and let $X := \{a_1, \ldots, a_n\} \subseteq A$ and $Y \subseteq B$. A bijection $p : X \to Y$, is called a uniform partial σ-isomorphism between $\mathfrak{A}$ and $\mathfrak{B}$, if*

$$\mathrm{tp}_{\mathfrak{A}}^\sigma[a_1, \ldots, a_n] = \mathrm{tp}_{\mathfrak{B}}^\sigma[p(a_1), \ldots, p(a_n)].$$

Quantification in $\mathcal{GF}$ over a model $\mathfrak{A}$ is restricted to *live* subsets of $\mathfrak{A}$, i.e., subsets of $\mathfrak{A}$ which are either singletons or are *contained* in a single tuple $\bar{a} \in R^{\mathfrak{A}}$, for some $R \in \sigma$. In the case of $\mathcal{UGF}_1$ we will need the following modified version of the notion of live set, which takes into account the requirement that our formulas are uniform.

Definition 6. *Let $\mathfrak{A}$ be a model and let $X \subseteq A$. We say that X is σ-live, if either $|X| \leq 1$ or there exists $R \in \sigma$ and $(a_1, \ldots, a_n) \in R^{\mathfrak{A}}$ so that $X = \{a_1, \ldots, a_n\}$.*

We are now ready to define the notion of bisimulation for $\mathcal{UGF}_1$.

Definition 7. *Let $\mathcal{Z}$ be a non-empty set of uniform partial σ-isomorphism be-tween two structures $\mathfrak{A}$ and $\mathfrak{B}$. Let $\bar{c} \in A^n$ and $\bar{d} \in B^n$ be tuples. We say that $\mathcal{Z}$ is a uniform guarded σ-bisimulation between $(\mathfrak{A}, \bar{c})$ and $(\mathfrak{B}, \bar{d})$, if for every $p : X \rightarrow Y \in \mathcal{Z}$ the following conditions hold:*

(cover) There exists $h \in \mathcal{Z}$ with $\bar{c} = \text{dom}(h)$ so that $h(\bar{c}) = \bar{d}$.
(forth) For any $a \in X$ and a σ-live set $X' \subseteq A$, with $a \in X'$, there exists $q : X' \rightarrow Y' \in \mathcal{Z}$ so that
$$p(a) = q(a).$$

(back) For any $b \in Y$ and a σ-live set $Y' \subseteq B$, with $b \in Y'$, there exists $q : X' \rightarrow Y' \in \mathcal{Z}$ so that

$$p^{-1}(b) = q^{-1}(b).$$

If there exists a guarded σ-bisimulation between $(\mathfrak{A}, \bar{c})$ and $(\mathfrak{B}, \bar{d})$, then we denote this by $(\mathfrak{A}, \bar{a}) \sim_\sigma (\mathfrak{B}, \bar{b})$.

In what follows we will often refer to uniform guarded bisimulations simply as guarded bisimulations. The following two lemmas establish that our notion of bisimulation is correct, the first of which can proved in a standard manner by using induction.

Lemma 1. *Let $\mathfrak{A}$ and $\mathfrak{B}$ be models, and let $\bar{c} \in A^n$ and $\bar{d} \in B^n$ be tuples so that $(\mathfrak{A}, \bar{c}) \sim_\sigma (\mathfrak{B}, \bar{d})$. Then $(\mathfrak{A}, \bar{c}) \equiv_\sigma (\mathfrak{B}, \bar{d})$.*

For the proof of the second lemma we need to recall the definition of ω-saturated model. A *elementary n-type* over a vocabulary σ is a consistent set of first-order formulas (not necessarily quantifier-free) with free variables in $\{x_1, \ldots, x_n\}$. Given a σ-model $\mathfrak{A}$, we say that it is ω-*saturated*, if for every tuple $\bar{a} \in A^n$ of elements of A we have that each elementary n-type over the extended vocabulary $\sigma \cup \{a_1, \ldots, a_n\}$, where each a_i denotes a constant to be interpreted as the element a_i, which is finitely consistent with the $\mathcal{FO}$-theory of $(\mathfrak{A}, \bar{a})$, is realized in $(\mathfrak{A}, \bar{a})$. It is well-known that every σ-model, where σ is finite and relational, has an ω-saturated elementary extension [3].

Lemma 2. *Let $\mathfrak{A}$ and $\mathfrak{B}$ be two ω-saturated models, and let $\bar{c} \in A^n$ and $\bar{d} \in B^n$ be tuples so that $(\mathfrak{A}, \bar{c}) \equiv_\sigma (\mathfrak{B}, \bar{d})$. Then $(\mathfrak{A}, \bar{c}) \sim_\sigma (\mathfrak{B}, \bar{d})$.*

Proof. Consider the following set

$$\mathcal{Z} := \{p : \bar{a} \to \bar{b} \mid (\mathfrak{A}, \bar{a}) \equiv_\sigma (\mathfrak{B}, \bar{b})\}.$$

We claim that $\mathcal{Z}$ is a guarded σ-bisimulation between $(\mathfrak{A}, \bar{c})$ and $(\mathfrak{B}, \bar{d})$. We first note that by assumption $\bar{c} \mapsto \bar{d} \in \mathcal{Z}$, and hence $\mathcal{Z}$ satisfies (cover). $\mathcal{Z}$ also clearly consists of uniform partial σ-isomorphism between $\mathfrak{A}$ and $\mathfrak{B}$. What remains to be proved is that $\mathcal{Z}$ also satisfies (forth) and (back). Since these two cases are analogous, we will concentrate on (forth).

Let $p : \bar{a} \to \bar{b} \in \mathcal{Z}$, $a \in X$ and $X' := \{c_1, \ldots, c_m\} \subseteq A$ be a σ-live set so that $a \in X'$. For simplicity we will assume that $a = c_1$. Consider now the following elementary m-type

$$\Sigma := \{\varphi(p(a), x_2, \ldots, x_m) \in \mathcal{UGF}_1[\sigma \cup \{p(a)\}] \mid \mathfrak{A} \models \varphi(a, c_2, \ldots, c_m)\}.$$

We claim that Σ is realized in $(\mathfrak{B}, p(a))$. Since $\mathfrak{B}$ is ω-saturated, it suffices to show that each finite subset of Σ is realized in $(\mathfrak{B}, p(a))$. Let

$$\psi_1(p(a), x_2, \ldots, x_m), \ldots, \psi_r(p(a), x_2, \ldots, x_m) \in \Sigma.$$

Since X' is σ-live, there exists an atomic formula $\alpha(x_1, \ldots, x_m)$ over σ with the property that

$$\mathfrak{A} \models \exists x_2 \ldots \exists x_m (\alpha(a, x_2, \ldots, x_m) \wedge \bigwedge_{1 \le i \le r} \psi_i(a, x_2, \ldots, x_m)).$$

Note that Definition 6 guarantees that this is indeed a formula of $\mathcal{UGF}_1[\sigma]$. Since $(\mathfrak{A}, \bar{a}) \equiv_\sigma (\mathfrak{B}, \bar{b})$, we know that

$$\mathfrak{B} \models \exists x_2 \ldots \exists x_m (\alpha(p(a), x_2, \ldots, x_m) \wedge \bigwedge_{1 \le i \le r} \psi_i(p(a), x_2, \ldots, x_m)).$$

Thus $\{\psi_1(p(a), x_2, \ldots, x_m), \ldots, \psi_r(p(a), x_2, \ldots, x_m)\}$ is satisfiable in $(\mathfrak{B}, p(a))$, and hence Σ is satisfiable in $(\mathfrak{B}, p(a))$, say by the tuple $(p(a), d_2, \ldots, d_m)$. Now $\bar{c} \mapsto \bar{d} \in \mathcal{Z}$ is the mapping we were after.

Remark 2. Using the two previous lemmas one prove in a standard manner that $\mathcal{UGF}_1$ is the maximal fragment of $\mathcal{FO}$ which is invariant under uniform guarded bisimulation, see for example [2].

4   Proof that $\mathcal{UGF}_1$ has CIP

In this section we will prove that $\mathcal{UGF}_1$ has CIP. We will start with the following lemma.

Lemma 3. *Let σ and τ be signatures, and let $\varphi \in \mathcal{UGF}_1[\sigma]$ and $\psi \in \mathcal{UGF}_1[\tau]$. Suppose that there is no $\chi \in \mathcal{UGF}_1[\sigma \cap \tau]$ with the property that $\varphi \models \chi$ and $\chi \models \psi$. Then there is a σ-model $\mathfrak{A}$ and a τ-model $\mathfrak{B}$ with the property that $\mathfrak{A} \models \varphi$, $\mathfrak{B} \not\models \psi$ and $\mathfrak{A} \equiv_{\sigma \cap \tau} \mathfrak{B}$.*

Proof. Essentially the same argument as the one used in the proof of Theorem 4.1 in [2] gives the result.

To give a high level overview of the rest of the proof, suppose that the assumption of Lemma 3 holds for sentences φ and ψ, which implies in particular that there are models $\mathfrak{A}$ and $\mathfrak{B}$ so that $\mathfrak{A} \sim_{\sigma \cap \tau} \mathfrak{B}$. Now, what we want to prove is that $\varphi \wedge \neg \psi$ is satisfiable. To do this, we will follow a standard approach in modal logic [2,11] by constructing an *amalgam* $\mathfrak{U}$ which has the property that $\mathfrak{U} \sim_\sigma \mathfrak{A}$ and $\mathfrak{U} \sim_\tau \mathfrak{B}$. In particular, it will be a model of $\varphi \wedge \neg \psi$, since $\mathfrak{A} \models \varphi$ and $\mathfrak{B} \models \neg \psi$.

Suppose now that $\mathfrak{A} \sim_{\sigma \cap \tau} \mathfrak{B}$ and let $\mathcal{Z}$ be a guarded $(\sigma \cap \tau)$-bisimulation which witnesses it. Given a pair $(\bar{a}, \bar{b})$ we will use $(\bar{a}, \bar{b}) \in \mathcal{Z}$ to denote the fact that there exists $p \in \mathcal{Z}$ with the property that $\bar{a} = \operatorname{dom}(p)$ and $p(\bar{a}) = \bar{b}$. In other words the relation $a_i \mapsto b_i$ induces a uniform partial $(\sigma \cap \tau)$-isomorphism which belongs to $\mathcal{Z}$.

Before describing the construction of $\mathfrak{U}$, we need to introduce some additional notation. Given two tuples $\bar{a}$ and $\bar{b}$ of the same length, we will let $(\bar{a} \otimes \bar{b})$ denote the following tuple:

$$((a_1, b_1), \ldots, (a_n, b_n))$$

Given $(\bar{a} \otimes \bar{b})$, we say that it is *left-good*, if for every $1 \leq i < j \leq n$ we have that if $a_i = a_j$, then $b_i = b_j$. [2] Similarly we say that $(\bar{a} \otimes \bar{b})$ is *right-good*, if for every $1 \leq i < j \leq n$ we have that if $b_i = b_j$, then $a_i = a_j$. Finally we say that $(\bar{a} \otimes \bar{b})$ is *good* if it is left-good and right-good. Note that if $(\bar{a} \otimes \bar{b})$ is of length n, $k \geq n$ and $\mu : [k] \to [n]$ is a surjection, then we have that if $(\bar{a} \otimes \bar{b})$ is left-good, then so is $(\bar{a} \otimes \bar{b})_\mu$. Analogous observation of course holds for right-good and good.

As the domain of the amalgam $\mathfrak{U}$ we will take the set $U = \{(a,b) \in A \times B \mid (a,b) \in \mathcal{Z}\}$, while the interpretations of relation symbols will be defined as follows. First, for every $R \in \sigma \cap \tau$ we define that

$$(\bar{a} \otimes \bar{b}) \in R^{\mathfrak{U}} \text{ iff } \bar{a} \in R^{\mathfrak{A}} \text{ and } (\bar{a}, \bar{b}) \in \mathcal{Z}$$

Then, for every $R \in (\sigma \backslash \tau)$ we define that $(\bar{a} \otimes \bar{b}) \in R^{\mathfrak{U}}$ iff $\bar{a} \in R^{\mathfrak{A}}$ and one of the following conditions holds:

- $(\bar{a}, \bar{b}) \in \mathcal{Z}$.
- $(\bar{a} \otimes \bar{b})$ is left-good and $\bar{a}$ is not $(\sigma \cap \tau)$-live.

Similarly, for every $R \in (\tau \backslash \sigma)$ we define that $(\bar{a} \otimes \bar{b}) \in R^{\mathfrak{U}}$ iff $\bar{b} \in R^{\mathfrak{B}}$ and one of the following conditions holds:

- $(\bar{a}, \bar{b}) \in \mathcal{Z}$.
- $(\bar{a} \otimes \bar{b})$ is right-good and $\bar{b}$ is not $(\sigma \cap \tau)$-live.

This concludes the construction of $\mathfrak{U}$. This construction is similar to the one given in [11] with the exception that we require tuples that are not $(\sigma \cap \tau)$-live to be either right-good or left-good.

---

[2] In other words, if $(\bar{a} \otimes \bar{b})$ is left-good, then the projection $(\bar{a} \otimes \bar{b}) \mapsto \bar{a}$ is an injection.

We now define

$$\mathcal{Z}_1 := \{(\overline{a} \otimes \overline{b}) \mapsto \overline{a} \mid (\overline{a} \otimes \overline{b}) \text{ is } \sigma\text{-live in } \mathfrak{U}.\}$$

and

$$\mathcal{Z}_2 := \{(\overline{a} \otimes \overline{b}) \mapsto \overline{b} \mid (\overline{a} \otimes \overline{b}) \text{ is } \tau\text{-live in } \mathfrak{U}.\}$$

Note that if $(\overline{a} \otimes \overline{b})$ is σ-live, then by construction it is also left-good (and an analogous observation obviously holds for τ-live tuples in $\mathfrak{U}$).

Lemma 4. $\mathcal{Z}_1$ *consists of uniform partial σ-isomorphism between $\mathfrak{U}$ and $\mathfrak{A}$, and $\mathcal{Z}_2$ consists of uniform partial τ-isomorphism between $\mathfrak{U}$ and $\mathfrak{B}$.*

Proof. We will only consider the case of $\mathcal{Z}_1$, since the case of $\mathcal{Z}_2$ is analogous. Let $(\overline{a} \otimes \overline{b}) \mapsto \overline{a} \in \mathcal{Z}_1$, where the length of $(\overline{a} \otimes \overline{b})$ is n. We will separately check that this mapping preserves 1-types and n-ary atomic formulas.

Let $1 \leq i \leq n$ and suppose that

$$((a_i, b_i), \ldots, (a_i, b_i)) \in R^{\mathfrak{U}},$$

where $R \in \sigma$. By construction we know that $(a_i, \ldots, a_i) \in R^{\mathfrak{U}}$. Suppose then that

$$(a_i, \ldots, a_i) \in R^{\mathfrak{A}}.$$

Since by definition of U we have that $(a_i, b_i) \in \mathcal{Z}$, we can conclude that

$$((a_i, b_i), \ldots, (a_i, b_i)) \in R^{\mathfrak{U}}.$$

Thus (a_i, b_i) and a_i have the same 1-types over σ.

We will then verify that the mapping preserves n-ary atomic formulas. Let $R \in \sigma$ be a k-ary relation, where $k \geq n$, and let $\mu : [k] \to [n]$ be a surjection. We need to show that $(\overline{a} \otimes \overline{b})_\mu \in R^{\mathfrak{U}}$ iff $\overline{a}_\mu \in R^{\mathfrak{A}}$. Again, the left to right direction follows immediately from the definition of $\mathfrak{U}$, so we will concentrate on the direction from right to left. First we note that if $\overline{a}$ is not $(\sigma \cap \tau)$-live, then we are done, since then also $\overline{a}_\mu$ is not $(\sigma \cap \tau)$-live.

Thus we can assume that $\overline{a}$ is $(\sigma \cap \tau)$-live. Now, due to the definition of $\mathcal{Z}_1$, we know that $(\overline{a} \otimes \overline{b})$ is σ-live in $\mathfrak{U}$. Hence, by definition of $\mathfrak{U}$, and the fact that $\overline{a}$ is $(\sigma \cap \tau)$-live, we know that $(\overline{a}, \overline{b}) \in \mathcal{Z}$, which is the same as $(\overline{a}_\mu, \overline{b}_\mu) \in \mathcal{Z}$. Now we can deduce, due to the definition of $\mathfrak{U}$, that $(\overline{a} \otimes \overline{b})_\mu \in R^{\mathfrak{U}}$. This, together with the fact that $(\overline{a} \otimes \overline{b}) \mapsto \overline{a}$ preserves 1-types over σ, allows us to conclude that $\mathrm{tp}_{\mathfrak{U}}^\sigma[\overline{a} \otimes \overline{b}] = \mathrm{tp}_{\mathfrak{A}}^\sigma[\overline{a}]$.

Lemma 5. $\mathcal{Z}_1$ *is a guarded σ-bisimulation between $\mathfrak{U}$ and $\mathfrak{A}$, and $\mathcal{Z}_2$ is a guarded τ-bisimulation between $\mathfrak{U}$ and $\mathfrak{B}$.*

Proof. Again, we will only consider the case of $\mathcal{Z}_1$, since the case of $\mathcal{Z}_2$ is analogous. Due to Lemma 4 we just need to verify (back) and (forth) conditions. Let $(\overline{a} \otimes \overline{b}) \mapsto \overline{a} \in \mathcal{Z}_1$, where the length of $\overline{a}$ and $\overline{b}$ is n.

(forth) Let $(a_i, b_i) \in X$ and let $X' \subseteq U$ be a σ-live set so that $(a_i, b_i) \in X'$. Since X' is σ-live, we know that it is of the form $\{(c_1, d_1), \ldots, (c_m, d_m)\}$, with $(\overline{c} \otimes \overline{d})$ being left-good. Now $(\overline{c} \otimes \overline{d}) \mapsto \overline{c} \in \mathcal{Z}_1$ is the required mapping.

(back) Let $a_i \in Y$ and let $Y' \subseteq A$ be a σ-live set so that $a_i \in Y$. For concreteness, suppose that $Y' = \{c_1, \ldots, c_m\}$. Consider first the case that Y' is not $(\sigma \cap \tau)$-live in $\mathfrak{A}$. For every $2 \leq i \leq m$ we will pick an element d_i so that $(c_i, d_i) \in \mathcal{Z}$. Note that such elements exists since each singleton is a live element. By construction $\{(c_1, d_1), \ldots, (c_m, d_m)\}$ is σ-live, and hence $(\overline{c} \otimes \overline{d}) \mapsto \overline{c} \in \mathcal{Z}_1$ is the required mapping we were after.

Suppose then that Y' is $(\sigma \cap \tau)$-live in $\mathfrak{A}$. Since $(a_i, b_i) \in U$, we know that $(a_i, b_i) \in \mathcal{Z}$. Since $\mathcal{Z}$ is a guarded $(\sigma \cap \tau)$-bisimulation, there exists a set $\{d_1, \ldots, d_m\} \subseteq B$ so that $(\overline{c}, \overline{d}) \in \mathcal{Z}$ and $(a_i, b_i) \in (\overline{c} \otimes \overline{d})$. In particular $(\overline{c} \otimes \overline{d})$ is σ-live in $\mathfrak{U}$, and hence $(\overline{c} \otimes \overline{d}) \mapsto \overline{d} \in \mathcal{Z}_1$, which is the mapping we were after.

Theorem 1. $\mathcal{UGF}_1$ *has Craig interpolation property.*

Proof. Let $\varphi \in \mathcal{UGF}_1[\sigma]$ and $\psi \in \mathcal{UGF}_1[\tau]$ be sentences so that $\varphi \models \psi$, but there is no interpolant for this entailment. By lemma 3 there exists a σ-model $\mathfrak{A}$ and a τ-model $\mathfrak{B}$ such that $\mathfrak{A} \models \varphi$, $\mathfrak{B} \not\models \psi$ and $\mathfrak{A} \equiv_{\sigma \cap \tau} \mathfrak{B}$. Take ω-saturated elementary extensions $\hat{\mathfrak{A}}$ and $\hat{\mathfrak{B}}$ of $\mathfrak{A}$ and $\mathfrak{B}$. Since $\hat{\mathfrak{A}} \equiv_{\sigma \cap \tau} \hat{\mathfrak{B}}$, by lemma 2 we have that $\hat{\mathfrak{A}} \sim_{\sigma \cap \tau} \hat{\mathfrak{B}}$. Using the construction presented in this section there exists a $(\sigma \cup \tau)$-model $\mathfrak{U}$ with the property that $\mathfrak{U} \sim_\sigma \hat{\mathfrak{A}}$ and $\mathfrak{U} \sim_\tau \hat{\mathfrak{B}}$. Thus $\mathfrak{U} \models \varphi \wedge \neg \psi$, i.e. $\varphi \wedge \neg \psi$ is consistent, which is a contradiction with the assumption that $\varphi \models \psi$.

5   Complexity of uniform $\mathcal{GF}$

In this section we will prove that the complexity of the satisfiability problem of uniform $\mathcal{GF}$ is in NExpTime. Since it was proved in [15] that the complexity of the satisfiability problem of $\mathcal{UGF}_1$ is NExpTime-hard, this upper bound is sharp.

5.1   Scott normal form

As usual, we will start by arguing that we can restrict our attention to sentences which are in a certain normal form. The normal form that we will use here has a somewhat awkward form, but the proof of Lemma 6 should clarify why we chose to use it.

Definition 8. *Let φ be a sentence of $\mathcal{UGF}$. We say that φ is in normal form, if it has the following shape*

$$\bigwedge_{t \in T} \exists z \lambda_t(z) \wedge \bigwedge_{i \in I} \forall \overline{x} (\alpha_i(\overline{x}) \to \exists \overline{y} (\beta_i(\overline{x}, \overline{y}) \wedge \psi_i(\overline{x}, \overline{y})))$$

$$\wedge \bigwedge_{j \in J} \forall \overline{x} (\kappa_j(\overline{x}) \to (\theta_j(\overline{x}) \to \forall \overline{y} (\gamma_j(\overline{x}, \overline{y}) \to \psi_j(\overline{x}, \overline{y})))),$$

where T, I, J are non-empty (finite) sets, λ_t, α_i, β_i, κ_j and γ_j are atomic formulas and ψ_i, θ_j and ψ_j are quantifier-free formulas.

Remark 3. In the definition of the normal form we do not require that the tuples $\overline{y}$ are necessarily non-empty, i.e., we allow formulas of the form $\forall \overline{x}(\alpha_i(\overline{x}) \rightarrow \psi_i(\overline{x}))$ in our normal forms. However, we do require that the tuples $\overline{x}$ are non-empty, and hence we do not allow formulas of the form $\exists \overline{y}(\beta_i(\overline{y}) \wedge \psi_i(\overline{y}))$, where the length of $\overline{y}$ is more than one.

If φ is a sentence of $\mathcal{UGF}$ in normal form, then we refer to its conjuncts of the form

$$\forall \overline{x}(\alpha_i(\overline{x}) \rightarrow \exists \overline{y}(\beta_i(\overline{x}, \overline{y}) \wedge \psi_i(\overline{x}, \overline{y})))$$

as the *existential requirements* and we will use $\varphi_i^{\exists}$ to denote them. Given a model $\mathfrak{A}$, an existential requirement $\varphi_i^{\exists}$ and $\overline{a} \in \alpha_i^{\mathfrak{A}}$ we say that a tuple $\overline{c}$ is a *witness for $\varphi_i^{\exists}$ and $\overline{a}$* if

$$\mathfrak{A} \models \beta_i(\overline{a}, \overline{c}) \wedge \psi_i(\overline{a}, \overline{c}).$$

Conjuncts of the form

$$\forall \overline{x}(\kappa_j(\overline{x}) \rightarrow (\theta_j(\overline{x}) \rightarrow \forall \overline{y}(\gamma_j(\overline{x}, \overline{y}) \rightarrow \psi_j(\overline{x}, \overline{y}))))$$

will be referred to as the *universal requirements* and we will use $\varphi_j^{\forall}$ to denote them.

Using standard renaming techniques one can establish the following.

Lemma 6. *There is a polynomial nondeterministic procedure, taking as its input a sentence $\varphi \in \mathcal{UGF}[\sigma]$ and producing a sentence $\varphi' \in \mathcal{UGF}[\sigma']$ in normal form, where $\sigma' \supset \sigma$, such that*

1. *if $\mathfrak{A} \models \varphi$ for some model $\mathfrak{A}$, then there is a run of the procedure producing a normal form φ' such that $\mathfrak{A}' \models \varphi'$ for some expansion $\mathfrak{A}'$ of $\mathfrak{A}$ to the vocabulary σ',*
2. *if the procedure has a run producing φ' and $\mathfrak{A}' \models \varphi'$, for some $\mathfrak{A}'$, then the σ-reduct $\mathfrak{A}$ of $\mathfrak{A}'$ satisfies φ.*

Proof. We will essentially follow the proof of lemma 1 in [15], with some small technical modifications. Let $\varphi \in \mathcal{UGF}[\sigma]$ be a sentence, which w.l.o.g contains only existential quantification. Let ψ be the innermost formula of φ which starts with a block of existential quantifiers. If ψ is a sentence, we will nondeterministically either replace it with $\bot$ or $\top$ and add ψ or $\neg\psi$ (depending on our guess) as a conjunct to the resulting formula. Suppose then that ψ is a formula of the form

$$\exists \overline{y}(\alpha(\overline{x}, \overline{y}) \wedge \psi(\overline{x}, \overline{y})).$$

Since φ was a sentence, ψ occurs in a scope of another formula of the form

$$\exists \overline{z}(\alpha'(\overline{x}) \wedge \psi'(\overline{x})),$$

where $\overline{z} \subseteq \overline{x}$. Let α' be the guard of the innermost such formula. We will now replace φ with the following formula

$$\varphi[\psi(\overline{x})/R(\overline{x})] \wedge \forall \overline{x}(R(\overline{x}) \rightarrow \exists \overline{y}(\alpha(\overline{x}, \overline{y}) \wedge \psi(\overline{x}, \overline{y})))$$

$$\wedge \forall \overline{x}(\alpha'(\overline{x}) \rightarrow (\neg R(\overline{x}) \rightarrow \forall \overline{y}(\alpha(\overline{x}, \overline{y}) \rightarrow \neg \psi(\overline{x}, \overline{y})))),$$

where $\varphi[\psi(\overline{x})/R(\overline{x})]$ is the sentence obtained from φ by replacing the previously mentioned subformula $\psi(\overline{x})$ with the atomic formula $R(\overline{x})$ which has a fresh relation symbol R. It is straightforward to verify that the resulting sentence is equi-satisfiable with φ.

Now one can repeat the above procedure until one is left with a sentence of the form

$$\bigwedge_{t \in T} \exists \overline{x}(\alpha_t(\overline{x}) \wedge \psi_t(\overline{x})) \wedge \bigwedge_{i \in I} \varphi_i^{\exists} \wedge \bigwedge_{j \in J} \varphi_j^{\forall},$$

where each $\varphi_i^{\exists}$ is an existential requirement, while each sentence $\varphi_j^{\forall}$ is an universal requirement. Now one can replace each conjunct $\exists x_1 \ldots \exists x_n(\alpha(x_1, \ldots, x_n) \wedge \psi_t(x_1 \ldots x_n))$ with a sentence of the form

$$\exists x \lambda_t(x) \wedge \forall x_1(\lambda_t(x_1) \rightarrow \exists x_2 \ldots \exists x_n(\alpha_t(x_1, \ldots, x_n) \wedge \psi_t(x_1, \ldots, x_n))),$$

where λ_t is a fresh unary relation symbol. The resulting sentence is clearly equi-satisfiable with the original sentence and furthermore it is in normal form.

5.2   Satisfiability Witnesses

A standard technique in proving that the complexity of the satisfiability problem of a given fragment of $\mathcal{FO}$ is in NExpTime is to show that each satisfiable sentence of this fragment has a finite model of size at most exponential with respect to the length of the sentence [8,12,15,16]. However, in the case of $\mathcal{UGF}$ it seems to be easier to show that we can associate to each of its sentences φ a different type of certificate, which is still at most exponential with respect to the length of the sentence, and which can be used to construct a (potentially infinite) model for φ.

Definition 9. *Let $\varphi \in \mathcal{UGF}[\sigma]$ be a sentence in normal form, P be a set of 1-types over σ and $\pi \in P$. A pair $(\mathfrak{A}, c)$, where $c \in A$, is called a (P, π)-witness for φ, if it satisfies the following requirements.*

1. *$\mathrm{tp}_{\mathfrak{A}}[c] = \pi$.*
2. *For every $a \in A$ we have that $\mathrm{tp}_{\mathfrak{A}}[a] \in P$.*
3. *For every existential requirement $\varphi_i^{\exists}$ and for every tuple $\overline{a}$ which contains c we have that if $\mathfrak{A} \models \alpha_i(\overline{a})$, then there exists a witness for φ_i and $\overline{a}$.*
4. *For every universal requirement $\varphi_j^{\forall}$ and for every tuple $\overline{a}$ which contains c we have that if $\mathfrak{A} \models \kappa_j(\overline{a}) \wedge \theta_j(\overline{a})$, then for every tuple $\overline{b}$ we have that*

$$\mathfrak{A} \models \gamma_j(\overline{a}, \overline{b}) \rightarrow \psi_j(\overline{a}, \overline{b}).$$

Here the intuition is that a (P, π)-witness $(\mathfrak{A}, c)$ is a local certificate; it certifies that we can provide witnesses for tuples which contain the element c. The main idea now is that if we have a (P, π)-witness for each $\pi \in P$, then we can use them to construct a proper model for φ.

Definition 10. *Let $\varphi \in \mathcal{UGF}[\sigma]$ be a sentence in normal form. A set of 1-types P over σ is a witness for φ, if it satisfies the following two requirements.*

1. *For every conjunct $\exists z \lambda_t(z)$ there exists $\pi \in P$ so that $\lambda_t(x) \in \pi$.*
2. *For every $\pi \in P$ there exists a (P, π)-witness for P.*

The following lemmas prove that an existence of a witness for φ is equivalent with the satisfiability of φ.

Lemma 7. *Let $\varphi \in \mathcal{UGF}$ be a sentence in normal form. If φ is satisfiable, then there exists a witness for it.*

Proof. Suppose that $\mathfrak{A} \models \varphi$. As the set of 1-types P we can take the set

$$\{\mathrm{tp}_{\mathfrak{A}}[a] \mid a \in A\}.$$

Clearly for every conjunct $\exists z \lambda_t(z)$ there exists a suitable 1-type in P. Towards verifying the second requirement let $\pi \in P$ and let $c \in A$ be an element which realizes π. Then $(\mathfrak{A}, c)$ is clearly a (P, π)-witness for φ.

Lemma 8. *Let $\varphi \in \mathcal{UGF}$ be a sentence in normal form. If there exists a witness for φ, then it is satisfiable.*

Proof. For simplicity we will assume that φ contains exactly one conjunct of the form $\exists z \lambda_t(z)$. Let P be a witness for φ. Thus for every $\pi \in P$ there exists a pair $(\mathfrak{A}^\pi, c)$ which is a (P, π)-witness for φ. Our goal is to use these witnesses to construct a sequence of models

$$\mathfrak{A}_1 \leq \mathfrak{A}_2 \leq \mathfrak{A}_3 \leq \ldots$$

so that their union is a model of φ.

Let $\pi \in P$ be a 1-type so that $\pi \models \lambda_t$. As the model $\mathfrak{A}_1$ we will take the model which contains a single element with 1-type π. Suppose then that we have defined $\mathfrak{A}_n$ in such a way that each 1-type realized in $\mathfrak{A}_n$ belongs to P. To define the model $\mathfrak{A}_{n+1}$ we will proceed as follows. Given $a \in \mathfrak{A}_n$, we will use W_a to denote the set $A^\pi - \{c\}$, where A^π refers to the domain of the model in the (P, π)-witness $(\mathfrak{A}^\pi, c)$ of $\pi := \mathrm{tp}_{\mathfrak{A}_n}[a]$. Without loss of generality we will assume that the sets W_a are pairwise disjoint. Now we will define $\mathfrak{A}_{n+1}$ as follows.

- The domain of the model is

$$A_n \cup \bigcup_{a \in A_n} W_a^*$$

- $\mathfrak{A}_{n+1} \restriction A_n$ is defined to be isomorphic with A_n.

- For each $a \in A_n$ and for each $\{c_1, \ldots, c_m\} \subseteq W_a$, we define that

$$\mathrm{tp}_{\mathfrak{A}_{n+1}}[a, c_1, \ldots, c_m] := \mathrm{tp}_{\mathfrak{A}^\pi}[c, c_1, \ldots, c_m],$$

where π is the 1-type of a.
- For every tuple $(a_1, \ldots, a_m)$ and a m-ary relation R for which we have not yet defined whether $(a_1, \ldots, a_m)$ belongs to $R^{\mathfrak{A}_{n+1}}$, we will simply define that it does not belong to it.

The last step guarantees that if a tuple, which contains more than one element, is live in $\mathfrak{A}_{n+1}$, then it was already alive in one of the models $\mathfrak{A}^\pi$. It is straightforward to verify that the union of the models $(\mathfrak{A}_n)_{n<\omega}$ is indeed a model of φ.

5.3   Complexity of $\mathcal{UGF}$

Although the size of a witness for φ is clearly only exponential with respect to $|\varphi|$, we do not yet have any upper bounds on the time it takes to verify that it really is a witness for φ. The following lemma gives us such a bound.

Lemma 9. *Let $\varphi \in \mathcal{UGF}$ be a sentence in normal form and let σ denote the vocabulary of φ. Let P be a set of 1-types over σ and $\pi \in P$. If there exists a (P, π)-witness for φ, then there exists one in which the size of the model is at most $2^{|\varphi|^{O(1)}}$.*

Proof. Let $(\mathfrak{A}, c)$ be a (P, π)-witness for φ and let $m = \max\{ar(R) \mid R \in \sigma\}$. Note that $m \leq |\varphi|$. Our goal is to construct a sequence

$$\mathfrak{B}_1 \leq \cdots \leq \mathfrak{B}_m$$

of models so that $(\mathfrak{B}_m, c)$ is a (P, π)-witness for φ and $|B_m| \leq 2^{|\varphi|^{O(1)}}$. As the model $\mathfrak{B}_1$ we will take the model which contains a single element with 1-type π; let e denote this element.

Before moving forward, we will introduce one auxiliary definition. Let $\bar{a} = (a_1, \ldots, a_n)$ and $\bar{b} = (b_1, \ldots, b_n)$ be tuples of elements from two models $\mathfrak{A}$ and $\mathfrak{B}$. Let $\{c_1, \ldots, c_m\}$ denote the set of distinct elements in $\bar{a}$. We say that $\bar{a}$ and $\bar{b}$ are *similar*, if the mapping $p : \bar{a} \to \bar{b}$, which was the mapping induced by the relation $a_i \mapsto b_i$, is a bijection and furthermore

$$\mathrm{tp}_{\mathfrak{A}}[c_1, \ldots, c_n] = \mathrm{tp}_{\mathfrak{B}}[p(c_1), \ldots, p(c_n)].$$

Suppose now that we have defined $\mathfrak{B}_k$, where $k < m$, and in such a way that for each σ-live tuple $\bar{b}$ for which $\mathrm{tp}_{\mathfrak{B}_k}[\bar{b}]$ has been defined, there exists a similar tuple $\bar{a}$ which consists of elements of $\mathfrak{A}$. Given an existential requirement $\varphi_i^{\exists}$ of φ and a tuple $\bar{b} \in \alpha_i^{\mathfrak{B}_k}$, which contains the element e, we say that $\bar{b}$ is a *i-defect* if there exists no witness for $\varphi_i^{\exists}$ and $\bar{b}$ in the model $\mathfrak{B}_k$. By construction, for each i-defect $\bar{b}$ we can find a tuple $\bar{a}$ of elements of $\mathfrak{A}$ so that $\bar{b}$ and $\bar{a}$ are similar. In particular $\bar{a} \in \alpha_i^{\mathfrak{A}}$, and hence there exists a witness $\bar{c}$ for $\varphi_i^{\exists}$ and $\bar{a}$ in $\mathfrak{A}$; let $W_{\bar{b}, i}$

denote the set of elements in $\bar{c}$ which were not contained in $\bar{a}$. Without loss of generality we will assume that the sets $W_{\bar{b},i}$ are pairwise disjoint. Now we will define $\mathfrak{B}_{k+1}$ as follows.

- The domain of the model is

$$B_k \cup \bigcup_{i \in I} \bigcup_{\bar{b} \text{ an } i\text{-defect}} W_{\bar{b},i}$$

- $\mathfrak{B}_{k+1} \restriction B_k$ is defined to be isomorphic with $\mathfrak{B}_k$.
- For each i-defect $\bar{b}$ and a set $W_{\bar{b},i} = \{c_1, \ldots, c_n\}$ we define that

$$\mathrm{tp}_{\mathfrak{B}_{k+1}}[d_1, \ldots, d_r, c_1, \ldots, c_n] = \mathrm{tp}_{\mathfrak{A}}[p(d_1), \ldots, p(d_r), c_1, \ldots, c_n],$$

  where $(d_1, \ldots, d_r)$ enumerates all the elements occurring in $\bar{b}$ and $p : \bar{b} \to \bar{a}$.
- For every tuple $(b_1, \ldots, b_n)$ and a n-ary relation R for which we have not yet defined whether $(b_1, \ldots, b_n)$ belongs to $R^{\mathfrak{B}_{k+1}}$, we will simply define that it does not belong to it.

This completes the construction of the models $\mathfrak{B}_1, \ldots, \mathfrak{B}_m$. To bound the size of $\mathfrak{B}_m$, we first note that $|B_{k+1}| \leq |B_k| + |\varphi||D_k|$, where D_k denotes the number of defects in $\mathfrak{B}_k$. By construction, for every defect $(d_1, \ldots, d_r)$ of $\mathfrak{B}_k$ the set $\{d_1, \ldots, d_r\}$ is a σ-live set which is not contained in $\mathfrak{B}_\ell$, for any $\ell < k$. If $k = 1$, then the number of such σ-live sets is one, and if $k > 1$, then the number of such σ-live sets is D_{k-1}. Since each σ-live set is of size at most $|\varphi|$, there are at most $|\varphi||\varphi|^{|\varphi|}D_{k-1} = 2^{|\varphi|^{O(1)}}D_{k-1}$ defects in $\mathfrak{B}_k$, i.e., $D_k \leq 2^{|\varphi|^{O(1)}}D_{k-1}$. Since $m \leq |\varphi|$, we have that $D_k \leq 2^{|\varphi|^{O(1)}}$, for any $k < m$, and hence $|B_m| \leq 2^{|\varphi|^{O(1)}}$.

Thus what remains to be proven is that $(\mathfrak{B}_m, e)$ is a (P, π)-witness for φ. Here the only non-trivial requirement that we need to verify is that $\mathfrak{B}_m$ satisfies the second item in definition 9. So, let $\varphi_i^{\exists}$ be an existential requirement and let $\bar{b} = (b_1, \ldots, b_n) \in \alpha_i^{\mathfrak{B}_m}$ be a tuple which contains e. We can clearly assume that $n < m$. It suffices to show that $\bar{b}$ is contained in $\mathfrak{B}_k$, for some $k < m$, since then by construction we know that it has a witness in $\mathfrak{B}_m$.

Aiming for a contradiction, suppose that $\bar{b}$ is contained in $\mathfrak{B}_m$, but it is not contained in $\mathfrak{B}_k$ for any $k < m$. By construction we know that, since $\bar{b}$ is σ-live, we assigned a table to some tuple $(b'_1, \ldots, b'_r)$, where $(b'_1, \ldots, b'_r)$ enumerates the set of distinct elements of $(b_1, \ldots, b_n)$. Again, by construction we know that we assigned a table to the tuple $(b'_1, \ldots, b'_r)$, because we wanted to provide a witness for some tuple $(d_1, \ldots, d_s)$, which contains e and for which $\{d_1, \ldots, d_s\}$ is a *strict* subset of $\{b'_1, \ldots, b'_r\}$.[3]

Now observe that $(d_1, \ldots, d_s)$ is a σ-live tuple containing e, which is contained in $\mathfrak{B}_{m-1}$ but is not contained in $\mathfrak{B}_k$ for any $k < m-1$. Indeed, if it were contained in $\mathfrak{B}_k$, for some $k < m-1$, then by construction we would have provided a witness for it in the model $\mathfrak{B}_{k+1}$, i.e., $(b_1, \ldots, b_n)$ would have been contained in $\mathfrak{B}_{k+1}$. But now we are in a position which is the same as the one that we started in; in

---

[3] If it were not, there would have been no need to provide a witness for it.

particular, we can repeat the above argument. After repeating the argument (at least) $(n-1)$-times we would end up with the conclusion that e is contained in some $\mathfrak{B}_k$, where $k > 1$, but it is not contained in $\mathfrak{B}_1$, which would be an obvious contradiction.

Now we can prove the main theorem of this section.

Theorem 2. *The satisfiability problem of* $\mathcal{UGF}$ *is* NExpTime-*complete.*

Proof. The lower bound follows from the proof of Theorem 3 in [15]. We will give an informal description of a non-deterministic procedure running in exponential time which determines whether a given sentence $\varphi \in \mathcal{UGF}$ is satisfiable. It starts by converting φ into an equi-satisfiable sentence $\varphi' \in \mathcal{UGF}$ in normal form, after which it guesses a set of 1-types P over the vocabulary of φ' and for each $\pi \in P$ a (P, π)-witness $(\mathfrak{A}, c)$ for φ, where the size of $\mathfrak{A}$ is at most $2^{|\varphi|^{O(1)}}$. Lemmas 6, 7, 8 and 9 guarantee that this procedure is correct. Since $|P| \leq 2^{|\varphi|}$, the algorithm runs in exponential time with respect to $|\varphi|$.

6   Conclusions

In this paper we have proved two results of quite distinct flavour on uniform guarded fragments. The first result was that although $\mathcal{GF}$ fails to have Craig interpolation, its one-dimensional uniform fragment does have it. The second result was that the complexity of the satisfiability problem of the uniform guarded fragment is NExpTime-complete. The results presented in this paper suggest several new research questions, but here we will mention just two of them.

The first question is whether or not the uniform $\mathcal{GF}$ has Craig interpolation property. While the correctness of the amalgam construction presented in Section 4 rests on the assumption of one-dimensionality, we have not been able to show that uniform $\mathcal{GF}$ would not have Craig interpolation property. This has led the author to conjecture that the uniform $\mathcal{GF}$ does in fact have Craig interpolation property.

The second question is whether or not uniform $\mathcal{GF}$ has the exponential model property (note that if uniform $\mathcal{GF}$ would have an exponential model property, then one would obtain Theorem 2 for free). As we saw in the proof of Lemma 9, the requirement of uniformity essentially prevents uniform $\mathcal{GF}$ from enforcing long paths, and this seems to suggest that uniform $\mathcal{GF}$ can only enforce exponentially long paths (which it can enforce, since it contains standard modal logic with the global diamond). Because of this, the author conjectures that uniform $\mathcal{GF}$ has the exponential model property.

of determining the complexity of the satisfiability problem of uniform guarded fragment. The author also wishes to thank Antti Kuusisto for pointing out rather silly mistakes in the original definitions of uniformity and the uniform guarded bisimulation. Finally, the author wishes to thank the anonymous reviewers for their useful remarks which improved the presentation of this paper.

References

1. Andréka, H., Németi, I., van Benthem, J.: Modal languages and bounded fragments of predicate logic. Journal of Philosophical Logic **27**, 217–274 (1998)
2. Bárány, V., Benedikt, M., Cate, B.T.: Some model theory of guarded negation. The Journal of Symbolic Logic **83**, 1307 – 1344 (2018)
3. Bell, J., Slomson, A.: Models and Ultraproducts: An Introduction. Dover Books on Mathematics Series, Dover Publications (2006)
4. van Benthem, J.: The many faces of interpolation. Synthese **164**(3), 451–460 (2008)
5. Blackburn, P., Rijke, M.d., Venema, Y.: Modal Logic. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press (2001)
6. Gabbay, D.M.: Craig's interpolation theorem for modal logics. In: Hodges, W. (ed.) Conference in Mathematical Logic — London '70. pp. 111–127. Springer Berlin Heidelberg, Berlin, Heidelberg (1972)
7. Grädel, E.: On the restraining power of guards. Journal of Symbolic Logic **64**(4), 1719–1742 (1999)
8. Grädel, E., Kolaitis, P., Vardi, M.: On the decision problem for two-variable first-order logic. Bulletin of Symbolic Logic **3**(1), 53–69 (1997)
9. Hella, L., Kuusisto, A.: One-dimensional fragment of first-order logic. In: Advances in Modal Logic. vol. 10, pp. 274–293 (08 2014)
10. Herzig, A.: A new decidable fragment of first order logic. In: 3rd Logical Biennial Summer School and Conference in Honour of SC Kleene (1990)
11. Hoogland, E., Marx, M.: Interpolation and definability in guarded fragments. Studia Logica: An International Journal for Symbolic Logic **70**(3), 373–409 (2002)
12. Jaakkola, R.: Ordered Fragments of First-Order Logic. In: 46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021). Leibniz International Proceedings in Informatics (LIPIcs), vol. 202, pp. 62:1–62:14 (2021)
13. Jung, J.C., Wolter, F.: Living without beth and craig: Definitions and interpolants in the guarded and two-variable fragments. 2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS) pp. 1–14 (2021)
14. Kieronski, E.: One-Dimensional Logic over Words. In: 25th EACSL Annual Conference on Computer Science Logic (CSL 2016). Leibniz International Proceedings in Informatics (LIPIcs), vol. 62, pp. 38:1–38:15 (2016)
15. Kieronski, E.: One-Dimensional Guarded Fragments. In: 44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019). Leibniz International Proceedings in Informatics (LIPIcs), vol. 138, pp. 16:1–16:14 (2019)
16. Kieronski, E., Kuusisto, A.: Complexity and expressivity of uniform one-dimensional fragment with equality. In: 39nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2014). Lecture Notes in Computer Science, vol. 8634, pp. 365–376 (2014)
17. Kierónski, E., Kuusisto, A.: Uniform One-Dimensional Fragments with One Equivalence Relation. In: 24th EACSL Annual Conference on Computer Science Logic (CSL 2015). Leibniz International Proceedings in Informatics (LIPIcs), vol. 41, pp. 597–615 (2015)

18. Kieronski, E., Kuusisto, A.: One-Dimensional Logic over Trees. In: 42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017). Leibniz International Proceedings in Informatics (LIPIcs), vol. 83, pp. 64:1–64:13 (2017)
19. Marx, M.: Interpolation in modal logic. In: Proceedings of the 7th International Conference on Algebraic Methodology and Software Technology. p. 154–163. AMAST '98, Springer-Verlag, Berlin, Heidelberg (1999)
20. Mogavero, F., Perelli, G.: Binding Forms in First-Order Logic. In: 24th EACSL Annual Conference on Computer Science Logic (CSL 2015). Leibniz International Proceedings in Informatics (LIPIcs), vol. 41, pp. 648–665 (2015)

Sweedler Theory of Monads

Dylan McDermott[1](✉) , Exequiel Rivas[2] , and Tarmo Uustalu[1,2]

[1] Dept. of Computer Science, Reykjavik University, Reykjavik, Iceland
[2] Dept. of Software Science, Tallinn University of Technology, Tallinn, Estonia
`dylanm@ru.is, exequiel.rivas@ttu.ee, tarmo@ru.is`

Abstract. Monad-comonad interaction laws are a mathematical concept for describing communication protocols between effectful computations and coeffectful environments in the paradigm where notions of effectful computation are modelled by monads and notions of coeffectful environment by comonads. We show that monad-comonad interaction laws are an instance of measuring maps from Sweedler theory for duoidal categories whereby the final interacting comonad for a monad and a residual monad arises as the Sweedler hom and the initial residual monad for a monad and an interacting comonad as the Sweedler copower. We then combine this with a (co)algebraic characterization of monad-comonad interaction laws to derive descriptions of the Sweedler hom and the Sweedler copower in terms of their coalgebras resp. algebras.

Keywords: (co)monads · (co)algebras · interaction laws · runners · duoidal categories · Sweedler operations

1 Introduction

The monad-comonad interaction laws of Katsumata et al. [16] are a mathematical concept for formalizing ways in which effectful programs (e.g., programs reading from and writing to a store, programs making nondeterministic choices) can be run. The idea is that effectful programs issue requests to the outside world; they can thus run on machines that can service such requests. Programs denote computations, machines implement environments. Notions of computation are modelled by monads in the manner first explained by Moggi [23], while notions of environment can be modelled by comonads. Interaction laws model protocols of cooperation between computations and environments. Ideally, interaction should result in a return value and a final state. But it may be that some effects cannot be serviced, in which case interaction yields a residual computation of a return value and a final state; another monad is then needed to model the suitable notion of residual computation. A monad-comonad interaction law is therefore given by a monad T, a comonad D and a monad R on a symmetric monoidal category with a family of maps $TX \otimes DY \to R(X \otimes Y)$ natural in X and Y and agreeing with the (co)units and (co)multiplications. If $R = \mathsf{Id}$, we have a non-residual interaction law.

It is natural to ask for useful methods for recognizing and constructing monad-comonad interaction laws. Specifically, it would be useful to find: a final

monad for a given interacting comonad and residual monad; a final interacting comonad for a given monad and residual monad; or an initial residual monad for a given monad and interacting comonad.

In this paper, we show how to find these universal (co)monads, elaborating on some ideas and results from prior work on interaction [16,33]. We emphasize that the most important structural foundation for interaction laws is the duoidal [10,2] interrelationship of the composition and Day convolution monoidal structures in endofunctor categories. It is so significant that some central statements about interaction laws can be made on the level of monoids and comonoids in general symmetric closed duoidal categories, completely suppressing any specifics about monads and comonads. In fact, it turns out that monad-comonad interaction laws are an instance of measuring maps from the Sweedler theory for duoidal categories as developed by López Franco and Vasilakopoulou [20]. The universal (co)monads are instances of the operations studied in this theory. In particular, the final interacting comonad is an instance of the Sweedler hom and the initial residual monad is an instance of the Sweedler copower.

To obtain results about monad-comonad interaction specifically, we combine this general perspective with the characterization of monad-comonad interaction laws by Uustalu and Voorneveld [33] as functors between the categories of (co)algebras of the (co)monads involved. This allows us to describe the Sweedler hom and the Sweedler power via their categories of (co)algebras in terms of what we call stateful and continuation-based runners.

We also discuss an enriched version of monad-comonad interaction laws, of which strong monad-comonad interaction laws are a special case. In this case, both kinds of runners of an enriched monad on a self-enriched category can be viewed as its algebras in another enriched category.

The paper is organized as follows. First, in Sect. 2, we review the basics of monad-comonad interaction laws. In Sect. 3, we show that monad-comonad interaction laws, the universal interacting comonad and the universal residual monad are an instance of measuring maps, the Sweedler hom and the Sweedler copower in symmetric closed duoidal categories. We then review the (co)algebraic perspective on monad-comonad interaction laws in Sect. 4, and apply it to derive (co)algebraic characterizations of the Sweedler hom and the Sweedler copower in Sect. 5. In Sect. 6, we comment on enriched monad-comonad interaction laws. We review some background category theory literature and related semantics work in Sect. 7. New material is primarily in Sects. 5, 6; some statements in Sect. 4 are also new.

We assume from the reader familiarity with the use of (strong) monads in mathematical semantics to model notions of effectful computation, and familiarity with the basics of the categorical machinery we need (monads and comonads, symmetric monoidal closed categories, accessibility [21,1], enrichment [17]).

2   Monad-Comonad Interaction Laws

We begin by reviewing the basics of monad-comonad interaction laws [16].

Consider a symmetric monoidal closed category $(\mathbb{C}, I, \otimes, \multimap)$, e.g., a Cartesian monoidal closed category, e.g., **Set**.

A *(residual) functor-functor interaction law* is given by endofunctors F, G, H on $\mathbb{C}$ together with a family of maps

$$\phi_{X,Y} : FX \otimes GY \to H(X \otimes Y)$$

natural in X, Y. We speak of a non-residual interaction law when $H = \mathsf{Id}$. A *map between (residual) functor-functor interaction laws* (F, G, H, ϕ) and (F', G', H', ϕ') is given by natural transformations $f : F \to F'$, $g : G' \to G$ and $h : H \to H'$ satisfying the equation

$$
\begin{array}{ccc}
 & FX \otimes GY \xrightarrow{\psi_{X,Y}} H(X \otimes Y) \\
FX \otimes G'Y \nearrow^{FX \otimes g_Y} & & \downarrow h_{X \otimes Y} \\
\searrow_{f_X \otimes G'Y} & F'X \otimes G'Y \xrightarrow{\psi'_{X,Y}} H'(X \otimes Y)
\end{array}
$$

Functor-functor interaction laws form a category that has a monoidal structure based on endofunctor composition.

A *(residual) monad-comonad interaction law* is given by a monad T, a comonad D and a monad R on $\mathbb{C}$ with a family of maps

$$\psi_{X,Y} : TX \otimes DY \to R(X \otimes Y)$$

natural in X, Y, that additionally satisfies the equations

$$
\begin{array}{ccc}
 & X \otimes Y =\!=\!= X \otimes Y \\
X \otimes DY \nearrow^{\mathsf{id} \otimes \varepsilon_Y} & & \downarrow \eta^R_{X,Y} \\
\searrow_{\eta_X \otimes \mathsf{id}} & TX \otimes DY \xrightarrow{\psi_{X,Y}} R(X \otimes Y)
\end{array}
\qquad
\begin{array}{ccc}
 & TTX \otimes DDY \xrightarrow{\psi_{TX,DY}} R(TX \otimes DY) \xrightarrow{R\psi_{X,Y}} RR(X \otimes Y) \\
TTX \otimes DY \nearrow^{\mathsf{id} \otimes \delta_Y} & & \downarrow \mu^R_{X,Y} \\
\searrow_{\mu_X \otimes \mathsf{id}} & TX \otimes DY \xrightarrow{\psi_{X,Y}} R(X \otimes Y)
\end{array}
$$

(Every such interaction law gives a functor-functor interaction law (UT, UD, UR, ψ), where U sends (co)monads to their underlying functors.) When $R = \mathsf{Id}$, we speak of a non-residual interaction law. A *map between (residual) monad-comonad interaction laws* (T, D, R, ψ) and (T', D', R', ψ') is given by a monad map $T \to T'$, a comonad map $D' \to D$ and a monad map $R \to R'$ that make a map between the underlying functor-functor interaction laws. Monad-comonad interaction laws form a category isomorphic to the category of monoid objects in the category of functor-functor interaction laws.

Example 1. Let $\mathbb{C} = \textbf{Set}$ (or any SMCC). Take $TX = S \Rightarrow (S \times X)$ (the state monad) and $DX = S_0 \times (S_0 \Rightarrow X)$ (the costate monad). There is a non-residual monad-comonad interaction law of T, D when $S = S_0$ and more generally when S, S_0 come with a lens structure $get : S_0 \to S$, $put : S_0 \times S \to S_0$; in fact, these laws are in bijection with lenses.

Let $\mathbb{C} = \textbf{Set}$ (or any extensive category that also has the relevant initial algebras and final coalgebras). Take $FX = 1 + X^2$ and T the free monad on

F, so $TX \cong \mu X'.X + 1 + X'^2$ (leaf-labelled nullary-binary trees). The only comonad D that can interact with T non-residually is $DY \cong 0$. If we take $RZ = 1 + Z$, we have an R-residual interaction law of T and D for example for $DY \cong \nu Y'.Y \times (2 \times Y')$ (node-labelled bitstreams), i.e., the cofree comonad for $GY = 2 \times Y$.

See [16,33] for further examples and their intuitive meaning for semantics.

Some equivalent formulations of interaction laws will be useful. Due to the bijections

$$\frac{\begin{array}{c} FX \otimes GY \to H(X \otimes Y) \text{ nat. in } X, Y \\ \hline \mathbb{C}(X \otimes Y, Z) \to \mathbb{C}(FX \otimes GY, HZ) \text{ nat. in } X, Y, Z \\ \hline \mathbb{C}(X, Y \multimap Z) \to \mathbb{C}(FX, GY \multimap HZ) \text{ nat. in } X, Y, Z \end{array}}{F(Y \multimap Z) \to GY \multimap HZ \text{ nat. in } Y, Z}$$

an H-residual functor-functor interaction law of F, G is the same as a family of maps

$$\phi_{Y,Z} : F(Y \multimap Z) \to GY \multimap HZ$$

natural in Y, Z. Under this view, the equation required of a functor-functor interaction law map (f, g, h) between (F, G, H, ϕ) and (F', G', H', ϕ') becomes

$$\begin{array}{ccc} F(Y \multimap Z) & \xrightarrow{\phi_{Y,Z}} & GY \multimap HZ \\ f_{Y \multimap Z} \downarrow & & \downarrow g_Y \multimap h_Z \\ F'(Y \multimap Z) & \xrightarrow{\phi'_{Y,Z}} & G'Y \multimap H'Z \end{array}$$

An R-residual monad-comonad interaction law of T, D is the same as a family of maps

$$\psi_{Y,Z} : T(Y \multimap Z) \to DY \multimap RZ$$

natural in Y, Z satisfying

$$\begin{array}{ccc} Y \multimap Z =\!\!=\!\!= Y \multimap Z & & TT(Y \multimap Z) \xrightarrow{T\psi_{Y,Z}} T(DY \multimap RZ) \xrightarrow{\psi_{DY,RZ}} DDY \multimap RRZ \\ \eta_{Y \multimap Z} \downarrow \quad\quad \downarrow \varepsilon_Y \multimap \eta_Z^R & \mu_{Y \multimap Z} \downarrow & \quad\quad\quad\quad \downarrow \delta_Y \multimap \mu_Z^R \\ T(Y \multimap Z) \xrightarrow{\psi_{Y,Z}} DY \multimap RZ & T(Y \multimap Z) \xrightarrow{\quad\quad\quad \psi_{Y,Z} \quad\quad\quad} DY \multimap RZ \end{array}$$

Suppose $F, G, H : \mathbb{C} \to \mathbb{C}$ are such that the coends and ends

$$\begin{array}{ll} (F \star G) Z = \int^{X,Y} \mathbb{C}(X \otimes Y, Z) \bullet (FX \otimes GY) & = \int^Y F(Y \multimap Z) \otimes GY \\ (G \to H) X = \int_{Y,Z} \mathbb{C}(X, Y \multimap Z) \pitchfork (GY \multimap HZ) & = \int_Y GY \multimap H(X \otimes Y) \end{array}$$

exist. ($F \star G$ is called the *Day convolution*.) Then, because of the bijections

$$\frac{\begin{array}{c} \int^{X,Y} \mathbb{C}(X \otimes Y, Z) \bullet (FX \otimes GY) \to HZ \text{ nat. in } Z \\ \hline \mathbb{C}(X \otimes Y, Z) \to \mathbb{C}(FX \otimes GY, HZ) \text{ nat. in } X, Y, Z \\ \hline \mathbb{C}(X, Y \multimap Z) \to \mathbb{C}(FX, GY \multimap HZ) \text{ nat. in } X, Y, Z \end{array}}{FX \to \int_{Y,Z} \mathbb{C}(X, Y \multimap Z) \pitchfork (GY \multimap HZ) \text{ nat. in } X}$$

an H-residual functor-functor interaction law of F, G turns out to be the same as a natural transformation $F \star G \to H$ or $F \to G \rightarrowtail H$. An R-residual monad-comonad interaction law of T, D is the same as a natural transformation $UT \star UD \to UR$ satisfying certain equations and also—by way of a particularly concise characterization—the same as a monad map $T \to D \rightarrowtail R$ where $D \rightarrowtail R$ is a certain canonical monad with $UD \rightarrowtail UR$ as the underlying functor.

Now, if $\mathbb{C}$ is locally presentable and F, G, H are accessible, then $F \star G$ and $G \rightarrowtail H$ are guaranteed to exist and be accessible. Writing $[\mathbb{C}, \mathbb{C}]_a$ for the category of accessible endofunctors on $\mathbb{C}$, we obtain functors $\star : [\mathbb{C}, \mathbb{C}]_a \times [\mathbb{C}, \mathbb{C}]_a \to [\mathbb{C}, \mathbb{C}]_a$ and $\rightarrowtail : [\mathbb{C}, \mathbb{C}]_a^{op} \times [\mathbb{C}, \mathbb{C}]_a \to [\mathbb{C}, \mathbb{C}]_a$. Together with $J \in [\mathbb{C}, \mathbb{C}]_a$ defined by $JZ = \mathbb{C}(\mathsf{I}, Z) \bullet \mathsf{I}$, the functor $\star$ equips $[\mathbb{C}, \mathbb{C}]_a$ with a symmetric monoidal structure. We also get that $- \star G \vdash G \rightarrowtail -$, i.e., this structure is closed.[3] The functor $\rightarrowtail : [\mathbb{C}, \mathbb{C}]_a^{op} \times [\mathbb{C}, \mathbb{C}]_a \to [\mathbb{C}, \mathbb{C}]_a$ is lax monoidal wrt. the composition monoidal structure on $[\mathbb{C}, \mathbb{C}]_a$. That $UD \rightarrowtail UR$ carries a monad structure if D is an accessible comonad and R is an accessible monad is a consequence of this.

These observations suggest the possibility of abstraction by switching to a more general setting. Instead of considering $[\mathbb{C}, \mathbb{C}]_a$, we can consider an arbitrary category $\mathbb{D}$ equipped with a monoidal structure and a symmetric monoidal structure that suitably agree. The appropriate notion of agreement is duoidality [10,2]. We will next consider this abstraction and see that monad-comonad interaction laws are the measuring maps of an instance of López Franco and Vasilakopoulou's Sweedler theory for duoidal categories [20].

3   Sweedler Theory for Duoidal Categories

We review the Sweedler theory for duoidal categories [20] and show that monads provide an instance.

Assume a symmetric duoidal category $(\mathbb{D}, I, \diamond, J, \star)$, i.e., a symmetric monoidal category in $\mathbf{MonCAT}_{oplax}$, that is also closed in the sense that $- \star G$ has a right adjoint $G \rightarrowtail -$ in $\mathbf{CAT}$. Explicitly, this means that we have a category $\mathbb{D}$ equipped with a monoidal structure $(I, \diamond)$, a symmetric monoidal closed structure $(J, \star, \rightarrowtail)$ and structural laws

$$J \to I \qquad\qquad J \to J \diamond J$$
$$I \star I \to I \qquad (F \diamond G) \star (H \diamond K) \to (F \star H) \diamond (G \star K)$$

satisfying appropriate equations witnessing oplax monoidality of $J : \mathbf{1} \to \mathbb{D}$ and $\star : \mathbb{D} \times \mathbb{D} \to \mathbb{D}$ as functors between monoidal categories for the $(I, \diamond)$ monoidal structure on $\mathbb{D}$.

---

[3] If $\mathbb{C}$ is locally κ-presentable with the κ-presentable objects closed under I and $\otimes$, then the κ-accessible endofunctors on $\mathbb{C}$ form a monoidal category with $\star$ as tensor. Garner and López Franco [13, Sect. 8.1] show that this monoidal category is closed, but their closed structure is different from ours. Our $G \rightarrowtail H$ has the property that natural transformations $F \to G \rightarrowtail H$ are H-residual functor-functor interaction laws of F, G even if F is not accessible; this is not the case for Garner and López Franco's. This is why we do not restrict to fixed κ, and instead use all of $[\mathbb{C}, \mathbb{C}]_a$.

The internal hom object $F \rightarrowtail I$ is called the *dual* of F. Stretching this terminology, the object $F \rightarrowtail H$ can be called the dual of F wrt. H.

We write $\mathbf{Mon}(\mathbb{D})$ (respectively $\mathbf{Comon}(\mathbb{D})$) for the categories of monoids (resp. comonoids) in $\mathbb{D}$ wrt. the $(I, \diamond)$ monoidal structure.

The composition monoidal and Day convolution symmetric monoidal closed structures $(\mathsf{Id}, \cdot)$ and $(J, \star, \rightarrowtail)$ on $[\mathbb{C}, \mathbb{C}]_a$ yield an example of such a symmetric duoidal category $\mathbb{D}$. The categories $\mathbf{Mon}([\mathbb{C}, \mathbb{C}]_a)$ and $\mathbf{Comon}([\mathbb{C}, \mathbb{C}]_a)$ are those of accessible monads and comonads.

The object J has a comonoid structure $J \to I$, $J \to J \diamond J$, and the functor $\rightarrowtail : \mathbb{D}^{\mathrm{op}} \times \mathbb{D} \to \mathbb{D}$ is lax monoidal wrt. the $(I, \diamond)$ monoidal structure. The operations

$$\star : \mathbb{D} \times \mathbb{D} \to \mathbb{D}$$
$$\rightarrowtail : \mathbb{D}^{\mathrm{op}} \times \mathbb{D} \to \mathbb{D}$$

lift to

$$\star : \mathbf{Comon}(\mathbb{D}) \times \mathbf{Comon}(\mathbb{D}) \to \mathbf{Comon}(\mathbb{D}) \quad \text{tensor of comonoids}$$
$$\rightarrowtail : (\mathbf{Comon}(\mathbb{D}))^{\mathrm{op}} \times \mathbf{Mon}(\mathbb{D}) \to \mathbf{Mon}(\mathbb{D}) \quad \text{power of a monoid}$$

in the sense that

via

$$\varepsilon = D_0 \star D_1 \xrightarrow{\varepsilon_0 \star \varepsilon_1} I \star I \longrightarrow I$$

$$\delta = D_0 \star D_1 \xrightarrow{\delta_0 \star \delta_1} (D_0 \diamond D_0) \star (D_1 \diamond D_1) \longrightarrow (D_0 \star D_1) \diamond (D_0 \star D_1)$$

$$\eta = I \longrightarrow I \rightarrowtail I \xrightarrow{\varepsilon \rightarrowtail \eta^R} D \rightarrowtail R$$

$$\mu = (D \rightarrowtail R) \diamond (D \rightarrowtail R) \longrightarrow (D \diamond D) \rightarrowtail (R \diamond R) \xrightarrow{\delta \rightarrowtail \mu^R} D \rightarrowtail R$$

Comonoid maps $D_0 \star D_1 \to D$ are the same as maps $\psi : UD_0 \star UD_1 \to UD$ satisfying

(omitting the Us in the equations). Such maps ψ could be called D-residual comonoid-comonoid interaction laws of D_0, D_1.

Monoid maps $T \to D \rightarrowtail R$ are in bijection with maps $\psi : UT \star UD \to UR$ that satisfy

(again omitting the Us in the equations), which are known as *measuring maps* from T to R by D and which we can also call R-residual monoid-comonoid interaction laws of T, R.

The three *Sweedler operations*

$$\mathcal{C} : (\mathbf{Comon}(\mathbb{D}))^{\mathrm{op}} \times \mathbf{Comon}(\mathbb{D}) \to \mathbf{Comon}(\mathbb{D}) \quad \text{internal hom of comonoids}$$
$$\rhd : \mathbf{Comon}(\mathbb{D}) \times \mathbf{Mon}(\mathbb{D}) \to \mathbf{Mon}(\mathbb{D}) \quad \text{Sweedler copower of a monoid}$$
$$\mathcal{M} : (\mathbf{Mon}(\mathbb{D}))^{\mathrm{op}} \times \mathbf{Mon}(\mathbb{D}) \to \mathbf{Comon}(\mathbb{D}) \quad \text{Sweedler hom of monoids}$$
$$\text{(univ. measuring comonoid)}$$

are everywhere defined by the following adjunctions if the adjoints exist.

$$
\begin{array}{ccc}
\mathbf{Comon}(\mathbb{D}) & \mathbf{Mon}(\mathbb{D}) & \mathbf{Comon}(\mathbb{D}) \\
{-\star D_1} \left(\quad \dashv \quad \right) \mathcal{C}(D_1,-) & D\rhd- \left(\quad \dashv \quad \right) D\rightarrowtail- & {-\rhd T} \left(\quad \dashv \quad \right) \mathcal{M}(T,-) \\
\mathbf{Comon}(\mathbb{D}) & \mathbf{Mon}(\mathbb{D}) & \mathbf{Mon}(\mathbb{D})
\end{array}
$$

They are defined for specific pairs of (co)monoids if the universal objects specified by the following bijections exist.

$$
\cfrac{UT \star UD \to UR \text{ meas.}}{\cfrac{T \to D \rightarrowtail R}{\cfrac{D \rhd T \to R}{D \to \mathcal{M}(T,R)}}}
$$

$$
\cfrac{D_0 \star D_1 \to D}{D_0 \to \mathcal{C}(D_1, D)}
$$

The comonoid $\mathcal{M}(T, I)$ is called the *Sweedler dual* of the monoid T.

By definition, the comonoid $\mathcal{C}(D_1, D)$ is the final comonoid interacting with the comonoid D_1 D-residually. The Sweedler hom $\mathcal{M}(T, R)$ is the final R-residually interacting comonoid for the monoid T. The Sweedler copower $D \rhd T$ is the initial residual monoid for monoid-comonoid interactions of T and D.

If the Sweedler operations are everywhere defined, for which it suffices that $\mathbb{D}$ is locally presentable [20, Thm. 20], then the category $(\mathbf{Comon}(\mathbb{D}), J, \star, \mathcal{C})$ is symmetric monoidal closed and the category $(\mathbf{Mon}(\mathbb{D}), \rhd, \rightarrowtail, \mathcal{M})$ is copowered, powered and enriched over $(\mathbf{Comon}(\mathbb{D}), J, \star, \mathcal{C})$. However, local presentability of $\mathbb{C}$ is not enough for local presentability (or even accessibility) of $[\mathbb{C}, \mathbb{C}]_{\mathrm{a}}$ (for example, $[\mathbf{Set}, \mathbf{Set}]_{\mathrm{a}}$ is not accessible). In Sect. 5, we return to the question of everywhere-definedness of the Sweedler operations for $[\mathbb{C}, \mathbb{C}]_{\mathrm{a}}$.

The Sweedler theory perspective allows us to establish some facts about interaction laws of free monads very easily. For example, we can straightforwardly derive a characterization of measuring maps from the free monoid F^* on F (assuming it exists).

Proposition 1. *Measuring maps $U(F^*) \star UD \to UR$ are in bijection with maps $F \star UD \to UR$.*

Proof. This is witnessed by the following chain of bijections.

$$
\cfrac{F \star UD \to UR}{\cfrac{F \to UD \rightarrowtail UR}{\cfrac{F \to U(D \rightarrowtail R)}{\cfrac{F^* \to D \rightarrowtail R}{U(F^*) \star UD \to UR \text{ meas.}}}}}
$$

$\square$

Similarly, we can calculate closed-form expressions for the Sweedler hom from a free monoid and the Sweedler copower of a free monoid. Here $G^\dagger$ denotes the cofree comonoid on G (if it exists).

Proposition 2. (i) $\mathcal{M}(F^*, R) \cong (F \rightarrowtail UR)^\dagger$. (ii) $D \rhd F^* \cong (F \star UD)^*$.

Proof. (i) As witnessed by the chain of bijections on the left below, comonoid maps $D \to \mathcal{M}(F^*, R)$ and comonoid maps $D \to (F \rightarrowtail UR)^\dagger$ are in bijection naturally in D. (ii) The chain of bijections on the right below composes to a bijection natural in R between monoid maps $D \rhd F^* \to R$ and monoid maps $(F \star UD)^* \to R$.

$$
\begin{array}{c}
\dfrac{D \to (F \rightarrowtail UR)^\dagger}{\dfrac{UD \to F \rightarrowtail UR}{\dfrac{F \to UD \rightarrowtail UR}{\dfrac{F \to U(D \rightarrowtail R)}{\dfrac{F^* \to D \rightarrowtail R}{D \to \mathcal{M}(F^*, R)}}}}}
\end{array}
\qquad\qquad
\begin{array}{c}
\dfrac{(F \star UD)^* \to R}{\dfrac{F \star UD \to UR}{\dfrac{F \to UD \rightarrowtail UR}{\dfrac{F \to U(D \rightarrowtail R)}{\dfrac{F^* \to D \rightarrowtail R}{D \rhd F^* \to R}}}}}
\end{array}
\qquad \square
$$

Example 2. Let $\mathbb{C} = \mathbf{Set}$. (i) Take $F = 0$, then $F^* \cong \mathsf{Id}$. We can calculate $F \rightarrowtail UR \cong 1$, therefore $\mathcal{M}(F^*, R) \cong \mathsf{Id}$, for any monad R.

Next take $FX = X^2$, then $F^*X \cong \mu X'. X + X'^2$ (these are leaf-labelled binary trees). We can calculate $(F \rightarrowtail UR)Y \cong R(2 \times Y)$, hence $\mathcal{M}(F^*, R)Y \cong \nu Y'. Y \times R(2 \times Y')$ (node-labelled streams of bits for $R = \mathsf{Id}$, node-labelled nonempty colists of bits for $RZ = 1 + Z$).

Finally, take $FX = 1 + X^2$, then $F^*X \cong \mu X'. X + 1 + X'^2$ (leaf-labelled nullary-binary trees). We calculate $(F \rightarrowtail UR)Y \cong R\,0 \times R(2 \times Y)$, hence $\mathcal{M}(F^*, R)Y \cong \nu Y'. Y \times R\,0 \times R(2 \times Y')$. For $R = \mathsf{Id}$ and any R such that $R\,0 \cong 0$, this means that $\mathcal{M}(F^*, R) \cong 0$. For $RZ = 1 + Z$, we get $\mathcal{M}(F^*, R)Y \cong \nu Y'. Y \times (1 + 2 \times Y')$ (node-labelled nonempty colists of bits).

(ii) Take $F = 0$, then $F^\star \cong \mathsf{Id}$. We can calculate $(F \star UD) \cong 0$, hence $D \rhd F^* \cong \mathsf{Id}$, for any comonad D.

Take $FX = X^2$, then $F^*X \cong \mu X'. X + X'^2$. We can calculate $(F \star UD)Z \cong D(Z^2)$, therefore $(D \rhd F^*)Z \cong \mu Z'. Z + D(Z'^2)$.

Take $FX = 1 + X^2$, then $F^*X \cong \mu X'. X + 1 + X'^2$. We can calculate $(F \star UD)Z \cong D\,1 + D(Z^2)$, therefore $(D \rhd F^*)Z \cong \mu Z'. Z + D\,1 + D(Z'^2)$.

These examples generalize to any wellpointed, locally presentable $\mathbb{C}$ with exponentials, when R and D are strong.

In exactly the same way as above, comonoid maps $D_0 \star D_1 \to G^\dagger$ are in bijection with maps $UD_0 \star UD_1 \to G$, and $\mathcal{C}(D_1, G^\dagger) \cong (UD_1 \rightarrowtail G)^\dagger$.

In the rest of this paper, we ignore comonad-comonad interaction laws and the internal hom of comonads since they are not our main focus. But developments similar to those for monad-comonad interaction laws and the Sweedler hom of monads and the Sweedler copower of a monad in Sects. 4, 5) below can be carried out for them as well.

4   Monad-comonad Interaction Laws (Co)algebraically

We now return to monad-comonad interaction laws specifically and explain the (co)algebraic perspective developed in [33]. (Props. 4 and 6 did not appear in [33].) First, monad-comonad interaction laws admit the following useful characterization in terms of (co)algebras of the (co)monads involved.

Proposition 3. *R-residual monad-comonad interaction laws ψ of T, D are in bijection with functors $\Psi : (\mathbf{Coalg}(D))^{\mathrm{op}} \times \mathbf{Alg}(R) \to \mathbf{Alg}(T)$ that internal-hom carriers, i.e., satisfy*

$$
\begin{array}{ccc}
(\mathbf{Coalg}(D))^{\mathrm{op}} \times \mathbf{Alg}(R) & \xrightarrow{\;\Psi\;} & \mathbf{Alg}(T) \\
{\scriptstyle U^{\mathrm{op}} \times U}\downarrow & & \downarrow{\scriptstyle U} \\
\mathbb{C}^{\mathrm{op}} \times \mathbb{C} & \xrightarrow{\;\;-\multimap\;\;} & \mathbb{C}
\end{array}
$$

Proof (sketch). Given an interaction law ψ, the functor Ψ is defined by

$$
\Psi((Y,\chi),(Z,\zeta)) = (Y \multimap Z,\; T(Y \multimap Z) \xrightarrow{\;\psi\;} DY \multimap RZ \xrightarrow{\;\chi \multimap \zeta\;} Y \multimap Z\;)
$$

Conversely, given a functor Ψ, the corresponding interaction law ψ is defined by

$$
\psi = \; T(Y \multimap Z) \xrightarrow{\;T(\varepsilon_Y \multimap \eta^R_Z)\;} T(DY \multimap RZ) \xrightarrow{\;\;\xi\;\;} DY \multimap RZ
$$

where $(DY \multimap RZ, \xi) = \Psi((DY, \delta_Y), (RZ, \mu^R_Z))$. $\square$

We remark that such functors Ψ are completely determined by their action on (co)free (co)algebras. To be precise, there is a bijection between these functors and functors $\Psi' : (\mathbf{CoKl}(D))^{\mathrm{op}} \times \mathbf{Kl}(R) \to \mathbf{Alg}(T)$ that satisfy

$$
\begin{array}{ccc}
(\mathbf{CoKl}(D))^{\mathrm{op}} \times \mathbf{Kl}(R) & \xrightarrow{\;\Psi'\;} & \mathbf{Alg}(T) \\
{\scriptstyle K^{\mathrm{op}} \times K}\downarrow & & \downarrow{\scriptstyle U} \\
\mathbb{C}^{\mathrm{op}} \times \mathbb{C} & \xrightarrow{\;\;-\multimap\;\;} & \mathbb{C}
\end{array}
$$

where $K : \mathbf{CoKl}(D) \to \mathbb{C}$ is the left adjoint of the coKleisli adjunction of D and $K : \mathbf{Kl}(R) \to \mathbb{C}$ is the right adjoint of the Kleisli adjunction of R.

The following reformulations of Prop. 1 enable a smooth derivation of further characterizations of monad-comonad interaction laws in terms of what we call runners, introduced next.

Corollary 1. *R-residual interaction laws of T, D are in bijection with functors $\Psi : \mathbf{Coalg}(D) \to [\mathbf{Alg}(R), \mathbf{Alg}(T)]^{\mathrm{op}}$ satisfying*

$$
\begin{array}{ccccc}
\mathbf{Coalg}(D) & \xrightarrow{\hspace{4cm}\Psi\hspace{4cm}} & & & [\mathbf{Alg}(R), \mathbf{Alg}(T)]^{\mathrm{op}} \\
{\scriptstyle U}\downarrow & & & & \downarrow{\scriptstyle [\mathbf{Alg}(R), U]^{\mathrm{op}}} \\
\mathbb{C} & \xrightarrow{\;(Y \mapsto Y \multimap -)^{\mathrm{op}}\;} [\mathbb{C}, \mathbb{C}]^{\mathrm{op}} & \xrightarrow{\;[U, \mathbb{C}]^{\mathrm{op}}\;} & & [\mathbf{Alg}(R), \mathbb{C}]^{\mathrm{op}}
\end{array}
$$

and also with functors $\Psi : \mathbf{Alg}(R) \to [\mathbf{Coalg}(D)^{\mathrm{op}}, \mathbf{Alg}(T)]$ *satisfying*

$$
\begin{array}{ccc}
\mathbf{Alg}(R) & \xrightarrow{\quad\Psi\quad} & [(\mathbf{Coalg}(D))^{\mathrm{op}}, \mathbf{Alg}(T)] \\
{\scriptstyle U}\downarrow & & \downarrow{\scriptstyle [(\mathbf{Coalg}(D))^{\mathrm{op}}, U]} \\
\mathbb{C} & \xrightarrow[(Z \mapsto -\multimap Z)]{} [\mathbb{C}^{\mathrm{op}}, \mathbb{C}] \xrightarrow[[U^{\mathrm{op}}, \mathbb{C}]]{} & [(\mathbf{Coalg}(D))^{\mathrm{op}}, \mathbb{C}]
\end{array}
$$

Stateful Runners

Say that an R-residual *stateful runner* of T is an object $Y \in \mathbb{C}$ together with a family of maps

$$\theta_X : TX \otimes Y \to R(X \otimes Y)$$

natural in X satisfying

$$
\begin{array}{ccc}
X \otimes Y == X \otimes Y \\
{\scriptstyle \eta_X \otimes Y}\downarrow \qquad \downarrow{\scriptstyle \eta^R_{X \otimes Y}} \\
TX \otimes Y \xrightarrow{\theta_X} R(X \otimes Y)
\end{array}
\qquad
\begin{array}{ccc}
TTX \otimes Y \xrightarrow{\theta_{TX}} R(TX \otimes Y) \xrightarrow{R\theta_X} RR(X \otimes Y) \\
{\scriptstyle \mu_X \otimes Y}\downarrow \qquad\qquad\qquad \downarrow{\scriptstyle \mu^R_{X \otimes Y}} \\
TX \otimes Y \xrightarrow{\qquad\qquad \theta_X \qquad\qquad} R(X \otimes Y)
\end{array}
$$

Maps $(Y, \theta) \to (Y', \theta')$ between stateful runners are maps $f : Y \to Y'$ satisfying $R(X \otimes f) \circ \theta_X = \theta'_X \circ (TX \otimes f)$. Stateful runners form a category $\mathbf{SRun}_R(T)$.

R-residual stateful runners of T with carrier Y are in bijection with monad maps $T \to \mathsf{St}^R_Y$ where St^R_Y is the R-transformed *state monad* for state object Y defined by $\mathsf{St}^R_Y X = Y \multimap R(X \otimes Y)$.

They are also in bijection with functors $\Theta : \mathbf{Alg}(R) \to \mathbf{Alg}(T)$ that internal-hom Y with the carrier, i.e., satisfy

$$
\begin{array}{ccc}
\mathbf{Alg}(R) & \xrightarrow{\quad\Theta\quad} & \mathbf{Alg}(T) \\
{\scriptstyle U}\downarrow & & \downarrow{\scriptstyle U} \\
\mathbb{C} & \xrightarrow{\quad Y \multimap - \quad} & \mathbb{C}
\end{array}
$$

Proof (sketch). Given a stateful runner θ, the functor Θ is defined by

$$\Theta(Z, \zeta) = T(Y \multimap Z) \xrightarrow{\theta_{Y \multimap Z}} Y \multimap R((Y \multimap Z) \otimes Y) \xrightarrow{Y \multimap \mathsf{Rev}} Y \multimap RZ \xrightarrow{Y \multimap \zeta} Y \multimap Z$$

Conversely, given a functor Θ, the stateful runner θ is

$$\theta_X = TX \xrightarrow{T\mathsf{coev}} T(Y \multimap X \otimes Y) \xrightarrow{T(Y \multimap \eta^R_{X \otimes Y})} T(Y \multimap R(X \otimes Y)) \xrightarrow{\xi} Y \multimap R(X \otimes Y)$$

where $(Y \multimap R(X \otimes Y), \xi) = \Theta(R(X \otimes Y), \mu^R_{X \otimes Y})$.   □

This observation is strengthened by the following proposition that also talks about stateful runner maps.

Proposition 4. *The following is pullback square:*

$$
\begin{array}{ccc}
\mathbf{SRun}_R(T) & \xrightarrow{\qquad\qquad\qquad} & [\mathbf{Alg}(R), \mathbf{Alg}(T)]^{\mathrm{op}} \\
{\scriptstyle U}\downarrow & & \downarrow{\scriptstyle [\mathbf{Alg}(R), U]^{\mathrm{op}}} \\
\mathbb{C} & \xrightarrow[(Y \mapsto Y \multimap -)^{\mathrm{op}}]{} [\mathbb{C}, \mathbb{C}]^{\mathrm{op}} \xrightarrow[[U, \mathbb{C}]^{\mathrm{op}}]{} & [\mathbf{Alg}(R), \mathbb{C}]^{\mathrm{op}}
\end{array}
$$

Combining Prop. 4 with Cor. 1, we obtain a characterization of monad-comonad interaction laws in terms of stateful runners.

Proposition 5. *R-residual monad-comonad interaction laws T, D are in a bijection with functors $\Psi : \mathbf{Coalg}(D) \to \mathbf{SRun}_R(T)$ preserving carriers, i.e., satisfying*

$$\mathbf{Coalg}(D) \xrightarrow{\;\;\Psi\;\;} \mathbf{SRun}_R(T)$$
$$\begin{array}{ccc} & U \searrow \quad \swarrow U & \\ & \mathbb{C} & \end{array}$$

Continuation-Based Runners

A D-fuelled *continuation-based runner* of T is an object $Z \in \mathbb{C}$ together with a family of maps

$$\theta_X : D(X \multimap Z) \to TX \multimap Z$$

natural in X satisfying

$$
\begin{array}{ccc}
D(X \multimap Z) \xrightarrow{\theta_X} TX \multimap Z & \qquad & D(X \multimap Z) \xrightarrow{\quad\theta_X\quad} TX \multimap Z \\
\varepsilon_{X \multimap Z} \downarrow \qquad \downarrow \eta_X \multimap Z & & \delta_{X \multimap Z} \downarrow \qquad\qquad\qquad\qquad \downarrow \mu_X \multimap Z \\
X \multimap Z \;=\!=\; X \multimap Z & & DD(X \multimap Z) \xrightarrow{D\theta_X} D(TX \multimap Z) \xrightarrow{\theta_{TX}} TTX \multimap Z
\end{array}
$$

These runners form a category $\mathbf{CRun}_D(T)$.

D-fuelled continuation-based runners of T with carrier Z are in bijection with monad maps $T \to \mathsf{Cnt}_Z^D$, where Cnt_Z^D is the D-transformed *continuation monad* for answer object Z defined by $\mathsf{Cnt}_Z^D X = D(X \multimap Z) \multimap Z$.

Continuation-based runners are also in bijection with functors $\Theta : (\mathbf{Coalg}(D))^{\mathrm{op}} \to \mathbf{Alg}(T)$ that internal-hom the carrier with Z, i.e., that satisfy

$$
\begin{array}{ccc}
(\mathbf{Coalg}(D))^{\mathrm{op}} & \xrightarrow{\;\Theta\;} & \mathbf{Alg}(T) \\
U^{\mathrm{op}} \downarrow & & \downarrow U \\
\mathbb{C}^{\mathrm{op}} & \xrightarrow{\;-\multimap Z\;} & \mathbb{C}
\end{array}
$$

Moreover:

Proposition 6. *The following is a pullback square:*

$$
\begin{array}{ccc}
\mathbf{CRun}_D(T) & \xrightarrow{\hspace{6cm}} & [(\mathbf{Coalg}(D))^{\mathrm{op}}, \mathbf{Alg}(T)] \\
U \downarrow & & \downarrow [(\mathbf{Coalg}(D))^{\mathrm{op}}, U] \\
\mathbb{C} \xrightarrow{Z \mapsto -\multimap Z} [\mathbb{C}^{\mathrm{op}}, \mathbb{C}] \xrightarrow{[U^{\mathrm{op}}, \mathbb{C}]} & & [(\mathbf{Coalg}(D))^{\mathrm{op}}, \mathbb{C}]
\end{array}
$$

Combining this proposition with Cor. 1, we obtain:

Proposition 7. *R-residual monad-comonad interaction laws of T, D are in bijection with functors $\Psi : \mathbf{Alg}(R) \to \mathbf{CRun}_D(T)$ that preserve carriers, i.e., that satisfy*

$$\mathbf{Alg}(R) \xrightarrow{\;\;\Psi\;\;} \mathbf{CRun}_D(T)$$
$$\begin{array}{ccc} & U \searrow \quad \swarrow U & \\ & \mathbb{C} & \end{array}$$

5 Combining Sweedler Theory and the (Co)algebraic Perspective

We now combine our (co)algebraic observations with Sweedler theory.

Sweedler Hom

By definition, the Sweedler hom between monads T, R, if it exists, is the comonad $\mathcal{M}(T,R)$ together with an monad-comonad interaction law υ such that, for any other comonad D and monad-comonad interaction law ψ, there exists a unique comonad map $g : D \to \mathcal{M}(T,R)$ satisfying

$$TX \otimes DY \xrightarrow[TX \otimes g_Y]{\psi_{X,Y}} TX \otimes \mathcal{M}(T,R)Y \xrightarrow{\upsilon_{X,Y}} R(X \otimes Y)$$

Comonad maps $D \to D'$ are in bijection with functors $\mathbf{Coalg}(D) \to \mathbf{Coalg}(D')$ that preserve carriers. Therefore, by Prop. 5, the Sweedler hom, if it exists, is the comonad $\mathcal{M}(T,R)$ together with a carrier-preserving functor $\Upsilon : \mathbf{Coalg}(\mathcal{M}(T,R)) \to \mathbf{SRun}_R(T)$ such that, for any other comonad D and carrier-preserving functor $\Psi : \mathbf{Coalg}(D) \to \mathbf{SRun}_R(T)$, there exists a unique carrier-preserving functor $\Gamma : \mathbf{Coalg}(D) \to \mathbf{Coalg}(\mathcal{M}(T,R))$ such that

$$
\begin{array}{ccc}
 & \Psi & \\
\mathbf{Coalg}(D) \xdashrightarrow{\Gamma} \mathbf{Coalg}(\mathcal{M}(T,R)) & \xrightarrow{\Upsilon} & \mathbf{SRun}_R(T) \\
 \searrow_U \quad \downarrow_U & & \swarrow_U \\
 & \mathbb{C} &
\end{array}
$$

It follows that, if $(\mathbf{SRun}_R(T), U)$ is strictly comonadic, then $\mathcal{M}(T,R)$ exists and $(\mathbf{Coalg}(\mathcal{M}(T,R)), U) \cong (\mathbf{SRun}_R(T), U)$. (Should $(\mathbf{SRun}_R(T), U)$ fail to be strictly comonadic, then $\mathcal{M}(T,R)$ may still exist, but with different algebras.) Easy calculations show that U strictly creates equalizers of U-split pairs. Hence, by the dual of Beck's monadicity theorem, U is strictly comonadic if it is a left adjoint. Under our assumptions on $\mathbb{C}$, T and R from Sect. 2, all is well.

Theorem 1. *If $\mathbb{C}$ is locally presentable and T and R are accessible monads on $\mathbb{C}$, then $\mathbf{SRun}_R(T)$ is locally presentable and the forgetful functor $U : \mathbf{SRun}_R(T) \to \mathbb{C}$ is a left adjoint. Hence the Sweedler hom $\mathcal{M}(T,R)$ exists, is accessible, and satisfies $(\mathbf{Coalg}(\mathcal{M}(T,R)), U) \cong (\mathbf{SRun}_R(T), U)$.*

Proof (sketch). We first show that $\mathbf{SRun}_R(T)$ is locally presentable. The functor $U : \mathbf{SRun}_R(T) \to \mathbb{C}$ strictly creates colimits by easy calculations, and hence $\mathbf{SRun}_R(T)$ is cocomplete. For local presentability, it therefore remains to show that $\mathbf{SRun}_R(T)$ is accessible, which we do by appealing to the fact that accessible categories are closed under *inserters* and *equifiers*. The category of F-coalgebras, for any accessible endofunctor F on $\mathbb{C}$, is an inserter of accessible

functors, and is therefore accessible by [1, Thm. 2.72]. For each Y, families of maps $\theta_X : TX \otimes Y \to R(X \otimes Y)$ natural in X are in bijection with maps $\chi : Y \to (T \twoheadrightarrow R)Y$, so that R-residual stateful runners of T are equivalently coalgebras (Y, χ) of the functor $T \twoheadrightarrow R$, satisfying two equations. One equation is an equality between two maps $Y \to (\mathsf{Id} \twoheadrightarrow R)Y$, the other between two maps $Y \to ((T \cdot T) \twoheadrightarrow R)Y$. It follows that $\mathbf{SRun}_R(T)$ is isomorphic to a full sub-category of the category $\mathbf{coalg}(T \twoheadrightarrow R)$ of $(T \twoheadrightarrow R)$-coalgebras, and that this full subcategory is the joint equifier of two natural transformations of accessible functors $\mathbf{coalg}(T \twoheadrightarrow R) \to \mathbf{coalg}(\mathsf{Id} \twoheadrightarrow R)$ and of two natural transformations of accessible functors $\mathbf{coalg}(T \twoheadrightarrow R) \to \mathbf{coalg}((T \cdot T) \twoheadrightarrow R)$. Accessible categories are closed under equifiers of natural transformations of accessible functors [1, Lemma 2.76], so $\mathbf{SRun}_R(T)$ is accessible and hence locally presentable.

As a colimit-preserving functor between locally presentable categories, U is a left adjoint by Freyd's special adjoint functor theorem, thus strictly comonadic. The induced comonad is the Sweedler hom $\mathcal{M}(T, R)$. Accessibility of $\mathcal{M}(T, R)$ follows from accessibility of the adjoints (the right adjoint by [1, Prop. 2.23]).   □

Example 3. Let $\mathbb{C} = \mathbf{Set}$. Take $TX = X^S$ (the reader monad for state object S). R-residual stateful runners of T are objects Y with families of maps $X^S \times Y \to R(X \times Y)$ natural in X or, equivalently, maps $Y \to R(S \times Y)$ constrained by two equations. For $R = \mathsf{Id}$ or $R = 1 + -$, these are in bijection with maps $Y \to S$. The comonad with such structured objects Y as coalgebras, which is the Sweedler hom of T and R, is $DY = S \times Y$ (the coreader monad for S). For a general accessible monad R, the Sweedler hom can be described as a subcomonad of the cofree comonad $DY = \nu Y'. \, Y \times R(S \times Y')$.

Take $TX = X^+ = \mu X'.X \times (1 + X')$ (the nonempty list monad with con-catenation as multiplication, free semigroup monad). R-residual stateful runners of T are objects Y with families of maps $X^+ \times Y \to R(X \times Y)$ natural in X satisfying two equations or, equivalently, maps $(X \times X) \times Y \to R(X \times Y)$ con-strained by one equation or, equivalently, maps $Y \to R(Y + Y)$ coassociative wrt. the coproduct monoidal structure of $\mathbf{Kl}(R)$, i.e., making Y into a cosemi-group. For $R = \mathsf{Id}$, the corresponding comonad is the cofree cosemigroup (wrt. the coproduct monoidal structure on $\mathbf{Set}$) comonad. Its underlying functor is $DY \cong Y \times (Y + Y)$.

These examples generalize to any wellpointed, locally presentable $\mathbb{C}$ with exponentials, when R is a strong monad.

Sweedler Copower

The Sweedler copower of a monad T by a comonad D, if it exists, is by definition the monad $D \triangleright T$ together with a monad-comonad interaction law υ such that, for any other monad R and monad-comonad interaction law ψ, there exists a unique monad map $g : D \triangleright T \to R$ satisfying

$$TX \otimes DY \xrightarrow[\upsilon_{X,Y}]{\overset{\psi_{X,Y}}{}} (D \triangleright T)(X \otimes Y) \xdashrightarrow{g_{X \otimes Y}} R(X \otimes Y)$$

Monad maps $R' \to R$ are in bijection with functors $\mathbf{Alg}(R) \to \mathbf{Alg}(R')$ that preserve carriers. Therefore, by Prop. 7, the Sweedler copower, if it exists, is the monad $D \triangleright T$ together with a carrier-preserving functor $\Upsilon : \mathbf{Alg}(D \triangleright T) \to \mathbf{CRun}_D(T)$ such that, for any other monad R and carrier-preserving functor $\Psi : \mathbf{Alg}(R) \to \mathbf{CRun}_D(T)$, there exists a unique carrier-preserving functor $\Gamma : \mathbf{Alg}(R) \to \mathbf{Alg}(D \triangleright T)$ such that

Consequently, if $(\mathbf{CRun}_D(T), U)$ is strictly monadic, then $D \triangleright T$ exists and $(\mathbf{Alg}(D \triangleright T), U) \cong (\mathbf{CRun}_D(T), U)$. This is the case as soon as U is a right adjoint by Beck's strict monadicity theorem, because U is easily verified to strictly create U-split coequalizers.

Theorem 2. *If $\mathbb{C}$ is locally presentable and T and D are accessible, then $\mathbf{CRun}_D(T)$ is locally presentable and the forgetful functor $U : \mathbf{CRun}_D(T) \to \mathbb{C}$ is a right adjoint. Hence the Sweedler copower $D \triangleright T$ exists, is accessible, and satisfies $(\mathbf{Alg}(D \triangleright T), U) \cong (\mathbf{CRun}_D(T), U)$.*

Proof (sketch). The proof is similar to that of Thm. 1. The functor U strictly creates limits, so $\mathbf{CRun}_D(T)$ is complete. The category $\mathbf{CRun}_D(T)$ is isomorphic to a full subcategory of the category of algebras of the functor $D \star T$, forming a joint equifier. Categories of algebras of accessible endofunctors on $\mathbb{C}$ are inserters of accessible functors, and hence form accessible categories. It follows that $\mathbf{CRun}_D(T)$ is also accessible, and hence locally presentable. The functor U strictly creates κ-filtered colimits, where κ is such that $\mathsf{Id} \star T$, $D \star T$, and $(D \cdot D) \star T$ are κ-accessible; in particular, U is accessible. Since U also strictly creates limits, it is therefore a right adjoint by [1, Theorem 1.66]. The induced monad is the Sweedler copower $D \triangleright T$, which is accessible because both adjoints are. □

Example 4. Let $\mathbb{C} = \mathbf{Set}$. Take $TX = M \times X$ where $(M, u, *)$ is a monoid (the writer monad) and $DY = S \times Y$ (the coreader comonad). D-fuelled continuation-based runners of T are objects Z with families of maps $S \times Z^X \to Z^{M \times X}$ natural in X or, equivalently, maps $(S \times M) \times Z \to Z$, subject to two equations. The monad with such structured objects Z as algebras, which is the Sweedler copower of T and D, is the writer monad for the free monoid on $S \times M$ quotiented by $(s, a) * (s, b) = (s, a * b)$ and $u = (s, u)$.

6   Enriched Interaction Laws

In Sects. 2, 4, 5 above, we worked with (a full subcategory of) the category $[\mathbb{C}, \mathbb{C}]$ of endofunctors on a SMCC $\mathbb{C}$ and natural transformations between them, and abstracted it to a duoidal category $\mathbb{D}$ in Sect. 3.

An alternative is to proceed from an SMCC $(\mathbb{V}, I, \otimes, \multimap)$ (copowered over itself by $\otimes$ and enriched and powered by $\multimap$) and another category $\mathbb{C}$ that is at least copowered or enriched over $\mathbb{V}$, or possibly both or even powered too. In this setting, a $\mathbb{V}$-*enriched functor-functor interaction law* is given by $\mathbb{V}$-enriched endofunctors F on $\mathbb{V}$ and G and H on $\mathbb{C}$ together with either a family of maps $\phi_{X,Y} : FX \bullet GY \to H(X \bullet Y)$ in $\mathbb{C}$ that are $\mathbb{V}$-natural in $X \in \mathbb{V}$ and $Y \in \mathbb{C}$ or, equivalently, a family of maps $\phi_{Y,Z} : F(\mathbb{C}(Y, Z)) \to \mathbb{C}(GY, HZ)$ in $\mathbb{V}$ that are $\mathbb{V}$-natural in $Y, Z \in \mathbb{C}$.

Two cases are of special interest.

- $\mathbb{V} = \mathbf{Set}$: Then the requirements that the category $\mathbb{C}$, the functors F, G, H and the natural transformation ϕ be $\mathbb{V}$-enriched are automatically met, but differently from the main setting of this paper, F is an endofunctor on a generally different category than G and H.
- $\mathbb{V} = \mathbb{C}$: Then the requirements that the functors F, G, H and the natural transformation ϕ be $\mathbb{V}$-enriched become real restrictions, but F, G, H remain endofunctors all on the same category.

The only case where the enriched setting agrees with the main one of this paper of Sects. 2–5, i.e., the concept of interaction law where there are no non-vacuous enrichment requirements and the endofunctors involved are all on the same category, is the intersection of the above two: $\mathbb{V} = \mathbb{C} = \mathbf{Set}$.

A more general situation in which the two settings do not differ too much is when $\mathbb{V} = \mathbb{C}$ and $\mathbb{C}$ is *monoidally wellpointed.* Then all functors with codomain $\mathbb{C}$ are uniquely $\mathbb{C}$-enriched (but may fail to admit an enrichment) and all natural transformations between $\mathbb{C}$-enriched functors with codomain $\mathbb{C}$ are $\mathbb{C}$-enriched.

In the case $\mathbb{V} = \mathbb{C}$, which is probably the most interesting case for mathematical semantics applications, the duoidal abstraction of Sect. 3 still applies. We can take $\mathbb{D}$ to be (a suitable full subcategory of) $\mathbb{C}\text{-}[\mathbb{C}, \mathbb{C}]$, where $\mathbb{C}\text{-}[\mathbb{C}, \mathbb{C}]$ is the ordinary category of $\mathbb{C}$-functors $\mathbb{C} \to \mathbb{C}$ (strong endofunctors).

In the case of a general $\mathbb{V}$, the simple duoidal abstraction ceases to apply. We need to switch to an action $\star : \mathbb{W} \times \mathbb{D} \to \mathbb{D}$ (in $\mathbf{MonCAT}_{\text{oplax}}$) of a symmetric duoidal category $(\mathbb{W}, I_{\mathbb{W}}, \diamond_{\mathbb{W}}, J_{\mathbb{W}}, \star_{\mathbb{W}})$ on a monoidal category $(\mathbb{D}, I, \diamond)$ together with a functor $\rightthreetimes : \mathbb{D}^{\text{op}} \times \mathbb{D} \to \mathbb{W}$ such that $- \star G \vdash G \rightthreetimes -$ (in $\mathbf{CAT}$). Crucially, the action $\star$ comes with structural laws

$$I_{\mathbb{W}} \star I \to I \qquad (F \diamond_{\mathbb{W}} G) \star (H \diamond K) \to (F \star H) \diamond (G \star K)$$

witnessing oplaxity of $\star$. Similarly to the simple duoidal situation, we get that $\star$ and $\rightthreetimes$ lift to functors $\star : \mathbf{Comon}(\mathbb{W}) \times \mathbf{Comon}(\mathbb{D}) \to \mathbf{Comon}(\mathbb{D})$ and $\rightthreetimes : (\mathbf{Comon}(\mathbb{D}))^{\text{op}} \times \mathbf{Mon}(\mathbb{D}) \to \mathbf{Mon}(\mathbb{W})$ and can then define measuring maps and Sweedler-like operations and ask if they are everywhere defined.

The instantiation is given by (suitable full subcategories of) $\mathbb{W} = \mathbb{V}\text{-}[\mathbb{V}, \mathbb{V}]$, $\mathbb{D} = \mathbb{V}\text{-}[\mathbb{C}, \mathbb{C}]$ and

$$(F \star G)\, Z = \int^{X,Y} \mathbb{C}(X \bullet Y, Z) \bullet (FX \bullet GY) \quad = \int^{Y} F(\mathbb{C}(Y, Z)) \bullet GY$$
$$(G \rightthreetimes H)\, X = \int_{Y,Z} (X \multimap \mathbb{C}(Y, Z)) \multimap \mathbb{C}(GY, HZ) = \int_{Y} \mathbb{C}(GY, H(X \bullet Y))$$

where the integral signs now stand for $\mathbb{V}$-enriched coends and ends.

Runners as Generalized Algebras

Enriched monad-comonad interaction laws can be characterized as enriched functors between categories of (co)algebras analogously to Props. 3, 5, 7. But one pleasant feature of the enriched setting is that enriched versions of both stateful and continuation-based runners of T can be described as algebras of T in a generalized sense.

Suppose we are given an SMCC $\mathbb{V}$ (copowered over itself by $\otimes$ and enriched and powered by $\multimap$) and a $\mathbb{V}$-enriched monad T on $\mathbb{V}$. For a category $\mathbb{K}$ that is enriched and powered over $\mathbb{V}$, we say that an *algebra* of T *in* $\mathbb{K}$ as an object Y of $\mathbb{K}$ together with family of maps $\chi_X : X \pitchfork Y \to TX \pitchfork Y$ in $\mathbb{K}$ that is $\mathbb{V}$-enriched natural in $X \in \mathbb{V}$ and satisfies the equations

$$
\begin{array}{ccc}
X \pitchfork Y \xrightarrow{\chi_X} TX \pitchfork Y & \qquad & X \pitchfork Y \xrightarrow{\chi_X} TX \pitchfork Y \\
\searequals\quad \downarrow{\scriptstyle \eta_X \pitchfork Y} & & {\scriptstyle \chi_X}\downarrow \qquad\qquad \downarrow{\scriptstyle \mu_X \pitchfork Y} \\
X \pitchfork Y & & TX \pitchfork Y \xrightarrow{\chi_{TX}} TTX \pitchfork Y
\end{array}
$$

If $\mathbb{V}$ has enough limits, then these form a $\mathbb{V}$-category $\mathbf{Alg}(T, \mathbb{K})$, and there is a forgetful $\mathbb{V}$-functor $U : \mathbf{Alg}(T, \mathbb{K}) \to \mathbb{C}$. (The limits are required to carve out the object of algebra maps $(Y, \chi) \to (Y', \chi')$ from the hom-object $\mathbb{K}(Y, Y')$.)

An algebra like this is equivalently an object $Y \in \mathbb{K}$ together with a $\mathbb{V}$-enriched monad map $T \to \mathsf{Knt}_Y$ where $\mathsf{Knt}_Y X = \mathbb{K}(X \pitchfork Y, Y)$. If $\mathbb{V} = \mathbb{K}$, an algebra of T in this sense is the same as an algebra in the standard sense. In this case, we have $\mathsf{Knt}_Y X = (X \multimap Y) \multimap Y$.

Enriched runners of T turn out to be algebras of T in this generalized sense. Given a category $\mathbb{C}$ enriched and copowered over $\mathbb{V}$ and a $\mathbb{V}$-enriched monad R on $\mathbb{C}$, an $\mathbb{V}$-*enriched* R-residual *stateful runner* of T is an object $Y \in \mathbb{C}$ together with a family of maps $\theta_X : TX \bullet Y \to R(X \bullet Y)$ in $\mathbb{C}$ $\mathbb{V}$-natural in $X \in \mathbb{V}$ and satisfying two equations. Enriched stateful runners of T are in bijection with algebras of T in $(\mathbf{Kl}(R))^{\mathrm{op}}$.

Proof (sketch). The statement is wellformed since, as soon as $\mathbb{C}$ is $\mathbb{V}$-enriched and copowered by a functor $\bullet : \mathbb{V} \otimes \mathbb{C} \to \mathbb{C}$, we have that $\mathbf{Kl}(R)$ is $\mathbb{V}$-enriched and copowered by a functor $\mathbb{V} \otimes \mathbf{Kl}(R) \to \mathbf{Kl}(R)$ that agrees with $\bullet$ on objects. Therefore $(\mathbf{Kl}(R))^{\mathrm{op}}$ is $\mathbb{V}$-enriched and powered by the opposite of that functor. We have the following chain of bijections:

$$
\frac{\dfrac{TX \bullet Y \to R(X \bullet Y) \text{ in } \mathbb{C} \ \mathbb{V}\text{-nat. in } X}{TX \bullet Y \to X \bullet Y \text{ in } \mathbf{Kl}(R) \ \mathbb{V}\text{-nat. in } X}}{X \pitchfork Y \to TX \pitchfork Y \text{ in } (\mathbf{Kl}(R))^{\mathrm{op}} \ \mathbb{V}\text{-nat. in } X} \qquad\qquad \square
$$

The statement about the category of enriched stateful runners is:

Proposition 8. *If* $\mathbf{Alg}(T, (\mathbf{Kl}(R))^{\mathrm{op}})$ *exists as a* $\mathbb{V}$-*category, then so does* $\mathbb{V}$-$\mathbf{SRun}_R(T)$, *and the following is a pullback square (in* $\mathbb{V}$-$\mathbf{CAT}$*).*

$$
\begin{array}{ccc}
\mathbb{V}\text{-}\mathbf{SRun}_R(T) & \longrightarrow & (\mathbf{Alg}(T, (\mathbf{Kl}(R))^{\mathrm{op}}))^{\mathrm{op}} \\
{\scriptstyle U}\downarrow & & \downarrow{\scriptstyle U^{\mathrm{op}}} \\
\mathbb{C} & \xrightarrow{\quad J \quad} & \mathbf{Kl}(R)
\end{array}
$$

In the special case when $\mathbb{V} = \mathbb{C}$ and $R = \mathsf{Id}_{\mathbb{C}}$, we get $(\mathbf{Coalg}(\mathcal{M}^{\mathbb{C}}(T, \mathsf{Id}), U) \cong ((\mathbf{Alg}(T, \mathbb{C}^{\mathrm{op}}))^{\mathrm{op}}, U^{\mathrm{op}})$ ("coalgebras" of the $\mathbb{C}$-monad T).

By the same token, given a $\mathbb{V}$-enriched and powered category $\mathbb{C}$ and a $\mathbb{V}$-enriched comonad D on $\mathbb{C}$, we can define what an $\mathbb{V}$-*enriched D*-fuelled *continuation based runner* of T is: an object $Z \in \mathbb{C}$ together with a family of maps $\theta_X : D(X \pitchfork Z) \to TX \pitchfork Z$ in $\mathbb{C}$ that is $\mathbb{V}$-natural in $X \in \mathbb{V}$ and satisfies two equations. Enriched continuation-based runners of T are in bijection with algebras of T in the coKleisli category of D. Moreover:

Proposition 9. *If* $\mathbf{Alg}(T, \mathbf{CoKl}(D))$ *exists as a* $\mathbb{V}$*-category, then so does* $\mathbb{V}$-$\mathbf{CRun}_D(T)$*, and the following is a pullback square:*

$$
\begin{array}{ccc}
\mathbb{V}\text{-}\mathbf{CRun}_D(T) & \longrightarrow & \mathbf{Alg}(T, \mathbf{CoKl}(D)) \\
{\scriptstyle U}\downarrow & & \downarrow{\scriptstyle U} \\
\mathbb{C} & \xrightarrow{\ \ J\ \ } & \mathbf{CoKl}(D)
\end{array}
$$

7 Related Work

In semantics work, the use of monads as notions of computation was pioneered by Moggi [23], but the first to study comonads (or algebraic theories comodelled) as notions of environment (not under that name) were Shkaravska and Power [29]. This work was developed further by Plotkin and Power [24] and then Møgelberg and Staton [22] (who considered the enriched setting). Stateful runners appeared in Uustalu's paper [32], who noticed that nonresidual stateful runners of a set monad induced by an algebraic theory are in bijection with coalgebras of the comonad induced by the same theory (comodels). The concept of monad-comonad interaction law was distilled by Katsumata et al. [16], who also noticed that the universal interacting comonad of a monad is an instance of the Sweedler hom from Sweedler theory for duoidal categories; they calculated the dual and Sweedler dual for a number of cases. Uustalu and Voorneveld [33] noticed the bijection between monad-comonad interaction laws and suitable functors between categories of (co)algebras and that, in addition to stateful runners, monad-comonad interaction laws relate to continuation-based runners. Garner [12,11] further developed this thread. In particular, he gave a formula for the Sweedler duals of polynomial monads, and demonstrated properties of the dual/Sweedler dual (costructure/cosemantics) adjunction for accessible **Set**-(co)monads, such as its idempotency. He also pointed out that, when T and R are accessible **Set**-monads, the coalgebras of the Sweedler hom $\mathcal{M}(T, R)$ are algebras of T in $(\mathbf{Kl}(R))^{\mathrm{op}}$ with, as maps between them, maps in **Set** that $J : \mathbf{Set} \to \mathbf{Kl}(R)$ sends to algebra maps.

Independently, and earlier than in the semantics community, monad-comonad interaction laws were discovered among functional programmers by Kmett [19] and Freeman [8].

There is a disconnected and more mature thread of work in universal algebra started by Freyd [9] (or even Kan [15]), and continued by Tall and Wraith

[31,34] and Bergman and Hausknecht [5], studying functors from coalgebras of a covariety to algebras (like those of our Prop. 3) in the case $\mathbb{V} = \mathbf{Set}$, $R = \mathsf{Id}_\mathbb{C}$ of our enriched setting. (There are also textbook expositions, by Popescu and Popescu [26, Ch. 3] and Bergman [4, Ch. 10].) Strangely, this thread seems to have never been picked up in semantics work. It was not cited in the work by Power and coauthors [29,24], and the later authors (except Garner) have been unaware of it.

Sweedler's original work [30] was for (co)algebras over a field. Anel and Joyal [3] studied the Sweedler theory in great detail for dg-(co)algebras [3]. It was abstracted for (co)monoids in symmetric monoidal closed categories by Porst and Street [28] and Hyland et al. [14] (the internal hom of comonoids is older and goes back to Porst [27]) and then generalized for duoidal categories by López Franco and Vasilakopoulou [20]. A typical example duoidal structure on a functor category is given by the Day convolution and pointwise tensor. Garner and López Franco [13] considered the example of composition and the Day convolution of endofunctors (κ-accessible for a fixed κ).

We do not know the earliest reference to generalized algebras of a monad, in particular, coalgebras of a monad. The latter were considered by Poinsot and Porst [25] (and models of algebraic theories elsewhere than **Set** are standard).

8   Conclusion and Future Work

We have studied universal (co)monads for monad-comonad interactions. We have shown that an elegant setting for such a study on a more general level is provided by Sweedler theory for general duoidal categories as developed by López Franco and Vasilakopoulou [20]. But for results about monad-comonad interaction specifically it is fruitful to combine it with the (co)algebraic perspective on monad-comonad interaction laws [33]. This makes it possible to characterize the universal (co)monads defined by Sweedler operations via their categories of (co)algebras in terms of different flavors of runners.

We have witnessed that there is the choice of whether to work with ordinary monad-comonad interaction laws or with the enriched version. It remains to be seen which option yields a richer or more useful theory. An issue with the enriched option is that we know little about accessibility for enriched categories, although some studies exist (e.g., [18,6,7]).

We refrained from discussing it in this paper altogether, but of course one can specifically study interaction laws of monads and comonads specified by algebraic theories. We intend to do this in a sequel paper. We also plan to explain properly the significance for semantics of the constructions of this paper by describing in detail how they work on semantics-motivated examples and what this means.

References

1. Adámek, J., Rosický, J.: Locally Presentable and Accessible Categories, London Math. Soc. Lecture Note Series, vol. 189. Cambridge University Press (1994)
2. Aguiar, M., Mahajan, S.: Monoidal Functors, Species and Hopf Algebras, CRM Monograph Series, vol. 29. Amer. Math. Soc. (2010)
3. Anel, M., Joyal, A.: Sweedler theory of (co)algebras and the bar-cobar construction. arXiv eprint 1309.6952 [math.CT] (2013), https://arxiv.org/abs/1309.6952
4. Bergman, G.M.: An Invitation to General Algebra and Universal Constructions. Universitext, Springer (2015). https://doi.org/10.1007/978-3-319-11478-1, author's revised version at https://math.berkeley.edu/~gbergman/245/
5. Bergman, G.M., Hausknecht, A.O.: Cogroups and Co-rings in Categories of Associative Rings, AMS Mathematical Surveys and Monographs, vol. 45. Amer. Math. Soc. (1996)
6. Bird, G.J.: Limits in 2-Categories of Locally-Presented Categories. Ph.D. thesis, University of Sydney (1984)
7. Borceux, F., Quinteiro, C.: Enriched accessible categories. Bull. Austral. Math. Soc. **54**, 489–501 (1996). https://doi.org/10.1017/s0004972700021900
8. Freeman, P.: Comonads as spaces (a series of blog posts) (2016), https://blog.functorial.com/posts/2016-08-07-Comonads-As-Spaces.html
9. Freyd, P.: Algebra valued functors in general and tensor products in particular. Coll. Math. **14**(1), 89–106 (1966). https://doi.org/10.4064/cm-14-1-89-106
10. Garner, R.: Understanding the small object argument. Appl. Categ. Struct. **17**(3), 247–285 (2009). https://doi.org/10.1007/s10485-008-9137-4
11. Garner, R.: Stream processors and comodels. In: Gadducci, F., Silva, A. (eds.) Proc. of 9th Conf. on Algebra and Coalgebra in Computer Science, CALCO 2021 (Salzburg, Aug./Sept. 2021), Leibniz Int. Proc. in Informatics, vol. 211, pp. 15:1–15:17. Dagstuhl Publishing (2021). https://doi.org/10.4230/lipics.calco.2021.15
12. Garner, R.: The costructure-cosemantics adjunction for comodels for computational effects. Math. Struct. Comput. Sci. (to appear). https://doi.org/10.1017/s0960129521000219
13. Garner, R., López Franco, I.: Commutativity. J. Pure Appl. Algebra **204**(2), 1707–1751 (2016). https://doi.org/10.1016/j.jpaa.2015.09.003
14. Hyland, M., López Franco, I., Vasilakopoulou, C.: Hopf measuring comonoids and enrichment. Proc. London Math. Soc. **115**(3), 1118–1148 (2017). https://doi.org/10.1112/plms.12064
15. Kan, D.M.: On monoids and their dual. Bol. Soc. Mat. Mexicana, Ser. 2 **3**, 52–61 (1958)
16. Katsumata, S., Rivas, E., Uustalu, T.: Interaction laws of monads and comonads. In: Proc. of 35th Ann. ACM/IEEE Symp. on Logic in Computer Science, LICS 2020 (Saarbrücken, July 2020), pp. 604–618. ACM (2020). https://doi.org/10.1145/3373718.3394808
17. Kelly, G.M.: Basic Concepts of Enriched Category Theory, London Math. Soc. Lecture Note Series, vol. 64. Cambridge University Press (1982), reprinted (2005) as: *Reprints in Theory and Applications of Categories* 10, http://www.tac.mta.ca/tac/reprints/articles/10/tr10abs.html
18. Kelly, M.G.: Structures defined by finite limits in the enriched context, I. Cahiers Topol. Géom. Différentielle Catégoriques **23**(1), 3–42 (1982)
19. Kmett, E.: Monads from comonads (a series of blog posts) (2011), http://comonad.com/reader/2011/monads-from-comonads/

20. López Franco, I., Vasilakopoulou, C.: Duoidal categories, measuring comonoids and enrichment. arXiv eprint 2005.01340 [math.CT] (2020), https://arxiv.org/abs/2005.01340
21. Makkai, M., Paré, R.: Accessible Categories: The Foundations of Categorical Model Theory: The Foundations of Categorical Model Theory, Contemporary Mathematics, vol. 104. Amer. Math. Soc. (1989)
22. Møgelberg, R.E., Staton, S.: Linear usage of state. Log. Methods Comput. Sci. **10**(1) (2014). https://doi.org/10.2168/lmcs-10(1:17)2014
23. Moggi, E.: Computational lambda-calculus and monads. In: Proc. of 4th Ann. Symp. on Logic in Computer Science, LICS '89, pp. 14–23. IEEE Press (1989). https://doi.org/10.1109/lics.1989.39155
24. Plotkin, G., Power, J.: Tensors of comodels and models for operational semantics. Electron. Notes Theor. Comput. Sci. **218**, 295–311 (2008). https://doi.org/10.1016/j.entcs.2008.10.018
25. Poinsot, L., Porst, H.E.: Internal coalgebras in cocomplete categories: Generalizing the Eilenberg-Watts theorem. J. Algebra Appl. **20**(9), art. 2510165 (2021). https://doi.org/10.1142/s0219498821501656
26. Popescu, N., Popescu, L.: Theory of Categories. Editura Academiei / Sijthoff & Noordhoff Int. Publishers (1979)
27. Porst, H.E.: On categories of monoids, comonoids, and bimonoids. Quaest. Math. **31**(2), 127–139 (2008). https://doi.org/10.2989/qm.2008.31.2.2.474
28. Porst, H.E., Street, R.: Generalizations of the Sweedler dual. Appl. Categ. Struct. **24**, 619–647 (2016). https://doi.org/10.1007/s10485-016-9450-2
29. Power, J., Shkaravska, O.: From comodels to coalgebras: State and arrays. Electron. Notes Theor. Comput. Sci. **106**, 297–314 (2004). https://doi.org/10.1016/j.entcs.2004.02.041
30. Sweedler, M.E.: Hopf Algebras. Math. Lecture Note Series, W. A. Benjamin (1969)
31. Tall, D.O., Wraith, G.C.: Representable functors and operations on rings. Proc. London Math. Soc., Ser. 3 **20**(4), 619–643 (1970). https://doi.org/10.1112/plms/s3-20.4.619
32. Uustalu, T.: Stateful runners for effectful computations. Electron. Notes Theor. Comput. Sci. **319**, 403–421 (2015). https://doi.org/10.1016/j.entcs.2015.12.024
33. Uustalu, T., Voorneveld, N.: Algebraic and coalgebraic perspectives on interaction laws. In: d. S. Oliveira, B.C. (ed.) Proc. of 18th Asian Symp. on Programming Languages and Systems, APLAS 2020 (Fukuoka, Nov./Dec. 2020), Lect. Notes Comput. Sci., vol. 12470, pp. 186–205. Springer (2020). https://doi.org/10.1007/978-3-030-64437-6_10
34. Wraith, G.C.: Algebraic Theories (Lectures Autumn 1969, Revised Version of Notes), Lecture Note Series, vol. 22. Aarhus Universitet, Matematisk Institut (1975)

Model Checking Temporal Properties of Recursive Probabilistic Programs

Tobias Winkler✉ , Christina Gehnen , and Joost-Pieter Katoen

RWTH Aachen University, Aachen, Germany
{tobias.winkler,katoen}@cs.rwth-aachen.de
christina.gehnen@rwth-aachen.de

Abstract. Probabilistic pushdown automata (pPDA) are a standard operational model for programming languages involving discrete random choices, procedures, and returns. Temporal properties are useful for gaining insight into the chronological order of events during program execution. Existing approaches in the literature have focused mostly on ω-regular and LTL properties. In this paper, we study the model checking problem of pPDA against ω-visibly pushdown languages that can be described by specification logics such as CaRet and are strictly more expressive than ω-regular properties. With these logical formulae, it is possible to specify properties that explicitly take the structured computations arising from procedural programs into account. For example, CaRet is able to match procedure calls with their corresponding future returns, and thus allows to express fundamental program properties like total and partial correctness.

Keywords: Probabilistic Recursive Programs · Model Checking · Probabilistic Pushdown Automata · Visibly Pushdown Languages · CaRet.

1 Introduction

Probabilistic programs extend traditional programs with the ability to flip coins or, more generally, sample values from probability distributions. These programs can be used to encode randomized algorithms and randomized mechanisms in security [7] in a natural way. The interest in probabilistic programs has significantly increased in recent years. To a large extent, this is due to the search in AI for more expressive and succinct languages than probabilistic graphical models for Bayesian inference [17]. Probabilistic programs have many applications [24]. They are used in, amongst others, machine learning, systems biology, security, planning and control, quantum computing, and software–defined networks. Probabilistic variants of many programming languages exist.

Procedural programs allow for declaration of procedures—small independent code blocks—and the ability to *call* procedures from one another, possibly in

```
proc void infectYoung() {              proc bool infectElder() {
    y := uniform(0, 3)                     y := uniform(0, 1)
    repeat y times {                       repeat y times {
       infectYoung()   }                      infectYoung()   }
    e := uniform(0, 2)                     e := uniform(0, 4)
    repeat e times {                       repeat e times {
       f := infectElder()   }                 infectElder()   }
    return   }                             f := bernoulli(0.01); return f   }
```

Fig. 1. Recursive probabilistic program modeling the outbreak of an infectious disease. $\texttt{uniform}(a, b)$ stands for the *discrete* uniform distribution on $[a, b]$.

a recursive fashion. Most common programming languages such as C, Python, or Java support procedures. It is thus not surprising that recursion is a key ingredient in many modern probabilistic programming languages (PPL). In fact, many early approaches to extend Bayesian networks focused on incorporating recursion [26,19,11,27]. Randomized algorithms such as Hoare's quicksort with random pivot selection can be straightforwardly programmed using recursion. Recursion is also a first-class citizen in modeling rule-based dependencies between molecules or populations in systems biology (e.g., modeling reproduction).

This paper studies the automated verification of *probabilistic pushdown automata* [14] (pPDA) as an explicit-state operational model of procedural probabilistic programs against temporal specifications. As a motivating example, let us consider a simple epidemiological model for the outbreak of an infectious disease in a large population where the number of susceptible individuals can be assumed to be infinite. Our example

	Y	E
Y	1.5	1
E	0.5	2

Fig. 2. Example infection rates by age groups.

model distinguishes young and elderly persons. Each affected individual infects a uniformly distributed number of others, with varying rates (expected values) according to the age groups (Figure 2). The fatality rate for infected elderly and young persons is 1% and 0%, respectively. Initially, we assume there is a single infected young person, i.e., the overall program is started by calling *infectYoung()*. It is an easy task for any working programmer to specify this model as a discrete probabilistic program with mutually recursive procedures (Figure 1). Note that this program can be easily amended to more realistic models involving, e.g., more age or gender groups, other distributions, hospitalization rate, etc.

The operational behavior of programs such as the one in Figure 1 can be naturally described by pPDA. The technical details of such a translation are beyond the scope of this paper but let us provide some intuition (more details can be found e.g. in [2]). Roughly, the *local* states of the procedures—the valuation of the local variables and the position of the program counter—constitute both the state space and the stack alphabet of the automaton. Procedure calls correspond to push transitions in the automaton in such a way that the program's *procedure*

stack is simulated by the automaton's pushdown stack, i.e., the current local state is saved on top of the stack. Accordingly, *returning* from a procedure corresponds to taking a pop transition in order to restore the local state of the caller. Returning a value can be handled similarly. Clearly, if the reachable local state spaces of the involved procedures are finite, then the resulting automaton will be finite as well.

A number of relevant questions such as "Will the virus eventually become extinct?" (termination probability) or "What is the expected number of fatalities?" (expected costs) can be decided on finite pPDA (see [9] for a survey). In this work, we focus on *temporal properties*, e.g., questions that involve reasoning about the chronological order of certain events of interest during the epidemic. An example are chains of infection: For instance, we might ask

> *What is the probability that eventually a young person with only young persons in their chain of infection passes the virus on to an elderly person who then dies?*

On the level of the program in Figure 1, this corresponds to the probability of reaching a *global* program configuration where the call stack only contains *infectYoung*() invocations and during execution of the current *infectYoung*(), the local variable f is eventually set to true. This requires reasoning about the nestings of calls and returns of a computation. In fact, in order to decide if $f = true$ in the current procedure, we must "skip" over all calls within it and only consider their local return values. This requirement and many others can be rather naturally expressed in the logic *CaRet* [3], an extension of LTL:

$$\Diamond^g \left(\Box^- p_Y \ \wedge \ p_Y \ \wedge \ \Diamond^a f \right) .$$

Here, p_Y is an atomic proposition that holds at states which correspond to being in procedure *infectYoung*, and f indicates that $f = true$. Intuitively, the above formula states that eventually (outer $\Diamond^g$), the computation reaches a (global) state where only *infectYoung* is on the call stack and the current procedure is *infectYoung* as well ($\Box^- p_Y \ \wedge \ p_Y$), and moreover the local—aka *abstract*—path *within* in the current procedure reaches a state where f is true ($\Diamond^a f$). Such properties are in general context-free but not always regular and thus cannot be expressed in LTL [3].

Technical Contribution. We are given a (finite) pPDA Δ and a CaRet formula φ and we are interested in determining the probability that a random trajectory of Δ satisfies φ. In order for this problem to be decidable [13], we need to impose a mild *visibility* restriction on Δ, yielding a probabilistic *visibly* pushdown automaton (pVPA). Just like several previous works on model checking pPDA against ω-regular specifications [14,10,21], we follow the automata-based approach (see Figure 3). More specifically, we first translate φ into an equivalent non-deterministic *Büchi visibly pushdown automaton* [4] (VPA) $\mathcal{A}$ and then determinize it using a result of [22]. The resulting DVPA $\mathcal{D}$ uses a so-called *stair-parity* [22] acceptance condition that is strictly more expressive than standard parity or Muller DVPA [4]. Stair-parity differs from usual parity in that
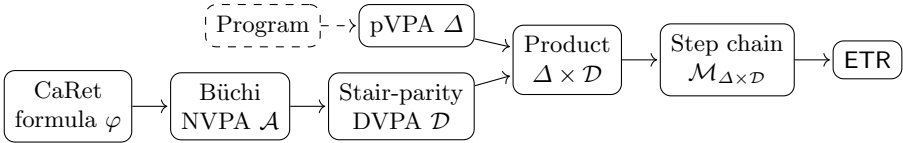
Fig. 3. Chain of reductions used in this paper. ETR stands for *existential theory of the reals*, i.e., the existentially quantified fragment of the FO-theory over $(\mathbb{R}, +, \cdot, \leq)$.

it only considers certain positions—called *steps* [22]—of an infinite word where the stack height never decreases again. We then construct a standard product $\Delta \times \mathcal{D}$. Here, the visibility conditions ensure that the automata synchronize their stack actions, yielding a product automaton that uses *a single stack* instead of two independent ones, which would lead to undecidability [13]. Finally, we are left with computing a stair-parity acceptance probability in the product, which is itself a pPDA. This is achieved by constructing a specific finite Markov chain associated to $\Delta \times \mathcal{D}$, called *step chain* in this paper. Intuitively, the step chain jumps from one step of a run to the next, and therefore we only need to evaluate *standard parity* rather than stair-parity on the step chain. The idea of step chains is due to [14] where they were used to show decidability against deterministic non-pushdown Büchi automata. For constructing the step chain, certain *termination probabilities* of the pPDA need to be computed. These are in general algebraic numbers that cannot always be expressed by radicals [16], let alone by rationals. However, the relevant problems are still decidable via an encoding in the existential fragment of the FO-theory of the reals (ETR) [21].

The resulting main contributions of this paper are complexity results, summarized in Figure 4, and algorithms for quantitative model checking of pPDA against ω-VPL given in terms of either deterministic automata, non-deterministic automata, or as CaRet formulae. As common in the literature, we consider the special case of qualitative, or *almost-sure* (a.-s.), model checking separately. To the best of our knowledge, none of these problems was known to be decidable before. The work of [13] proved decidability of model checking against deterministic Muller VPA which capture a strict subset of the CaRet-definable languages [4]. As a lemma of independent interest, we show that the step chain can be used for checking all kinds of measurable properties defined on steps, even beyond parity.

Related work. We have already mentioned various works on recursion in probabilistic graphical models (and PPL) as well as on verifying pPDA and the equivalent model of recursive Markov chains [16]. The analysis of these models focuses on reachability probabilities, ω-regular properties or (fragments of) probabilistic CTL, expected costs, and termination probabilities. The computation of termination probabilities in recursive Markov chains and variations thereof with non-determinism is supported by the software tool PReMo [29]. Our paper can be seen as a natural extension from checking pPDA against ω-regular properties to ω-visibly pushdown languages. In contrast to these algorithmic approaches, various deductive reasoning methods have been developed for recursive

ω-VPL given in terms of ...	qualitative	quantitative
Deterministic stair-parity VPA [Theorem 3]	in PSPACE	in PSPACE
Non-deterministic Büchi VPA [Theorem 4]	EXPTIME-compl.	in EXPSPACE
CaRet formula [Theorem 5]	in 2EXPTIME	in 2EXPSPACE

Fig. 4. Complexity results of this paper.

probabilistic programs. Proof rules for recursion were first provided in [20], and later extended to proof rules in a weakest-precondition reasoning style [23,25]. Olmedo *et al.* [25] also address the connection to pPDA and provide proof rules for expected run-time analysis. A mechanized method for proving properties of randomized algorithms, including recursive ones, for the Coq proof assistant is presented in [5]. The Coq approach is based on higher–order logic using a monadic interpretation of programs as probabilistic distributions.

Organization. We review the basics about VPA and CaRet in Section 2. Section 3 introduces probabilistic *visibly* pushdown automata (pVPA). The stair-parity DVPA model checking procedure is presented in Section 4, and the results for Büchi VPA and CaRet in Section 5. We conclude the paper in Section 6.

2   Visibly Pushdown Languages

We fix some general notation for words first. Given a non-empty alphabet Σ, let Σ^* be the set of finite words (this includes the empty word ϵ), and let Σ^ω be the set of infinite words over Σ. For $i \geq 0$, the i-th symbol of a word $w \in \Sigma^* \cup \Sigma^\omega$ is denoted $w(i)$ if it exists. $|w|$ denotes the length of w.

2.1   Visibly Pushdown Automata

A finite alphabet Σ is called a *pushdown alphabet* if it is equipped with a partition $\Sigma = \Sigma_{\mathsf{call}} \uplus \Sigma_{\mathsf{int}} \uplus \Sigma_{\mathsf{ret}}$ into three—possibly empty—subsets of *call*, *internal*, and *return* symbols. A *visibly pushdown automaton* [4] (VPA) over Σ is like a standard pushdown automaton with the additional syntactic restriction that reading a call or return symbol triggers a push or a pop transition, respectively. Reading an internal symbol, on the other hand, does not affect the stack at all.

Definition 1 (VPA [4]). *Let Σ be a pushdown alphabet. A* visibly pushdown automaton (VPA) *over Σ is a tuple $\mathcal{A} = (S, s_0, \Gamma, \bot, \delta, \Sigma)$ with S a finite set of states, $s_0 \in S$ an initial state, Γ a finite stack alphabet, $\bot \in \Gamma$ a special bottom-of-stack symbol, and $\delta = (\delta_{\mathsf{call}}, \delta_{\mathsf{int}}, \delta_{\mathsf{ret}})$ a triple of relations*

$$\delta_{\mathsf{call}} \subseteq (S \times \Sigma_{\mathsf{call}}) \times (S \times \Gamma_{\text{-}\bot}) \,, \quad \delta_{\mathsf{int}} \subseteq (S \times \Sigma_{\mathsf{int}}) \times S \,, \quad \delta_{\mathsf{ret}} \subseteq (S \times \Sigma_{\mathsf{ret}} \times \Gamma) \times S$$

where $\Gamma_{-\perp} = \Gamma \setminus \{\perp\}$. For $s, t \in S$, $Z \in \Gamma$, and $a \in \Sigma$, we use the shorthand notations $s \xrightarrow{a} tZ$, $s \xrightarrow{a} t$, $sZ \xrightarrow{a} t$ to indicate that there exist transitions $(s, a, t, Z) \in \delta_{\mathsf{call}}$, $(s, a, t) \in \delta_{\mathsf{int}}$, $(s, a, Z, t) \in \delta_{\mathsf{ret}}$, respectively. Note that e.g. $s \xrightarrow{a} tZ$ implies implicitly that $a \in \Sigma_{\mathsf{call}}$ and $Z \neq \perp$, and similar for internal and return transitions. Intuitively, call transitions push a new symbol Z onto the stack, internal transitions ignore the stack, and return transitions pop the topmost symbol Z from the stack (unless $Z = \perp$, in which case nothing is popped). A *configuration* of VPA $\mathcal{A}$ is a tuple $(s, \gamma) \in S \times \Gamma^*$, written more succinctly as $s\gamma$ in the sequel. Let $w \in \Sigma^\omega$ be an infinite input word. An infinite sequence $\rho = s_0\gamma_0, s_1\gamma_1 \ldots$ of configurations is called a *run* of $\mathcal{A}$ on w if $s_0\gamma_0 = s_0\perp$ and for all $i \geq 0$, exactly one of the following cases applies:

- $w(i) \in \Sigma_{\mathsf{call}}$ and $\gamma_{i+1} = \gamma_i Z$ for some $Z \in \Gamma_{-\perp}$ such that $s_i \xrightarrow{w(i)} s_{i+1}Z$; or
- $w(i) \in \Sigma_{\mathsf{int}}$ and $\gamma_{i+1} = \gamma_i$ and $s_i \xrightarrow{w(i)} s_{i+1}$; or
- $w(i) \in \Sigma_{\mathsf{ret}}$ and $\gamma_{i+1}Z = \gamma_i$ for some $Z \in \Gamma_{-\perp}$ such that $s_iZ \xrightarrow{w(i)} s_{i+1}$, or $\gamma_i = \gamma_{i+1} = \perp$ and $s_i\perp \xrightarrow{w(i)} s_{i+1}$.

A *Büchi acceptance condition* for $\mathcal{A}$ is a subset $F \subseteq S$. A VPA equipped with a Büchi condition is called a *Büchi VPA*. An infinite word $w \in \Sigma^\omega$ is accepted by a Büchi VPA if there exists a run $s_0\gamma_0, s_1\gamma_1, \ldots$ of $\mathcal{A}$ on w such that $s_i \in F$ for infinitely many $i \geq 0$. The ω-language of words accepted by a Büchi VPA $\mathcal{A}$ is denoted $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^\omega$.

Definition 2 (ω-VPL [4]). *Let Σ be a pushdown alphabet. $L \subseteq \Sigma^\omega$ is an ω-visibly pushdown language (ω-VPL) if $L = \mathcal{L}(\mathcal{A})$ for a Büchi VPA $\mathcal{A}$ over Σ.*

A VPA is *deterministic* (DVPA) if it has exactly one run on each input word. In this case, $\delta_{\mathsf{call}}, \delta_{\mathsf{int}}$, and δ_{ret} can be viewed as (total) functions. As for standard NBA, the class of languages recognized by Büchi DVPA is a strict subset of the languages recognized by non-deterministic Büchi VPA. Unlike in the non-pushdown case, DVPA with Muller or parity conditions are also strictly less expressive than non-deterministic Büchi VPA [4]. A deterministic automaton model for ω-VPL was given in [22]. It uses a so-called *stair-parity* acceptance condition which is the topic of the next subsection.

2.2 Steps and Stair-parity Conditions

Let us fix a pushdown alphabet Σ and a VPA $\mathcal{A}$ over Σ. Consider a run $\rho = s_0\gamma_0, s_1\gamma_1, \ldots$ of $\mathcal{A}$ on an infinite word $w \in \Sigma^\omega$. We define the *stack height* of the i-th configuration as $sh(\rho(i)) = |\gamma_i| - 1$ (the bottom symbol $\perp$ does not count to the stack height). The stair-parity condition relies on the notion of *steps*:

Definition 3 (Step). *Let ρ be a run of $\mathcal{A}$. Position $i \geq 0$ is a* step *of ρ if*

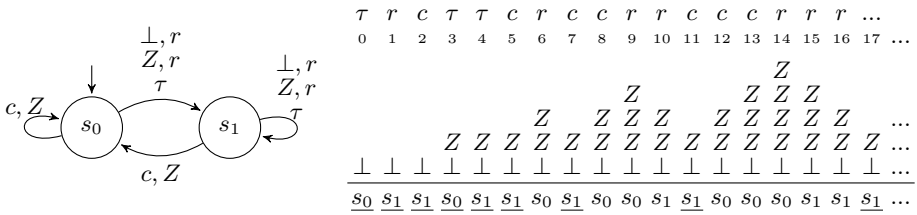$$\forall n \geq i: \quad sh(\rho(n)) \ \geq \ sh(\rho(i)) \ .$$

Fig. 5. Left: An example VPA (in fact, a DVPA) with $\Gamma = \{Z, \bot\}$ over input alphabet $\Sigma = \{c\} \uplus \{\tau\} \uplus \{r\}$. Transitions labeled c, Z are *call transitions* which push Z on the stack, the transitions labeled with τ are *internal* ones that ignore the stack, and those labeled Z, r and $\bot, r$ are *return transitions* that are only enabled if Z ($\bot$, resp.) is on top of the stack; when executing Z, r we also pop Z from the stack. However, the special bottom-of-stack symbol $\bot$ can never be popped (see e.g. pos. 1). Right: The *unique* run of the DVPA on input word $\tau\, r\, c\, \tau\, \tau\, c\, r\, c^2\, r^2\, c^3\, r^3 \dots$. Steps are underlined.

Abusing terminology, we may also refer to the configurations at the step positions of a run as steps.

Example 1. Figure 5 depicts a DVPA and the initial fragment of its unique run ρ on the input word $\tau\, r\, c\, \tau\, \tau\, c\, r\, c^2\, r^2\, c^3\, r^3 \dots$. The step positions are underlined, i.e., positions 0-5, 7, 11, and 17 are steps. Note that if $\rho(i) = s\bot$ for some $s \in S$ then i is a step, i.e., *bottom configurations* are always steps.

Steps play a central role in the rest of the paper. We therefore explain some of their fundamental properties.

- If positions $i < j$ are adjacent steps, then $sh(\rho(j)) - sh(\rho(i)) \in \{0, 1\}$, i.e., the stack height from one step to the next increases by either zero or one. More precisely, if the symbol at step position i is internal (e.g. $i = 0, 3, 4$ in Figure 5) or a return (e.g. $i = 1$) then the next step is simply the next configuration $j = i + 1$ and the stack height does not increase. If the symbol at position i is a call, then one of two cases occurs: Either the call has no matching future return (e.g. $i = 2$); in this case, the next step is the next configuration $j = i + 1$. Otherwise the call is eventually matched (e.g. $i = 5, 7, 11$) and the next step $j > i + 1$ occurs after the corresponding matching return is read and has the same stack height.
- Each infinite run has infinitely many steps since the above discussion also implies that each step has a successor. Notice though that the difference between two adjacent step positions may grow unboundedly as in the example.
- As a consequence, the stack height at the steps either grows unboundedly or eventually stabilizes (the latter occurs in Figure 5).

Remark 1. One can also define the steps of a *word* $w \in \Sigma^\omega$ as the positions where a run of *any arbitrary* VPA on w has a step. Due to the visibility restriction, the actual behaviour of the VPA does not influence the step positions [22]. In other words, the step positions are predetermined by the input word. Thus, we can also speak of the stack height $sh(w(i))$ of word w at position i.

We need one last notion before defining stair-parity. The *footprint* of an infinite run $\rho = s_0\gamma_0, s_1\gamma_1, \ldots$ is the infinite sequence $\rho\!\downarrow_{Steps} = s_{n_0}s_{n_1}\ldots \in S^\omega$ where for all $i \geq 0$ the position n_i is the i-th step of ρ. Phrased differently, $\rho\!\downarrow_{Steps}$ is the projection of the run ρ onto the states occurring at its steps. For the example run in Figure 5 (right), $\rho\!\downarrow_{Steps} = s_0s_1s_1s_0s_1^\omega$.

Definition 4 (Stair-parity [22]). *Let $\mathcal{A}$ be a VPA over pushdown alphabet Σ. A* stair-parity *acceptance condition for $\mathcal{A}$ is defined in terms of a priority function $\Omega\colon S \to \mathbb{N}_0$. i.e. A word $w \in \Sigma^\omega$ is accepted if $\mathcal{A}$ has a run ρ on ω s.t.*

$$\min \{\, k \in \mathbb{N}_0 \mid \overset{\infty}{\exists} i\colon\ \Omega(\,\rho\!\downarrow_{Steps}(i)\,) = k \,\}$$

is even. The language accepted by $\mathcal{A}$ is denoted $\mathcal{L}(\mathcal{A})$.

Example 2. The DVPA in Figure 5 with $\Omega(s_0) = 1$ and $\Omega(s_1) = 2$ accepts

$$\mathcal{L}_{repbdd} = \{\, w \in \Sigma^\omega \mid \exists B \geq 0,\ \overset{\infty}{\exists} i \geq 0\colon\ sh(w(i)) \ \leq\ B \,\} \,,$$

the language of *repeatedly bounded* words [22], i.e., words whose stack height (cf. Remark 1) is infinitely often at most a constant B. It is known that $\mathcal{L}_{repbdd}$ is not expressible by DVPA with usual parity conditions [4].

Theorem 1 ([22, Thm. 1]). *For every non-deterministic Büchi VPA $\mathcal{A}$ there exists a deterministic stair-parity DVPA $\mathcal{D}$ with $2^{\mathcal{O}(|S|^2)}$ states such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{D})$. Moreover, $\mathcal{D}$ can be constructed in exponential time in the size of $\mathcal{A}$.*

It was also shown in [22] that stair-parity DVPA characterize exactly the class of ω-VPL (and are thus not more expressive than non-deterministic Büchi VPA).

2.3   CaRet, a Temporal Logic of Calls and Returns

Specifying requirements directly in terms of automata is tedious in practice. *CaRet* [3] is an extension of Linear Temporal Logic (LTL) that can be used to describe ω-VPL. Its syntax is defined as follows:

Definition 5 (CaRet [3]). *Let AP be a finite set of atomic propositions. The logic CaRet adheres to the grammar*

$$\varphi \;\coloneqq\; p \mid \varphi \vee \varphi \mid \neg\varphi \mid \bigcirc^g\varphi \mid \varphi\,\mathcal{U}^g\varphi \mid \bigcirc^a\varphi \mid \varphi\,\mathcal{U}^a\varphi \mid \bigcirc^-\varphi \mid \varphi\,\mathcal{U}^-\varphi \,,$$

where $p \in AP \cup \{\, \mathsf{call}, \mathsf{int}, \mathsf{ret} \,\}$.

Other common modalities such as $\Diamond^b$ and $\Box^b$ for $b \in \{\, g, a, - \,\}$ are defined as usual via $\Diamond^b\varphi = true\,\mathcal{U}^b\,\varphi$, and $\Box^b\varphi = \neg\Diamond^b\neg\varphi$. We briefly explain the semantics of CaRet, the formal definition can be found in [3] or the full version [28]. We assume familiarity with LTL. CaRet formulae are interpreted over infinite words from the pushdown alphabet $\Sigma = 2^{AP} \times \{\, \mathsf{call}, \mathsf{int}, \mathsf{ret} \,\}$. $\bigcirc^g$ and $\mathcal{U}^g$ are the standard next and until modalities from LTL (called *global* next and until
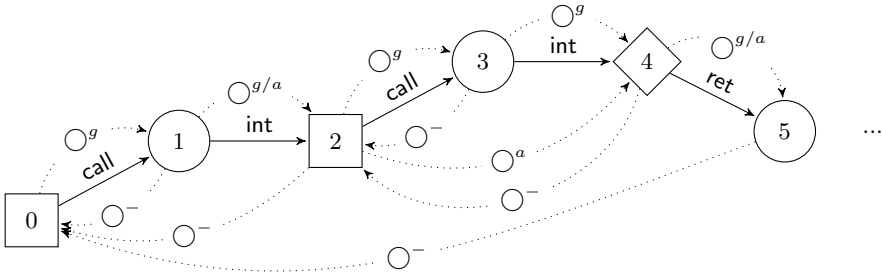
Fig. 6. CaRet's various next modalities applied to the initial fragment of an example word. Call, internal, and return positions are depicted as boxes, circles, and rhombs, resp. Note that $\bigcirc^a$ of position 3 is undefined because $\bigcirc^g$ is a return.

in CaRet). CaRet extends LTL by two key operators, the *caller* modality $\bigcirc^-$ and the *abstract successor* $\bigcirc^a$, see Figure 6. The former is a *past* modality that refers to the position of the last pending call. For internal and return symbols, the abstract successor $\bigcirc^a$ behaves like $\bigcirc^g$ unless the latter is a return, in which case $\bigcirc^a$ is undefined (e.g. pos. 3 in the example). On the other hand, the abstract successor of a call symbol is its *matching return* if it exists, or undefined otherwise. The until modalities $\mathcal{U}^-$ and $\mathcal{U}^a$ are defined over the paths induced by the callers and abstract successors, respectively. Note that the caller path is always finite and the abstract path can be either finite or infinite. A prime application of CaRet is to state Hoare-like total correctness of a procedure F [3]:

$$\varphi_{total} \quad = \quad \square^g \left(\mathsf{call} \wedge p \wedge p_F \ \rightarrow \ \bigcirc^a q \right)$$

where p and q are atomic propositions that hold at the states where the pre- and post-condition is satisfied, respectively, and p_F is an atomic proposition marking the calls to F. Another example is the language of repeatedly bounded words from Example 2; it is $\mathcal{L}_{repbdd} = \mathcal{L}(\lozenge^g \square^g (\mathsf{call} \rightarrow \bigcirc^a \mathsf{ret}))$. Further examples are given in [3]. The language defined by a CaRet formula φ is denoted $\mathcal{L}(\varphi)$.

Theorem 2 ([1, Thm. 5.1]). *CaRet-definable languages are ω-VPL: For each CaRet formula φ there exists a (non-deterministic) Büchi VPA $\mathcal{A}$ such that $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A})$, and $\mathcal{A}$ can be constructed in time $2^{\mathcal{O}(|\varphi|)}$.*

The above theorem is well-known in the literature [1,2] even though it is usually stated for *Nested Word Automata* (NWA) which are equivalent to VPA, and it is more common to state a space bound on $\mathcal{A}$ rather than a time bound for the construction. The theorem also applies to more expressive extensions of CaRet [1] which we do not consider here for the sake of simplicity.

3   Probabilistic Visibly Pushdown Automata

As explained in the introductory section, we employ *probabilistic pushdown automata* [14] (pPDA) as an operational model for procedural probabilistic pro-

grams. pPDA thus play a fundamentally different role in this paper than VPA (cf. Definition 1): While the former are used to model the *system*, the latter encode the *specification*. Consequently, our pPDA do *not* read an input word like VPA do, but instead take their transitions randomly, according to fixed probability distributions. In this way, they define a probability space over their possible traces, i.e., runs projected on their labeling sequence. These traces constitute the input words of the VPA. In order for the model checking problems to be decidable [13], a syntactic visibility restriction related—but not exactly analogous—to the one required by VPA needs to be imposed on pPDA. In a nutshell, the condition is that each state only has outgoing transitions of one *type*, i.e., push, internal, or pop. This means that the stack operation is *visible in the states* (recall that for VPA, the stack operation is visible in the input symbol). This restriction is not severe in the context of modeling programs (see Remark 2 further below) and leads to our notion of probabilistic *visibly* pushdown automata (pVPA) which we now define formally.

Given a finite set X, we write $\mathfrak{D}(X) = \{\, f \colon X \to [0,1] \mid \sum_{a \in X} f(a) = 1 \,\}$ for the set of probability distributions on X.

Definition 6 (pVPA). *A probabilistic visibly pushdown automaton (pVPA) is a tuple $\Delta = (Q, q_0, \Gamma, \bot, P, \Sigma, \lambda)$ where Q is a finite set of* states *partitioned into $Q = Q_{\mathsf{call}} \uplus Q_{\mathsf{int}} \uplus Q_{\mathsf{ret}}$, $q_0 \in Q$ is an* initial state, *Γ is a finite* stack alphabet, *$\bot \in \Gamma$ is a special* bottom-of-stack symbol, *$P = (P_{\mathsf{call}}, P_{\mathsf{int}}, P_{\mathsf{ret}})$ is a triple of functions with signature*

$$P_{\mathsf{call}} \colon Q_{\mathsf{call}} \to \mathfrak{D}(Q \times \Gamma_{-\bot}) \;, \quad P_{\mathsf{int}} \colon Q_{\mathsf{int}} \to \mathfrak{D}(Q) \;, \quad P_{\mathsf{ret}} \colon Q_{\mathsf{ret}} \times \Gamma \to \mathfrak{D}(Q) \;,$$

$\Sigma = \Sigma_{\mathsf{call}} \uplus \Sigma_{\mathsf{int}} \uplus \Sigma_{\mathsf{ret}}$ is a pushdown alphabet, and $\lambda \colon Q \to \Sigma$ is a state labeling function consistent with the visibility condition, i.e., for all $\mathsf{type} \in \{\mathsf{call}, \mathsf{int}, \mathsf{ret}\}$ and all $q \in Q$, we have that $q \in Q_{\mathsf{type}}$ iff $\lambda(q) \in \Sigma_{\mathsf{type}}$.

Intuitively, the behavior of a pVPA Δ is as follows. If the current state q is a call state, then the probability distribution $P_{\mathsf{call}}(q)$ determines a random successor state and stack symbol to be pushed on the stack ($\bot$ cannot be pushed). Similarly, if the current state is internal, then $P_{\mathsf{int}}(q)$ is the distribution over possible successor states and the stack is ignored completely. Lastly, if the current state is a return state and symbol $Z \in \Gamma$ is on top of the stack, then $P_{\mathsf{ret}}(q, Z)$ once again determines the probability distribution of successor states, and additionally Z is removed from the stack. Similar to VPA, the bottom symbol $\bot$ is the only exception to this rule, it can never be removed. Thus, pVPA are a generalization of labeled *Markov chains*, which correspond to the special case $Q = Q_{\mathsf{int}}$.

We now define the semantics of pVPA more formally. For $q, r \in Q, Z \in \Gamma$ and $p > 0$ we use the shorthand notations $q \xrightarrow{p} rZ$, $q \xrightarrow{p} r$, and $qZ \xrightarrow{p} r$ to indicate that $P_{\mathsf{call}}(q)(r, Z) = p$, $P_{\mathsf{int}}(q)(r) = p$, and $P_{\mathsf{ret}}(q, Z)(r) = p$, respectively. As for VPA, a *configuration* of a pVPA is an element $q\gamma \in Q \times \Gamma^*$. An (infinite) *run* of a pVPA is a sequence of configurations $\rho = q_0\gamma_0, q_1\gamma_1, \dots$ such that $q_0\gamma_0 = q_0\bot$ and for all $i \geq 0$ we have that either

1. $q_i \in Q_{\mathsf{call}}$, $\gamma_{i+1} = \gamma_i Z$ for some $Z \in \Gamma_{-\bot}$ and $q_i \xrightarrow{p} q_{i+1}Z$;

2. $q_i \in Q_{\text{int}}$, $\gamma_{i+1} = \gamma_i$ and $q_i \xrightarrow{p} q_{i+1}$; or
3. $q_i \in Q_{\text{ret}}$, $\gamma_{i+1}Z = \gamma_i$ for some $Z \in \Gamma_\perp$ and $q_iZ \xrightarrow{p} q_{i+1}$, or $\gamma_{i+1} = \gamma_i$ and $q_i\perp \xrightarrow{p} q_{i+1}$ (because the bottom symbol $\perp$ is never popped).

Note that our pVPA only produce infinite runs and do not simply "terminate" upon reaching the empty stack as in e.g. [14]. In fact, in our case the stack cannot be empty due to the special bottom symbol $\perp$ that can never be popped. We have chosen to avoid finite pVPA runs for compatibility with CaRet which describes ω-languages per definition. Nonetheless, terminating behavior can be easily simulated in our framework by moving to a dedicated sink state once the pVPA attempts to pop $\perp$ for the first time.

The set of all runs of a pVPA Δ is denoted $Runs_\Delta$. We extend Δ's labeling function λ to runs $\rho \in Runs_\Delta$ by applying it to each state along ρ individually, yielding a word $\lambda(\rho) \in \Sigma^\omega$. Steps of pVPA runs are defined as in Definition 3. An example pVPA and its possible runs are depicted in Figure 7 on page 14.

We can view the set of all configurations $Q \times \Gamma^*$ as the (infinite) state space of a discrete-time Markov chain. In this way, we obtain a probability space $(Runs_\Delta, \mathcal{F}, \mathbb{P})$ via the usual cylinder set construction [6, Ch. 10].

Remark 2. The visibility restriction of our pVPA is slightly different from the definition given in [13] which requires all *incoming* transitions to a state to be of the same type, i.e., call, internal, or return. Our definition, on the other hand, imposes the same requirement on the states' *outgoing* transitions. We believe that our condition is more natural for pVPA obtained from procedural programs, such as the one in Figure 1. In fact, programs where randomness is restricted to *internal* statements such as $x := \texttt{bernoulli}(0.5)$ or $x := \texttt{uniform}(0,3)$ naturally comply with our visibility condition because all call and return states of such programs are deterministic and thus cannot violate visibility. However, the alternative condition of [13] is not necessarily fulfilled for such programs.

We can now formally state our main problem of interest:

Definition 7 (Probabilistic CaRet Model Checking). *Let AP be a finite set of atomic propositions, φ be a CaRet formula over AP, Δ be a pVPA with labels from the pushdown alphabet $\Sigma = 2^{AP} \times \{\, \textsf{call}, \textsf{int}, \textsf{ret} \,\}$, and $\theta \in [0,1] \cap \mathbb{Q}$. The* quantitative CaRet Model Checking problem *is to decide whether*

$$\mathbb{P}(\{\, \rho \in Runs_\Delta \mid \lambda(\rho) \in \mathcal{L}(\varphi) \,\}) \geq_? \theta .$$

The qualitative *CaRet Model Checking problem is the special case where $\theta = 1$.*

The probabilities in Definition 7 are well-defined as ω-VPL are measurable [22].

4   Model Checking against Stair-parity DVPA

In this section, we show that model checking pVPA (Definition 6) against VPL given in terms of a stair-parity DVPA (Definition 4) is decidable. This is achieved by first computing an automata-theoretic product of the pVPA and the DVPA and then evaluating the acceptance condition in the product automaton.

4.1 Products of Visibly Pushdown Automata

In general, pushdown automata are not closed under taking products as this would require *two* independent stacks. However, the visibility conditions on VPA and pVPA ensure that their product is again an automaton with just a single stack because the stack operations (push, nop, or pop) are forced to synchronize.

We now define the product formally. An *unlabeled* pVPA is a pVPA where the labeling function λ and alphabet Σ are omitted.

Definition 8 (Product $\Delta \times \mathcal{D}$). *Let $\Delta = (Q, q_0, \Gamma, \bot, P, \Sigma, \lambda)$ be a pVPA, and $\mathcal{D} = (S, s_0, \Gamma', \bot, \delta, \Sigma)$ be a DVPA over pushdown alphabet Σ. The product of Δ and $\mathcal{D}$ is the unlabeled pVPA*

$$\Delta \times \mathcal{D} \;=\; (\, Q \times S, \; (q_0, s_0), \; \Gamma \times \Gamma', \; \langle \bot, \bot \rangle, \; P_{\Delta \times \mathcal{D}} \,) \,,$$

where $P_{\Delta \times \mathcal{D}}$ is the smallest set of transitions satisfying the following rules for all $q, r \in Q$, $Z \in \Gamma$, $s, t \in S$, and $Y \in \Gamma'$:

$$\frac{q \xrightarrow{p}_\Delta rZ \;\wedge\; s \xrightarrow{\lambda(q)}_\mathcal{D} tY}{(q,s) \xrightarrow{p}_{\Delta \times \mathcal{D}} (r,t)\langle Z, Y\rangle} \quad \frac{q \xrightarrow{p}_\Delta r \;\wedge\; s \xrightarrow{\lambda(q)}_\mathcal{D} t}{(q,s) \xrightarrow{p}_{\Delta \times \mathcal{D}} (r,t)} \quad \frac{qZ \xrightarrow{p}_\Delta r \;\wedge\; sY \xrightarrow{\lambda(q)}_\mathcal{D} t}{(q,s)\langle Z,Y\rangle \xrightarrow{p}_{\Delta \times \mathcal{D}} (r,t)} \,.$$

$$\textit{(call)} \qquad\qquad\qquad \textit{(internal)} \qquad\qquad\qquad \textit{(return)}$$

If the DVPA $\mathcal{D}$ is equipped with a priority function $\Omega \colon S \to \mathbb{N}_0$, then we extend Ω to $\Omega' \colon Q \times S \to \mathbb{N}_0$ via $\Omega'(q, s) = \Omega(s)$.

It is not difficult to show that $\Delta \times \mathcal{D}$ is indeed a well-defined pVPA and moreover satisfies the following property (the proof is standard, see [28]):

Lemma 1 (Soundness of $\Delta \times \mathcal{D}$). *Let Δ be a pVPA and $\mathcal{D}$ be a stair-parity DVPA with priority function Ω, both over pushdown alphabet Σ. Then the product pVPA $\Delta \times \mathcal{D}$ with priority function Ω' as in Definition 8 satisfies*

$$\mathbb{P}(\{\, \rho \in Runs_\Delta \mid \lambda(\rho) \in \mathcal{L}(\mathcal{D}) \,\}) \;=\; \mathbb{P}(\{\, \rho \in Runs_{\Delta \times \mathcal{D}} \mid \rho\!\downarrow_{Steps} \in Parity_{\Omega'} \,\}),$$

where $Parity_{\Omega'}$ denotes the set of words in $(Q \times S)^\omega$ satisfying the standard *parity condition defined by Ω'. Moreover, $\Delta \times \mathcal{D}$ can be constructed in polynomial time.*

Remark 3. It is *not* actually important that the product satisfies the visibility condition. All techniques we apply to the product also work for general pPDA.

4.2 Stair-parity Acceptance Probabilities in pVPA

Lemma 1 effectively reduces model checking pVPA against stair-parity DVPA to computing stair-parity acceptance in the product, which is again an (unlabeled) pVPA. We therefore focus on pVPA in this section and do not consider DVPA.

Throughout the rest of this section, let $\Delta = (\, Q, q_0, \Gamma, \bot, P\,)$ be an unlabeled pVPA. On the next pages we describe the construction of a finite Markov chain $\mathcal{M}_\Delta$ that we call the *step chain* of Δ. Loosely speaking, $\mathcal{M}_\Delta$ simulates jumping from one *step* (see Definition 3) of a run of Δ to the next. A similar idea first appeared in [14]. Our construction, however, differs from the original one in various aspects. We discuss this in detail in Remark 5 further below.

Steps as events. For all $n \in \mathbb{N}_0$, we define a *random* variable $V^{(n)}$ on $Runs_\Delta$ whose value is either the state q of Δ at the n-th step, or the *extended state* $q\bot$ in the special case where the n-th step occurs at a bottom configuration of the form $q\bot$, for some $q \in Q$. We denote the set of all such extended states with $Q\bot = \{\, q\bot \mid q \in Q \,\}$. Formally, $V^{(n)} \colon Runs_\Delta \to Q \cup Q\bot$ is defined as

$$V^{(n)}(\rho) \quad = \quad \begin{cases} q & \text{if } step_n(\rho) = q\gamma \text{ and } \gamma \neq \bot \\ q\bot & \text{if } step_n(\rho) = q\bot\ , \end{cases}$$

where $step_n(\rho)$ denotes the configuration at the n-th step of ρ. Note that $V^{(0)} = q_0\bot$ because the first position of a run is always a step.

Lemma 2. *For all $n \in \mathbb{N}_0$ and $v \in Q \cup Q\bot$, the event $V^{(n)} = v$ is measurable, and thus $V^{(n)}$ is a well-defined random variable.*

We can view the sequence $V^{(0)}, V^{(1)} \ldots$ of random variables as a *stochastic process*. It is intuitively clear that for all $n \in \mathbb{N}_0$, the value of $V^{(n+1)}$ depends only on $V^{(n)}$, but not on $V^{(i)}$ for $i < n$. This is due to the more general observation that the state q at any step configuration $q\gamma$ (with $\gamma \neq \bot$) *fully determines the future of the run* because being a step already implies that no symbol in γ can ever be read as reading it implies popping it from the stack. In particular, q determines the probability distribution over possible next steps. A similar observation applies to bottom configurations of the form $q\bot$. Phrased in probability theoretical terms, the process $V^{(0)}, V^{(1)} \ldots$ has the *Markov property*, i.e.,

$$\mathbb{P}(V^{(n)}{=}v_n \mid V^{(n-1)}{=}v_{n-1} \wedge \ldots \wedge V^{(0)}{=}v_0) \;=\; \mathbb{P}(V^{(n)}{=}v_n \mid V^{(n-1)}{=}v_{n-1}) \quad (1)$$

holds for all values of $v_0, \ldots, v_n$ such that the above conditional probabilities are well-defined [1]. This was proved in detail in [14]. It is also clear that the Markov process is time-homogeneous in the sense that

$$\mathbb{P}(V^{(n+1)} = v \mid V^{(n)} = v') \quad = \quad \mathbb{P}(V^{(n'+1)} = v \mid V^{(n')} = v')$$

holds for all $n, n' \in \mathbb{N}_0$ for which the two conditional probabilities are well-defined. The following example provides some intuition on these facts.

Example 3. Consider the pVPA in Figure 7 (left). The initial fragments of its two equiprobable runs are depicted in the middle. In this example, it is easy to read off the next-step probabilities $\mathbb{P}(V^{(n)} = v_n \mid V^{(n-1)} = v_{n-1})$ for all $n \in \mathbb{N}_0$ and $v_n, v_{n-1} \in Q \cup Q\bot$. They are summarized in the Markov chain on the right. For example, $V^{(0)} = q_0\bot$ holds with probability 1, and $V^{(1)} = q_1$ and $V^{(1)} = q_3\bot$ hold with probability 1/2 each because the second step occurs either at position 1 with configuration $q_1\bot Z$ or at position 3 with configuration $q_3\bot$,

---

[1] A conditional probability is well-defined if the *condition*, i.e., the event on the right hand side of the vertical bar, has positive probability. Expressions like the one in (1) are thus not necessarily well-defined because the probability that $V^{(n-1)} = v_{n-1}$ might be zero for certain values of n and v_{n-1}.
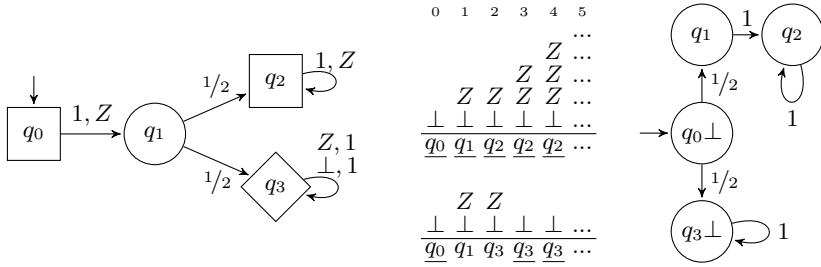
Fig. 7. Left: An example (unlabeled) pVPA Δ. Call, internal, and return states are depicted as squares, circles, and rhombs, respectively. The format of the transition labels is analogous to Figure 5 (left). Middle: Initial fragments of the two possible runs of Δ. Steps are underlined. Right: Its step Markov chain $\mathcal{M}_\Delta$ (Definition 10, page 15).

	$q \to r$	$q\bot \to r$	$q\bot \to r\bot$	$q \to r\bot$
$q \in Q_{\text{call}}$	$\frac{[r\uparrow]}{[q\uparrow]}\left(\sum_{r',Z} P_{\text{call}}(q,r'Z)[r'Z\downarrow r] + \sum_Z P_{\text{call}}(q,rZ)\right)$	$\sum_Z P_{\text{call}}(q,rZ)[r\uparrow]$	$\sum_{r',Z} P_{\text{call}}(q,r'Z)[r'Z\downarrow r]$	0
$q \in Q_{\text{int}}$	$\frac{[r\uparrow]}{[q\uparrow]} P_{\text{int}}(q,r)$	0	$P_{\text{int}}(q,r)$	0
$q \in Q_{\text{ret}}$	n/a	0	$P_{\text{ret}}(q\bot,r)$	n/a

Fig. 8. Next-step probabilities of the step Markov chain. P_{type} for type $\in \{\text{call}, \text{int}, \text{ret}\}$ are the probabilities of the pVPA's call, internal, and return transitions, respectively. The values $[r'Z\downarrow r]$ and $[q\uparrow]$ are the return and diverge probabilities from Definition 9.

and both options are equally likely. The case $\mathbb{P}(V^{(2)} = q_2 \mid V^{(1)} = q_1) = 1$ is slightly more interesting: Given that a configuration $q_1\gamma$ with $\gamma \neq \bot$ is a step, we know that the next state must be q_2 (which is then also a step). Even though there is a transition from q_1 to q_3 in Δ, the next state cannot be q_3 because the latter is a return state which would immediately decrease the stack height of γ. This shows that, intuitively speaking, *conditioning on being a step influences the probabilities of a state's outgoing transitions.*

Probabilities of next steps, returns, and diverges. Our next goal is to provide expressions for the next-step probabilities $\mathbb{P}(V^{(n+1)} = v' \mid V^{(n)} = v)$ as we did in Example 3. It turns out that those can be stated in terms of the *return* and *diverge* probabilities of Δ.

Definition 9. *Let $p, q \in Q$, $Z \in \Gamma$, and $\gamma \in \Gamma^*$. We define*

- *the* return *probability $[pZ\downarrow q]$ as the probability to reach configuration $q\gamma$ from $p\gamma Z$ without visiting another configuration of the form $r\gamma$ for some $r \in Q$ in between; and*
- *the* diverge *probability $[p\uparrow]$ as the probability to never decrease the stack height below $|\gamma Z|$ when starting in $p\gamma Z$, i.e., $[p\uparrow] = 1 - \sum_{q \in Q}[pZ\downarrow q]$.*

Note that $[p{\uparrow}]$ is indeed independent of Z because the only way to read Z is by popping it from the stack which decreases the stack height. The diverge probabilities are closely related to steps. Indeed, the probability that a configuration $p\gamma$ with $\gamma \neq \bot$ is a step is equal to $[p{\uparrow}]$. For example, in the pVPA in Figure 7 the configuration $q_1{\bot}Z$ is a step with probability $[q_1{\uparrow}] = 1/2$.

It is known that the return and diverge probabilities are in general *non-rational*. As a minimal example, consider a pVPA that repeats the following steps until emptying its stack or getting stuck: (i) It pushes four symbols with probability $1/6$, or (ii) pops one symbol with probability $1/2$, or (iii) gets stuck otherwise. The resulting return probability is the least solution of $x = (1/6)x^5 + 1/2$, a non-rational number that is not even solvable by radicals [16, Thm. 3.2(1)].

Remark 4. The terms *return* and *diverge* are natural. When modeling procedural probabilistic programs as pVPA, $[pZ{\downarrow}q]$ is just the probability to eventually *return* from local state p of the current procedure to local state q of the calling procedure (the return address is stored on the stack in Z). Similarly, $[p{\uparrow}]$ is the probability that the current procedure diverges, i.e., it never returns to the calling context. Clearly, this is independent of the return address.

Lemma 3. *The conditional next-step probabilities in Figure 8 are correct in the sense that if $\mathbb{P}(V^{(n+1)} = v' \mid V^{(n)} = v)$ is defined for $n \in \mathbb{N}_0$ and $v, v' \in Q \cup Q\bot$ then it is equal to the probability in the respective column "$v \to v'$".*

Proof sketch. We only provide some intuition for two important cases; formal derivations are in [28]. Let $r \in Q$ be arbitrary.

- If $q \in Q_{\mathsf{int}}$ then $\mathbb{P}(V^{(n+1)} = r \mid V^{(n)} = q) = P_{\mathsf{int}}(q, r)[r{\uparrow}]/[q{\uparrow}]$: Suppose that the n-th step takes place at position i of the run. Since the n-th step occurs at an *internal* state q, the $n+1$-st step must necessarily occur immediately at position $i+1$. The factor $P(q, r)[r{\uparrow}]$ is proportional to the probability to take an (internal) transition from q to r and then diverge in r, which is necessary in order for the next configuration to be a step. However, the values $\{\, P(q, r)[r{\uparrow}] \mid r \in Q \,\}$ do not form a probability distribution in general. This justifies the division by the normalizing constant $[q{\uparrow}] = \sum_{r \in Q} P(q, r)[r{\uparrow}]$.
- If $q \in Q_{\mathsf{call}}$ then $\mathbb{P}(V^{(n+1)} = r\bot \mid V^{(n)} = q\bot) = \sum_{r', Z} P_{\mathsf{call}}(q, r'Z)[r'Z{\downarrow}r]$: If the n-th step occurs at bottom configuration $q\bot$, then the $n+1$-st step can only occur at *bottom* configuration $r\bot$ if the symbols pushed by q's outgoing transitions are eventually popped. The expression in the sum equals the probability to take a push-transition from q to r' that pushes Z onto the stack multiplied by the probability to *return* from r' (with Z on top) to r.

The step chain. It is convenient to view the stochastic process $V^{(0)}, V^{(1)} \ldots$ as an explicit (graphical) Markov chain.

Definition 10 (The Step Chain $\mathcal{M}_\Delta$). *$\mathcal{M}_\Delta$ is the Markov chain with states*

$$M \quad = \quad \{\, q \in Q_{\mathsf{call}} \cup Q_{\mathsf{int}} \mid [q{\uparrow}] > 0 \,\} \ \cup \ Q\bot \,,$$
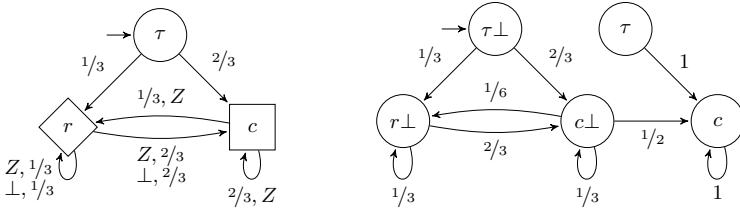
Fig. 9. Left: Example pVPA with the following return-diverge probabilities: $[cZ{\downarrow}c] = \frac{1}{6}$, $[cZ{\downarrow}r] = \frac{1}{12}$, $[rZ{\downarrow}r] = \frac{1}{3}$, $[rZ{\downarrow}c] = \frac{2}{3}$, and $[c{\uparrow}] = \frac{3}{4}$, $[\tau{\uparrow}] = \frac{1}{2}$, $[r{\uparrow}] = 0$. Even though it is the case here, these probabilities are not always rational [16]. Right: Its step Markov chain according to Definition 10. The transition probabilities can be computed using the return and diverge probabilities and Figure 8.

initial state $q_0{\perp}$, *and for all* $v, v' \in M$, *the probability of transition* $v \to v'$ *is defined according to Figure 8.*

Figure 9 depicts a non-trivial pVPA and its step chain. In this example, all return and diverge probabilities are rational. In general, however, the return and diverge probabilities (Definition 9) are algebraic numbers that are not always rational or even expressible by radicals [16]. As a consequence, one cannot easily perform numerical computations on the step chain. However, the probabilities can be encoded implicitly as the unique solution of an *existential theory of the reals* (ETR) formula, i.e. an existentially quantified FO-formula over $(\mathbb{R}, +, \cdot, \leq)$ [14]. Since the ETR is decidable, many questions about the step chain are still decidable as well. We will make use of this in Theorem 3 below.

The property of $\mathcal{M}_\Delta$ that is most relevant to us is given by the following Lemma 4. We call $\rho{\Downarrow}_{Steps} = V^{(0)}(\rho)V^{(1)}(\rho)\ldots$ the *extended footprint* of run ρ.

Lemma 4 (Soundness of $\mathcal{M}_\Delta$). *Let Δ be a pVPA with step chain $\mathcal{M}_\Delta$. Let M be the states of the step chain and consider a measurable set $R \subseteq M^\omega$. Then*

$$\mathbb{P}(\{\, \rho \in Runs_\Delta \mid \rho{\Downarrow}_{Steps} \in R \,\}) \quad = \quad \mathbb{P}(R) \ .$$

Proof sketch. For basic cylinder sets of the form $R = w \cdot M^\omega$ for some $w \in M^*$, the claim follows from the Markov property (1) together with the correctness of the transition probabilities of $\mathcal{M}_\Delta$ according to Lemma 3. For other measurable sets, it can be shown by induction over the levels of the Borel hierarchy [28].

Remark 5. The step chain as presented here differs from the original definition in [14] in at least two important aspects. First, we have to take the semantics of our special bottom symbol $\perp$ into account. This is why our chain uses a subset of $Q \cup Q{\perp}$ as states—it must distinguish whether a step occurs at a bottom configuration. The pPDA in [14], on the other hand, may have both finite and infinite runs, and this needs to be handled differently in the step chain. Second, we use step chains for a different purpose than [14], namely to show that general measurable properties defined on steps—this includes stair-parity—can be evaluated on pVPA (Lemma 4).
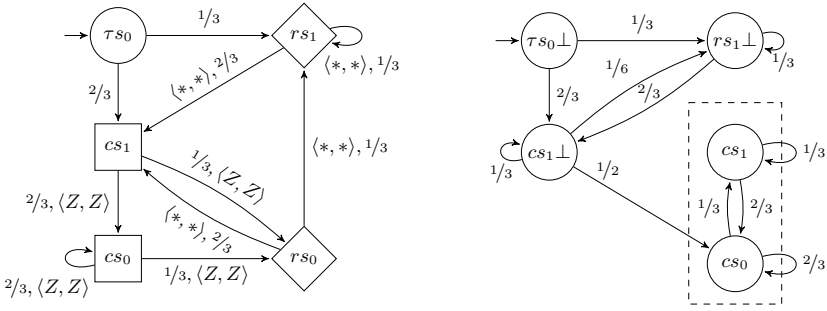
Fig. 10. Left: The product of the pVPA from Figure 9 (left) and the DVPA from Figure 5 (left) on page 7. Right: Its step chain according to Definition 10. The dashed region is the only BSCC. It violates the parity condition $\Omega(s_0) = 1$ and $\Omega(s_1) = 2$ inherited from the DVPA (see Example 2 on page 8) since every run reaching the BSCC visits cs_0 infinitely often with probability 1. Only reachable states are depicted.

Putting it all together. We can now prove the main result of this section.

Theorem 3. *Let Δ be a pVPA and let $\mathcal{D}$ be a stair-parity DVPA, both over the same pushdown alphabet Σ. Then for all $\theta \in [0,1] \cap \mathbb{Q}$, the problem $\mathbb{P}(\{\rho \in Runs_\Delta \mid \lambda(\rho) \in \mathcal{L}(\mathcal{D})\}) \geq_? \theta$ is decidable in* PSPACE.

Proof sketch. We first construct the product $\Delta \times \mathcal{D}$ according to Definition 8. By Lemma 1 we need to compute the stair-parity acceptance probability of $\Delta \times \mathcal{D}$. Lemma 4 reduces this to computing a usual parity acceptance probability in the step chain $\mathcal{M}_{\Delta \times \mathcal{D}}$. This can be achieved through finding the bottom strongly connected components (BSCC) of $\mathcal{M}_{\Delta \times \mathcal{D}}$, classifying them as *good* (the minimum priority of a BSCC state is even) or otherwise *bad*, and running a standard reachability analysis wrt. the good states. See Figure 10 for an example. The remaining technical difficulty is that the transition probabilities of $\mathcal{M}_{\Delta \times \mathcal{D}}$ are not rational in general. However, this can be dealt with using the fact that these probabilities are expressible in the ETR [14] (see [28] for the details).

4.3 Probabilistic One-counter Automata

A probabilistic visibly *one-counter automaton* (pVOC) is the special case of a pVPA with unary stack alphabet, i.e., $|\Gamma_{\!-\!\perp}| = 1$. For example, the pVPA in Figure 9 (left) is a pVOC. For many problems, better complexity bounds are known for pVOC than for the general case. In particular, $[p\!\uparrow] > 0$ can be decided in P [9, Thm. 4]. We can exploit this to improve Theorem 3 in the pVOC case:

Corollary 1. *Let Δ be a pVOC and $\mathcal{D}$ be a stair-parity DVPA over pushdown alphabet Σ. The problem $\mathbb{P}(\{\rho \in Runs_\Delta \mid \lambda(\rho) \in \mathcal{L}(\mathcal{D})\}) =_? 1$ is decidable in* P.

Corollary 1 implies that there exist efficient algorithms for many properties of pVOC-expressible random walks on $\mathbb{N}_0$. In fact, a.-s. satisfaction of each *fixed* visibly-pushdown property can be decided in P. For instance, using the DVPA from Figure 5 we can decide if a random walk is a.-s. repeatedly bounded.

5   Model Checking against Büchi VPA and CaRet

With Theorems 1 and 3 it follows immediately that quantitative model checking of pVPA against non-deterministic Büchi VPA is decidable in EXPSPACE. We can improve the complexity in the qualitative case:

Theorem 4. *Let Δ be a pVPA and $\mathcal{A}$ be a (non-deterministic) Büchi VPA over the same pushdown alphabet. The problem $\mathbb{P}(\{\rho \in Runs_\Delta \mid \lambda(\rho) \in \mathcal{L}(\mathcal{A})\}) =_? 1$ is EXPTIME-complete.*

In the above result, membership in EXPTIME relies on the fact that one can construct the underlying *graph* of a step chain $\mathcal{M}_{\Delta \times \mathcal{D}}$ in time exponential in the size of Δ but *polynomial* in the size of $\mathcal{D}$; see [28]. EXPTIME-hardness follows from [15, Thm. 8]. In fact, qualitative model checking of pPDA against *non-pushdown* Büchi automata is also EXPTIME-complete [15]. With Theorems 1 to 4 we immediately obtain the following complexity results for CaRet model checking:

Theorem 5. *The quantitative and qualitative probabilistic CaRet model checking problems (Def. 7) are decidable in 2EXPSPACE and 2EXPTIME, respectively.*

Both problems are known to be EXPTIME-hard [30].

6   Conclusion

We have presented the first decidability result for model checking pPDA—an operational model of procedural discrete probabilistic programs—against CaRet, or more generally, against the class of ω-VPL. We heavily rely on the determinization procedure from [22] and the notion of a step chain used in previous works. These two constructions turn our to be natural match.

We conjecture that our complexity bounds are not the best possible which is often the case in purely automata-based model checking. Future work is thus to investigate whether the doubly-exponential complexity can be lowered to singly-exponential, e.g. by generalizing the automata-less algorithm from [30]. Other topics are to explore to what extent algorithms for probabilistic CTL can be generalized to the branching-time variant of CaReT [18], to consider more expressive logics such as visibly LTL [8] or OPTL [12], and to study the interplay of *conditioning* and recursion [27] through the lens of pPDA.

References

1. Alur, R., Arenas, M., Barceló, P., Etessami, K., Immerman, N., Libkin, L.: First-Order and Temporal Logics for Nested Words. In: 22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10-12 July 2007, Wroclaw, Poland, Proceedings. pp. 151–160. IEEE Computer Society (2007). https://doi.org/10.1109/LICS.2007.19
2. Alur, R., Bouajjani, A., Esparza, J.: Model Checking Procedural Programs. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 541–572. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8_17
3. Alur, R., Etessami, K., Madhusudan, P.: A Temporal Logic of Nested Calls and Returns. In: Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings. Lecture Notes in Computer Science, vol. 2988, pp. 467–481. Springer (2004). https://doi.org/10.1007/978-3-540-24730-2_35
4. Alur, R., Madhusudan, P.: Visibly Pushdown Languages. In: Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004. pp. 202–211. ACM (2004). https://doi.org/10.1145/1007352.1007390
5. Audebaud, P., Paulin-Mohring, C.: Proofs of randomized algorithms in Coq. Sci. Comput. Program. **74**(8), 568–589 (2009). https://doi.org/10.1016/j.scico.2007.09.002
6. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
7. Barthe, G., Köpf, B., Olmedo, F., Béguelin, S.Z.: Probabilistic Relational Reasoning for Differential Privacy. ACM Trans. Program. Lang. Syst. **35**(3), 9:1–9:49 (2013). https://doi.org/10.1145/2492061
8. Bozzelli, L., Sánchez, C.: Visibly Linear Temporal Logic. J. Autom. Reason. **60**(2), 177–220 (2018). https://doi.org/10.1007/s10817-017-9410-z
9. Brázdil, T., Esparza, J., Kiefer, S., Kucera, A.: Analyzing probabilistic pushdown automata. Formal Methods Syst. Des. **43**(2), 124–163 (2013). https://doi.org/10.1007/s10703-012-0166-0
10. Brázdil, T., Kucera, A., Strazovský, O.: On the Decidability of Temporal Properties of Probabilistic Pushdown Automata. In: STACS 2005, 22nd Annual Symposium on Theoretical Aspects of Computer Science, Stuttgart, Germany, February 24-26, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3404, pp. 145–157. Springer (2005). https://doi.org/10.1007/978-3-540-31856-9_12
11. Casini, L., Illari, P.M., Russo, F., Williamson, J.: Recursive Bayesian Networks. Theoria. Revista de Teoria, Historia y Fundamentos de la Ciencia **26**(1), 5–33 (2008)
12. Chiari, M., Mandrioli, D., Pradella, M.: Operator precedence temporal logic and model checking. Theor. Comput. Sci. **848**, 47–81 (2020). https://doi.org/10.1016/j.tcs.2020.08.034
13. Dubslaff, C., Baier, C., Berg, M.: Model checking probabilistic systems against pushdown specifications. Inf. Process. Lett. **112**(8-9), 320–328 (2012). https://doi.org/10.1016/j.ipl.2012.01.006
14. Esparza, J., Kucera, A., Mayr, R.: Model Checking Probabilistic Pushdown Automata. In: 19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings. pp. 12–21. IEEE Computer Society (2004). https://doi.org/10.1109/LICS.2004.1319596

15. Etessami, K., Yannakakis, M.: Algorithmic Verification of Recursive Probabilistic State Machines. In: Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3440, pp. 253–270. Springer (2005). https://doi.org/10.1007/978-3-540-31980-1_17

16. Etessami, K., Yannakakis, M.: Recursive markov chains, stochastic grammars, and monotone systems of nonlinear equations. J. ACM **56**(1), 1:1–1:66 (2009). https://doi.org/10.1145/1462153.1462154

17. Gordon, A.D., Henzinger, T.A., Nori, A.V., Rajamani, S.K.: Probabilistic programming. In: Proceedings of the on Future of Software Engineering, FOSE 2014, Hyderabad, India, May 31 - June 7, 2014. pp. 167–181. ACM (2014). https://doi.org/10.1145/2593882.2593900

18. Gutsfeld, J.O., Müller-Olm, M., Nordhoff, B.: A Branching Time Variant of CaRet. In: Model Checking Software - 25th International Symposium, SPIN 2018, Malaga, Spain, June 20-22, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10869, pp. 153–170. Springer (2018). https://doi.org/10.1007/978-3-319-94111-0_9

19. Jaeger, M.: Complex Probabilistic Modeling with Recursive Relational Bayesian Networks. Ann. Math. Artif. Intell. **32**(1-4), 179–220 (2001). https://doi.org/10.1023/A:1016713501153

20. Jones, C.: Probabilistic non-determinism. Ph.D. thesis, University of Edinburgh, UK (1990), http://hdl.handle.net/1842/413

21. Kucera, A., Esparza, J., Mayr, R.: Model Checking Probabilistic Pushdown Automata. Log. Methods Comput. Sci. **2**(1) (2006). https://doi.org/10.2168/LMCS-2(1:2)2006

22. Löding, C., Madhusudan, P., Serre, O.: Visibly Pushdown Games. In: FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Chennai, India, December 16-18, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3328, pp. 408–420. Springer (2004). https://doi.org/10.1007/978-3-540-30538-5_34

23. McIver, A., Morgan, C.: Partial correctness for probabilistic demonic programs. Theor. Comput. Sci. **266**(1-2), 513–541 (2001). https://doi.org/10.1016/S0304-3975(00)00208-5

24. van de Meent, J., Paige, B., Yang, H., Wood, F.: An Introduction to Probabilistic Programming. CoRR **abs/1809.10756** (2018), http://arxiv.org/abs/1809.10756

25. Olmedo, F., Kaminski, B.L., Katoen, J., Matheja, C.: Reasoning about Recursive Probabilistic Programs. In: Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016. pp. 672–681. ACM (2016). https://doi.org/10.1145/2933575.2935317

26. Pfeffer, A., Koller, D.: Semantics and Inference for Recursive Probability Models. In: Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA. pp. 538–544. AAAI Press / The MIT Press (2000), http://www.aaai.org/Library/AAAI/2000/aaai00-082.php

27. Stuhlmüller, A., Goodman, N.D.: A Dynamic Programming Algorithm for Inference in Recursive Probabilistic Programs. CoRR **abs/1206.3555** (2012), http://arxiv.org/abs/1206.3555

28. Winkler, T., Gehnen, C., Katoen, J.: Model Checking Temporal Properties of Recursive Probabilistic Programs. CoRR **abs/2111.03501** (2021), https://arxiv.org/abs/2111.03501

29. Wojtczak, D., Etessami, K.: PReMo : An Analyzer for Probabilistic Recursive Models. In: Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 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. Lecture Notes in Computer Science, vol. 4424, pp. 66–71. Springer (2007). https://doi.org/10.1007/978-3-540-71209-1_7

30. Yannakakis, M., Etessami, K.: Checking LTL Properties of Recursive Markov Chains. In: Second International Conference on the Quantitative Evaluaiton of Systems (QEST 2005), 19-22 September 2005, Torino, Italy. pp. 155–165. IEEE Computer Society (2005). https://doi.org/10.1109/QEST.2005.8

Author Index