Helmut Seidl (Ed.)

LNCS 4423

Foundations of Software Science and Computational Structures

10th International Conference, FOSSACS 2007
Held as Part of the Joint European Conferences
on Theory and Practice of Software, ETAPS 2007
Braga, Portugal, March/April 2007, Proceedings

European Joint Conferences on
Theory
And
Practice of
Software
2007

Springer

Foreword

ETAPS 2007 is the tenth instance of the European Joint Conferences on Theory and Practice of Software, and thus a cause for celebration.

The events that comprise ETAPS address various aspects of the system development process, including specification, design, implementation, analysis and improvement. The languages, methodologies and tools which support these activities are all well within its scope. Different blends of theory and practice are represented, with an inclination towards theory with a practical motivation on the one hand and soundly based practice on the other. Many of the issues involved in software design apply to systems in general, including hardware systems, and the emphasis on software is not intended to be exclusive.

History and Prehistory of ETAPS

ETAPS as we know it is an annual federated conference that was established in 1998 by combining five conferences [Compiler Construction (CC), European Symposium on Programming (ESOP), Fundamental Approaches to Software Engineering (FASE), Foundations of Software Science and Computation Structures (FOSSACS), Tools and Algorithms for Construction and Analysis of Systems (TACAS)] with satellite events.

All five conferences had previously existed in some form and in various colocated combinations: accordingly, the prehistory of ETAPS is complex. FOSSACS was earlier known as the Colloquium on Trees in Algebra and Programming (CAAP), being renamed for inclusion in ETAPS as its historical name no longer reflected its contents. Indeed CAAP's history goes back a long way; prior to 1981, it was known as the Colleque de Lille sur les Arbres en Algebre et en Programmation. FASE was the indirect successor of a 1985 event known as Colloquium on Software Engineering (CSE), which together with CAAP formed a joint event called TAPSOFT in odd-numbered years. Instances of TAPSOFT, all including CAAP plus at least one software engineering event, took place every two years from 1985 to 1997 inclusive. In the alternate years, CAAP took place separately from TAPSOFT.

Meanwhile, ESOP and CC were each taking place every two years from 1986. From 1988, CAAP was colocated with ESOP in even years. In 1994, CC became a "conference" rather than a "workshop" and CAAP, CC and ESOP were thereafter all colocated in even years.

TACAS, the youngest of the ETAPS conferences, was founded as an international workshop in 1995; in its first year, it was colocated with TAPSOFT. It took place each year, and became a "conference" when it formed part of ETAPS 1998. It is a telling indication of the importance of tools in the modern field of informatics that TACAS today is the largest of the ETAPS conferences.

The coming together of these five conferences was due to the vision of a small group of people who saw the potential of a combined event to be more than the sum of its parts. Under the leadership of Don Sannella, who became the first ETAPS steering committee chair, they included: Andre Arnold, Egidio Astesiano, Hartmut Ehrig, Peter Fritzson, Marie-Claude Gaudel, Tibor Gyimothy, Paul Klint, Kim Guldstrand Larsen, Peter Mosses, Alan Mycroft, Hanne Riis Nielson, Maurice Nivat, Fernando Orejas, Bernhard Steffen, Wolfgang Thomas and (alphabetically last but in fact one of the ringleaders) Reinhard Wilhelm.

ETAPS today is a loose confederation in which each event retains its own identity, with a separate programme committee and proceedings. Its format is open-ended, allowing it to grow and evolve as time goes by. Contributed talks and system demonstrations are in synchronized parallel sessions, with invited lectures in plenary sessions. Two of the invited lectures are reserved for "unifying" talks on topics of interest to the whole range of ETAPS attendees. The aim of cramming all this activity into a single one-week meeting is to create a strong magnet for academic and industrial researchers working on topics within its scope, giving them the opportunity to learn about research in related areas, and thereby to foster new and existing links between work in areas that were formerly addressed in separate meetings.

ETAPS 1998–2006

The first ETAPS took place in Lisbon in 1998. Subsequently it visited Amsterdam, Berlin, Genova, Grenoble, Warsaw, Barcelona, Edinburgh and Vienna before arriving in Braga this year. During that time it has become established as the major conference in its field, attracting participants and authors from all over the world. The number of submissions has more than doubled, and the numbers of satellite events and attendees have also increased dramatically.

ETAPS 2007

ETAPS 2007 comprises five conferences (CC, ESOP, FASE, FOSSACS, TACAS), 18 satellite workshops (ACCAT, AVIS, Bytecode, COCV, FESCA, FinCo, GT-VMT, HAV, HFL, LDTA, MBT, MOMPES, OpenCert, QAPL, SC, SLA++P, TERMGRAPH and WITS), three tutorials, and seven invited lectures (not including those that were specific to the satellite events). We received around 630 submissions to the five conferences this year, giving an overall acceptance rate of 25%. To accommodate the unprecedented quantity and quality of submissions, we have four-way parallelism between the main conferences on Wednesday for the first time. Congratulations to all the authors who made it to the final programme! I hope that most of the other authors still found a way of participating in this exciting event and I hope you will continue submitting.

ETAPS 2007 was organized by the Departamento de Informática of the Universidade do Minho, in cooperation with

– European Association for Theoretical Computer Science (EATCS)
– European Association for Programming Languages and Systems (EAPLS)
– European Association of Software Science and Technology (EASST)
– The Computer Science and Technology Center (CCTC, Universidade do Minho)
– Camara Municipal de Braga
– CeSIUM/GEMCC (Student Groups)

The organizing team comprised:

– João Saraiva (Chair)
– José Bacelar Almeida (Web site)
– José João Almeida (Publicity)
– Luís Soares Barbosa (Satellite Events, Finances)
– Victor Francisco Fonte (Web site)
– Pedro Henriques (Local Arrangements)
– José Nuno Oliveira (Industrial Liaison)
– Jorge Sousa Pinto (Publicity)
– António Nestor Ribeiro (Fundraising)
– Joost Visser (Satellite Events)

ETAPS 2007 received generous sponsorship from Fundação para a Ciência e a Tecnologia (FCT), Enabler (a Wipro Company), Cisco and TAP Air Portugal.

Overall planning for ETAPS conferences is the responsibility of its Steering Committee, whose current membership is:

Perdita Stevens (Edinburgh, Chair), Roberto Amadio (Paris), Luciano Baresi (Milan), Sophia Drossopoulou (London), Matt Dwyer (Nebraska), Hartmut Ehrig (Berlin), José Fiadeiro (Leicester), Chris Hankin (London), Laurie Hendren (McGill), Mike Hinchey (NASA Goddard), Michael Huth (London), Anna Ingólfsdóttir (Aalborg), Paola Inverardi (L'Aquila), Joost-Pieter Katoen (Aachen), Paul Klint (Amsterdam), Jens Knoop (Vienna), Shriram Krishnamurthi (Brown), Kim Larsen (Aalborg), Tiziana Margaria (Göttingen), Ugo Montanari (Pisa), Rocco de Nicola (Florence), Jakob Rehof (Dortmund), Don Sannella (Edinburgh), João Saraiva (Minho), Vladimiro Sassone (Southampton), Helmut Seidl (Munich), Daniel Varro (Budapest), Andreas Zeller (Saarbrücken).

I would like to express my sincere gratitude to all of these people and organizations, the programme committee chairs and PC members of the ETAPS conferences, the organizers of the satellite events, the speakers themselves, the many reviewers, and Springer for agreeing to publish the ETAPS proceedings. Finally, I would like to thank the organizing chair of ETAPS 2007, João Saraiva, for arranging for us to have ETAPS in the ancient city of Braga.

Edinburgh, January 2007

Perdita Stevens
ETAPS Steering Committee Chair

Preface

The present volume contains the proceedings of the international conference Foundations of Software Science and Computations Structures (FOSSACS) 2007, held in Braga, Portugal, March 26-28, 2007. FOSSACS is an event of the Joint European Conferences on Theory and Practice of Software (ETAPS). The previous nine FOSSACS conferences took place in Lisbon (1998), Amsterdam (1999), Berlin (2000), Genoa (2001), Grenoble (2002), Warsaw (2003), Barcelona (2004), Edinburgh (2005) and Vienna (2006).

FOSSACS presents original papers on foundational research with a clear significance to software science. The Program Committee invited papers on theories and methods to support analysis, synthesis, transformation and verification of programs and software systems. We identified the following topics, in particular: algebraic models, automata and language theory, behavioral equivalences, categorical models, computation processes over discrete and continuous data, infinite state systems computation structures, logics of programs, modal, spatial, and temporal logics, models of concurrent, reactive, distributed, and mobile systems, process algebras and calculi, semantics of programming languages, software specification and refinement, type systems and type theory, fundamentals of security, semi-structured data, program correctness and verification. We ultimately received 103 submissions.

This proceedings volume consists of the abstract of our invited talk together with 25 contributed papers. The contributed papers were selected for publication by the Program Committee during a two-week electronic discussion.

I sincerely thank all the authors of papers submitted to FOSSACS 2007. We were pleased by the number and quality of the submissions. Moreover, I would like to thank the members of the Program Committee for the excellent job they did during the selection process. Clearly, all this would not have been possible without the valuable and detailed reports provided by the sub-reviewers.

To administer submission and evaluation of papers, we relied on the Web-based tool OCS from Dortmund; thanks to Martin Karusseit for his patience and immediate help in cases of emergency. Finally, I would also like to thank the ETAPS 2007 Organizing Committee chaired by João Alexandre Saraiva and the ETAPS Steering Committee for their efficient coordination of all the activities leading up to FOSSACS 2007.

January 2007 Helmut Seidl

Organization

Program Committee

Martin Abadi (University of California at Santa Cruz and Microsoft Research)
Michael Benedikt (Bell Laboratories)
Ahmed Bouajjani (Université Paris 7)
Cristiano Calcagno (Imperial College London)
Didier Caucal (IRISA-CNRS, Rennes)
Flavio Corradini (University of Camerino)
Robert van Glabbeek (National ICT Australia, Sydney)
Andrew D. Gordon (Microsoft Research, Cambridge)
Hendrik Jan Hoogeboom (Leiden University)
Anna Ingolfsdottir (Aalborg University)
Florent Jacquemard (LSV, ENS de Cachan)
Werner Kuich (TU Wien)
Kamal Lodaya (Institute of Mathematical Sciences, Chennai)
Antoine Miné (ENS Rue d'Ulm, Paris)
Damian Niwinski (Warsaw University)
David A. Schmidt (University of Kansas)
Stefan Schwoon (Universität Stuttgart)
Helmut Seidl (TU München)
Scott A. Smolka (State University of New York at Stony Brook)
P.S. Thiagarajan (National University of Singapore)
Sophie Tison (Université des Sciences et Technologies de Lille)
Heiko Vogler (TU Dresden)
Christoph Weidenbach (Max-Planck-Institut für Informatik, Saarbrücken)

Reviewers

Luca Aceto
Thorsten Altenkirch
S. Arun-Kumar
Eugene Asarin
Franz Baader
Matthias Baaz
Eric Badouel
Jos Baeten
Christel Baier
Ezio Bartocci
Frédérique Bassino
Emmanuel Beffara
Jesper Bengtson
Martin Berger
Alexandru Berlea
Jean Berstel
Frédéric Besson
Bruno Blanchet
Frederic Blanqui
Mikolaj Bojanczyk
Viviana Bono
Michele Boreale
Richard Bornat
Laurent Braud

Peter Buchholz
Elie Bursztein
Nadia Busi
Diletta Romana Cacciagrano
Silvio Capobianco
Arnaud Carayol
Christian Choffrut
Mireille Clerbout
Veronique Cortier
Bruno Courcelle
Patrick Cousot
Adrian Curic
Deepak D'Souza
Alexandre David
Francesco De Angelis
Stéphanie Delaune
Stéphane Demri
Yuxin Deng
Raymond Devillers
Maria Rita Di Berardini
Dino Distefano
Javier Esparza
Kousha Etessami
Jérôme Feret
Emmanuel Filiot
Marcelo Fiore
Vojtech Forejt
Julien Forest
Joern Freiheit
Rudolf Freund
Murdoch Gabbay
Blaise Genest
Konstantinos Georgatos
Jeremy Gibbons
Jürgen Giesl
Hugo Gimbert
Jens C. Godskesen
Massimilian Goldwurm
Annegret Habel
Peter Habermehl
Sebastian Hack
Jo Hannay
Masahito Hasegawa
Ichiro Hasuo
Thomas Hildebrandt
Jane Hillston
Daniel Hirschkoff
Markus Holzer
Kohei Honda
Matthias Horbach
Hans Hüttel
Radu Iosif
Radha Jagadeesan
Alan Jeffrey
Yan Jurski
Kalpesh Kapoor
Stefan Kiefer
Bartek Klin
Eryk Kopczynski
Adam Koprowski
Orna Kupferman
Dietrich Kuske
Barbara König
Morten Khnrich
Slawek Lasota
Michel Latteux
Paul Levy
Francesco Logozzo
Gavin Lowe
Michael Luttenberger
Bas Luttik
Florent Madelaine
Andreas Maletti
Thierry Massart
Laurent Mauborgne
Annabelle McIver
Ingmar Meinecke
Emanuela Merelli
Antoine Meyer
Sayan Mitra
David Monniaux
Carroll Morgan
Christophe Morvan
Mohammad Reza Mousavi
Madhavan Mukund
Filip Murlak
Anca Muscholl
Anders Møller
Sebasitan Nanz
K. Narayan Kumar

Mark-Jan Nederhof
Uwe Nestmann
Frank Neven
Linh Anh Nguyen
Joachim Niehren
Michael Norrish
Dirk Nowotka
Javier Oliver
P. K. Pandya
Matthew Parkinson
Joachim Parrow
Dirk Pattison
Karl-Heinz Pennemann
Mati Pentus
Andrew Pitts
Damien Pous
K.V.S. Prasad
George Rahonis
R. Ramanujam
Jacob I. Rasmussen
Barbara Re
Steve Reeves
Oliviero Riganelli
Chloé Rispal
Andrey Rybalchenko
Christian Rétoré
Kai Salomaa
Davide Sangiorgi
Manfred Schmidt-Schauss
Olivier Serre
Anil Seth
Sunil Simon
Isabelle Simplot-Ryl
David Simplot-Ryl
Anu Singh
Pawel Sobocinski
Jiri Srba
Eugene Stark
Ian Stark
Alin Stefanescu
Frank Stephan
Colin Stirling
S. P. Suresh
Dejvuth Suwimonteerabuth
Jean-Marc Talbot
Luca Tesei
Alwen Tiu
Richard Trefler
Ralf Treinen
Christian Urban
Tarmo Uustalu
Frits Vaandrager
Kumar Neeraj Verma
Janis Voigtlaender
Igor Walukiewicz
Daria Walukiewicz-Chrzaszcz
Thomas Wilke
Hongwei Xi
Hongseok Yang
Shaofa Yang
Sergio Yovine
Hans Zantema
Yu Zhang
Valeria de Paiva
Michel de Rougemont
Erik de Vink
Roel de Vrijer
Maurice ter Beek

Table of Contents

Formal Foundations for Aspects

Radha Jagadeesan

DePaul University, USA
rjagadeesan@cs.depaul.edu

Abstract. Aspects have emerged as a powerful tool in the design and development of systems. Aspect-orientation ideas are paradigm independent and have been developed for object-oriented, imperative and functional languages.

This talk will discuss a suite of results that aim to level the foundational playing field between aspects and other programming paradigms. In this context, we will argue that aspects are no more intractable than stateful higher order programs.

The talk is based on joint work with Glenn Bruns, Alan Jeffrey, Corin Pitcher and James Riely.

Sampled Universality of Timed Automata

Parosh Aziz Abdulla, Pavel Krcal, and Wang Yi

Uppsala University, Sweden
{parosh,pavelk,yi}@it.uu.se

Abstract. Timed automata can be studied in not only a dense-time setting but also a discrete-time setting. The most common example of discrete-time semantics is the so called *sampled* semantics (i.e., discrete semantics with a fixed time granularity ε). In the real-time setting, the universality problem is known to be undecidable for timed automata. In this work, we study the universality question for the languages accepted by timed automata with sampled semantics. On the negative side, we show that deciding whether for all sampling periods ε a timed automaton accepts all timed words in ε-sampled semantics is as hard as in the real-time case, i.e., undecidable. On the positive side, we show that checking whether there is a sampling period such that a timed automaton accepts all untimed words in ε-sampled semantics is decidable. Our proof uses clock difference relations, developed to characterize the reachability relation for timed automata in connection with sampled semantics.

1 Introduction

Timed automata [3] are considered as one of the standard models for timed systems. The semantics of these models can be defined over various time domains. The most common one is the set of nonnegative real numbers, giving *dense* time semantics. The dense time semantics allows for the description of how a system behaves at every real-valued time point with arbitrarily fine precision, and thus one needs not consider time granularity in modeling and verification. To study systems which have a fixed granularity of time (e.g., clock cycles), discrete time semantics, and in particular, *sampled* semantics with fixed time step ε are often considered, e.g., [10, 5]. In such a case, the time domain is $\{k \cdot \varepsilon | k \in \mathbb{N}_0\}$, where $\varepsilon = 1/n$ for some $n \in \mathbb{N}$.

In this paper, we study the universality question for the languages accepted by timed automata in sampled semantics. Let A be a timed automaton and $L_\varepsilon(A)$ denote the sampled language accepted by A in the ε-sampled semantics, i.e., the set of timed traces where all events are associated with a timestamp which is $n * \varepsilon$ for some natural number n. More precisely we study the following problems:

1. *Existential timed universality* which is to check whether $L_\varepsilon(\mathcal{A})$ is universal for some sampling period ε.
2. *Universal timed universality* which is to check whether $L_\varepsilon(\mathcal{A})$ is universal for all sampling periods ε.

3. *Existential untimed universality* which is to check whether the untimed language of $L_\varepsilon(\mathcal{A})$ is universal for some sampling period ε.
4. *Universal untimed universality* which is to check whether the untimed language of $L_\varepsilon(\mathcal{A})$ is universal for all sampling periods ε.

We show that Problem 1 and 4 are easy to check. In fact, they are equivalent to checking whether $L_1(\mathcal{A})$ is timed (and untimed) universal (in the sampled semantics with sampling period 1). As our main results, we prove the (un)decidability of the remaining two questions for both finite and infinite words. On the negative side, we show that Problem 2 is as hard as the universality problem of timed automata in the real-time setting, that is undecidable (in fact, Π_1^1-hard). On the positive side, we show that Problem 3 is decidable.

The decidability proof extends the standard subset construction technique by a novel procedure for detection of loops which enforce nonimplementable behaviors, i.e., behaviors which cannot be realized by a system whenever we fix a sampling period. This procedure is based on a recently described technique – *clock difference relations* [12]. This technique has been developed in connection with sampled semantics to characterize the reachability relation for timed automata. It can be used to detect behaviors of timed automata in the dense time semantics which are not present in a sampled semantics for any sampling period.

Related Work. Universality of timed automata has been shown Π_1^1-hard in the seminal paper [3] for the real time semantics. Later papers study the universality (or the language inclusion) problem for subclasses of timed automata, e.g., closed/open timed automata [16], robust automata [11], or timed automata with one clock [17, 1]. There has been a considerable amount of work related to discretization issues and verifying dense time properties using discrete time methods, e.g., [10, 13, 15, 5]. The main difference compared to our work is that usually only a fixed sampling rate is considered. Practical aspects of verification with the use of sampled semantics are discussed in [7, 6, 4]. These works are concerned mainly with data structures for representing sets of discrete valuations (e.g., different types of decision diagrams). Implementability issues are discussed in connection with robust semantics of timed automata in [19, 18]. The reachability relations for timed automata were also characterized using the additive theory of real numbers in [8] and using $2n$-automata in [9].

2 Preliminaries

We consider the standard model of timed automata [3].

Definition 1. *A* timed automaton $\mathcal{A}$ *is a tuple* $\langle \mathcal{C}, \Sigma, N, l_0, E, F \rangle$ *where*

- $\mathcal{C}$ *is a set of real-valued clocks,*
- Σ *is a finite alphabet of events,*
- N *is a finite set of locations,*
- $l_0 \in N$ *is the initial location,*

- *$E \subseteq N \times \Phi(\mathcal{C}) \times \Sigma \times 2^{\mathcal{C}} \times N$ is the set of edges (describing possible transitions), and*
- *$F \subseteq N$ is the set of accepting locations.*

The set $\Phi(\mathcal{C})$ of clock constraints (guards) g is defined as a set of conjunctive formulas of atomic constraints in the form $x \sim m$ where $x \in \mathcal{C}$ is a clock, $\sim \in \{\leq, <, \geq, >\}$, and m is a natural number. A clock valuation $\nu \in [\mathcal{C} \to \mathbb{R}_{\geq 0}]$ is a function mapping clocks to non-negative real numbers. We use $\nu + t$ to denote the clock valuation which maps each clock x to the value $\nu(x) + t$, and $\nu[r \mapsto 0]$ for $r \subseteq \mathcal{C}$ to denote the clock valuation which maps each clock in r to 0 and agrees with ν for the other clocks (i.e., $\mathcal{C} \backslash r$). An edge (l_1, g, e, r, l_2) represents a transition from location $l_1 \in N$ to location $l_2 \in N$ accepting an input symbol (an *event*) $e \in \Sigma$, and resetting clocks in $r \subseteq \mathcal{C}$ to zero. The transition can be performed only if the current values of clocks satisfy g.

We present the definition of runs, accepting runs, and the language of timed automaton relative to a time domain on which the automaton operates. A *time domain* $\mathcal{T}$ is a subset of nonnegative real numbers satisfying the following properties. If $a, b \in \mathcal{T}$ then $a + b \in \mathcal{T}$ and $0 \in \mathcal{T}$. If we take $\mathcal{T} = \mathbb{R}_{\geq 0}$ then we get the standard semantics of timed automata. A *$\mathcal{T}$-timed event* is a pair (t, e), where $e \in \Sigma$ is an event and $t \in \mathcal{T}$ is called a *timestamp* of the event e. A *$\mathcal{T}$-timed trace* is a (possibly infinite) sequence of $\mathcal{T}$-timed events $\xi = (t_1, e_1)(t_2, e_2)...$, $e_i \in \Sigma$, $t_i \in \mathcal{T}$ and $t_i \leq t_{i+1}$ for all $i \geq 1$.

Definition 2. *A* run *of a timed automaton $\mathcal{A} = \langle \mathcal{C}, \Sigma, N, l_0, E, F \rangle$ over a $\mathcal{T}$-timed trace $\xi = (t_1, e_1)(t_2, e_2)(t_3, e_3) \ldots$, is a (possibly infinite) sequence of the form*

$$(l_0, \nu_0) \xrightarrow[t_1]{e_1} (l_1, \nu_1) \xrightarrow[t_2]{e_2} (l_2, \nu_2) \xrightarrow[t_3]{e_3} \ldots$$

where (l_i, ν_i) are states *of $\mathcal{A}$, $l_i \in N$, ν_i is a clock valuation, satisfying the following conditions:*

- *$\nu_0(x) = 0$ for all $x \in \mathcal{C}$.*
- *for all $i \geq 1$, there is an edge $(l_{i-1}, g_i, e_i, r_i, l_i)$ such that $(\nu_{i-1} + t_i - t_{i-1})$ satisfies g_i (we define $t_0 = 0$) and $\nu_i = (\nu_{i-1} + t_i - t_{i-1})[r_i \mapsto 0]$.*

A finite run of a timed automaton is accepting if $l_n \in F$ for its last state (l_n, ν_n). An infinite run of a timed automaton is accepting if an accepting location $l \in F$ occurs on it infinitely often (standard Büchi acceptance condition). The (finite word) timed language of a timed automaton $\mathcal{A}$ with respect to the time domain $\mathcal{T}$, denoted $L_{\mathcal{T}}(\mathcal{A})$, is the set of all finite $\mathcal{T}$-time traces for which there is an accepting run of $\mathcal{A}$. Analogously, the timed ω-language of a timed automaton $\mathcal{A}$ with respect to the time domain $\mathcal{T}$ denoted $L^{\omega}_{\mathcal{T}}(\mathcal{A})$ is the set of all infinite $\mathcal{T}$-time traces for which there is an accepting run of $\mathcal{A}$.

An untime function $\mathcal{U}$ maps a timed trace into a word over Σ by projecting out the timestamps, i.e., $\mathcal{U}((t_1, e_1)(t_2, e_2)(t_3, e_3) \ldots) = e_1 e_2 e_3 \ldots$. A natural extension of $\mathcal{U}$ to sets of timed traces maps timed languages into their untimed counterparts.

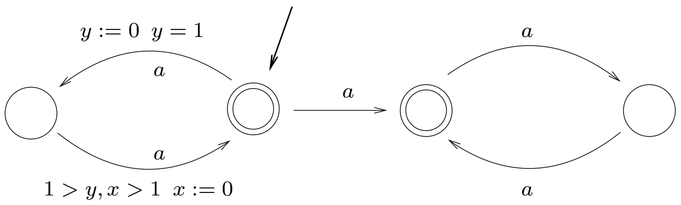


Fig. 1. An example of an automaton which is untimed universal in the real time semantics, but there is no ε such that it is untimed universal in the ε-sampled semantics

We say that languages $L_{\mathcal{T}}(\mathcal{A})$, $L^{\omega}_{\mathcal{T}}(\mathcal{A})$ are *timed universal* if they contain all finite, resp. infinite, $\mathcal{T}$-timed traces over Σ and $\mathcal{T}$. The languages $L_{\mathcal{T}}(\mathcal{A})$, $L^{\omega}_{\mathcal{T}}(\mathcal{A})$ are *untimed universal* if $\mathcal{U}(L_{\mathcal{T}}(\mathcal{A})) = \Sigma^*$, $\mathcal{U}(L^{\omega}_{\mathcal{T}}(\mathcal{A})) = \Sigma^{\omega}$, respectively.

The time domains of special interest in this paper are the sampled (digital) time domains $\mathcal{T}_\varepsilon = \{k \cdot \varepsilon | k \in \mathbb{N}_0\}$, where ε is a sampling period, $\varepsilon = 1/n$ for some $n \in \mathbb{N}$. To simplify the notation, we write $L_\varepsilon(\mathcal{A})$, $L^{\omega}_\varepsilon(\mathcal{A})$ instead of $L_{\mathcal{T}_\varepsilon}(\mathcal{A})$, $L^{\omega}_{\mathcal{T}_\varepsilon}(\mathcal{A})$, respectively. In the following, we always assume that $\varepsilon = 1/n$ for some $n \in \mathbb{N}$. We will also write $L(\mathcal{A})$, $L^{\omega}(\mathcal{A})$ when $\mathcal{T} = \mathbb{R}_{\geq 0}$ (standard real time semantics). Figure 1 shows an automaton $\mathcal{A}$ such that $L(\mathcal{A})$ is untimed universal, but there is no ε such that $L_\varepsilon(\mathcal{A})$ is also untimed universal. For any ε, there is $k \in \mathbb{N}$ such that the word a^{2k} does not belong to $\mathcal{U}(L_\varepsilon(\mathcal{A}))$.

For a timed automaton $\mathcal{A} = \langle \mathcal{C}, \Sigma, N, l_0, E, F \rangle$ we will use the standard notion of region automaton [3]. States of a region automaton consist of a location and a region. A region is a set of valuations (an equivalence class of so called region equivalence). These sets can be represented in a finite way, because the region equivalence has finite index for each timed automaton. Regions are denoted by $D, D', \ldots$. Transitions in a region automaton $(l, D) \overset{a}{\mapsto} (l', D')$ or $(l, D) \overset{t}{\mapsto} (l', D')$ are labeled by either an $a \in \Sigma$ or by a special symbol t denoting an immediate time successor (a time delay). In the following, we denote states of the region automaton by $R, R_1, R_2, \ldots$ and the initial state of the region automaton by R_0. By a path π in the region automaton starting from R and leading to R' labeled by w (denoted $\pi = R \xrightarrow{w} R'$) we mean a sequence of transitions starting from R and leading to R' labeled by w' such that $w = w' \upharpoonright t$ (w is equal to w' with all labels t projected out) and the last letter of w' is different from t (the last transition of the path is labeled by some $a \in \Sigma$). Note that a path does not have to be uniquely determined by R, R' and w. Paths are denoted by $\pi, \pi', \bar{\pi}, \ldots$

When we say that there is a run of a timed automaton $\mathcal{A}$ over a path $\pi = R \xrightarrow{w} R'$ in an ε-sampled semantics (real time semantics) then we mean that there is a run of $\mathcal{A}$ over ξ_w where ξ_w is a $\mathcal{T}_\varepsilon$-timed trace ($\mathbb{R}_{\geq 0}$-timed trace) such that $w = \mathcal{U}(\xi_w)$ going through π. If we need to specify the starting and the finishing state, then we write $(l, \nu) \longrightarrow^{\pi}_{\varepsilon} (l', \nu')$ or $(l, \nu) \longrightarrow^{\pi} (l', \nu')$.

3 Reachability Relations

As a technical tool in our proofs, we will use clock difference relations [12] to characterize reachability relations. The notion of the reachability relation describes exactly the valuations which can be reached in a timed automaton $\mathcal{A}$ from a given valuation while going through a given path in the region automaton for $\mathcal{A}$. Let $\pi = (l, D) \xrightarrow{w} (l', D')$ be a path in the region automaton for $\mathcal{A}$. We want to characterize which concrete states $(l', \nu'), \nu' \in D'$ can be reached from a concrete state $(l, \nu), \nu \in D$ by a run of $\mathcal{A}$ going through π in the real time semantics. We are not interested in the clock values when they grow over the greatest constant in $\mathcal{A}$, denoted by K. To abstract from such clocks, we define a relation $\sim_K$ on clock valuations as follows: $\nu \sim_K \nu'$ iff for all $x \in \mathcal{C} : \nu(x) = \nu'(x)$ or $\nu(x) > K \wedge \nu'(x) > K$. Formally, the *reachability relation* of a path $\pi = (l, D) \xrightarrow{w} (l', D')$ is a relation on valuations $C^\pi \subseteq D \times D'$ such that for each $\nu \in D, \nu' \in D'$:

$$(\nu, \nu') \in C^\pi \iff \exists \nu'' \sim_K \nu' : (l, \nu) \longrightarrow^\pi (l', \nu'').$$

This relation can be characterized by a structure over a set of clocks $\mathcal{C}$ called *clock difference relations*. This structure is a set of (in)equalities of the following form:

- $x' - y' \bowtie u - v$
- $x' - y' \bowtie 1 - (u - v)$

where $\bowtie \in \{<, >, =\}$, $x, y, u, v \in \mathcal{C}$. We use primed clock names on the left hand side of the (in)equalities to denote the fact that these clocks are interpreted in the target valuation, whereas the unprimed clocks are interpreted in the starting valuation. The semantics of a clock difference relation B is defined as follows. For any $\delta \in \mathbb{R}$, $\mathsf{fr}(\delta)$ denotes the fractional part of δ. We say that a pair of valuations (ν, ν') satisfies B if and only if:

- if $x' - y' \bowtie u - v \in B$ then $\mathsf{fr}(\nu'(x)) - \mathsf{fr}(\nu'(y)) \bowtie \mathsf{fr}(\nu(u)) - \mathsf{fr}(\nu(v))$,
- if $x' - y' \bowtie 1 - (u - v) \in B$ then $\mathsf{fr}(\nu'(x)) - \mathsf{fr}(\nu'(y)) \bowtie 1 - (\mathsf{fr}(\nu(u)) - \mathsf{fr}(\nu(v)))$.

The semantics of clock difference relations can be extended to sets of clock difference relations. A pair of valuations (ν, ν') satisfies a set of clock difference relations if it satisfies at least one of them. By $\sigma = R \xrightarrow{w} R'$ we denote a set of (not necessarily all) paths which start in R, lead to R', and are labeled by the same word w.

Lemma 1 ([12]). *For a given set of paths $\sigma = R \xrightarrow{w} R'$, the reachability relation $\bigcup_{\pi \in \sigma} C^\pi$ is effectively definable as a (finite) set of clock difference relations.*

It follows from this lemma that for any two given sets of paths $\sigma = R \xrightarrow{w} R', \sigma' = R \xrightarrow{w'} R'$ can one algorithmically check whether $\bigcup_{\pi \in \sigma} C^\pi = \bigcup_{\pi \in \sigma'} C^\pi$ and whether there is ν such that $(\nu, \nu) \in \bigcup_{\pi \in \sigma} C^\pi$.

Now we define two concepts characterizing properties of loops in timed automata and state that they are equivalent for the sampled semantics. Let us assume that for a given timed automaton $\mathcal{A}$, σ is a set of (not necessarily all) paths from R back to itself labeled by w in the region automaton. We call this set of paths a *loop* $\sigma = R \xrightarrow{w} R$. This loop symbolically represents (fragments of) runs of $\mathcal{A}$ which start in a state $(l, \nu) \in R$ and end in some state $(l, \nu') \in R$ over some $\pi \in \sigma$. A crucial fact for our decision procedure is whether we can start again from (l, ν') and run over some $\pi' \in \sigma$, ending in some $(l, \nu'') \in R$, and whether we can iterate the loop like this unboundedly many times.

Definition 3. *For a timed automaton $\mathcal{A}$ and a loop $\sigma = R \xrightarrow{w} R$, $R = (l, D)$, we say that σ* can be iterated *in the ε-sampled semantics if for any $k \in \mathbb{N}$ there is a concrete run $(l, \nu_0) \longrightarrow_\varepsilon^{\pi_1} (l, \nu_1) \longrightarrow_\varepsilon^{\pi_2} \ldots \longrightarrow_\varepsilon^{\pi_k} (l, \nu_k)$, where $\pi_i \in \sigma, \nu_i \in D$ for all $0 \le i \le k$.*

In the real time semantics, any loop $\sigma = R \xrightarrow{w} R$ can be iterated, because for any $(l, \nu) \in R$ and for any $\pi \in \sigma$ there is a state $(l, \nu') \in R$ such that there is a run of $\mathcal{A}$ over π starting from (l, ν) and ending in (l, ν'). Therefore, one can compose these runs into an arbitrarily long one. This is not true in the sampled time domains. There are timed automata such that some loops in their region automata can be for any ε iterated only finitely many times. Intuitively, to constitute a real loop it requires that the automaton can get back exactly to the same concrete valuation in which it started after several iterations of the loop. This can be characterized in the terms of the reachability relations.

Definition 4. *For a timed automaton $\mathcal{A}$ and a loop $\sigma = R \xrightarrow{w} R$, $R = (l, D)$, we say that σ is a* real loop *if and only if there is $k \in \mathbb{N}$ and a clock valuation $\nu \in D$ such that $(\nu, \nu) \in (\bigcup_{\pi \in \sigma} C^\pi)^k$.*

The following lemma characterizes precisely when loops in a region automaton can be iterated unboundedly many times also in the sampled time domains.

Lemma 2 (**[12]**)
For a timed automaton $\mathcal{A}$ and a loop $\sigma = R \xrightarrow{w} R$, where $R = (l, D)$, there is an ε such that σ can be iterated in ε-sampled semantics if and only if σ is a real loop.

The following observation can give some intuition for the fact that $\sigma = R \xrightarrow{w} R$ cannot be iterated when there is no k such that $(\nu, \nu) \in (\bigcup_{\pi \in \sigma} C^\pi)^k$. For any ε, there are only finitely many different valuations of the clocks in R. But if for all $k \in \mathbb{N}$, $(\nu, \nu) \notin (\bigcup_{\pi \in \sigma} C^\pi)^k$ then each iteration of $R \xrightarrow{w} R$ has to result in a new clock valuation. Thus, it is not possible to iterate the loop again after sufficiently many iterations. The clock difference relation for the left loop of the automaton in Figure 1 is $y' - x' > y - x$. After every iteration of the left loop, the difference between the fractional parts of the clocks x and y grows but it has to be smaller than 1 all the time. Therefore, the computation will be blocked after at most $1/\varepsilon$ iterations.

4 Existential Untimed Sampled Universality (Finite Words)

In this part we study the decidability of the question whether there exists an ε such that the untimed ε-sampled language is universal. We give a positive answer for finite word languages in this section and for infinite word languages in the next section.

Theorem 1. *For a given timed automaton $\mathcal{A}$, the question whether there exists an ε such that $L_\varepsilon(\mathcal{A})$ is untimed universal is decidable.*

Note that the untimed universality is decidable for dense time semantics (both $L(\mathcal{A})$ and $L^\omega(\mathcal{A})$, [3]). For some timed automata, sampled semantics cuts out some words from their untimed language, i.e., it can happen that for each ε there is some untimed word w such that $w \in \mathcal{U}(L(\mathcal{A}))$ and $w \notin \mathcal{U}(L_\varepsilon(\mathcal{A}))$.

To solve the problem, we present an algorithm such that for a given timed automaton $\mathcal{A}$ the algorithm answers 'YES' if there is an ε such that $L_\varepsilon(\mathcal{A})$ is untimed universal. Otherwise, it answers 'NO' and gives a procedure which for each ε returns a counterexample for the untimed universality of $L_\varepsilon(\mathcal{A})$.

Before presenting the algorithm, we define several auxiliary concepts which are needed for the description of the loops in the procedure resembling the subset construction for region automata. We assume a fixed timed automaton $\mathcal{A}$ and its region automaton. Let σ be a set of (not necessarily all) paths in the region automaton labeled by w. For all prefixes w' of w, $\mathcal{R}(\sigma, w')$ denotes the set of states of the region automaton reachable by the prefixes π' of paths $\pi \in \sigma$ such that π' is labeled by w'. We say that a set of paths σ over w is a *sequence* if for all paths π over w in the region automaton the following holds: if $\mathcal{R}(\sigma, w') = \mathcal{R}(\sigma \cup \{\pi\}, w')$ for all prefixes w' of w then $\pi \in \sigma$ (we say that a sequence is *state complete*). Note that for a given region automaton each sequence σ over w is fully determined by the sets $\mathcal{R}(\sigma, w')$ for all prefixes w' of w. Let us assume that σ is a sequence over w and u, v are words such that uv is a prefix of w. For two states of the region automaton $R \in \mathcal{R}(\sigma, u), R' \in \mathcal{R}(\sigma, uv)$ we define a sequence $\sigma' = R \xrightarrow{v} R'$ *relative to* σ, u, v, R, R' as a set of paths π from R to R' labeled by v such that there are paths π' and π'', π' is labeled by u, and $\pi'\pi\pi'' \in \sigma$. Finally, let $\mathsf{Pattern}(\sigma, u, v)$ denote a partial mapping which for a pair of states of the region automaton returns a reachability relation according to the following rule:

$$\mathsf{Pattern}(\sigma,u,v)(R,R') = \begin{cases} \bigcup_{\pi\in\sigma'} C^\pi & \text{if } R \in \mathcal{R}(\sigma,u) \text{ and } R' \in \mathcal{R}(\sigma,uv); \\ & \sigma' = R \xrightarrow{u} R' \text{ is a sequence} \\ & \text{relative to } \sigma,u,v,R,R' \\ \perp & \text{otherwise} \end{cases}$$

Intuitively, $\mathsf{Pattern}(\sigma, u, v)$ summarizes the reachability pattern relative to σ (all paths involved have to be the corresponding fragments of the paths in σ) between states reachable by u and states reachable by uv. A schema of such a mapping is depicted in Figure 2.

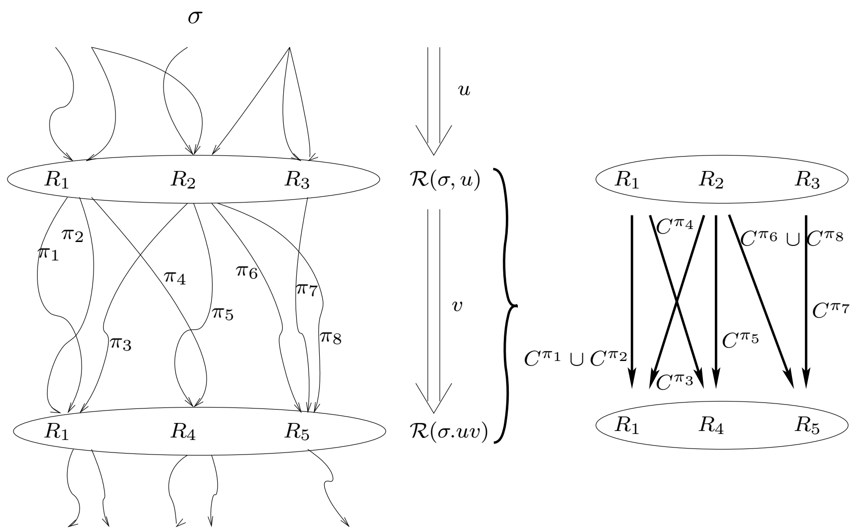


Fig. 2. A schema of $\mathsf{Pattern}(\sigma, u, v)$. For instance, $\mathsf{Pattern}(\sigma, u, v)(R_2, R_5) = C^{\pi_6} \cup C^{\pi_8}$. When there is no path in σ between two states of the region automaton labeled by v (e.g., between R_3 and R_4) then the corresponding reachability relation is empty (e.g., $\mathsf{Pattern}(\sigma, u, v)(R_3, R_4) = \emptyset$).

The algorithm works with the set of colored sequences, i.e., sequences where every path is either red or green. We define one concept and two simple operations which the algorithm uses. For a colored sequence σ over w:

– a division of w into $w = w_1v_1v_2v_3w_2$ is a *full loop* of σ if $|v_1|, |v_2|, |v_3| > 0$, $\mathcal{R}(\sigma, w_1) = \mathcal{R}(\sigma, w_1v_1) = \mathcal{R}(\sigma, w_1v_1v_2) = \mathcal{R}(\sigma, w_1v_1v_2v_3)$, $\mathsf{Pattern}(\sigma, w_1, v_1) = \mathsf{Pattern}(\sigma, w_1v_1, v_2) = \mathsf{Pattern}(\sigma, w_1v_1v_2, v_3) = \mathsf{Pattern}(\sigma, w_1, v_1v_2)$, and it is the first such division, i.e., there is no such division $w = \bar{w}_1\bar{v}_1\bar{v}_2\bar{v}_3\bar{w}_2$ where $\bar{w}_1\bar{v}_1\bar{v}_2\bar{v}_3$ is a strict prefix of $w_1v_1v_2v_3$.[1]
– $\mathsf{reduce}(\sigma,\ w_1v_1v_2v_3w_2)$, where $w_1v_1v_2v_3w_2$ is a full loop, is a colored sequence σ' built in two steps. First, for every path π in σ such that $\pi = R_0 \xrightarrow{w_1v_1} R \xrightarrow{v_2} R \xrightarrow{v_3w_2} R'$, put a path $\pi' = R_0 \xrightarrow{w_1v_1} R \xrightarrow{v_3w_2} R'$ into σ' (take only paths which loop over v_2 and cut this loop out, preserve the colors). Secondly, if a path $\pi \in \sigma'$, $\pi = R_0 \xrightarrow{w_1v_1} R \xrightarrow{v_3w_2} R'$, is green, $R = (l, D)$ and there is no $\nu \in D$ such that $(\nu, \nu) \in \mathsf{Pattern}(\sigma', w_1v_1, v_2)(R, R)$ then change the color of π to red.
– $\mathsf{extend}(\sigma,\ a)$ where $a \in \Sigma$ is the colored sequence σ' over wa such that if $\pi \in \sigma$ and π' is an extension of π by t^*a in the region automaton with the same color as π then $\pi' \in \sigma'$ (σ' is the greatest extension of σ over wa).

The full loop concept can be seen as a partial function which for a sequence σ returns a division such that a set of states repeats four times, creating three

[1] If the prefix is nonstrict then we fix any order on such divisions and pick the first one.

segments. These segments should have the same reachability pattern and also their composition should have this pattern. Note, that also $\mathsf{Pattern}(\sigma, w_1, v_1v_2v_3) = \mathsf{Pattern}(\sigma, w_1v_1, v_2v_3) = \mathsf{Pattern}(\sigma, w_1, v_1)$. If there are several divisions of the required properties, we choose the shortest such division, i.e., we want w_2 to be as big as possible. If there are several such divisions then we fix some rule to pick one. This decision does not play any role in the correctness of the algorithm, i.e., the algorithm works for any fixed rule.

The reduction step works with the full loop of a colored sequence and returns another colored sequence. It first chooses only paths which create a loop over the middle segment of the full loop (v_2). For each such path it checks whether the loop over v_2 can be iterated. If no then the color of the path is changed to red. Finally, the loops are cut out and the reduced paths are added into the new colored sequence. Clearly, this sequence is state complete.

The extension step just performs one more symbolic transition for each letter from the alphabet. For each path, it checks whether it can be extended by some time delay transitions and a discrete transition in the region automaton. If it is the case, then all such extensions (for any number of time delays and nondeterministic choices of the transition) are added into the new colored sequence (colors are preserved). Clearly, the new sequence is state complete. This step corresponds to the computation of the successor of a set of states in the standard subset construction.

The set of colored sequences together with the transitions given by the reduction and the extension function forms a transition system. Each colored sequence is a state in the transition system. If the colored sequence can be reduced, then the only transition outgoing from this sequence is the reduction transition. Otherwise, there is an extension transition outgoing from this sequence for each letter from the alphabet. The algorithm performs a standard reachability procedure in this transition system, looking for a colored sequence which does not contain a green path leading into an accepting state of the region automaton. It uses two sets of colored sequences, $\mathsf{Visited}$ and Passed, as data structures.

The Algorithm. Put the set containing the empty green path (over an empty word) starting in R_0 into $\mathsf{Visited}$.

1. Pick one colored sequence σ over w from $\mathsf{Visited}$.
2. **Counterexample?** Let $\rho \subseteq \sigma$ be the set of all green paths in σ. If $\mathcal{R}(\rho, w)$ does not contain any accepting state, stop and answer 'NO' (the sequence of steps leading to σ gives a counterexample).
3. **Extension.** If there is no full loop of σ then remove σ from $\mathsf{Visited}$, add σ to Passed, and for all $a \in \Sigma$, if $\mathsf{extend}(\sigma, a) \notin \mathsf{Passed}$ then add $\mathsf{extend}(\sigma, a)$ into $\mathsf{Visited}$.
4. **Reduction.** If $w = w_1v_1v_2v_3w_2$ is the full loop of σ then let $\sigma' = \mathsf{reduce}(\sigma, w_1v_1v_2v_3w_2)$. Remove σ from $\mathsf{Visited}$ and if $\sigma' \notin \mathsf{Passed}$ then add σ' into $\mathsf{Visited}$.
5. **Universality?** If $\mathsf{Visited}$ is empty then stop and answer 'YES'. Otherwise, go to Step 1.

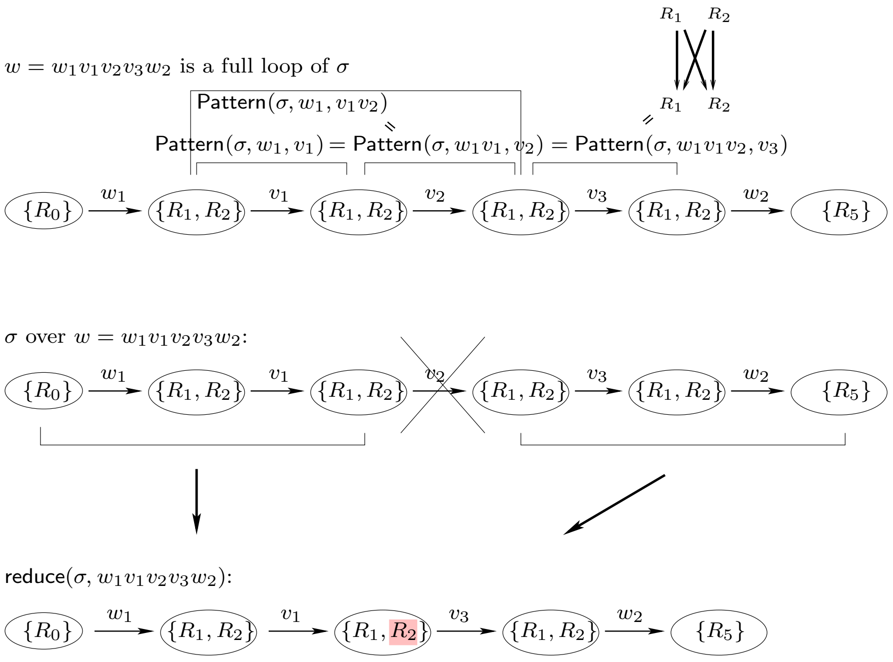

$w = w_1v_1v_2v_3w_2$ is a full loop of σ
$\mathsf{Pattern}(\sigma, w_1, v_1v_2)$
$\mathsf{Pattern}(\sigma, w_1, v_1) = \mathsf{Pattern}(\sigma, w_1v_1, v_2) = \mathsf{Pattern}(\sigma, w_1v_1v_2, v_3)$
R_1
R_2
$\{R_0\}$
$\{R_1, R_2\}$
$\{R_5\}$
w_1
v_1
v_2
v_3
w_2
σ over $w = w_1v_1v_2v_3w_2$:
$\mathsf{reduce}(\sigma, w_1v_1v_2v_3w_2)$:

Fig. 3. The reduction step of the algorithm. There is a full loop $w = w_1v_1v_2v_3w_2$ of σ. Let us assume that there is no ν such that $(\nu, \nu) \in \mathsf{Pattern}(\sigma, w_1, v_1)(R_2, R_2)$ (no loops over v_2 starting at R_2 can be iterated). Then $\mathsf{reduce}(\sigma, w_1v_1v_2v_3w_2)$ changes the color of all paths that go through R_2 after reading w_1v_1 to red.

A situation where a colored sequence has a full loop and it is reduced is depicted in Figure 3.

The correctness of the algorithm is based on the following two facts. At first, the set of the reachable colored sequences is finite, because for each timed automaton $\mathcal{A}$ there is a constant $\mathcal{H}_{\mathcal{A}}$ such that each sequence over w, where $|w| > \mathcal{H}_{\mathcal{A}}$, has a full loop. This means that it can be (and will be) shortened by the reduction step. The algorithm extends only irreducible colored sequences. A bound on the value of $\mathcal{H}_{\mathcal{A}}$ depending on the timed automaton parameters is given in [2].

Secondly, there is an ε such that the untimed ε-sampled language of the timed automaton is not untimed universal if and only if a colored sequence σ over w which does not contain any green path leading into an accepting state of the region automaton is reachable. The rest of this section is devoted to this argument.

We state that if $L_\varepsilon(\mathcal{A})$ is untimed nonuniversal for all ε, but $L(\mathcal{A})$ is untimed universal, then the language $\Sigma^* - \bigcap_\varepsilon \mathcal{U}(L_\varepsilon(\mathcal{A}))$ (set of words which do not belong to some $L_\varepsilon(\mathcal{A})$) has a special structure. There is an infinite subset of this language which can be obtained by pumping a word of a certain form. Also, if $L_\varepsilon(\mathcal{A})$ is untimed universal for all ε then for each word there is an accepting path of a special structure defined by the following inductive definition.

Definition 5. *Given a colored sequence σ over w, a path $\pi \in \sigma$ is* fully accepting *if it is accepting, green and either*

– *σ does not have a full loop, or*
– *$w = w_1v_1v_2v_3w_2$ is a full loop of σ , $\pi = R_0 \xrightarrow{w_1v_1} R \xrightarrow{v_2} R \xrightarrow{v_3w_2} R_f$ (denote $\bar{\pi} = R \xrightarrow{v_2} R$ the segment of π over v_2), there is a valuation ν such that $(\nu, \nu) \in C^{\bar{\pi}}$, and $\pi' = R_0 \xrightarrow{w_1v_1} R \xrightarrow{v_3w_2} R'$ is fully accepting for* $\mathsf{reduce}(\sigma, w_1v_1v_2v_3w_2)$ *over $w_1v_1v_3w_2$.*

Let us for a given timed automaton $\mathcal{A}$ and a word w denote by $\sigma^w_{\mathcal{A}}$ a set of all paths starting in R_0 and labeled by w' such that $w = w' \upharpoonright t$ in the region automaton of $\mathcal{A}$. Let also all paths in $\sigma^w_{\mathcal{A}}$ be green. This set is state complete, hence a sequence. The proofs of the following lemmata can be found in [2].

Lemma 3. *For a given timed automaton $\mathcal{A}$, if for all words w there is a fully accepting path for $\sigma^w_{\mathcal{A}}$ then there is an ε such that $L_\varepsilon(\mathcal{A})$ is untimed universal.*

To show the converse, namely that if there is a word w without a fully accepting path over $\sigma^w_{\mathcal{A}}$ then there is no ε such that $L_\varepsilon(\mathcal{A})$ is untimed universal, we define a recursive function pumped which for a given sequence, a word, and a number returns a word. If σ over w does not have a full loop then $\mathsf{pumped}(\sigma, w, n) = w$. If $w = w_1v_1v_2v_3w_2$ is a full loop then the function creates two auxiliary words $w' = w_1v_1 \star_{v_2} v_3w_2$ which contains a special mark $\star_{v_2}$ instead of the subword v_2 and $w'' = v_2^n$. In case that v_2 contained some marks, they occur in every copy of v_2. Then it assigns to $\bar{w}$ the result of $\mathsf{pumped}(\mathsf{reduce}(\sigma, w_1v_1v_2v_3w_2), w', n)$ and replaces every occurrence of $\star_{v_2}$ in $\bar{w}$ by w''.

Lemma 4. *For a given timed automaton $\mathcal{A}$, if there is a word w such that no path π is fully accepting for $\sigma^w_{\mathcal{A}}$ then for every ε there is an $n \in \mathbb{N}$ such that* $\mathsf{pumped}(\sigma^w_{\mathcal{A}}, w, n) \notin L_\varepsilon(\mathcal{A})$.

This lemma follows from Lemma 2, because there is at least one loop which cannot be iterated on each accepting path over w. Therefore, it suffices to choose n so that none of these loops can be iterated n times to guarantee that there is no concrete run going through any of the accepting paths for $\mathsf{pumped}(\sigma^w_{\mathcal{A}}, w, n)$. It remains to show that the counterexample produced by the algorithm corresponds to such a word with no fully accepting path and if there is such a word then the algorithm finds it.

Lemma 5 (Correctness). *For a given timed automaton $\mathcal{A}$, the algorithm always stops and it answers 'YES' if and only if there exists an ε such that $L_\varepsilon(\mathcal{A})$ is untimed universal.*

Proof (Sketch). If for a given timed automaton the algorithm stops and answers 'NO' then it reports a sequence of steps as a counterexample. Let us consider the word w obtained by concatenating the letters from the extend operations in this sequence of steps. Since the algorithm follows Definition 5 and reaches a colored sequence with green paths leading only into nonaccepting states, there is no fully accepting path in $\sigma_{\mathcal{A}}^{w}$. From Lemma 4, there is no ε such that $L_\varepsilon(\mathcal{A})$ is untimed universal.

If the timed automaton $\mathcal{A}$ is not untimed universal in the real time semantics then the algorithm stops and answers 'NO'. If $\mathcal{A}$ is untimed universal in the real time semantics but there is no ε such that $L_\varepsilon(\mathcal{A})$ is untimed universal then from Lemma 3 there is a word w such that no path in $\sigma_{\mathcal{A}}^{w}$ is fully accepting. Because the algorithm follows Definition 5, it will stop and answer 'NO'.

5 Existential Untimed Sampled Universality (Infinite Words)

In this section, we give the positive answer to the decidability of the problem whether for a given timed automaton $\mathcal{A}$ there is an ε such that the untimed ε-sampled ω-language of $\mathcal{A}$ is universal.

Theorem 2. *For a given timed automaton $\mathcal{A}$, the question whether there exists an ε such that $L_\varepsilon^\omega(\mathcal{A})$ is untimed universal is decidable.*

We show that the algorithm for finite words can be modified to work also for infinite words. For a given timed automaton $\mathcal{A}$, $L_\varepsilon^\omega(\mathcal{A})$ is clearly untimed nonuniversal if $L_\varepsilon(\mathcal{A}')$ is not untimed universal, where $\mathcal{A}'$ is obtained from $\mathcal{A}$ by changing all locations to accepting. Otherwise, we need to find a word $w_1 w_2$ such that for each ε there is an n such that there is no accepting run of $\mathcal{A}$ over $\mathsf{pumped}(\sigma_{\mathcal{A}}^{w_1 w_2}, w_1 w_2, n) \cdot (\mathsf{pumped}(\sigma^{w_2}, w_2, n))^\omega$ in ε-sampled semantics (by σ^{w_2} we denote the sequence obtained from $\sigma_{\mathcal{A}}^{w_1 w_2^2}$ by cutting off all prefixes labeled by $w_1 w_2$). But for this is it sufficient to find a colored sequence σ over some $w_1 w_2$ reachable by the algorithm such that $\mathcal{R}(\sigma, w_1) = \mathcal{R}(\sigma, w_1 w_2) = \mathcal{R}(\sigma, w_1 w_2^2)$, $\mathsf{Pattern}(\sigma, w_1, w_2)) = \mathsf{Pattern}(\sigma, w_1, w_2^2)$, and the following condition holds. Let $\rho \subseteq \sigma$ be the set of all green paths in σ. For all $R \in \mathcal{R}(\rho, w)$ there is no green path $\pi \in \sigma^{w_2} \cdot \sigma^{w_2} \cdot \sigma^{w_2}$ going through an accepting state such that $\pi = R \xrightarrow{w_2^3} R$ and there is a ν such that $(\nu, \nu) \in C^\pi$.

Therefore, it is enough to modify the condition for answering 'NO' and reporting a counterexample to follow the description above. The algorithm does not compute the colored sequence $\sigma^{w_2} \cdot \sigma^{w_2} \cdot \sigma^{w_2}$, but it can be obviously done during the check for a counterexample.

Lemma 6 (Correctness). *For a given timed automaton $\mathcal{A}$, the modified algorithm always stops and it answers 'YES' if and only if there exists an ε such that $L_\varepsilon^\omega(\mathcal{A})$ is untimed universal.*

6 Universal Timed Sampled Universality

A dual question to the one studied in the previous subsection is whether for all ε it holds that $L_\varepsilon(\mathcal{A})$ respectively $L_\varepsilon^\omega(\mathcal{A})$ is timed universal. We show that these questions are undecidable. The finite word case is equivalent to the dense time universality and a technique from [1] applies for the infinite word case.

Theorem 3. *For a given timed automaton $\mathcal{A}$, the question whether for all ε it holds that $L_\varepsilon(\mathcal{A})$ ($L^\omega_\varepsilon(\mathcal{A})$) is timed universal is undecidable.*

Proof. In the case of finite timed traces, we show that the timed universality of $L_\varepsilon(\mathcal{A})$ is equivalent to the timed universality of $L(\mathcal{A})$, which has been proved undecidable in [3]. Assume that the answer for the sampled universality problem for a given automaton $\mathcal{A}$ is 'NO'. Then $L(\mathcal{A})$ is not dense time universal, because for each timed trace $w \notin L_\varepsilon(\mathcal{A})$ it holds that $w \notin L(\mathcal{A})$. To show the other direction, assume that for a given timed automaton $\mathcal{A}$ the language $L(\mathcal{A})$ is not timed universal. Then there is a timed trace w such that $w \notin L(\mathcal{A})$ and all timestamps are rational. This means that all runs of the automaton $\mathcal{A}$ over this timed trace are nonaccepting. But then there is an ε for which this timed trace is also a $\mathcal{T}_\varepsilon$-timed trace (and also not accepted).

In the case of infinite timed traces, this argument does not work, because an infinite timed trace with rational timestamps violating universality in the dense time case does not have to be a $\mathcal{T}_\varepsilon$-timed trace for any ε.

Due to Mayr [14], the existence of a space-bounded recurrent-state computation with insertion errors for alternating channel machines is undecidable. We sketch an adaptation of the reduction of this problem to the universality checking for one clock Büchi timed automata from [1]. We need to build a timed automaton $\mathcal{A}$ for every alternating channel machine M such that M has a space-bounded recurrent-state computation with insertion errors if and only if for all ε, $L^\omega_\varepsilon(\mathcal{A})$ is timed universal.

The proof in [1] specifies five conditions in Definition 4 for an ω-language so that its words correspond exactly to space-bounded recurrent-state computations with insertion errors of M. We build $\mathcal{A}$ such that it accepts precisely those words that fail to satisfy any of the conditions 1–4 (there is such an automaton even with one clock). The last condition from this definition says that the maximal number of the events in any time unit is bounded. But if there is an ε such that $L^\omega_\varepsilon(\mathcal{A})$ is not timed universal then it means that the words accepted by $\mathcal{A}$ in ε-sampled semantics also satisfy this condition.

Conversely, if M has no space-bounded recurrent-state computation with insertion errors then each $\mathcal{T}_\varepsilon$-timed trace must violate one of the conditions 1–4, because it automatically satisfies the condition 5. Therefore, it is accepted by $\mathcal{A}$.

7 Remaining Variants

There are two other decision problems arising naturally in our scheme. Both of them are decidable, but as we show, the problems degenerate to checking (un)timed universality of $L_\varepsilon(\mathcal{A})$ or $L^\omega_\varepsilon(\mathcal{A})$ for one fixed ε, namely $\varepsilon = 1$.

The first problem is to decide whether there is an ε for a timed automaton $\mathcal{A}$ such that $L_\varepsilon(\mathcal{A})$ is timed universal. But if this is not true for $\varepsilon = 1$ then it is not true for any $\varepsilon < 1$. If there is a $\mathcal{T}_1$-timed trace for which there is no run of $\mathcal{A}$ then this trace is also a $\mathcal{T}_\varepsilon$-timed trace for every $\varepsilon < 1$ and there is no run of $\mathcal{A}$ over this trace. The same reasoning applies also for $L^\omega_\varepsilon(\mathcal{A})$.

The other problem is to decide whether for a timed automaton $\mathcal{A}$ it is true that for all ε, $L_\varepsilon(\mathcal{A})$ (or $L^\omega_\varepsilon(\mathcal{A})$) is untimed universal. But if this is true for $\varepsilon = 1$ then it is true for any $\varepsilon < 1$. If there is a word w for which there is an accepting run of $\mathcal{A}$ in 1-sampled semantics then this run also accepts w in ε-sampled semantics for every $\varepsilon < 1$.

Therefore, for both cases, it is enough to check whether $L_1(\mathcal{A})$ is (un)timed universal or whether $L^\omega_1(\mathcal{A})$ is (un)timed universal and this gives us an answer for all ε or constitutes a witness of existence of an ε with the checked property.

8 Conclusions and Future Work

In this paper, we have studied the universality problems of timed automata in sampled semantics. We have shown that the question whether for all sampling periods ε a timed automaton accepts all timed words in ε-sampled semantics is undecidable. As a main result, we have presented a novel proof for the decidability of checking whether there is a sampling period such that a timed automaton accepts all untimed words in ε-sampled semantics is decidable. We believe that the proof techniques may be used to study other properties of timed systems, in particular the implementability of timed automata. As future work, we plan to extend our results to language inclusion checking, i.e, the problem: given timed automata A and B, whether there exists a sampling period ε such that $L_\varepsilon(A) \subseteq L_\varepsilon(B)$ and the related question for the untimed case. We shall also study the corresponding questions within the context of timed and untimed bisimulation.

Acknowledgments. We thank Radek Pelánek for his comments on previous drafts of this paper.

References

[1] P. A. Abdulla, J. Deneux, J. Ouaknine, and J. Worrell. Decidability and complexity results for timed automata via channel machines. In *Proc. of ICALP'05*, volume 3580 of *LNCS*, pages 1089–1101. Springer, 2005.

[2] P. A. Abdulla, P. Krcal, and W. Yi. Sampled universality of timed automata. Technical Report 2007-001, IT Department, Uppsala University, Jan 2007.

[3] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[4] E. Asarin, M. Bozga, A. Kerbrat, O. Maler, A. Pnueli, and A. Rasse. Data-structures for the verification of timed automata. In *Proc. of HART'97*, volume 1201 of *LNCS*, pages 346–360. Springer, 1997.

[5] E. Asarin, O. Maler, and A. Pnueli. On discretization of delays in timed automata and digital circuits. In *Proc. of CONCUR'98*, volume 1466 of *LNCS*, pages 470–484. Springer, 1998.

[6] D. Beyer. Improvements in BDD-based reachability analysis of timed automata. In *Proc. of FME 2001*, volume 2021 of *LNCS*, pages 318–343. Springer, 2001.

[7] M. Bozga, O. Maler, and S. Tripakis. Efficient verification of timed automata using dense and discrete time semantics. In *Proc. of Charme'99*, volume 1703 of *LNCS*, pages 125–141. Springer, 1999.

[8] H. Comon and Y. Jurski. Timed automata and the theory of real numbers. In *Proc. of CONCUR'99*, volume 1664 of *LNCS*, pages 242–257. Springer, 1999.

[9] C. Dima. Computing reachability relations in timed automata. In *Proc. of LICS'02*. IEEE Computer Society Press, 2002.

[10] T. A. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks? In *Proc. of ICALP'92*, volume 623 of *LNCS*, pages 545–558. Springer, 1992.

[11] T. A. Henzinger and J.-F. Raskin. Robust undecidability of timed and hybrid systems. In *HSCC'00*, volume 1790 of *LNCS*, pages 145–159. Springer, 2000.

[12] P. Krčál and R. Pelánek. On sampled semantics of timed systems. In *Proc. of FSTTCS'05*, volume 3821 of *LNCS*, pages 310–321. Springer, 2005.

[13] K. G. Larsen and W. Yi. Time-abstracted bisimulation: Implicit specifications and decidability. *Information and Computation*, 134(2):75–101, 1997.

[14] R. Mayr. Undecidable problems in unreliable computations. *Theoretical Computer Science*, 297:347–354, 2003.

[15] J. Ouaknine and J. Worrell. Revisiting digitization, robustness, and decidability for time d automata. In *Proc. of LICS'03*, pages 198–207. IEEE Computer Society Press, 2003.

[16] J. Ouaknine and J. Worrell. Universality and language inclusion for open and closed timed automata. In *Proc. of HSCC'03*, volume 2623 of *LNCS*, pages 375–388. Springer, 2003.

[17] J. Ouaknine and J. Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *Proc. of LICS 2004*, pages 54–63. IEEE Computer Society, 2004.

[18] A. Puri. Dynamical properties of timed automata. *Discrete Event Dynamic Systems*, 10(1-2):87–113, 2000.

[19] M. De Wulf, L. Doyen, and J.-F. Raskin. Almost ASAP semantics: From timed models to timed implementations. In *Proc. of HSCC'04*, volume 2993 of *LNCS*, pages 296–310. Springer, 2004.

Iterator Types[⋆]

Sandra Alves[1,⋆⋆], Maribel Fernández[2], Mário Florido[1], and Ian Mackie[2,3,⋆⋆⋆]

[1] University of Porto, Department of Computer Science & LIACC, R. do Campo Alegre 823, 4150-180, Porto, Portugal
[2] King's College London, Department of Computer Science, Strand, London, WC2R 2LS, U.K
[3] LIX, École Polytechnique, 91128 Palaiseau Cedex, France

Abstract. System $\mathcal{L}$ is a linear λ-calculus with numbers and an iterator, which, although imposing linearity restrictions on terms, has all the computational power of Gödel's System $\mathcal{T}$. System $\mathcal{L}$ owes its power to two features: the use of a closed reduction strategy (which permits the construction of an iterator on an open function, but only iterates the function after it becomes closed), and the use of a liberal typing rule for iterators based on iterative types. In this paper, we study these new types, and show how they relate to intersection types. We also give a sound and complete type reconstruction algorithm for System $\mathcal{L}$.

1 Introduction

Recently new insights into linearity have lead to the development of rich computational models (see for instance [12,13,1,19,3]). To support them, new strategies of reduction and new notions of types and typing rules have been introduced.

System $\mathcal{L}$, as defined in [3], extends the linear λ-calculus with numbers, booleans, pairs, and an iterator. Unlike previous linear versions of System $\mathcal{T}$, System $\mathcal{L}$ permits to build an iterator term with an open function, but uses a reduction strategy that will block such subterms until the function becomes closed (thus preserving linearity). This reduction strategy, which we call *closed reduction*, has its roots in work by Girard on cut-elimination strategies [14], and was used to devise efficient evaluation strategies in the λ-calculus (see [11,13]).

Although linear systems are known to be computationally weak, System $\mathcal{L}$ has all the power of Gödel's System $\mathcal{T}$ (see [3] for details of the encoding of System $\mathcal{T}$ in System $\mathcal{L}$). The use of closed reduction (or more precisely, the fact that using closed reduction a linear system can deal with more general classes of terms) is one of the keys to the power of System $\mathcal{L}$: in [2] two linear versions of

⋆ Research partially supported by the British Council Treaty of Windsor Grant: "Linearity: Programming Languages and Implementations", and by funds granted to *LIACC* through the *Programa de Financiamento Plurianual, Fundação para a Ciência e Tecnologia* and *FEDER/POSI*.

⋆⋆ Programa Gulbenkian de Estímulo à Investigação.

⋆⋆⋆ Projet Logical, Pôle Commun de Recherche en Informatique du plateau de Saclay, CNRS, École Polytechnique, INRIA, Université Paris-Sud.

System $\mathcal{T}$, with and without closed-reduction, are analysed; the first is strictly more powerful, it can represent Ackermann's function whereas the latter cannot.

The other distinctive feature of System $\mathcal{L}$ is the use of a more liberal rule to type iterators, introducing *iterative types*. More precisely, in System $\mathcal{L}$ it is possible to construct iterators where in some cases the iterated function is used with different types each time (so we have a form of polymorphic iteration [18]).

In this paper we study these new types, give a Curry-style type system for System $\mathcal{L}$, and relate it to intersection type assignment systems. Intersection types were introduced by Coppo and Dezani in [7], and since then they have been used to characterise classes of terms with specific normalisation properties (see e.g. [23,5]), to define type systems with principal typings [17], to define models for the λ-calculus [6], etc. General intersection type assignment systems are undecidable, but several decidable restrictions have been defined (see for example [4,16,10]). Iterative types can be seen as a new decidable restriction of intersection types based on iteration. The type system of System $\mathcal{L}$ is decidable: one of the main contributions of this paper is a type reconstruction algorithm for System $\mathcal{L}$.

The rest of this paper is structured as follows. In the next section we recall System $\mathcal{L}$. Section 3 gives a type reconstruction algorithm, including the iterator types, with soundness and completeness proofs. Section 4 contains a detailed analysis of iterator types. Section 5 concludes the paper.

2 Linear λ-Calculus with Iterator: System $\mathcal{L}$

In this section we recall the syntax, reduction rules and typing rules of System $\mathcal{L}$ (for more details we refer the reader to [3]).

The set of linear λ-terms is built from: variables $x, y, \ldots$; linear abstraction $\lambda x.t$, where $x \in \mathsf{fv}(t)$; and application tu, where $\mathsf{fv}(t) \cap \mathsf{fv}(u) = \varnothing$. Here $\mathsf{fv}(t)$ denotes the set of free variables of t. Because x is used at least once in the body of the abstraction, and the condition on the application ensures that all variables are used at most once, these terms are syntactically linear (variables occur exactly once in each term).

Since we are in a linear calculus, we cannot have the usual notion of pairs and projections; instead, we have pairs and splitters:

$$\begin{array}{ll} \langle t, u\rangle & \text{if } \mathsf{fv}(t) \cap \mathsf{fv}(u) = \varnothing \\ \texttt{let } \langle x, y\rangle = t \texttt{ in } u & \text{if } x, y \in \mathsf{fv}(u) \text{ and } \mathsf{fv}(t) \cap \mathsf{fv}(u) = \varnothing \end{array}$$

Note that when projecting from a pair, we use both projections. A simple example is the swapping function: $\lambda x.\texttt{let } \langle y, z\rangle = x \texttt{ in } \langle z, y\rangle$.

Finally, we have booleans true and false, with a linear conditional: $\texttt{cond}\ t\ u\ v$ where $\mathsf{fv}(t) \cap \mathsf{fv}(u) = \varnothing$ and $\mathsf{fv}(u) = \mathsf{fv}(v)$; and numbers (built from 0 and S), with a linear iterator: $\mathsf{iter}\ t\ u\ v$ where $\mathsf{fv}(t) \cap \mathsf{fv}(u) = \mathsf{fv}(u) \cap \mathsf{fv}(v) = \mathsf{fv}(v) \cap \mathsf{fv}(t) = \varnothing$. $\mathsf{S}^n 0$ denotes n applications of S to 0. Table 1 summarises the syntax of System $\mathcal{L}$.

The dynamics of the system is given by a set of conditional reduction rules (which can be seen as a higher-order membership conditional rewrite system,

Table 1. Terms

Construction	**Variable Constraint**	**Free Variables** (fv)
0, true, false	–	$\varnothing$
S t	–	fv(t)
iter $t\ u\ v$	fv$(t) \cap$ fv$(u) =$ fv$(u) \cap$ fv$(v) = \varnothing$ fv$(t) \cap$ fv$(v) = \varnothing$	fv$(t) \cup$ fv$(u) \cup$ fv(v)
x	–	$\{x\}$
tu	fv$(t) \cap$ fv$(u) = \varnothing$	fv$(t) \cup$ fv(u)
$\lambda x.t$	$x \in$ fv(t)	fv$(t) \smallsetminus \{x\}$
$\langle t, u\rangle$	fv$(t) \cap$ fv$(u) = \varnothing$	fv$(t) \cup$ fv(u)
let $\langle x, y\rangle = t$ in u	fv$(t) \cap$ fv$(u) = \varnothing,\ x, y \in$ fv(u)	fv$(t) \cup ($fv$(u) \smallsetminus \{x, y\})$
cond $t\ u\ v$	fv$(u) =$ fv(v), fv$(t) \cap$ fv$(u) = \varnothing$	fv$(t) \cup$ fv(u)

see [25,26]). The conditions on the rewrite rules ensure that *Beta* only applies to redexes where the argument is a closed term (which implies that α-conversion is not needed to implement substitution), and only closed functions are iterated. Table 2 gives the reduction rules for System $\mathcal{L}$, substitution is a meta-operation defined as usual. Reductions can take place in any context.

Table 2. Closed reduction

Name	Reduction		Condition
Beta	$(\lambda x.t)v$	$\longrightarrow t[v/x]$	fv$(v) = \varnothing$
Let	let $\langle x, y\rangle = \langle t, u\rangle$ in v	$\longrightarrow (v[t/x])[u/y]$	fv$(t) =$ fv$(u) = \varnothing$
Cond	cond true $u\ v$	$\longrightarrow u$	
Cond	cond false $u\ v$	$\longrightarrow v$	
Iter	iter (S t) $u\ v$	$\longrightarrow v($iter $t\ u\ v)$	fv$(tv) = \varnothing$
Iter	iter $0\ u\ v$	$\longrightarrow u$	fv$(v) = \varnothing$

We give some examples to illustrate the system.

– Erasing numbers: although we are in a linear system, we can erase numbers by using them in iterators.

$$\begin{aligned}\mathsf{fst} &= \lambda x.\mathtt{let}\ \langle u, v\rangle = x\ \mathtt{in}\ \mathsf{iter}\ v\ u\ (\lambda z.z)\\ \mathsf{snd} &= \lambda x.\mathtt{let}\ \langle u, v\rangle = x\ \mathtt{in}\ \mathsf{iter}\ u\ v\ (\lambda z.z)\end{aligned}$$

– Copying numbers: $C = \lambda x.\mathsf{iter}\ x\ \langle 0, 0\rangle\ (\lambda x.\mathtt{let}\ \langle a, b\rangle = x\ \mathtt{in}\ \langle \mathsf{S}\ a, \mathsf{S}\ b\rangle)$ takes a number n and returns a pair $\langle n, n\rangle$.
– Addition: $\mathsf{add} = \lambda mn.\mathsf{iter}\ m\ n\ (\lambda x.\mathsf{S}\ x)$
– Multiplication: $\lambda mn.\mathsf{iter}\ m\ 0\ (\mathsf{add}\ n)$
– Predecessor: $\lambda n.\mathsf{fst}(\mathsf{iter}\ n\ \langle 0, 0\rangle\ (\lambda x.\mathtt{let}\ \langle t, u\rangle = C(\mathsf{snd}\ x)\ \mathtt{in}\ \langle t, \mathsf{S}\ u\rangle))$
– Ackermann: $ack(m, n) = (\mathsf{iter}\ m\ (\lambda x.\mathsf{S}\ x)\ (\lambda gu.\mathsf{iter}\ (\mathsf{S}\ u)\ (\mathsf{S}\ 0)\ g))\ n$

To type the terms in System $\mathcal{L}$ we use a set of *linear types*:

$$A, B ::= \mathsf{Nat} \mid \mathtt{Bool} \mid \alpha \mid A \multimap B \mid A \otimes B$$

where `Nat` and `Bool` are the types of numbers and booleans, and α is a type variable.

Let $A_0, \dots, A_n$ be a (non-empty) list of linear types. $It(A_0, \dots, A_n)$ denotes a non-empty set of *iterative types* defined by induction on n:

$$\begin{aligned}
&n = 0 : It(A_0) = \{A_0 \multimap A_0\}\\
&n = 1 : It(A_0, A_1) = \{A_0 \multimap A_1\}\\
&n \geq 2 : It(A_0, \dots, A_n) = It(A_0, \dots, A_{n-1}) \cup \{A_{n-1} \multimap A_n\}
\end{aligned}$$

Iterative types will serve to type the functions used in iterators. Note that $It(A_0) = It(A_0, A_0) = It(A_0, \dots, A_0)$.

The typing rules specifying how to assign types to untyped terms are given in Figure [1], where we use the following abbreviations: $\Gamma \vdash_{\mathcal{L}} t : It(A_0, \dots, A_n)$ iff $\Gamma \vdash_{\mathcal{L}} t : B$ for each $B \in It(A_0, \dots, A_n)$. It is a Curry-style type system (there are no type decorations in terms). We do not have Weakening and Contraction rules: we are in a linear system; the logical rules split the context between the premises. For terms of the form $\mathsf{iter}\ t\ u\ v$, we check that t is a term of type Nat and that v and u are compatible. There are two cases: if t is $\mathsf{S}^n 0$ then we require v to be a function that can be iterated n times on u. Otherwise, if t is not (yet) a number, we require v to have a type that allows it to be iterated any number of times (i.e. u has type A and $v : A \multimap A$, for some type A).

All the examples above can be typed in a straightforward way. More interestingly, the term $D = \lambda z.\mathsf{iter}\ (\mathsf{S}^2 0)\ (\lambda xy.\langle x, y\rangle)\ (\lambda x.xz)$ which allows us to copy arbitrary closed terms in System $\mathcal{L}$ (for any closed term t, $D\ t \longrightarrow^* \langle t, t\rangle$, see [3] for more details), is typable. We show a type derivation for D, which illustrates the use of iterative types. In the following type derivation N denotes Nat and B denotes $A \otimes A$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash_{\mathcal{L}} 0 : \mathsf{N}}}{\vdash_{\mathcal{L}} \mathsf{S}\,0 : \mathsf{N}}}{\vdash_{\mathcal{L}} \mathsf{S}^2 0 : \mathsf{N}} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{x : A \vdash_{\mathcal{L}} x : A} \quad \dfrac{}{y : A \vdash_{\mathcal{L}} y : A}}{x : A, y : A \vdash_{\mathcal{L}} \langle x, y\rangle : B}}{x : A \vdash_{\mathcal{L}} \lambda y.\langle x, y\rangle : A \multimap B}}{\vdash_{\mathcal{L}} \lambda xy.\langle x, y\rangle : A \multimap A \multimap B} \quad z : A \vdash_{\mathcal{L}} (\lambda x.xz) : It(A \multimap A \multimap B, A \multimap B, B)}{z : A \vdash_{\mathcal{L}} \mathsf{iter}\ (\mathsf{S}^2 0)\ (\lambda xy.\langle x, y\rangle)\ (\lambda x.xz) : B}}{\vdash_{\mathcal{L}} \lambda z.\mathsf{iter}\ (\mathsf{S}^2 0)\ (\lambda xy.\langle x, y\rangle)\ (\lambda x.xz) : A \multimap B}}$$

Note that

$$\dfrac{\dfrac{\dfrac{}{x : A \multimap A \multimap B \vdash_{\mathcal{L}} x : A \multimap A \multimap B} \qquad \dfrac{}{z : A \vdash_{\mathcal{L}} z : A}}{x : A \multimap A \multimap B, z : A \vdash_{\mathcal{L}} xz : A \multimap B}}{z : A \vdash_{\mathcal{L}} (\lambda x.xz) : (A \multimap A \multimap B) \multimap (A \multimap B)}$$

and

$$\dfrac{\dfrac{\dfrac{}{x : A \multimap B \vdash_{\mathcal{L}} x : A \multimap B} \qquad \dfrac{}{z : A \vdash_{\mathcal{L}} z : A}}{x : A \multimap B, z : A \vdash_{\mathcal{L}} xz : B}}{z : A \vdash_{\mathcal{L}} (\lambda x.xz) : (A \multimap B) \multimap B}$$

Axiom and **Structural Rule:**

$$\frac{}{x : A \vdash_{\mathcal{L}} x : A}\ (\mathsf{Axiom}) \qquad \frac{\Gamma, x : A, y : B, \Delta \vdash_{\mathcal{L}} t : C}{\Gamma, y : B, x : A, \Delta \vdash_{\mathcal{L}} t : C}\ (\mathsf{Exchange})$$

Logical Rules:

$$\frac{\Gamma, x : A \vdash_{\mathcal{L}} t : B}{\Gamma \vdash_{\mathcal{L}} \lambda x.t : A \multimap B}\ (\multimap\mathsf{Intro}) \qquad \frac{\Gamma \vdash_{\mathcal{L}} t : A \multimap B \qquad \Delta \vdash_{\mathcal{L}} u : A}{\Gamma, \Delta \vdash_{\mathcal{L}} tu : B}\ (\multimap\mathsf{Elim})$$

$$\frac{\Gamma \vdash_{\mathcal{L}} t : A \quad \Delta \vdash_{\mathcal{L}} u : B}{\Gamma, \Delta \vdash_{\mathcal{L}} \langle t, u\rangle : A \otimes B}\ (\otimes\mathsf{Intro}) \qquad \frac{\Gamma \vdash_{\mathcal{L}} t : A \otimes B \quad \Delta, x : A, y : B \vdash_{\mathcal{L}} u : C}{\Gamma, \Delta \vdash_{\mathcal{L}} \mathtt{let}\ \langle x, y\rangle = t\ \mathtt{in}\ u : C}\ (\otimes\mathsf{Elim})$$

Numbers

$$\frac{}{\vdash_{\mathcal{L}} 0 : \mathsf{Nat}}\ (\mathsf{Zero}) \qquad \frac{\Gamma \vdash_{\mathcal{L}} n : \mathsf{Nat}}{\Gamma \vdash_{\mathcal{L}} \mathsf{S}\ n : \mathsf{Nat}}\ (\mathsf{Succ})$$

$$\frac{\Gamma \vdash_{\mathcal{L}} t : \mathsf{Nat} \quad \Theta \vdash_{\mathcal{L}} u : A_0 \quad \Delta \vdash_{\mathcal{L}} v : It(A_0, \ldots, A_n) \quad (\star)}{\Gamma, \Theta, \Delta \vdash_{\mathcal{L}} \mathsf{iter}\ t\ u\ v : A_n}\ (\mathsf{Iter})$$

($\star$) where if $t \equiv \mathsf{S}^m 0$ then $n = m$ otherwise $n = 0$

Booleans

$$\frac{}{\vdash_{\mathcal{L}} \mathsf{true} : \mathtt{Bool}}\ (\mathsf{True}) \qquad \frac{}{\vdash_{\mathcal{L}} \mathsf{false} : \mathtt{Bool}}\ (\mathsf{False})$$

$$\frac{\Delta \vdash_{\mathcal{L}} t : \mathtt{Bool} \quad \Gamma \vdash_{\mathcal{L}} u : A \quad \Gamma \vdash_{\mathcal{L}} v : A}{\Gamma, \Delta \vdash_{\mathcal{L}} \mathtt{cond}\ t\ u\ v : A}\ (\mathsf{Cond})$$

Fig. 1. Type System for System $\mathcal{L}$

Therefore $z : A \vdash_{\mathcal{L}} (\lambda x.xz) : It(A \multimap A \multimap B, A \multimap B, B)$

We recall from [3] that System $\mathcal{L}$ is confluent, reductions preserve types, and typable terms are strongly normalisable[1].

3 Linear Type Reconstruction

This section develops a type reconstruction algorithm for System $\mathcal{L}$. Our algorithm is in a similar style to that of Damas-Milner [9]. We begin by giving a presentation of the type assignment rules which will suggest a type reconstruction algorithm. We will prove it to be both sound and complete with respect to these rules. We refer the reader to [22,9,8] for background to this work.

System $\mathcal{L}$ is a resource sensitive calculus, and we place a restriction on the use of assumptions in a derivation: namely use them all exactly once. Its type system is given in a multiplicative style, where each term is provided with the exact number of type assumptions for its free variables. Following [20], we will simulate a multiplicative system using a hybrid (in between multiplicative and additive) presentation of the rules. We will write typing judgements in the following way:

[1] In [3] there are no type variables, but the same results hold here since we don't have instantiation rules.

$$\Gamma \mid \Theta \vdash_{\mathcal{L}} t : A$$

where Γ and Θ are lists such that the elements in Θ are also in Γ, and $\Gamma \setminus \Theta$ contains precisely the assumptions necessary to type t; we call Γ the *before-set* and Θ the *after-set*, indicating that the derivation uses the assumptions only in $\Gamma \setminus \Theta$. The idea is best explained by an example. Consider the rule:

$$\frac{\Gamma \mid \Delta \vdash_{\mathcal{L}} t : A \multimap B \quad \Delta \mid \Theta \vdash_{\mathcal{L}} u : A}{\Gamma \mid \Theta \vdash_{\mathcal{L}} tu : B}\ (\multimap\mathsf{Elim})$$

This rule states that if we type tu using Γ then Θ will be left over. We give t all of the assumptions, and the remaining Δ are given to u. The ones that are not consumed here are exactly those which are left over in typing tu. The rationale for choosing this notation will become more apparent when we present the type reconstruction algorithm.

The full type assignment for System $\mathcal{L}$ using the "before-and-after" presentation is given in Figure 2, where we write $\Gamma, x : A$ to denote the list obtained by adding to Γ the element $x : A$ at the end (and in general we write Γ, Δ for list concatenation), and $x : A \in \Gamma$ holds if $x : A$ is the last assumption for x in the list Γ. The notation $\Gamma \setminus \{x : A\}$ represents the list Γ where we have deleted the last assumption for x (and in general, $\Gamma \setminus \Delta$ denotes the list Γ without the elements in Δ).

To relate the two versions of the type system (see Figures 1 and 2) we need some lemmas, where we use the following notation: if $\Gamma \mid \Delta$ is a type environment in the hybrid system, then we write $\overline{\Gamma} \mid \overline{\Delta}$ to denote any permutation of Γ that preserves the relative order of assumptions for the same variable (that is, all the assumptions for x occur in the same order in Γ and $\overline{\Gamma}$) and the corresponding sub-list $\overline{\Delta}$.

Lemma 1 (Permutations). *If $\Gamma \mid \Delta \vdash_{\mathcal{L}} t\colon A$, then $\overline{\Gamma} \mid \overline{\Delta} \vdash_{\mathcal{L}} t\colon A$.*

Proof. By induction on the derivation. In the permutation, only the relative order of the assumptions for x is relevant in the Axiom.

Lemma 2 (Monotonicity). *$\Gamma \mid \Gamma' \vdash_{\mathcal{L}} t\colon A$ if and only if $\Delta, \Gamma \mid \Delta, \Gamma' \vdash_{\mathcal{L}} t\colon A$.*

Proof. By induction on the type derivation.

As a consequence of these lemmas, $\Gamma \mid \Gamma' \vdash_{\mathcal{L}} t : A$ implies $\Gamma \setminus \Gamma' \mid \varnothing \vdash_{\mathcal{L}} t : A$ (since the elements in Γ' are also in Γ).

The relationship between the multiplicative and the hybrid versions of System $\mathcal{L}$ is as follows:

Theorem 1.
- *If $\Gamma \vdash_{\mathcal{L}} t : A$ then $\overline{\Gamma} \mid \varnothing \vdash_{\mathcal{L}} t : A$ for any permutation $\overline{\Gamma}$.*
- *If $\overline{\Gamma} \mid \varnothing \vdash_{\mathcal{L}} t : A$ for some permutation $\overline{\Gamma}$ then $\Gamma \vdash_{\mathcal{L}} t : A$.*

Proof. $\Rightarrow$) By induction on the type derivation, using the previous lemmas. We distinguish cases according to the last rule applied; some interesting cases are:

Axiom:

$$\frac{x : A \in \Gamma}{\Gamma \mid \Gamma \setminus \{x : A\} \vdash_{\mathcal{L}} x : A}\ (\mathsf{Axiom})$$

Logical Rules:

$$\frac{\Gamma, x : A \mid \Delta \vdash_{\mathcal{L}} t : B}{\Gamma \mid \Delta \vdash_{\mathcal{L}} \lambda x.t : A \multimap B}\ (\multimap\mathsf{Intro}) \qquad \frac{\Gamma \mid \Gamma' \vdash_{\mathcal{L}} t : A \multimap B \qquad \Gamma' \mid \Delta \vdash_{\mathcal{L}} u : A}{\Gamma \mid \Delta \vdash_{\mathcal{L}} tu : B}\ (\multimap\mathsf{Elim})$$

$$\frac{\Gamma \mid \Gamma' \vdash_{\mathcal{L}} t : A \qquad \Gamma' \mid \Delta \vdash_{\mathcal{L}} u : B}{\Gamma \mid \Delta \vdash_{\mathcal{L}} \langle t, u \rangle : A \otimes B}\ (\otimes\mathsf{Intro})$$

$$\frac{\Gamma \mid \Gamma' \vdash_{\mathcal{L}} t : A \otimes B \qquad \Gamma', x : A, y : B \mid \Delta \vdash_{\mathcal{L}} u : C}{\Gamma \mid \Delta \vdash_{\mathcal{L}} \mathtt{let}\ \langle x, y \rangle = t\ \mathtt{in}\ u : C}\ (\otimes\mathsf{Elim})$$

Numbers

$$\frac{}{\Gamma \mid \Gamma \vdash_{\mathcal{L}} 0 : \mathsf{Nat}}\ (\mathsf{Zero}) \qquad \frac{\Gamma \mid \Gamma' \vdash_{\mathcal{L}} n : \mathsf{Nat}}{\Gamma \mid \Gamma' \vdash_{\mathcal{L}} \mathsf{S}\ n : \mathsf{Nat}}\ (\mathsf{Succ})$$

$$\frac{\Gamma \mid \Gamma' \vdash_{\mathcal{L}} t : \mathsf{Nat} \quad \Gamma' \mid \Theta \vdash_{\mathcal{L}} u : A_0 \quad \Theta \mid \Delta \vdash_{\mathcal{L}} v : It(A_0, \ldots, A_n) \quad (\star)}{\Gamma \mid \Delta \vdash_{\mathcal{L}} \mathsf{iter}\ t\ u\ v : A_n}\ (\mathsf{Iter})$$

($\star$) where if $t \equiv \mathsf{S}^m\ 0$ then $n = m$ otherwise $n = 0$

Booleans

$$\frac{}{\Gamma \mid \Gamma \vdash_{\mathcal{L}} \mathsf{true} : \mathtt{Bool}}\ (\mathsf{True}) \qquad \frac{}{\Gamma \mid \Gamma \vdash_{\mathcal{L}} \mathsf{false} : \mathtt{Bool}}\ (\mathsf{False})$$

$$\frac{\Gamma \mid \Delta \vdash_{\mathcal{L}} t : \mathtt{Bool} \quad \Delta \mid \Theta \vdash_{\mathcal{L}} u : A \quad \Delta \mid \Theta \vdash_{\mathcal{L}} v : A}{\Gamma \mid \Theta \vdash_{\mathcal{L}} \mathtt{cond}\ t\ u\ v : A}\ (\mathsf{Cond})$$

Fig. 2. Hybrid Type System for System $\mathcal{L}$

– Exchange: Since the type environment contains the same elements in the premise and conclusion, the result follows directly by induction.
– $\multimap$Intro: By induction, $\overline{\Gamma, x : A} \mid \varnothing \vdash_{\mathcal{L}} t : B$, for any permutation of $\Gamma, x : A$. In particular, $\overline{\Gamma}, x : A \mid \varnothing \vdash_{\mathcal{L}} t : B$, and the result follows using $\multimap$Intro in the hybrid system.
– Iter: By induction, $\overline{\Gamma} \mid \varnothing \vdash_{\mathcal{L}} t : \mathsf{Nat}$, $\overline{\Theta} \mid \varnothing \vdash_{\mathcal{L}} u : A_0$, and $\overline{\Delta} \mid \varnothing \vdash_{\mathcal{L}} v : It(A_0, \ldots, A_n)$. By Monotonicity, $\overline{\Delta}, \overline{\Theta}, \overline{\Gamma} \mid \overline{\Delta}, \overline{\Theta} \vdash_{\mathcal{L}} t : \mathsf{Nat}$ and $\overline{\Delta}, \overline{\Theta} \mid \overline{\Theta} \vdash_{\mathcal{L}} u : A_0$. Then, using rule Iter in the hybrid system we obtain: $\overline{\Delta}, \overline{\Theta}, \overline{\Gamma} \mid \varnothing \vdash_{\mathcal{L}}$ $\mathsf{iter}\ t\ u\ v : It(A_0, \ldots, A_n)$. The result follows using the Permutation lemma.

$\Leftarrow$) We assume $\overline{\Gamma} \mid \varnothing \vdash_{\mathcal{L}} t : A$ for some permutation, and proceed by induction on t, using the previous lemmas. Again, we distinguish cases according to the last rule applied, and show only some interesting cases.

– $\multimap$Elim: The premises are: $\overline{\Gamma} \mid \Gamma' \vdash_{\mathcal{L}} t : A \multimap B$, and $\Gamma' \mid \varnothing \vdash_{\mathcal{L}} u : A$, then by Monotonicity we also have $\overline{\Gamma} \setminus \Gamma' \mid \varnothing \vdash_{\mathcal{L}} t : A \multimap B$. By induction: $\Gamma \setminus \Gamma' \vdash_{\mathcal{L}} t : A$ and $\Gamma' \vdash_{\mathcal{L}} u : A$, then $\Gamma, \Gamma' \vdash_{\mathcal{L}} tu : B$, using $\multimap$Elim in the multiplicative version.
– Iter: The premises are: $\overline{\Gamma} \mid \Gamma' \vdash_{\mathcal{L}} t : \mathsf{Nat}$, and $\Gamma' \mid \Theta \vdash_{\mathcal{L}} u : A_0$, and $\Theta \mid \varnothing \vdash_{\mathcal{L}} v : It(A_0, \ldots, A_n)$. Then by Monotonicity we also have $\overline{\Gamma} \setminus \Gamma' \mid \varnothing \vdash_{\mathcal{L}} t : \mathsf{Nat}$, and $\Gamma' \setminus \Theta \mid \varnothing \vdash_{\mathcal{L}} u : A_0$. By induction: $\Gamma \setminus \Gamma' \vdash_{\mathcal{L}} t : \mathsf{Nat}$, $\Gamma' \setminus \Theta \vdash_{\mathcal{L}} u : A_0$, and $\Theta \vdash_{\mathcal{L}} v : It(A_0, \ldots, A_n)$. Since $\Gamma = \Gamma \setminus \Gamma' \cup \Gamma' \setminus \Theta \cup \Theta$, the result follows using Iter in the multiplicative version, and the Permutation lemma.

3.1 The Type Reconstruction Algorithm $\mathcal{L}$

Our presentation of the algorithm $\mathcal{L}$ will assume that the terms are syntactically linear. It is a trivial extension to the algorithm to perform this kind of checking—we just need extra conditions to be satisfied.

We will need unification of types in this section, a simple extension to the unification algorithm used in Damas-Milner's system, based on a variant of Robinson's theorem [24]. The definition is standard (see for instance [21]). Substitutions are mappings from type variables to types. They are associative and idempotent; composition is denoted by juxtaposition. We assume that $\mathsf{mgu}AB$ gives the *most general unifier* of A and B, that is, a substitution U such that: $UA = UB$; if V also unifies A and B then V is a substitution instance of U, *i.e.* $V = SU$ for some substitution S; and the final requirement is that U only involves variables in A and B—no new variables are introduced during unification. If A, B are not unifiable then $\mathsf{mgu}AB$ fails. Our type reconstruction algorithm will take as input a term and a list of type assumptions for variables. To reflect the linearity constraint that all assumptions must be used exactly once, we treat type assumptions as resources—once an assumption is used, we remove it. To this end our type reconstruction algorithm will return a triple (rather than a pair as in the case of $\mathcal{W}$), which consists of a substitution, a type, and the assumptions not yet used.

We write R, S to range over substitutions, α, β to range over type variables, Γ, Γ' to range over lists of assumptions. We write id for the identity substitution, and substitution over lists is defined element-wise. For a substitution R, we write $R(\Gamma \mid \Gamma')$ for $R\Gamma \mid R\Gamma'$, and define substitution on judgements by : $R(\Gamma \mid \Gamma' \vdash_{\mathcal{L}} t : A) = R\Gamma \mid R\Gamma' \vdash_{\mathcal{L}} t : RA$. We assume that the function *new* returns a fresh type variable each time it is called.

Definition 1 (Type Reconstruction Algorithm $\mathcal{L}$). $\mathcal{L}(\Gamma, e) = (T, \tau, \Gamma')$ *where:*

1. *If e is the identifier x, and $x : A \in \Gamma$ then $T = \mathsf{id}$, $\tau = A$, $\Gamma' = \Gamma \setminus \{x : A\}$.*
2. *If e is of the form $\langle t, u\rangle$, let*

$$\begin{array}{l}(R, A, \Gamma_1) = \mathcal{L}(\Gamma, t)\\ (S, B, \Gamma_2) = \mathcal{L}(R\Gamma_1, u)\end{array}$$

then $T = SR, \tau = SA \otimes B, \Gamma' = \Gamma_2$.

3. *If e is of the form $\mathtt{let}\ \langle x, y\rangle = t\ \mathtt{in}\ u$, let*

$$\begin{array}{ll}(R, A, \Gamma_1) & = \mathcal{L}(\Gamma, t)\\ U & = \mathsf{mgu}\ A\ \alpha \otimes \beta; \quad \alpha, \beta \textit{ new}\\ (S, B, \Gamma_2) & = \mathcal{L}((UR\Gamma_1, x : U\alpha, y : U\beta), u)\end{array}$$

then $T = SUR, \tau = B, \Gamma' = \Gamma_2$.

4. *If e is of the form $\lambda x.t$, let*

$$(R, B, \Gamma_1) = \mathcal{L}((\Gamma, x : \alpha), t); \quad \alpha \textit{ new}$$

then $T = R, \tau = R\alpha \multimap B, \Gamma' = \Gamma_1$.

5. *If e is of the form tu, let*

$$\begin{array}{ll}(R, C, \Gamma_1) & = \mathcal{L}(\Gamma, t)\\ (S, A, \Gamma_2) & = \mathcal{L}(R\Gamma_1, u)\\ U & = \mathsf{mgu}\ (SC)\ (A \multimap \beta); \quad \beta \textit{ new}\end{array}$$

then $T = USR, \tau = U\beta, \Gamma' = \Gamma_2$.

6. *If e is 0 then $T = \mathsf{id}$, $\tau = \mathsf{Nat}$ and $\Gamma' = \Gamma$.*
7. *If e is $\mathsf{S}\ t$, and $\mathcal{L}(\Gamma, t) = (R, A, \Gamma_1)$, and $\mathsf{mgu}\ A\ \mathsf{Nat} = U$, then $T = UR$, $\tau = \mathsf{Nat}$ and $\Gamma' = \Gamma_1$.*
8. *If e is of the form $\mathsf{iter}\ t\ u\ v$, where $t \neq \mathsf{S}^m 0$, let*

$$\begin{array}{ll}(R, C, \Gamma_1) & = \mathcal{L}(\Gamma, t)\\ U & = \mathsf{mgu}\ C\ \mathsf{Nat}\\ (S, A, \Gamma_2) & = \mathcal{L}(UR\Gamma_1, u)\\ (T', B, \Gamma_3) & = \mathcal{L}(S\Gamma_2, v)\\ V & = \mathsf{mgu}\ B\ (T'(A \multimap A))\end{array}$$

then $T = VT'SUR, \tau = VT'A, \Gamma' = \Gamma_3$.

9. *If e is of the form $\mathsf{iter}\ (\mathsf{S}^m 0)\ u\ v$, let*

$$\begin{array}{lll} & (S, B, \Gamma_0) & = \mathcal{L}(\Gamma, u)\\ & (R, A, \Gamma_1) & = \mathcal{L}(S\Gamma_0, v)\\ & B_0 & = RB\\ & S_0 & = RS\\ \textit{for } i = 1 \cdots m\{ & U_i & = \mathsf{mgu}\ A\ (B_{i-1} \multimap \beta_i); \quad \beta_i \textit{ new}\\ & B_i & = U_i\beta_i\\ & S_i & = U_iS_0\\ \} & & \\ & \textit{Condition} & : S_1\Gamma_0 = \cdots = S_m\Gamma_0\end{array}$$

then $T = S_m, \tau = B_m, \Gamma' = \Gamma_1$.

10. *If e is* true *or* false *then* $T = \mathsf{id}$, $\tau = \mathtt{Bool}$ *and* $\Gamma' = \Gamma$.
11. *If e is of the form* $\mathsf{cond}\ t\ u\ v$, *let*

$$\begin{array}{ll}(R, C, \Gamma_1) & = \mathcal{L}(\Gamma, t)\\ U & = \mathsf{mgu}\ C\ \mathtt{Bool}\\ (S, \rho, \Gamma_2) & = \mathcal{L}(UR\Gamma_1, u)\\ (S', \sigma, \Gamma_3) & = \mathcal{L}(SUR\Gamma_1, v)\\ Condition & :\ \Gamma_3 = \Gamma_2\\ V & = \mathsf{mgu}\ \sigma\ S'\rho\end{array}$$

then $T = VS'SUR, \tau = V\sigma, \Gamma' = \Gamma_2 (= \Gamma_3)$.

Note that $\mathcal{L}$ fails if it is not one of the above forms.

Cases 1-7, 10 and 11 are standard for a linear λ-calculus with numbers, booleans and pairs (see [20]). Cases 8 and 9 deal with iterator terms. In case 8 we first check that t can be given type Nat, then type u with the remaining assumptions, and finally type v using only the assumptions not consumed in the typing of t and u, checking that v has an arrow type of the correct form. The interesting case is 9: here we deal with an iterator term in which the number of iterations is known. We type u and v as in case 8, and then check that v can be given a set of iterative types.

Soundness of $\mathcal{L}$. If the algorithm $\mathcal{L}$ succeeds in typing a term e under some assumptions, then we want to be sure that e actually is typable. This is called Soundness and states that our algorithm is safe—it produces no wrong results.

Lemma 3 (Substitution). *If there is a derivation $\Gamma \mid \Gamma' \vdash_{\mathcal{L}} e : \tau$ then, for any substitution S, there is also a derivation for $S(\Gamma \mid \Gamma') \vdash_{\mathcal{L}} e : S\tau$.*

Proof. By induction over the length of the derivation.

Theorem 2 (Soundness of $\mathcal{L}$). *If $\mathcal{L}(\Gamma, e)$ succeeds with (S, τ, Γ') then there is a derivation of $S(\Gamma \mid \Gamma') \vdash_{\mathcal{L}} e : \tau$.*

Proof. By induction on the structure of terms e, using the Substitution Lemma and the fact that substitutions are idempotent. We show two cases:

1. If e is of the form $\langle t, u\rangle$ then $\mathcal{L}(\Gamma, t)$ succeeds with (R, A, Γ_1) and $\mathcal{L}(R\Gamma_1, u)$ succeeds with (S, B, Γ_2). By induction twice, there are derivations $R(\Gamma \mid \Gamma_1) \vdash_{\mathcal{L}} t : A$ and $S(R\Gamma_1 \mid \Gamma_2) \vdash_{\mathcal{L}} u : B$. Since Γ_2 is included in $R\Gamma_1$, and R is idempotent, also $SR(\Gamma_1 \mid \Gamma_2) \vdash u : B$. By Lemma 3 we can write the first derivation as $SR(\Gamma \mid \Gamma_1) \vdash t : SA$. Now, by $\otimes\mathsf{Intro}$ $SR(\Gamma \mid \Gamma_2) \vdash_{\mathcal{L}} e : SA \otimes B$.
2. If e is of the form $\mathsf{iter}\ t\ u\ v$ and $t \neq S^m 0$, then $\mathcal{L}(\Gamma, t)$ succeeds with (R, C, Γ_1) and $\mathsf{mgu}\ C\ \mathsf{Nat}$ succeeds with a substitution U. $\mathcal{L}(UR\Gamma_1, u)$ succeeds with (S, A, Γ_2), $\mathcal{L}(S\Gamma_2, v)$ succeeds with (T', B, Γ_3), and $\mathsf{mgu}\ B\ (T'A \multimap T'A)$ succeeds with a substitution V. Now by induction and Lemma 3 there are derivations ending in $VT'SUR(\Gamma \mid \Gamma_1) \vdash_{\mathcal{L}} t : \mathsf{Nat}$, $VT'SUR(\Gamma_1 \mid \Gamma_2) \vdash_{\mathcal{L}} u : VT'A$, and $VT'SUR(\Gamma_2 \mid \Gamma_3) \vdash_{\mathcal{L}} v : VT'A \multimap VT'A$, and the result follows by the Iter rule.

Completeness of $\mathcal{L}$. If a term can be typed using the inference rules, then we would require our algorithm to also be able to compute the type of this term. The proof follows closely the proof of the completeness of $\mathcal{W}$ in [8]. Note that a notion of *principal* type follows as an immediate corollary of the completeness theorem.

Theorem 3 (Completeness of $\mathcal{L}$). *If there is a derivation $S(\Gamma \mid \Gamma_1) \vdash_{\mathcal{L}} e : \tau$ for some substitution S, then:*

1. *$\mathcal{L}(\Gamma, e)$ succeeds with (R, A, Γ_1) for some R, A.*
2. *There exists a substitution T such that: $TR(\Gamma \mid \Gamma_1) = S(\Gamma \mid \Gamma_1)$ and $TA = \tau$.*

Proof. By induction over the structure of e.

4 Iterative Types

In this section we present two intersection type systems closely related to System $\mathcal{L}$. The first one is based on Damas's type system[8], a less known polymorphic type system with the same power of the Hindley-Milner system. The second is based on rank-2 intersection types [4,16].

Iterative types are a compact way of expressing several type derivations for an iterated function. Consider the iterator function itself $\lambda x.\mathsf{iter}\ t\ u\ x$. When this function is applied to a term v our type rules assume that v must be typed with every type in $It(A_0, \ldots, A_n)$. Another way to see it is to type $\mathsf{iter}\ t\ u\ x$ with multiple assumptions for x, and then type v with all the elements of the set of types declared for x. One standard way to extend a type system by allowing multiple assumptions for free variables is by using intersection types.

4.1 Polymorphic Iteration

In the system presented here, which we call System $\mathcal{L}_{\mathcal{I}}$, intersection types are only used in the set of assumptions for free variables. This kind of restriction to intersection type systems was first used in Damas's PhD thesis [8] in the definition of a system (later called Damas's System T [15]) with the same set of typable expressions as the widely known Hindley-Milner system, but that instead of using $\forall$-quantified types, allows multiple types in the set of assumptions for each free variable. We will use a similar method to type iterators.

We consider the set *Types*, of linear types defined in Section 2. Let S range over the set of all finite non-empty subsets of *Types*. The set *Inter* of *intersection types* is defined as follows: $\bar{A} ::= \wedge S$. The type environments (or bases in the terminology of intersection systems) of the systems presented in this section represent a total function from the set of variables of the term to *Inter*. Bases that associate to term-variables elements of *Types* will be called *monomorphic*.

System $\mathcal{L}_{\mathcal{I}}$ is obtained from System $\mathcal{L}$ by replacing the rule for (Iter) by the two rules given in Figure 3.

$$\frac{\Gamma \vdash_{\mathcal{L}_\mathcal{I}} t : \mathsf{Nat} \qquad \Delta \vdash_{\mathcal{L}_\mathcal{I}} u : A_0}{\Gamma, x : \wedge It(A_0, \dots, A_n), \Delta \vdash_{\mathcal{L}_\mathcal{I}} \mathsf{iter}\ t\ u\ x : A_n}\ (\mathsf{VarIter})$$

($\star$) where if $t \equiv \mathsf{S}^m 0$ then $n = m$ otherwise $n = 0$

$$\frac{\Gamma, x : \wedge S \vdash_{\mathcal{L}_\mathcal{I}} \mathsf{iter}\ t\ u\ x : A \qquad \forall B_i \in S. \Delta \vdash_{\mathcal{L}_\mathcal{I}} v : B_i}{\Gamma, \Delta \vdash_{\mathcal{L}_\mathcal{I}} \mathsf{iter}\ t\ u\ v : A}\ (\mathsf{Iter})$$

Fig. 3. System $\mathcal{L}$ with Intersection Types

System $\mathcal{L}_\mathcal{I}$ allows multiple types in the set of assumptions for each free variable. This can be seen as using intersection types for free variables and System $\mathcal{L}_\mathcal{I}$ can be seen as a restriction of a system of rank-2 intersection types.

We now show two results relating System $\mathcal{L}$ and System $\mathcal{L}_\mathcal{I}$.

Theorem 4. *If there is a derivation $\Gamma \vdash_\mathcal{L} e : \tau$, then $\Gamma \vdash_{\mathcal{L}_\mathcal{I}} e : \tau$*

Proof. We only show the case for Iter, as the other cases are trivial by induction.

If e is of the form $\mathsf{iter}\ t\ u\ v$, then $\Gamma, \Theta, \Delta \vdash_\mathcal{L} \mathsf{iter}\ t\ u\ v : A_n$ if $\Gamma \vdash_\mathcal{L} t : \mathsf{Nat}$, $\Theta \vdash_\mathcal{L} u : A_0$ and $\Delta \vdash_\mathcal{L} v : It(A_0, \dots, A_n)$, where if $t \equiv \mathsf{S}^m 0$ then $n = m$ otherwise $n = 0$. By induction, $\Gamma \vdash_{\mathcal{L}_\mathcal{I}} t : \mathsf{Nat}$ and $\Theta \vdash_{\mathcal{L}_\mathcal{I}} u : A_0$. Therefore by VarIter and Exchange, $\Gamma, \Theta, x : \wedge It(A_0, \dots, A_n) \vdash_{\mathcal{L}_\mathcal{I}} \mathsf{iter}\ t\ u\ x : A_n$. Again by induction, $\forall B_i \in It(A_0, \dots, A_n). \Delta \vdash_{\mathcal{L}_\mathcal{I}} v : B_i$. Thus, by Iter

$$\Gamma, \Theta, \Delta \vdash_{\mathcal{L}_\mathcal{I}} \mathsf{iter}\ t\ u\ v : A_n.$$

This last result shows that any term typable in System $\mathcal{L}$ is also typable in $\mathcal{L}_\mathcal{I}$. The opposite does not hold, i.e, system $\mathcal{L}_\mathcal{I}$ allows more typings than System $\mathcal{L}$. In particular, when typing an open term of the form $\mathsf{iter}\ (\mathsf{S}^m 0)\ u\ x$, it allows x to have an iterative type $It(A_0, \dots, A_n)$. For example, we can have the following derivation in System $\mathcal{L}_\mathcal{I}$ (consider for example, $\Gamma = \{x : \wedge It(A \multimap \mathsf{Nat} \multimap \mathsf{Nat} \otimes \mathsf{Nat}, \dots, \mathsf{Nat} \otimes \mathsf{Nat})\}$):

$$\Gamma \vdash_{\mathcal{L}_\mathcal{I}} (\lambda y.\mathsf{fst}\ y)(\mathsf{iter}\ (\mathsf{S}^2 0)\ (\lambda x_1 x_2.\langle x_1, x_2\rangle)\ x) : \mathsf{Nat}$$

but the term $(\lambda y.\mathsf{fst}\ y)(\mathsf{iter}\ (\mathsf{S}^2 0)\ (\lambda x_1 x_2.\langle x_1, x_2\rangle)\ x)$ is not typable in System $\mathcal{L}$, because, for $(\mathsf{iter}\ (\mathsf{S}^2 0)\ (\lambda x_1 x_2.\langle x_1, x_2\rangle)\ x)$, we can only have derivations of the form (consider $\Gamma = \{x : (A \multimap A \multimap A \otimes A) \multimap (A \multimap A \multimap A \otimes A)\}$):

$$\Gamma \vdash_\mathcal{L} \mathsf{iter}\ (\mathsf{S}^2 0)\ (\lambda x_1 x_2.\langle x_1, x_2\rangle)\ x : A \multimap A \multimap A \otimes A.$$

Note however that, if the bases used in derivations in System $\mathcal{L}_\mathcal{I}$ are monomorphic, then those terms are also typable in System $\mathcal{L}$.

Theorem 5. *If there is a derivation $\Gamma \vdash_{\mathcal{L}_\mathcal{I}} e : \tau$, with Γ monomorphic, then $\Gamma \vdash_\mathcal{L} e : \tau$.*

Proof. We only show the case for Iter, as the other cases are trivial by induction.

If e is of the form $\mathsf{iter}\ t\ u\ v$, then $\Gamma, \Delta \vdash_{\mathcal{L}_\mathcal{I}} \mathsf{iter}\ t\ u\ v : A_n$ if

$$\Gamma, x : \wedge It(A_0, \ldots, A_n) \vdash_{\mathcal{L}_\mathcal{I}} : \mathsf{iter}\ t\ u\ x : A_n \quad \Delta \vdash_{\mathcal{L}_\mathcal{I}} v : It(A_0, \ldots, A_n)$$

Also, $\Gamma, x : \wedge It(A_0, \ldots, A_n) \vdash_{\mathcal{L}_\mathcal{I}} : \mathsf{iter}\ t\ u\ x : A_n$ if $\Gamma' \vdash_{\mathcal{L}_\mathcal{I}} t : \mathsf{Nat}$ and $\Gamma'' \vdash_{\mathcal{L}_\mathcal{I}} u : A_0$ where if $t \equiv \mathsf{S}^m 0$ then $n = m$ otherwise $n = 0$, and $\Gamma = \Gamma', \Gamma''$. By induction hypothesis: $\Gamma' \vdash_{\mathcal{L}} t : \mathsf{Nat} \quad \Gamma'' \vdash_{\mathcal{L}} u : A_0 \quad \Delta \vdash_{\mathcal{L}} v : It(A_0, \ldots, A_n)$. Thus, by Iter $\Gamma, \Delta \vdash_{\mathcal{L}} \mathsf{iter}\ t\ u\ v : A_n$.

In particular for closed terms, the two systems are equivalent.

Corollary 1. $\vdash_{\mathcal{L}} e : \tau$ *iff* $\vdash_{\mathcal{L}_\mathcal{I}} e : \tau$.

4.2 Rank 2 Intersection Types: System $\mathcal{L}^2_\mathcal{I}$

Being able to type terms of the form $\mathsf{iter}\ (\mathsf{S}^m 0)\ u\ x$ using intersections like we do in System $\mathcal{L}_\mathcal{I}$, does not really give us more interesting terms, because we can not abstract on x, therefore it will never be replaced by the function to iterate. The system presented now extends System $\mathcal{L}_\mathcal{I}$ in that sense.

The rank 2 intersection type assignment for System $\mathcal{L}$ (which we call System $\mathcal{L}^2_\mathcal{I}$) is obtained from System $\mathcal{L}_\mathcal{I}$, by replacing the rules $\multimap\mathsf{Intro}$ and $\multimap\mathsf{Elim}$ by the two rules given in Figure 4. Note that we do not distinguish the types $\wedge\{A\}$

$$\frac{\Gamma, x : \wedge S \vdash_{\mathcal{L}^2_\mathcal{I}} t : B}{\Gamma \vdash_{\mathcal{L}^2_\mathcal{I}} \lambda x.t : \wedge S \multimap B}\ (\multimap\mathsf{Intro})$$

$$\frac{\Gamma \vdash_{\mathcal{L}^2_\mathcal{I}} t : \wedge S \multimap B \qquad \forall A_i \in S. \Delta \vdash_{\mathcal{L}^2_\mathcal{I}} u : A_i}{\Gamma, \Delta \vdash_{\mathcal{L}^2_\mathcal{I}} tu : B}\ (\multimap\mathsf{Elim})$$

Fig. 4. Rank 2 Intersection Types version of System $\mathcal{L}$

and A. This system corresponds to a linear version of a rank 2 intersection type system with iterators, and it includes System $\mathcal{L}$ (and System $\mathcal{L}_\mathcal{I}$).

Theorem 6. *If there is a derivation* $\Gamma \vdash_{\mathcal{L}} e : \tau$*, then* $\Gamma \vdash_{\mathcal{L}^2_\mathcal{I}} e : \tau$

Proof. Similar to Theorem 4.

Note that terms typable in System $\mathcal{L}_\mathcal{I}$ are also typable in System $\mathcal{L}^2_\mathcal{I}$, since the rules $\multimap\mathsf{Intro}$ and $\multimap\mathsf{Elim}$ of System $\mathcal{L}_\mathcal{I}$, are a subcase of the same rules in System $\mathcal{L}^2_\mathcal{I}$.

System $\mathcal{L}^2_\mathcal{I}$ is stronger than System $\mathcal{L}_\mathcal{I}$ (therefore, than System $\mathcal{L}$), since it allows abstractions on polymorphic variables. Note however, that polymorphic variables are only introduced through $\mathsf{VarIter}$. For example, we can have the following typing in System $\mathcal{L}^2_{\mathcal{I}}$:

$$\vdash_{\mathcal{L}^2_{\mathcal{I}}} (\lambda y.(\lambda x.\mathsf{fst}\ x)(\mathsf{iter}\ (\mathsf{S}^2 0)\ (\lambda x_1 x_2.x_1 x_2)\ y))(\lambda z.z(\mathsf{S}^3 0)) : \mathsf{Nat}$$

but this term is not typable in System $\mathcal{L}$. Note also that System $\mathcal{L}^2_{\mathcal{I}}$ allows us to write more compact versions of admissible linear terms. Consider for example F to be a closed function with types $It(A \multimap \mathsf{Nat} \multimap \mathsf{Nat} \otimes \mathsf{Nat}, \ldots, \mathsf{Nat} \otimes \mathsf{Nat})$, then $(\lambda fp.\mathsf{cond}\, p\ (\mathsf{iter}\ (\mathsf{S}^2 0)\ 0\ f)\ (\mathsf{iter}\ (\mathsf{S}^2 0)\ (\mathsf{S}0)\ f))F$ is typable in System $\mathcal{L}^2_{\mathcal{I}}$.

Subject reduction for Systems $\mathcal{L}_{\mathcal{I}}$ and $\mathcal{L}^2_{\mathcal{I}}$ is proved in a similar way as for System $\mathcal{L}$. As for confluence, since we proved confluence for untyped terms in [3], that result, together with subject reduction, implies confluence for terms typable in Systems $\mathcal{L}_{\mathcal{I}}$ and $\mathcal{L}^2_{\mathcal{I}}$.

Summarising, we have shown how iterative types are related with intersection types, which in turn shows the expressiveness of System $\mathcal{L}$. The relation between the set of terms typable in the three systems is: $\mathcal{L} \subset \mathcal{L}_{\mathcal{I}} \subset \mathcal{L}^2_{\mathcal{I}}$. Furthermore, for closed terms (therefore programs): $\mathcal{L} = \mathcal{L}_{\mathcal{I}}$.

5 Conclusions

We have studied a new type construct, shown its relationship with intersection types, and given a type reconstruction algorithm for it. Since it is known that the calculus is strongly normalising, type reconstruction has the usual applications. The results relating iterative types and intersection types, together with the results in [3] which show that System $\mathcal{L}$ can simulate Gödel's System $\mathcal{T}$, indicate that System $\mathcal{L}$ is even more expressive than System $\mathcal{T}$, it actually corresponds to a version of System $\mathcal{T}$ with a restricted form of intersection types.

References

1. S. Abramsky. Computational Interpretations of Linear Logic. *Theoretical Computer Science*, 111:3–57, 1993.
2. S. Alves, M. Fernández, M. Florido, and I. Mackie. The power of closed-reduction strategies. In *Proceedings of WRS 2006, 6th International Workshop on Rewriting Strategies, FLOC 2006, Seattle*, 2006.
3. S. Alves, M. Fernández, M. Florido, and I. Mackie. The power of linear functions. In *Proceedings of Computer Science Logic, CSL 2006*, LNCS. Springer Verlag, 2006.
4. S. Bakel. *Intersection Type Disciplines in Lambda Calculus and Applicative Term Rewriting Systems.* PhD thesis, Department of Computer Science, University of Nijmegen, 1993.
5. S. Bakel. Intersection type assignment systems. *Theoretical Computer Science*, 151(2):385–435, 1995.
6. H. P. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type-assignment. *J. Symbolic Logic*, 48:931–940, 1983.
7. M. Coppo and M. Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame Journal of Formal Logic*, 21(4):685–693, 1980.

8. L. M. M. Damas. *Type Assignment in Programming Languages.* PhD thesis, University of Edinburgh, 1985.
9. L. M. M. Damas and R. Milner. Principal type schemes for functional programs. In *Conference Record of the Ninth Annual ACM Symposium on the Principles of Programming Languages*, pages 207–212, 1982.
10. F. Damiani. Rank-2 intersection and polymorphic recursion, 2005.
11. M. Fernández, I. Mackie, and F.-R. Sinot. Closed reduction: explicit substitutions without alpha conversion. *Mathematical Structures in Computer Science*, 15(2):343–381, 2005.
12. J.-Y. Girard. Linear Logic. *Theoretical Computer Science*, 50(1):1–102, 1987.
13. J.-Y. Girard. Geometry of interaction 1: Interpretation of System F. In R. Ferro, C. Bonotto, S. Valentini, and A. Zanardo, editors, *Logic Colloquium 88*, volume 127 of *Studies in Logic and the Foundations of Mathematics*, pages 221–260. North Holland Publishing Company, Amsterdam, 1989.
14. J.-Y. Girard. Towards a geometry of interaction. In J. W. Gray and A. Scedrov, editors, *Categories in Computer Science and Logic: Proc. of the Joint Summer Research Conference*, pages 69–108. American Mathematical Society, Providence, RI, 1989.
15. C. Haack and J. B. Wells. Type error slicing in implicitly typed higher-order languages. *Sci. Comput. Program.*, 50(1-3):189–224, 2004.
16. T. Jim. Rank 2 type systems and recursive definitions. Technical report, Massachusetts Institute of Technology, 1995.
17. T. Jim. What are principal typings and what are they good for? In *Proceedings of the 23rd ACM Symposium on Principles of Programming Languages (POPL'96).* ACM Press, 1996.
18. A. J. Kfoury, H. G. Mairson, F. A. Turbak, and J. B. Wells. Relating typability and expressibility in finite-rank intersection type systems. In *Proceedings of the 1999 International Conference on Functional Programming*, pages 90–101. ACM Press, 1999.
19. Y. Lafont. Interaction nets. In *Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL'90)*, pages 95–108. ACM Press, Jan. 1990.
20. I. Mackie. Lilac: A functional programming language based on linear logic. *Journal of Functional Programming*, 4(4):395–433, 1994.
21. A. Martelli and U. Montanari. An efficient unification algorithm. *Transactions on Programming Languages and Systems*, 4(2):258–282, 1982.
22. R. Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17(3):348–375, 1978.
23. G. Pottinger. A type assignment for strongly normalizable λ-terms. In *To H.B. Curry, Essays in Combinatory Logic, Lambda-Calculus and Formalism*, pages 535–560. Academic Press, 1980.
24. J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.
25. Y. Toyama. Confluent term rewriting systems with membership. In *Proceedings of the 1st International Workshop on Conditional Term Rewriting Systems, CTRS'87, Orsay, France*, volume 308 of *LNCS*, pages 228–241. Springer-Verlag, 1988.
26. J. Yamada. Confluence of terminating membership conditional trs. In *Proceedings of the 3rd International Workshop on Conditional Term Rewriting Systems, CTRS'92, Pont--Mousson, France*, volume 656 of *LNCS*, pages 378–392. Springer-Verlag, 1993.

Types and Effects for Resource Usage Analysis

Massimo Bartoletti, Pierpaolo Degano, Gian Luigi Ferrari, and Roberto Zunino

Dipartimento di Informatica, Università di Pisa, Italy
{bartolet, degano, giangi, zunino}@di.unipi.it

Abstract. An extension of the λ-calculus is proposed, to study resource usage analysis and verification. Resources can be dynamically created, and passed / returned by functions; their usages have side effects, represented by events. Usage policies are properties over histories of events, and have a possibly nested, local scope. A type and effect system over-approximates the set of histories a program can generate at run-time. A crucial point solved here concerns correctly associating fresh resources with their usages within approximations. A second issue is that these approximations may contain an unbounded number of fresh resources. Despite of that, we have devised a technique to model-check validity of approximations. A program with a valid approximation is resource-safe: no run-time monitor is needed to safely drive its executions.

1 Introduction

An important aspect of programming language design and implementation is how to ensure that resources are used correctly. The typical run-time mechanisms for enforcing resource usage policies are *execution monitors*, which abort executions whenever about to violate the usage policy prescribed by the programmer. The events observed by these monitors are accesses to sensible resources, e.g. opening socket connections, reading/writing files, allocating/deallocating memory. A main issue is finding a compromise between the expressivity of usage policies and the efficiency of the enforcement mechanism. Static analysis techniques may be applied to improve efficiency, but this often results in an unacceptable restriction of the expressive power of policies.

A common mechanism for enforcing usage policies consists in guarding with *local checks* the program points where critical resources can be accessed [10,18]. Local checks have a main drawback: they must be explicitly inserted into code by the programmer. Since forgetting even a single check might compromise the safety of the whole application, programmers have to inspect very carefully their code. This may be cumbersome even for small programs, and it may easily lead to unnecessary checking.

A safer approach is that of *global policies*, where the execution monitor enforces a global invariant that must hold at any point of the execution. This may involve guarding each resource access, and ad-hoc optimizations are then in order to recover efficiency, e.g. compiling the global policy to local checks [7,14]. Furthermore, a large monolithic policy may be hard to understand, and not very flexible either. Indeed, one has to imagine all the possible resource usage scenarios in advance. If an unexpected situation occurs at run-time (e.g. a piece of mobile code with specific resource usage requirements), the global policy must be dynamically updated, if possible at all.

A more flexible approach consists in attaching usage policies to resources, so to adapt them to the context where a resource is used. For example, one may restrain the capabilities before calling untrusted code. In [12], a type system extracts from programs an approximation of their possible run-time usage behaviour. Usage policies are arbitrary sets of permitted histories, so statically verifying whether the permitted usages include the extracted approximation is undecidable. Run-time monitoring is thus still needed, unless one restricts to some decidable fragments. As for expressiveness, a limitation is that you can only control the usage of resources you have created. In a mobile code scenario, e.g. a browser that runs untrusted applets, it is also important that you can impose constraints on how external programs manage the resources created in your local environment. For example, an applet may create an unbounded number of resources on the browser site, and never release them. This clearly leads to denial-of-service attacks, that may eventually crash the whole system.

We consider here a language that aims at reconciling expressivity of resource usage policies with efficiency of the enforcement mechanism. This language, called $\lambda^{[]}$ (lambda-box), has primitives for creating and accessing resources, and for defining *local* resource usage policies. Sequences of resource accesses in executions are called *histories*; a *policy* is a regular property of histories. A program fragment e protected by a policy φ is written $\varphi[e]$, called *policy framing*. Roughly, while evaluating e, the histories must respect the policy φ. Of course, framings can be nested.

Local policies generalise both local checks and global policies. They smoothly allow for safe composition of programs with their own private policies, also in mobile code scenarios. Indeed, there is no need to dynamically accommodate the local private policies into a single global one, possibly invalidating syntax-directed optimizations of the enforcement mechanism. Local policies may offer protection also in the web-services scenario [3]: there, one has not full control on the code to run, and thus inserting local checks is infeasible. For example, a browser must obey a usage policy specified by the user. Additionally, the browser can invoke a policy provider to obtain a stricter security policy, used for dynamically sandboxing applets. This rich interplay between policies seems difficult to express in the above-mentioned approaches.

In $\lambda^{[]}$, efficiency of resource usage control is obtained through a suitable combination of static techniques. The type and effect system over-approximates the run-time usage behaviour of a program, by inferring a *history expression* that denotes all the possible histories resulting from executions. A history expression is *valid* when it contains permitted usage patterns only; a program with a valid history expression will never go wrong. Validity of history expressions is then verified through model-checking.

This approach was originally introduced in [1] to deal with *history-based access control*. The present version extends [1] with dynamic creation of resources. This apparently little extension demands for addressing a more general problem, from various viewpoints: one has to correctly bind the creation of new resources to their usages. The solution to this problem deeply affects the techniques of [1], with respect to the following points: (i) the enforcement mechanism, (ii) the semantics of history expressions, (iii) the type and effect system, and (iv) the verification technique.

For the first point, we introduce *template usage automata*: they are an extension of finite state automata (FSA) where the input alphabet is parametrized over resources. A policy φ is represented by a template usage automaton $A_{\varphi(x)}$. To enforce φ, the usage histories of each resource r must be accepted by the FSA $A_{\varphi(r)}$, obtained by instantiating $A_{\varphi(x)}$ on r. For (ii), the semantics of a history expression is a set of histories: the problem here is to equate those histories that only differ in the name of fresh resources. For (iii), the problem is to correctly record the binding of fresh names in history expressions. Constructing the history expression of a program is a basic step in our approach: indeed, checking that a program obeys the usage policies requires knowing all its possible histories in their entirety — history safety is *not* compositional. Technically, we explore a novel approach to quantify types over freshly created resources — a sort of polymorphism à la ML on *both* types and effects. We avoid using explicit binders in types: the definition/use of resources is determined after the type & effect has been inferred. Living without binders made the type and effect system simpler (and required some little ingenuities in proofs). For (iv), the creation of new resources may give rise to an infinite number of formulae to be inspected while verifying validity. We solve this problem by suitably grouping resources with equivalent usage constraints. This allows us to extract from a history expression a Basic Process Algebra [5] and a regular formula, to be used in model-checking validity [9].

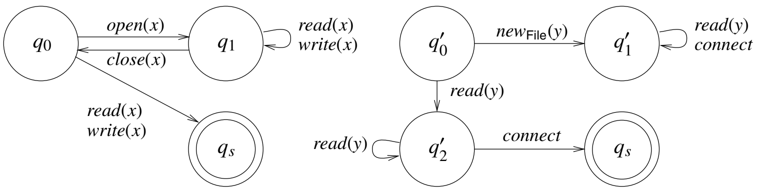
A key point of our proposal is that we offer a comprehensive framework for safely handling resources, within a linguistic setting. On the one hand, our calculus has an expressive and flexible way to compose and enforce usage policies. On the other hand, resource usage control is made feasible by suitably extending and integrating techniques from type theory and model-checking.

2 Programming Model

We consider a call-by-value λ-calculus enriched with primitives for creating and accessing resources, and with local usage policies. Resources $r, r', \ldots \in \mathsf{Res}$ are system objects that can be either statically available in $\mathsf{Res}_0 \subset \mathsf{Res}$ or created dynamically. We assume that resources can be accessed through a given finite set of actions $\alpha, \alpha', \ldots \in \mathsf{Act}$. This set is partitioned to reflect the kinds of resources, i.e. $\mathsf{Act} = \bigcup_i \mathsf{Act}_i = \mathsf{File} \cup \mathsf{Socket} \cup \cdots$, where each element of the partition contains the actions admissible for the given kind (e.g. $\mathsf{File} = \{new_{\mathsf{File}}, open, close, read, write\}$). The action new_{Act_i} represents the creation of a new resource of kind Act_i. An *event* $\alpha(r)$ denotes accessing the resource r through the action α. We assume a global *capability environment* Γ_0 that maps each resource in Res_0 to the set of actions it admits. A *history* η is a sequence of access events.

Usage automata. Usage policies $\varphi, \varphi', \ldots \in \mathsf{Pol}$ are regular properties of histories. Each of them is modelled by a *template usage automaton* $A_{\varphi(x)} = \langle Q, q_0, q_s, E \rangle$, which gives rise to a FSA when the parameter x is instantiated to an actual resource r. As usual, Q is a *finite* set of states, $q_0 \in Q$ is the start state, while $q_s \in Q$ is the final *sink* state, and E is a finite set of *template edges* of the form $q \xrightarrow{\vartheta} q'$, where $\vartheta \in \{\alpha(x), \alpha(\bar{x}), \alpha(r) \mid \alpha \in \mathsf{Act} \wedge r \in \mathsf{Res}_0\}$. The wildcard $\bar{x}$ stands for "any resource different from x".

Example 1. Consider a file usage policy φ saying that only open files can be read or written, and a security policy φ' saying that, after having read a file you have not created, you can no longer connect to the network (a sort of "Chinese Wall" property). These policies are described below by the template automata $A_{\varphi(x)}$ (left) and $A_{\varphi'(y)}$ (right, where the resource associated with the action *connect* is irrelevant).



A template usage automaton is *well-kinded* when the resources r in its edges are accessed according to the capability environment Γ_0, i.e. an edge labelled $\alpha(r)$ requires that $\alpha \in \Gamma_0(r)$. Also, the parameter x must be used consistently: for example, if x is used as a file, e.g. in $read(x)$, then it cannot be used also as a socket, or as a printer. To this purpose, we define the kinding function:

$$\kappa(\varphi) = \kappa(\{\, \alpha \mid \exists \zeta \in \{x, \bar{x}\} : q \xrightarrow{\alpha(\zeta)} q' \in E_{\varphi(x)} \,\} \quad \kappa(\gamma) = \begin{cases} \mathsf{Act}_i & \text{if } \exists i.\, \emptyset \subset \gamma \subseteq \mathsf{Act}_i \\ \emptyset & \text{otherwise} \end{cases}$$

and we require that $\kappa(\varphi) \neq \emptyset$ for all φ. We assume our automata be always well-kinded.

Given a finite set of resources R, a template usage automaton $A_{\varphi(x)}$ is instantiated into a FSA $A_{\varphi(r,R)}$ by binding x to the resource $r \in R$ (we simply write $A_{\varphi(r)}$ when unambiguous). Intuitively, a template edge $q \xrightarrow{\alpha(x)} q'$ results in a transition $\langle q, \alpha(r), q' \rangle$. The instantiation $A_{\varphi(r,R)}$ is $\langle Q, q_0, \Sigma, \delta, F \rangle$, where $\Sigma = \{\alpha(r') \mid \alpha \in \mathsf{Act} \wedge r' \in R\}$, $F = \{q_s\}$, and the transition relation $\delta : Q \times \Sigma \times Q$ is defined as follows:

$$\bar{\delta} = \{\langle q, \alpha(r), q' \rangle \mid q \xrightarrow{\alpha(x)} q' \,\} \cup \{\langle q, \alpha(r'), q' \rangle \mid q \to \alpha(r') q' \} \cup \bigcup_{r' \in R \setminus \{r\}} \{\langle q, \alpha(r'), q' \rangle \mid q \xrightarrow{\alpha(\bar{x})} q'\}$$

$$\delta = \bar{\delta} \cup \{\, \langle q, \alpha(r'), q \rangle \mid r' \in R, \nexists \langle q, \alpha(r'), q' \rangle \in \bar{\delta} \,\} \cup \{\, \langle q, \alpha(?), q' \rangle \mid \exists \zeta : q \xrightarrow{\alpha(\zeta)} q' \,\}$$

In the first line we instantiate x to the given resource r, we maintain the transitions $\alpha(r')$ for $r' \in \mathsf{Res}_0$, and we instantiate $\alpha(\bar{x})$ with all $r' \neq r$. In the second line we add self-loops for all the events not explicitly mentioned in the template automaton. The last set is only used in the verification phase; the meaning of the special symbol ? will be explained later. Note that finiteness of R and of Act guarantees that $A_{\varphi(r)}$ is always a finite state automaton. The assumption that R is finite causes no loss of generality, because each time a template usage automaton is instantiated in $\lambda^{[\,]}$ executions, the number of resources occurring in the history is finite. We denote with $\mathcal{L}(\varphi(r, R))$ the language *not* accepted by $A_{\varphi(r)}$ — thus going into the sink state represents a violation of the policy. Also, $\ell(\varphi)$ stands for the set of actions and resources labelling the template edges.

The language $\boldsymbol{\lambda^{[\,]}}$**.** The syntax of $\lambda^{[\,]}$ comprises variables $x, y, \ldots \in \mathsf{Var}$, resources $r, r', \ldots \in \mathsf{Res}$, events $\alpha(e)$, abstractions $\lambda_z x.\, e$ (where z within e stands for the whole abstraction), applications $e\, e'$, conditional expressions `if` b `then` e `else` e' (the definition of guard b is irrelevant here), policy framings $\varphi[e]$, and resource creation `new` $x : \gamma$ `in` e, where $\gamma \subseteq \mathsf{Act}$ is the set of capabilities associated with the new resource.

Variables, resources, abstractions, and *failures* are the values $v, v', \dots$ of $\lambda^{[]}$. A failure fail_κ occurs when a computation is about to access a resource of the wrong kind. A failure $\mathit{fail}_{\varphi(r)}$ is raised when about to violate the policy $\varphi(r)$. Let fail denote both kinds of failures. We assume that, for any expression e, policy φ, and action α: $\mathit{fail} = e\,\mathit{fail} = \mathit{fail}\ e = \varphi[\mathit{fail}] = \alpha(\mathit{fail})$. We write $*$ for a fixed, closed, access-free, non-failure value, and $\lambda.\,e$ for $\lambda x.\,e$ when $x \notin \mathit{fv}(e)$ (x not free in e). The following abbreviations are standard: $e; e' = (\lambda.\,e')\,e$, and $\mathtt{let}\,x = e\,\mathtt{in}\,e' = (\lambda x.\,e')\,e$. We write α instead of $\alpha(r)$ when the parameter r is immaterial. W.l.o.g. we assume that each framing has an opening event, i.e. for all $\varphi[e]$, the expression e has the form $\alpha; e'$, for some α and e'. The opening event can be dummy, with no influence on usage policies.

We define the behaviour of $\lambda^{[]}$ expressions through the following small-step operational semantics. A transition $\eta, \Gamma, e \to \eta', \Gamma', e'$ means that, starting from a history η and capability environment Γ, the expression e may evolve to e' (possibly a failure), the history η to η' and the capability environment Γ to Γ'. Initial configurations have the form ε, Γ_0, e, where ε denotes the empty history.

$$\frac{\eta,\Gamma,e_1 \to \eta',\Gamma',e_1'}{\eta,\Gamma,e_1 e_2 \to \eta',\Gamma',e_1' e_2} \qquad \frac{\eta,\Gamma,e_2 \to \eta',\Gamma',e_2'}{\eta,\Gamma,v\,e_2 \to \eta',\Gamma',v\,e_2'}$$

$$\eta,\Gamma,(\lambda_z x.e)v \to \eta,\Gamma,e\{v/x, \lambda_z x.e/z\} \qquad \eta,\Gamma,\mathtt{if}\,b\,\mathtt{then}\,e_{tt}\,\mathtt{else}\,e_{ff} \to \eta,\Gamma,e_{\mathcal{B}(b)}$$

$$\frac{\eta,\Gamma,e \to \eta',\Gamma',e'}{\eta,\Gamma,\alpha(e) \to \eta',\Gamma',\alpha(e')} \qquad \frac{\alpha \in \Gamma(r)}{\eta,\Gamma,\alpha(r) \to \eta\alpha(r),\Gamma,*} \qquad \frac{\alpha \notin \Gamma(r)}{\eta,\Gamma,\alpha(r) \to \eta,\Gamma,\mathit{fail}_\kappa}$$

$$\frac{\eta,\Gamma,e \to \eta',\Gamma',e' \quad \eta' \models \varphi}{\eta,\Gamma,\varphi[e] \to \eta',\Gamma',\varphi[e']} \qquad \frac{\eta \models \varphi}{\eta,\Gamma,\varphi[v] \to \eta,\Gamma,v} \qquad \frac{\eta,\Gamma,e \to \eta',\Gamma',e' \quad \eta' \not\models_r \varphi}{\eta,\Gamma,\varphi[e] \to \eta,\Gamma,\mathit{fail}_{\varphi(r)}}$$

$$\eta,\Gamma,\mathtt{new}\,x:\gamma\,\mathtt{in}\,e \to \eta\,\mathit{new}_{\kappa(\gamma)}(r), \Gamma \cup \{\gamma/r\}, e\{r/x\} \qquad \mathit{if}\ \kappa(\gamma) \neq \emptyset,\ r\ \mathit{fresh}$$

An access $\alpha(r)$ can be executed if the capabilities associated with r include the action α, otherwise it generates a failure. A new resource r is created through the primitive $\mathtt{new}\,x : \gamma\,\mathtt{in}\,e$, which binds the scope of the *fresh* name r in e, and extends the capability environment Γ. For conditionals, we assume as given a total function $\mathcal{B}$ that evaluates the boolean guards. An expression $\varphi[e]$ can evolve to $\varphi[e']$, provided that the resulting history η' satisfies all the relevant instantiations $\varphi(r)$. A failure $\mathit{fail}_{\varphi(r)}$ occurs when, for some resource r, $\varphi(r)$ is violated by the extended history η'. Formally, let $\eta|_\varphi$ be the (longest) subsequence of η containing only the accessess $\alpha(r)$ such that $\alpha \in \ell(\varphi)$, and let R be the set of resources mentioned in $\eta|_\varphi$. We say that a history η obeys a policy φ, in symbols $\eta \models \varphi$, when $\eta|_\varphi \in \mathcal{L}(\varphi(r, R))$ for each $r \in R$. Similarly, we write $\eta \not\models_r \varphi$ when $\eta|_\varphi \notin \mathcal{L}(\varphi(r, R))$ for some $r \in R$.

Example 2. Let $\eta = \mathit{new}_{\mathsf{File}}(r_0)\,\mathit{open}(r_0)\,\mathit{write}(r_0)\,\mathit{close}(r_0)\,\mathit{open}(r_1)\,\mathit{read}(r_1)\,\mathit{connect}$, and let φ, φ' be the file usage and Chinese Wall policies of Ex. 1. Then, $\eta \models \varphi$, because $\eta|_\varphi = \mathit{open}(r_0)\mathit{write}(r_0)\mathit{close}(r_0)\mathit{open}(r_1)\mathit{read}(r_1)$, $R = \{r_0, r_1\}$ and so $\eta|_\varphi \in \mathcal{L}(\varphi(r_0, R)) \cap \mathcal{L}(\varphi(r_1, R))$. Instead, $\eta \not\models_{r_1} \varphi'$, because $\eta|_{\varphi'} = \mathit{new}_{\mathsf{File}}(r_0)\mathit{read}(r_1)\mathit{connect} \notin \mathcal{L}(\varphi'(r_1, R))$.

3 Static Semantics

We statically predict the histories generated by programs at run-time through a type and effect system, building upon [1,18]. The types extend those of the implicitly-typed λ-calculus, and the effects are *history expressions*, which over-approximate the aspects of the program behaviour that are relevant for resource usage. History expressions include the empty ε, events $\alpha(\rho)$, where $\rho \in \mathsf{Res} \cup \mathsf{Nam} \cup \{?\}$ ($n, n', \ldots \in \mathsf{Nam}$ are *names*, to be instantiated to fresh resources, and ? is a wildcard for all names), resource binding $\nu n.H$, sequencing $H \cdot H'$, non-deterministic choice $H + H'$, policy framing $\varphi[H]$, and recursion $\mu h.H$, where μ binds the occurrences of h in H.

Histories. The intended meaning of a history expression is a set of histories, extended to keep track of the policy framings through the special *framing events* $[_\varphi$ and $]_\varphi$, that stand respectively for opening and closing the scope of the policy φ. For example, an (extended) history $\alpha[_\varphi\alpha']_\varphi$ represents a computation that (i) generates an access α, (ii) enters the scope of a framing $\varphi[\cdots]$, (iii) generates α' within the scope of φ, and (iv) leaves the scope of φ. Note that histories with no framing events were enough to give the operational semantics of $\lambda^{[\,]}$, where the role of framing events is played by framed expressions. Hereafter, a history may end with the truncation marker $! \notin \mathsf{Act}$. The history $\eta!$ represents a prefix of a possibly non-terminating computation that generates the sequence of events η. We assume that histories are undistinguishable after truncation, i.e. $\eta!$ followed by η' equals to $\eta!$. A history η is *balanced* when either η is empty, or η is an access event, or $\eta =!$, or $\eta = [_\varphi\eta']_\varphi$ with η' balanced, or $\eta = \eta'\eta''$ with both η' and η'' balanced. For example, $\alpha[_\varphi\alpha'[_{\varphi'}\alpha'']_{\varphi'}]_\varphi$ is balanced, while $\alpha[_\varphi\alpha'[_{\varphi'}\alpha'']_\varphi$ is not. In what follows, we will only consider *well-formed* histories that are prefixes of some balanced history. Non well-formed histories, like e.g. $]_\varphi\alpha$, are not interesting, because they do not correspond to any $\lambda^{[\,]}$ computation.

The denotation of $H = (\nu n.\,\alpha(n)) \cdot (\nu n.\,\alpha(n))$ will contain *all* the histories $\alpha(r)\alpha(r')$ for $r \neq r'$. To this purpose we introduce *template histories* $\nabla\boldsymbol{n}.\,\eta$, where η may possibly contain events of the form $\alpha(n)$, and ∇ acts as a binder of the names in the finite set $\boldsymbol{n}$. Back to our example, the semantics of H is rendered by $\nabla n, n'.\,\alpha(n)\alpha(n')$. Bound names in template histories are α-convertible. We write η for $\nabla\emptyset.\,\eta$, and $\nabla\boldsymbol{n}.\,\nabla\boldsymbol{m}.\,\eta$ for $\nabla\boldsymbol{nm}.\,\eta$. A template history $\nabla\boldsymbol{n}.\,\eta$ is balanced when η is such. Let $\mathcal{H}, \mathcal{H}'$ range over sets of balanced template histories (BTH for short) and let $N(\eta)$ be the set of names occurring in η. The set $\varphi[\mathcal{H}]$ denotes $[_\varphi\mathcal{H}\,]_\varphi$. Also, we denote with $\mathcal{H}\mathcal{H}'$ the set:

$$\{\,\nabla\boldsymbol{nm}.\,\eta\eta' \mid \nabla\boldsymbol{n}.\,\eta \in \mathcal{H}, \nabla\boldsymbol{m}.\,\eta' \in \mathcal{H}',\ \boldsymbol{n} \cap N(\eta') = \emptyset = \boldsymbol{m} \cap N(\eta)\,\}$$

For example, since $\nabla n.\beta(n)$ can be α-converted to $\nabla m.\beta(m)$, then:

$$(\nabla n.\,\alpha(n))\,(\nabla n.\beta(n)) \;=\; (\nabla n.\,\alpha(n))(\nabla m.\beta(m)) = \nabla n, m.\,\alpha(n)\beta(m)$$

Semantics of history expressions. The *denotational semantics* $[\![H]\!]_\chi$ of history expressions maps H to a set $\mathcal{H}$ of BTH, in an environment χ that maps variables h to sets of BTH. We assume that a truncated history always denotes all its truncated prefixes, i.e., whenever $\nabla\boldsymbol{n}.\,\eta\eta'! \in \mathcal{H}$, then $\nabla\boldsymbol{n}.\,\eta! \in \mathcal{H}$.

$$[\![\varepsilon]\!]_\chi = \{\varepsilon\} \qquad [\![\alpha(r)]\!]_\chi = \{\alpha(r)\} \qquad [\![\alpha(?)]\!]_\chi = \{\alpha(?)\} \qquad [\![\nu n.\, H]\!]_\chi = \nabla n.\, [\![H]\!]_\chi$$
$$[\![\varphi[H]]\!]_\chi = \varphi[[\![H]\!]_\chi] \qquad [\![H \cdot H']\!]_\chi = [\![H]\!]_\chi\, [\![H']\!]_\chi \qquad [\![H + H']\!]_\chi = [\![H]\!]_\chi \cup [\![H']\!]_\chi$$
$$[\![h]\!]_\chi = \chi(h) \qquad [\![\mu h.H]\!]_\chi = \bigcup_{k>0} f^k(!) \quad \text{where } f(X) = [\![H]\!]_{\chi\{X/h\}}$$

Example 3. Let $H_0 = \mu h.\, \alpha \cdot h$, let $H_1 = \mu h.\, h \cdot \alpha$, and let $H_2 = \mu h.\, \nu n.\, \alpha(n) \cdot h$. Then, $[\![H_0]\!]_\emptyset = \alpha^*!$, i.e. H_0 generates histories with an arbitrary number of α, and never terminates. Instead, $[\![H_1]\!]_\emptyset = \{!\}$, i.e. H_1 loops forever, without generating events. The semantics of H_2 consists of all the histories $\nabla n_1, \ldots, n_k.\, \alpha(n_1) \cdots \alpha(n_k)!$, for $k > 0$. □

Equational theory. History expressions enjoy some equational properties. Intuitively, the equation $H = H'$ implies that $[\![H]\!]_\chi = [\![H']\!]_\chi$ for all χ. The operation $+$ is associative, commutative and idempotent; $\cdot$ is associative, has the identity ε, and distributes over $+$. The binders νn and μh can be rearranged, and μh can be introduced/eliminated when h does not occur free. The ν binder can be extruded when it does not bind free names (as usual, n is free in H if it is not in the scope of a νn, otherwise it is bound). Note that νn cannot be always lifted to the top-level: e.g., $\mu h.\nu n.H \neq \nu n.\mu h.H$ in general, because the leftmost history expression represents a loop that creates a new resource at each iteration, while in the rightmost one the new resource is created just before entering the loop. The last two rules allow for introduction/elimination of name binders, and for α-conversion. The set $N(H)$ denotes the names in H.

$$H + H = H \qquad (H + H') + H'' = H + (H' + H'') \qquad H + H' = H' + H$$
$$(H \cdot H') \cdot H'' = H \cdot (H' \cdot H'') \qquad \varepsilon \cdot H = H = H \cdot \varepsilon$$
$$H \cdot (H' + H'') = H \cdot H' + H \cdot H'' \qquad (H + H') \cdot H'' = H \cdot H'' + H' \cdot H''$$
$$\nu n.\, \nu n'.\, H = \nu n'.\nu n.H \qquad \mu h.\, \mu h'.\, H = \mu h'.\, \mu h.\, H \qquad \varphi[\nu n.\, H] = \nu n.\, \varphi[H]$$
$$\nu n.(H \cdot H') = (\nu n.H) \cdot H' \;\; \text{if } n \notin \mathit{fn}(H') \qquad \nu n.(H \cdot H') = H \cdot (\nu n.H') \;\; \text{if } n \notin \mathit{fn}(H)$$
$$\nu n.(H + H') = (\nu n.H) + H' \;\; \text{if } n \notin \mathit{fn}(H') \qquad \mu h.H = H\{\mu h.H/h\}$$
$$\nu n.H = H \quad \text{if } n \notin \mathit{fn}(H) \qquad \nu n.H = \nu m.H\{m/n\} \quad \text{(capture-avoiding)}$$

Note that we could replace the two constructs $\nu n.H$ and $\mu h.H$ with a single construct $\mu h.\nu n.H$, so defining a *standard form* for history expressions. For example, $\mu h.\, (\varepsilon + \nu n.\, \nu n'.\, \alpha(n) \cdot h \cdot \alpha(n'))$ can be rewritten as $\mu h.\nu n.\, \mu h'.\nu n'.\, \varepsilon + \alpha(n) \cdot h \cdot \alpha(n')$.

Unbound history expressions. Unbound history expressions are history expressions without ν-binders. Binding names in unbound history expressions is driven by the events *new*. For instance, in the unbound $H = \mathit{new}(n) \cdot \alpha(n) + \mathit{new}(m) \cdot \beta(m)$ the event $\mathit{new}(n)$ binds the name n, while $\mathit{new}(m)$ binds m, i.e. H is “bindified” to the history expression $(\nu n.\mathit{new}(n) \cdot \alpha(n)) + (\nu m.\mathit{new}(m) \cdot \beta(m))$.

Unbound history expressions have an equational theory $\approx$, which is a subtheory of the relation $=$ on history expressions. In particular, the last three equations (folding /unfolding, introduction/elimination of ν and α-conversion) are not permitted on unbound history expressions. Also, the right-distributivity of $\cdot$ over $+$ has the side condition $\mathit{fn}((H + H') \cdot H'') = \mathit{fn}(H \cdot H'' + H' \cdot H'')$. The definition of bound and free names in unbound history expressions slightly differs from the standard one: $bn(new(n)) = \{n\}$, $fn(new(n)) = \emptyset, fn(H \cdot H') = fn(H) \cup (fn(H') \setminus bn(H))$. We also define the set $rn(H) = \{ n \mid H \approx C(\mu h.\, \nu n.\, H'), h \in fv(H') \}$ of *recursive names* in H, where $C(\bullet)$ is a context.

To obtain a history expression from an unbound one, we will now introduce the *bindify* transformation ω. This transformation will insert the ν binders at the right points, provided that the introduced scopes of names do not interfere dangerously. For instance, ω is undefined on the unbound history expression $H = new(n) \cdot new(n) \cdot \alpha(n)$, because it is unclear whether the action α is performed on the resource n created first or the second *new*). It is then not sound choosing the above H to approximate the histories of e.g. $e = \texttt{new}\ x\ \texttt{in}\ \texttt{new}\ y\ \texttt{in}\ \alpha(y)$, because $new(r)new(r')\alpha(r')$ is not represented by H.

$$\omega(\alpha(\rho)) = \alpha(\rho) \quad \text{if } \alpha \neq new \qquad \omega(new(n)) = \nu n.\, new(n) \qquad \omega(h) = h$$

$$\omega(H \cdot H') = \omega(H) \odot \omega(H') \quad \text{if } bn(H) \cap bn(H') = \emptyset$$

$$\omega(H + H') = \omega(H) \oplus \omega(H') \qquad \omega(\mu h.\, H) = \mu h.\, \omega(H) \qquad \omega(\varphi[H]) = \varphi[\omega(H)]$$

$$H \odot H' = \begin{cases} \nu n.\, (\bar{H} \odot H') & \text{if } H \approx \nu n.\, \bar{H}, n \notin rn(H) \\ H \cdot H' & \text{if } rn(H) \cap fn(H') = \emptyset \end{cases}$$

$$H \oplus H' = \begin{cases} \nu n.\, (\bar{H} \oplus \bar{H}') & \text{if } H \approx \nu n.\, \bar{H}, H' \approx \nu n.\, \bar{H}', n \notin rn(H + H') \\ H + H' & \text{otherwise} \end{cases}$$

The event $new(n)$ drives the introduction of the actual binder νn in history expressions: the scope of n in H is entered just before the $new(n)$, and it is left as soon as needed no longer, e.g. $new(n) \cdot (\mu h.\, \varepsilon + new(n') \cdot \alpha(n) \cdot \alpha(n') \cdot h) \cdot \alpha(n)$ is bindified into $(\nu n.\, new(n) \cdot (\mu h.\, \varepsilon + (\nu n'.\, new(n') \cdot \alpha(n) \cdot \alpha(n') \cdot h)) \cdot \alpha(n))$. Instead, ω is not defined on $(\mu h.\, \varepsilon + new(n) \cdot h) \cdot \alpha(n)$, because the name n accessed through α could be any name generated by the *new* inside the loop.

Type & Effect system. We define below a type and effect system for $\lambda^{[\,]}$. Effects H are unbound history expressions. Types τ comprise the unit $\mathbf{1}$, sets $R \subseteq (\mathsf{Res} \cup \mathsf{Nam}) \times 2^{\mathsf{Act}}$, and arrows $\tau \xrightarrow{H} \tau$. For instance, a resource r with capabilities γ has the singleton type $\{(r, \gamma)\}$ (we omit the capabilities when irrelevant). Type environments have the form $\Delta;\ \xi : \tau$ where $\xi \in \mathsf{Var} \cup \mathsf{Res}$ is not already in $dom(\Delta)$. A typing judgment $\Delta \vdash e : \tau \triangleright H$ means that, in a type environment Δ, the expression e evaluates to a value of type τ, and produces a history belonging to the effect H. In the functional type $\tau \xrightarrow{H} \tau'$, H describes the *latent* effect associated with an abstraction, i.e. one of the histories represented by H will be generated when such an abstraction is applied to a value.

To keep our type system as simple as possible, and still allowing to deal with the *escape* of freshly created resources, we avoid to explicitly introduce binders on types. Instead, we have raised the action *new* to the key role of an implicit binder. E.g., the type $\mathbf{1} \xrightarrow{new(n) \cdot \alpha(n)} \{n\}$ is for a function that generates a fresh resource upon each invocation, accesses it through the action α, and then returns it. The bindify transformation, together with some side-conditions on the typing rules, ensure that the typing derivation do not exploit the absence of explicit binders to identify names that should be kept distinct. As a global invariant on typing derivations, we require that in a type & effect $\tau \triangleright H$, the bound names of H are disjoint from those of τ (i.e. the bound names in latent effects).

The relation $\sqsubseteq$ is used to define subtypes and subeffects. Roughly, $H \sqsubseteq H'$ means that $[\![\omega(H)]\!] \subseteq [\![\omega(H')]\!]$ — when both $\omega(H)$ and $\omega(H')$ are defined and closed. For instance, $C(H) \sqsubseteq C(H + H')$, in any context $C(H)$. The relation $\sqsubseteq$ comprises a version of folding/unfolding that creates *fresh* names upon unfolding (so not to prevent from bindification). For example, if $H = \mu h.\, new(n) \cdot \alpha(n) \cdot h$, then $new(n') \cdot \alpha(n') \cdot H \sqsubseteq H$. Subtypes are defined as usual, contravariant in the argument type and covariant in the return type (the latent effect is invariant).

$$\mathbf{1} \sqsubseteq \mathbf{1}\, R \sqsubseteq R' \ \text{ if } R \subseteq R' \ \{(n,\gamma)\} \cup R \sqsubseteq \{(?,\gamma)\} \cup R \quad \tau_0 \xrightarrow{H} \tau_0' \sqsubseteq \tau_1 \xrightarrow{H} \tau_1' \ \text{ if } \begin{array}{l}\tau_0 \sqsupseteq \tau_1\\ \tau_0' \sqsubseteq \tau_1'\end{array}$$

$$H \sqsubseteq H' \text{ if } H \approx H' \ H \sqsubseteq H + H' \ H\theta\{\mu h.\, H/h\} \sqsubseteq \mu h.\, H \ \ (\theta \text{ maps } bn(H) \text{ into fresh names})$$

$$C(H) \sqsubseteq C(H') \quad \text{if } H \sqsubseteq H',\ (bn(H) \setminus bn(H')) \cap N(C) = \emptyset,\ fn(C(H)) \subseteq fn(C(H'))$$

We now introduce the type and effect system for $\lambda^{[\,]}$. An access $\alpha(e)$ has type $\mathbf{1}$, provided that the type of e is a set of resources R, and each resource in R has the capability α. The effect of $\alpha(e)$ can be any of the accesses $\alpha(\rho)$ for $(\rho,\gamma) \in R$. The effects in the rule for application are concatenated according to the evaluation order of the call-by-value semantics (function, argument, latent effect). The side condition ensures that the free names in the effect of the argument are not captured by the effect of the function. The actual effect of an abstraction is the empty history expression, while its latent effect is equal to the actual effect of the function body. Note that the rule for abstraction constrains the premise to equate the actual and latent effects. A resource creation generates a fresh name, and binds it in the effect through the event *new*. The last two rules allow for *weakening* of types/effects and α-conversion, respectively. The side condition † on weakening requires that names created through subeffecting are disjoint from the names in the type (e.g. the weakening $1 \xrightarrow{\alpha(n)} 1 \rhd \varepsilon \sqsubseteq 1 \xrightarrow{\alpha(n)} 1 \rhd \varepsilon + new(n)$ is not permitted, because the *new* event would capture the free n in the type). Although α-conversion is not permitted on unbound history expressions, we allow it on types, e.g. $\mathbf{1} \xrightarrow{new(n)\cdot\alpha(n)} \{n\}$ can be α-converted to $\mathbf{1} \xrightarrow{new(m)\cdot\alpha(m)} \{m\}$.

$$\frac{\Delta \vdash e : R \rhd H \quad \forall(\rho,\gamma) \in R.\ \alpha \in \gamma}{\Delta \vdash \alpha(e) : \mathbf{1} \rhd H \cdot \Sigma_{(\rho,\gamma)\in R}\ \alpha(\rho)} \qquad \frac{\xi : \tau \in \Delta}{\Delta \vdash \xi : \tau \rhd \varepsilon} \qquad \frac{\Delta; x : \tau; z : \tau \xrightarrow{H} \tau' \vdash e : \tau' \rhd H}{\Delta \vdash \lambda_z x.e : \tau \xrightarrow{H} \tau' \rhd \varepsilon}$$

$$\frac{\Delta \vdash e : \tau \xrightarrow{H''} \tau' \rhd H \quad \Delta \vdash e' : \tau \rhd H'}{\Delta \vdash e\,e' : \tau' \rhd H \cdot H' \cdot H''}\ bn(H) \cap fn(H') = \emptyset \qquad \frac{\Delta \vdash e : \tau \rhd H}{\Delta \vdash \varphi[e] : \tau \rhd \varphi[H]}$$

$$\frac{\Delta; x : \{(n,\gamma)\} \vdash e : \tau \rhd H \quad \kappa(\gamma) \neq \emptyset}{\Delta, \vdash \mathtt{new}\, x : \gamma\, \mathtt{in}\, e : \tau \rhd new_{\kappa(\gamma)}(n) \cdot H}\ \begin{array}{l} n \notin \Delta \\ n \notin bn(\tau) \\ n \notin bn(H)\end{array} \qquad \frac{\Delta \vdash e : \tau \rhd H \quad \Delta \vdash e' : \tau \rhd H}{\Delta \vdash \mathtt{if}\, b\, \mathtt{then}\, e\, \mathtt{else}\, e' : \tau \rhd H}$$

$$\frac{\Delta \vdash e : \tau \rhd H}{\Delta \vdash e : \tau' \rhd H'}\ \begin{array}{l}\tau \sqsubseteq \tau' \\ H \sqsubseteq H'\end{array}\dagger \qquad \frac{\Delta \vdash e : \tau \xrightarrow{H} \tau' \rhd H'}{\Delta \vdash e : \tau \xrightarrow{H\theta} \tau'\theta \rhd H'}\ \begin{array}{l} dom(\theta) \cap bn(H') = \emptyset \\ \theta \text{ capture-avoiding}\end{array}$$

$$\dagger \ \ (bn(H) \setminus bn(H')) \cap N(\tau) = \emptyset = (bn(H') \setminus bn(H)) \cap N(\tau')$$

Example 4. We have the following typing judgements (see App. **??** for details):

$$\begin{aligned}
&\emptyset \vdash \alpha(\mathtt{new}\, x : \gamma\, \mathtt{in}\, \mathtt{new}\, y : \gamma'\, \mathtt{in}\, \mathtt{if}\, b\, \mathtt{then}\, x\, \mathtt{else}\, y) : \mathbf{1}\\
&\quad \triangleright H_1 = \mathit{new}_{\kappa(\gamma)}(n) \cdot \mathit{new}_{\kappa(\gamma')}(n') \cdot (\alpha(n) + \alpha(n')) \quad \text{if } \alpha \in \gamma \cap \gamma'\\
&\quad \omega(H_1) = \nu n.\ \nu n'.\ \mathit{new}_{\kappa(\gamma)}(n) \cdot \mathit{new}_{\kappa(\gamma')}(n') \cdot (\alpha(n) + \alpha(n'))\\
&\emptyset \vdash \mathtt{let}\, f = (\lambda x.\, \mathtt{new}\, n : \gamma\, \mathtt{in}\, \alpha(n); n)\, \mathtt{in}\, \alpha'(f*; f*) : \mathbf{1}\\
&\quad \triangleright H_2 = \mathit{new}_{\kappa(\gamma)}(n) \cdot \alpha(n) \cdot \mathit{new}_{\kappa(\gamma)}(n') \cdot \alpha(n') \cdot \alpha'(n') \quad \text{if } \alpha, \alpha' \in \gamma\\
&\quad \omega(H_2) = (\nu n.\ \mathit{new}_{\kappa(\gamma)}(n) \cdot \alpha(n)) \cdot (\nu n'.\ \mathit{new}_{\kappa(\gamma)}(n') \cdot \alpha(n') \cdot \alpha'(n'))\\
&\emptyset \vdash \mathtt{let}\, g = (\mathtt{new}\, n : \gamma\, \mathtt{in}\, \lambda x.\, \alpha(n); n)\, \mathtt{in}\, \alpha'(g*; g*) : \mathbf{1}\\
&\quad \triangleright H_3 = \mathit{new}_{\kappa(\gamma)}(n) \cdot \alpha(n) \cdot \alpha(n) \cdot \alpha'(n) \quad \text{if } \alpha, \alpha' \in \gamma\\
&\quad \omega(H_3) = \nu n.\ \mathit{new}_{\kappa(\gamma)}(n) \cdot \alpha(n) \cdot \alpha(n) \cdot \alpha'(n)\\
&\emptyset \vdash (\lambda_z x.\, \mathtt{new}\, n : \gamma\, \mathtt{in}\, \mathtt{if}\, b\, \mathtt{then}\, \alpha(n)\, \mathtt{else}\, \alpha'(n); zx) * : \mathbf{1}\\
&\quad \triangleright H_4 = \mu h.\ \mathit{new}_{\kappa(\gamma)}(n) \cdot (\alpha(n) + \alpha'(n) \cdot h) \quad \text{if } \alpha, \alpha' \in \gamma\\
&\quad \omega(H_4) = \mu h.\ \nu n.\ \mathit{new}_{\kappa(\gamma)}(n) \cdot (\alpha(n) + \alpha'(n) \cdot h)\\
&\emptyset \vdash \alpha((\lambda_z x.\, \mathtt{new}\, n : \gamma\, \mathtt{in}\, \mathtt{if}\, b\, \mathtt{then}\, n\, \mathtt{else}\, \alpha'(n); zx) *) : \mathbf{1}\\
&\quad \triangleright H_5 = (\mu h.\ \mathit{new}_{\kappa(\gamma)}(n) \cdot (\varepsilon + \alpha'(n) \cdot h)) \cdot \alpha(?) \quad \text{if } \alpha, \alpha' \in \gamma\\
&\quad \omega(H_5) = (\mu h.\ \nu n.\ \mathit{new}_{\kappa(\gamma)}(n) \cdot (\varepsilon + \alpha'(n) \cdot h)) \cdot \alpha(?)
\end{aligned}$$

Type safety. Let $\eta = \beta_1\beta_2 \cdots$ be a history. We define $\eta^\flat$ as the history obtained from η by erasing all the framing events, and η^∂ as the set of all the prefixes of η, without !. E.g., $(\alpha\alpha'[_\varphi\alpha'']_\varphi^\flat)^\partial = (\alpha\alpha'\alpha'')^\partial = \{\varepsilon, \alpha, \alpha\alpha', \alpha\alpha'\alpha''\}$. Let Δ_0 comprise $r : \{(r, \gamma)\}$ whenever $(r, \gamma) \in \Gamma_0$. Our type and effect system correctly approximates the actual run-time histories; as usual, precision is lost with conditionals and with recursive functions. Also, you may lose the identity of names exported by recursive functions (see H_5 above).

Theorem 1. *Let $\Delta_0 \vdash e : \tau \triangleright H$, $\omega(H)$ closed, and $\varepsilon, \Gamma_0, e \rightarrow^* \eta, \Gamma, e'$. Then, $\eta \in [\![\omega(H)]\!]^{\flat\partial}$.*

A *valid* history does not violate any resource usage constraint. Consider the security policy φ' of Ex. 1: the history $\eta = \mathit{open}(r)\mathit{read}(r)\varphi'[\mathit{connect}]$ is *not* valid, because the *connect* occurs within a framing enforcing φ', and $\mathit{open}(r)\mathit{read}(r)\mathit{connect}$ does not obey φ'. To formally define validity, we introduce the notion of safe-sets. The history η above has one safe-set: $\varphi'[\{\mathit{open}(r)\mathit{read}(r), \mathit{open}(r)\mathit{read}(r)\mathit{connect}\}]$, meaning that the scope of the framing $\varphi'[\cdots]$ encloses the two histories within the curly brackets. To have a short, inductive definition of the safe-sets $S(\eta)$ we first balance all the framings of η, e.g. $[_\varphi\alpha$ becomes $[_\varphi\alpha]_\varphi = \varphi[\alpha]$. Then, we define:

$$S(\varepsilon) = \emptyset \qquad S(\eta\,\alpha(r)) = S(\eta) \qquad S(\eta_0\,\varphi[\eta_1]) = S(\eta_0\,\eta_1) \cup \varphi[\eta_0^\flat\,(\eta_1^\flat)^\partial]$$

A history η is *valid* when $\varphi[\mathcal{H}] \in S(\eta)$ implies $\eta' \models \varphi$ for all $\eta' \in \mathcal{H}$. Note that past events cannot be hidden, because policy framings can always inspect the whole past history. For example, a history $\alpha_1\varphi[\alpha_2]\alpha_3$ is valid when $\alpha_1 \models \varphi$ and $\alpha_1\alpha_2 \models \varphi$ (even if α_1 is outside of the safety framing), while $\alpha_1\alpha_2\alpha_3$ is not required to satisfy φ any longer. A history expression H is *valid* when all the histories in $[\![H]\!]$ are such. Our type and effect system guarantees the following type safety property.

Theorem 2 (Type Safety). *Let $\Delta_0 \vdash e : \tau \rhd H$. Then:*

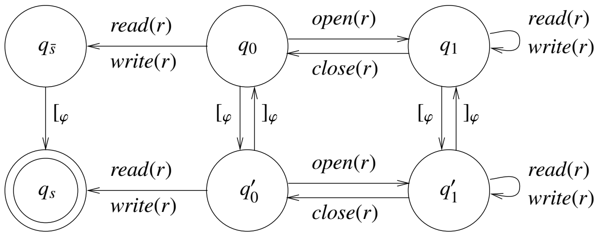
(i) $\varepsilon, \Gamma_0, e \not\to^* \eta, \Gamma, \mathit{fail}_\kappa$ *(e respects capabilities)*
(ii) if $\omega(H)$ is valid, then $\varepsilon, \Gamma_0, e \not\to^* \eta, \Gamma, \mathit{fail}_{\varphi(r)}$ *(e respects the prescribed usages).*

4 Static Verification

We verify the validity of history expressions by model-checking Basic Process Algebras (BPAs) with finite state automata. The standard decision procedure for verifying that a BPA process p satisfies a regular property φ amounts to constructing the pushdown automaton for p and the automaton for $\neg\varphi$. Then, the property holds if the (context-free) language accepted by the conjunction of the above, which is still a pushdown automaton, is empty. This problem is known to be decidable, and several algorithms and tools show this approach feasible [9].

A first problem, solved in [1], is that the arbitrary nesting of framings makes validity of histories a non-regular, property. For example, $[\![\mu h.\, \alpha + h \cdot h + \varphi[h]]\!]$ denotes histories with unbounded pairs of balanced $[_\varphi$ and $]_\varphi$, so it is a context-free, non-regular language. In [1] we defined a *regularization* $\downarrow$ of history expressions such that H is valid if and only if each $\eta \in [\![H \downarrow]\!]$ satisfies $\varphi_{[\,]}$. The formula $\varphi_{[\,]}$ is defined through the automaton $A_{\varphi_{[\,]}(x)}$, a smooth transformation of $A_{\varphi(x)}$ taking into account entering/leaving the frame $\varphi[\cdots]$. Hereafter, we assume that history expressions have been regularized (a simple extension of [1] suffices).

Example 5. The framed version of the file usage policy $\varphi(r)$ of Ex. 1 is described by the automaton $A_{\varphi_{[\,]}(r)}$ below. The top (resp. bottom) layer models being outside (resp. inside) the scope of φ. All states have self-loops (not displayed in the figure) for the irrelevant events. For instance, the history $[_\varphi \mathit{open}(r)\, \mathit{close}(r)\, \mathit{read}(r)$ is not accepted.



A second problem, solved here, is that now history expressions may create new names, while BPAs cannot handle fresh names. Verifying validity would thus need to check an unbounded set of policies $\varphi(r)$, e.g. the histories denoted by $H = \varphi[\mu h.\, \varepsilon + \nu n.\alpha(n) \cdot h]$ must satisfy all the policies $\varphi(r_0), \varphi(r_1), \ldots$ for each fresh resource created within the loop. Thus, we would have to intersect an infinite number of finite state automata to verify H valid, which is unfeasible. As a first contribution, we extract from a history expression H a BPA and a *finite* set of usage policies, that suffice for verifying H valid. The intuition is that a new resource r created under a μh lives for a *single* iteration of the loop, and in the other iterations we do not care of the actual resources generated (therefore we denote with _ these "dummy" resources). Formally, the function $\mathcal{M}(H)_\Theta$

takes as input a history expression H and a function Θ from history variables h to BPA variables X. Its output is a guarded BPA process p, a finite set of definitions Δ for BPA variables, and a finite set of usage policies Π. Without loss of generality, we assume that H is regularized, and that its variables are all distinct.

$$\mathcal{M}(\varepsilon)_\Theta = \langle \varepsilon, \emptyset, \emptyset \rangle \qquad \mathcal{M}(h)_\Theta = \langle \Theta(h), \emptyset, \emptyset \rangle$$

$$\mathcal{M}(\alpha(\zeta))_\Theta = \langle \alpha(\zeta), \emptyset, \{ \varphi(\zeta) \mid \alpha \in \ell(\varphi) \wedge \kappa(\{\alpha\}) = \kappa(\varphi)\} \rangle \qquad \zeta \in \mathsf{Res} \cup \{?\}$$

$$\mathcal{M}(H_0 \cdot H_1)_\Theta = \langle p_0 \cdot p_1, \Delta_0 \cup \Delta_1, \Pi_0 \cup \Pi_1 \rangle, \text{ where } \mathcal{M}(H_i)_\Theta = \langle p_i, \Delta_i, \Pi_i \rangle$$

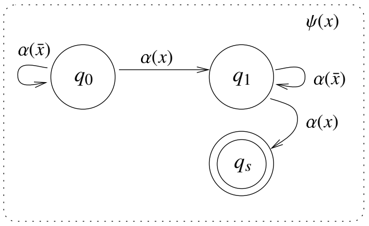
$$\mathcal{M}(H_0 + H_1)_\Theta = \langle p_0 + p_1, \Delta_0 \cup \Delta_1, \Pi_0 \cup \Pi_1 \rangle, \text{ where } \mathcal{M}(H_i)_\Theta = \langle p_i, \Delta_i, \Pi_i \rangle$$

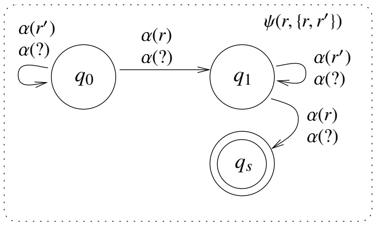
$$\mathcal{M}(\varphi[H])_\Theta = \langle [_\varphi \cdot p \cdot]_\varphi, \Delta, \Pi \rangle, \text{ where } \mathcal{M}(H)_\Theta = \langle p, \Delta, \Pi \rangle$$

Access events, variables, concatenation and choice are mapped into the corresponding BPA counterparts. An expression $\varphi[H]$ is mapped to the BPA for H, surrounded by the opening and closing events of the φ-framing. The tricky case is that of recursion and new name generation, not shown above (the items Δ and Θ will indeed be populated and exploited in the recursive case). We shall outline, with the help of the following examples, the stages that lead to the correct definition of $\mathcal{M}(H)$ in such cases.

The component Π in $\mathcal{M}(H)$ contains the set of all the usage policies that are needed in verifying the validity of H. Let R be the (finite) set of resources in the BPA of $\mathcal{M}(H)$. For each event $\alpha(r)$ contained in $\mathcal{M}(H)$ (for $r \neq \_$), the set Π comprises all the policies $\varphi(r, R)$ such that the kind of φ is consistent with that of α (i.e. $\kappa(\varphi) = \kappa(\{\alpha\})$, and $A_{\varphi(x)}$ has some edge labelled with α (i.e. $\alpha \in \ell(\varphi)$).

Example 6. Consider the history expression $H = \nu n. \nu n'. \alpha(n) \cdot \alpha(n') \cdot \alpha(?)$. Then, a sound BPA for $\mathcal{M}(H)$ is $\alpha(r) \cdot \alpha(r') \cdot \alpha(?)$ where r and r' are two distinct resources. For instance, consider a policy $\psi(x)$ saying that the action α cannot be performed twice on the same resource (left-hand side of the figure below).

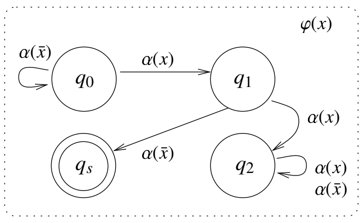


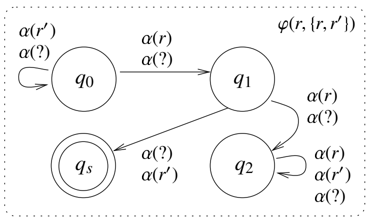


Clearly, the above-mentioned BPA violates $\psi(r)$ (right-hand side of the figure above), consistently with the fact that the wildcard ? represents any resource, including r (e.g. $\alpha(r)\alpha(r')\alpha(r) \in [\![H]\!]$ violates ψ). Instead, the BPA above correctly respects a policy ψ' requiring that α is not executed *three* times on the same resource. So, $\mathcal{M}(H)$ correctly reflects violations and obeyance to the relevant policies. In this sense we can say that $\mathcal{M}(H)$ is sound and complete (see Theorem 4 below).

Example 7. Let $H = \mu h. (\varepsilon + \nu n. \alpha(n) \cdot h)$. A first, naïve solution to obtain $\mathcal{M}(H)$ would be that of picking out a resource r, and then modelling the BPA as a recursive process, where at each step the event $\alpha(r)$ is executed (left-hand side of the picture below).

$$\begin{tikzcd} & \varepsilon & \\ X \arrow[ur] \arrow[rr] & & \alpha(r) \cdot X \arrow[ll, bend right=60] \end{tikzcd}$$

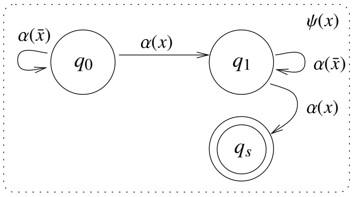


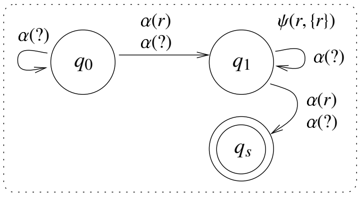


However, this solution is *not* sound. To see why, consider a policy $\varphi(x)$ (modelled by the template usage automaton $A_{\varphi(x)}$ central in the picture above), saying that, for each resource x, the first event $\alpha(x)$ is necessarily followed by another $\alpha(x)$. Clearly, H violates the policy (e.g. $\eta = \alpha(r)\alpha(r') \in [\![H]\!]$, and $\eta \not\models \varphi(r, \{r, r'\})$, see the instantiated automaton on the right-hand side of the picture above). Instead, the BPA does *not* violate the policy, and so it is unsound.

As a second try, consider a slight variation of the BPA above (left-hand side of the picture below), where, at each step, one among the events $\alpha(r)$ or $\alpha(r')$ can be executed. This BPA correctly violates $\varphi(r, \{r, r'\})$ above.

$$\begin{tikzcd} & \varepsilon & \\ X \arrow[ur] \arrow[rr] \arrow[drr] & & \alpha(r) \cdot X \arrow[ll, bend right=60] \\ & & \alpha(r') \cdot X \arrow[ull, bend left=60] \end{tikzcd}$$





Although this solution is sound, it is not complete. Consider for instance the policy $\psi(x)$ saying that the action α cannot be performed twice on the same resource x (see the template automaton in the center of the figure above). Although H obeys ψ, the BPA does not: indeed, the BPA trace $\alpha(r)\alpha(r)$ violates $\psi(r, \{r\})$ (the instantiated automaton $A_{\psi(r,\{r\})}$ is depicted in the right-hand side of the picture above).

To recover completeness, we must ensure that the BPA does not execute the same event $\alpha(r)$ twice. To do that, the BPA is composed of two loops: the first loop executes $\alpha(\_)$ on a dummy resouce $\_$, then, the BPA executes $\alpha(r)$ once, and finally the second loop executes $\alpha(\_)$ (see the left part in the figure below). Template automata are only instantiated with the resource r (and not with $\_$).

$$\begin{tikzcd} & \varepsilon & & & \\ X \arrow[ur] \arrow[rr] \arrow[drr] & & \alpha(\_) \cdot X \arrow[ll, bend right=60] & \varepsilon & \\ & & \alpha(r) \cdot X' \arrow[ur] \arrow[rr] & & \alpha(\_) \cdot X' \arrow[ll, bend right=60] \end{tikzcd}$$

This solution is both sound and complete. For soundness, the BPA correctly violates $\varphi(r, \{r, \_\})$, e.g. with the trace $\alpha(r)\alpha(\_)$. For completeness, the BPA respects $\psi(r, \{r, \_\})$ (note that here it is important that $\psi(\_)$ is not considered).

More generally, when H is a loop, then $\mathcal{M}(H)$ is a BPA which (i) runs the loop an arbitrary number of times, substituting _ for the actual freshly generated resources, (ii) runs a *single* iteration of the loop, using a unique instantiation of names, and (iii), runs the same loop as (i). Special care is needed to avoid replication of the same names, e.g. in case of nested recursion.

Example 8. Let φ_k require that no more than k files can be created, i.e. $A_{\varphi_k(x)}$ has states $q_0, \ldots, q_k, q_s$ and edges $q_i \xrightarrow{new_{\mathsf{File}}(\zeta)} q_{i+1}$ for $i \in 0..k-1$ and $q_k \xrightarrow{new_{\mathsf{File}}(\zeta)} q_s$, for $\zeta \in \{x, \bar{x}\}$. Let $H = \varphi[\varphi_k[\mu h.\, \varepsilon + \nu n.\, new_{\mathsf{File}}(n) \cdot open(n) \cdot read(n) \cdot close(n) \cdot h]]$, where φ is the file usage policy of Ex.1. Then, $\mathcal{M}(H)_\emptyset = \langle [_\varphi \cdot [_{\varphi_k} \cdot X \cdot]_{\varphi_k} \cdot]_\varphi,\, \Delta,\, \{\varphi(r), \varphi_k(r)\}\rangle$, where Δ comprises the following definitions (we abbreviate new_{File} with α_n, *read* with α_r, etc.):

$$X \triangleq \varepsilon + \Big(\alpha_n(\_) \cdot \alpha_o(\_) \cdot \alpha_r(\_) \cdot \alpha_c(\_) \cdot X\Big) + \Big(\alpha_n(r) \cdot \alpha_o(r) \cdot \alpha_r(r) \cdot \alpha_c(r) \cdot X'\Big)$$
$$X' \triangleq \varepsilon + \alpha_n(\_) \cdot \alpha_o(\_) \cdot \alpha_r(\_) \cdot \alpha_c(\_) \cdot X'$$

Note that each computation of the BPA $\langle p, \Delta\rangle$ obeys the file usage policy $\varphi(r, \{r, \_\})$, while there exist computations that violate $\varphi_k(r, \{r, \_\})$.

We now state the correspondence between history expressions and BPAs. The prefixes of the histories generated by a history expression H (i.e. $[\![H]\!]^\partial$) are all and only the strings that label the computations of the extracted BPA, after a renaming of resources. A special case is that of ?, which may stand for any resource. To deal with it, we define the "up-to-?" relation $=_?$ between histories: $=_?$ is the least reflexive relation such that, for any r, $\eta_0\alpha(r)\eta_1 =_? \eta'_0\alpha(?)\eta'_1$ whenever $\eta_0 =_? \eta'_0$ and $\eta_1 =_? \eta'_1$.

Theorem 3. *Let $\mathcal{M}(H)_\emptyset = \langle p, \Delta, \Pi\rangle$. For each $\eta \in [\![H]\!]^\partial$, there exist $\eta' \in [\![p, \Delta]\!]$ and a substitution ξ from* **Res** *to* **Res** $\cup \{\_\}$ *such that $\xi(\eta) =_? \eta'$ ($[\![p, \Delta]\!]$ is the trace semantics). Conversely, for each $\eta \in [\![p, \Delta]\!]$, there exists some ξ and $\eta' \in [\![H]\!]^\partial$ such that $\xi(\eta') =_? \eta$.*

Example 9. Let $H = \mu h.\, \nu n.\, \varepsilon + \alpha(n) \cdot h$. Then, the BPA extracted from H is $\langle X, \Delta\rangle$, where $\Delta = \{X \triangleq \varepsilon + \alpha(\_) \cdot X + \alpha(r) \cdot X', X' \triangleq \varepsilon + \alpha(\_) \cdot X'\}$. Let $\eta = \alpha(r_0)\alpha(r_1)\alpha(r_2) \in [\![H]\!]^\partial$, and let $\eta' = \alpha(\_)\alpha(r)\alpha(\_)$ be a string in $[\![X, \Delta]\!]$. If $\xi = \{\_/r_0, r/r_1, \_/r_2\}$, then $\xi(\eta) = \eta'$.

The theorem above enables us to verify the validity of a (regularized) history expression H by extracting $\mathcal{M}(H)_\emptyset = \langle p, \Delta, \Pi\rangle$ and then model-checking the BPA $\langle p, \Delta\rangle$ against the finite set of policies Π. Indeed, a valid computation of the BPA is recognized by the intersection of the finite state automata $A_{\varphi_{[\,]}(r)}$, for all $\varphi(r)$ in Π. Together with Theorem 2, a $\lambda^{[\,]}$ expression *never goes wrong* if its effect is checked valid.

Theorem 4. *Let $\mathcal{M}(H)_\emptyset = \langle p, \Delta, \Pi\rangle$. Then, H is valid iff $[\![p, \Delta]\!] \models \bigwedge \{\, \varphi_{[\,]}(r) \mid \varphi(r) \in \Pi \,\}$.*

5 Related Work and Conclusions

We proposed a novel approach to the resource usage problem, within an extension of the λ-calculus that features creation/access to resources, and regular usage policies with a local scope. To efficiently enforce policies, we have exploited a two-step static analysis.

We defined a type and effect system that over-approximates the run-time behaviour of a program as a history expression. In spite of the augmented flexibility given by the nesting of policy scopes and by resource creation, we transformed history expressions so that model checking their validity is decidable. Our technique manages to represent the generation of an unbounded number of resources in a finitary manner. Yet, we do not lose the possibility of verifying interesting properties of programs (see Ex. 9). When a history expression is valid, we can safely dispose the execution monitor. Otherwise, the soft-typing approach in [2] allows for substituting local checks for local policies, thus making the dynamic control of accesses efficient. Although our policies can always inspect the whole past history, one can easily limit the scope from the side of the past: it suffices to mark in the history the point in time β_φ from which checking a policy φ has to start; the corresponding automaton discards then all the events before β_φ. Type inference has not been considered here, but we do not see major obstacles in extending [18] to our case. Another research direction consists in extending $\lambda^{[]}$ in a distributed setting, to study secure discovery and composition of services [3].

Many authors [7,8,14,21] mixed static and dynamic techniques to transform programs and make them obey a given global policy. Colcombet and Fradet [7] abstracted a program into an instrumented control flow graph, then minimized and converted back to a program that is guaranteed to abort just before violating the property. Marriot, Stuckey and Sulzmann [14] over-approximated the run-time behaviour of a program through a context-free grammar. A finite-state automaton models the permitted resource usages. If the language generated by the grammar is not included in the language accepted by automaton, the program is instrumented with the local checks and the tracking operations needed to make it obey the policy, similarly to [2]. Our programming model allows for local policies and access events parametrized over dynamically created resources, while [7,8,14,21] only consider global policies and no parametrized events.

Igarashi and Kobayashi [12] extended the λ-calculus with primitives for creating and accessing resources, and for defining their permitted usage patterns. An execution is resource-safe when the possible patterns are within the permitted ones. A type system guarantees well-typed expressions to be resource-safe. However, they do not present any algorithm to effectively check whether the inferred usages conform to the permitted ones. Instead, here we provided $\lambda^{[]}$ with a static verification technique; clearly, also [12] might be amenable to static verification, if one restricts the language of permitted usages to a decidable subset. Furthermore, the policies of [12] can only speak about the usage of *single* resources, while ours can span over many resources, of different kinds, e.g. the Chinese Wall of Ex. 1.

Skalka and Smith [18] proposed a λ-calculus with local checks that enforce linear μ-calculus properties [6,13] on the past history. A type and effect system approximates the possible run-time histories, whose validity can be statically verified by model checking μ-calculus formulae over Basic Process Algebras [5,9]. Compared to [18], we feature dynamic resource creation, and local policies instead of local checks. On a more concrete level, the same ideas are applied in [19] to define a type and effect system for an extension of Featherweight Java, featuring histories and security checks.

Walker [22] mixed static and dynamic techniques with proof-carrying code [15]. Properties are specified by *security automata* [4,17]. When a security-unaware program is compiled, a centralized policy tells where to insert local checks, in order to obtain provably-secure compiled code. An optimization phase follows: whenever a check is removed, it is replaced by a proof that the optimized code is still safe. Before executing a piece of code, a certified verifier ensures that it respects the centralized security policy. Thus, compilers are no longer required to belong to the trusted computing base.

Acknowledgments. We thank Luís Caires and the anonymous referees for their comments. Research partially supported by EU-FETPI Global Computing Project IST-2005-16004 SENSORIA (Software Engineering for Service-Oriented Overlay Computers).

References

1. M. Bartoletti, P. Degano, and G. L. Ferrari. History based access control with local policies. *Proc. Fossacs (LNCS 3441)*, 2005.
2. M. Bartoletti, P. Degano, and G. L. Ferrari. Checking risky events is enough for local policies. Proc. 9th ICTCS (LNCS 3701), 2005.
3. M. Bartoletti, P. Degano, and G. L. Ferrari. Types and effects for secure service orchestration. *Proc. CSFW*, 2006.
4. L. Bauer, J. Ligatti, and D. Walker. More enforceable security policies. *Proc. FCS*, 2002.
5. J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37:77–121, 1985.
6. J. C. Bradfield. On the expressivity of the modal μ-calculus. *Proc. of Int. Symp. on Theoretical Aspects of Computer Science*, 1996.
7. T. Colcombet and P. Fradet. Enforcing trace properties by program transformation. *Proc. POPL*, 2000.
8. Ú. Erlingsson and F.B. Schneider. SASI enforcement of security policies: a retrospective. *Proc. of 7th New Security Paradigms Workshop*, 1999.
9. J. Esparza. On the decidability of model checking for several μ-calculi and Petri nets. *Proc. of 19th Int. Colloquium on Trees in Algebra and Programming*, 1994.
10. C. Fournet and A.D. Gordon. Stack inspection: theory and variants. *ACM Transactions on Programming Languages and Systems*, 25(3):360–399, 2003.
11. Li Gong. Inside Java 2 platform security: architecture, API design, and implementation. Addison-Wesley, 1999.
12. A. Igarashi and N. Kobayashi. Resource usage analysis. *Proc. POPL*, 2002.
13. D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
14. K. Marriott, P. J. Stuckey, and M. Sulzmann. Resource usage verification. *Proc. APLAS (LNCS 3302)*, 2003.
15. G. C. Necula. Proof-carrying code. *Proc. POPL*, 1997.
16. F. Nielson, H. Riis Nielson, and C. Hankin. *Principles of Program Analysis*. Springer, 1999.
17. F. B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(1):30–50, 2000.
18. C. Skalka and S. Smith. History effects and verification. *Proc. of APLAS*, 2004.
19. C. Skalka. Trace Effects and Object Orientation. *Proc. PPDP*, 2005.
20. J.-P. Talpin and P. Jouvelot. The type and effect discipline. *Proc. 7th IEEE Symposium on Logic in Computer Science*, 1992.
21. P. Thiemann. Enforcing Safety Properties Using Type Specialization *Proc. ESOP*, 2001.
22. D. Walker. A type system for expressive security policies. *Proc. POPL*, 2000.

The Complexity of Generalized Satisfiability for Linear Temporal Logic

Michael Bauland[1], Thomas Schneider[2], Henning Schnoor[1], Ilka Schnoor[1], and Heribert Vollmer[1]

[1] Theoret. Informatik, Universität Hannover, Appelstr. 4, 30167 Hannover, Germany
{bauland,henning.schnoor,ilka.schnoor,vollmer}@thi.uni-hannover.de
[2] Informatik, Friedrich-Schiller-Universität, 07737 Jena, Germany
schneider@cs.uni-jena.de

Abstract. In a seminal paper from 1985, Sistla and Clarke showed that satisfiability for Linear Temporal Logic (LTL) is either NP-complete or PSPACE-complete, depending on the set of temporal operators used. If, in contrast, the set of propositional operators is restricted, the complexity may decrease. This paper undertakes a systematic study of satisfiability for LTL formulae over restricted sets of propositional and temporal operators. Since every propositional operator corresponds to a Boolean function, there exist infinitely many propositional operators. In order to systematically cover all possible sets of them, we use Post's lattice. With its help, we determine the computational complexity of LTL satisfiability for all combinations of temporal operators and all but two classes of propositional functions. Each of these infinitely many problems is shown to be either PSPACE-complete, NP-complete, or in P.

Keywords: computational complexity, linear temporal logic.

1 Introduction

Linear Temporal Logic (LTL) was introduced by Pnueli in [Pnu77] as a formalism for reasoning about the properties and the behavior of parallel programs and concurrent systems, and has widely been used for these purposes. Because of the need to perform reasoning tasks — such as deciding satisfiability, validity, or truth in a structure generated by binary relations — in an automated manner, their decidability and computational complexity is an important issue.

It is known that in the case of full LTL with the operators F (eventually), G (invariantly), X (next-time), U (until), and S (since), satisfiability and determination of truth are PSPACE-complete [SC85]. Restricting the set of temporal operators leads to NP-completeness in some cases [SC85]. These results imply that reasoning with LTL is difficult in terms of computational complexity.

This raises the question under which restrictions the complexity of these problems decreases. Since the semantics of LTL is rather fixed, such restrictions can only be of syntactic nature. However, there are several possible constraints that can be posed on the syntax. One possibility is to restrict the set of temporal operators, which has been done almost exhaustively in [SC85].

Another constraint is to allow only a certain "degree of propositionality" in the language, i. e., to restrict the set of allowed propositional operators. Every propositional operator represents a Boolean function — e. g., the operator $\wedge$ (*and*) corresponds to the binary function whose value is 1 if and only if both arguments have value 1. There are infinitely many Boolean functions and hence an infinite number of propositional operators.

If these propositional restrictions are considered in a systematic way, this will lead to a complete classification of the complexity of the reasoning problems for LTL. Not only will this reveal all cases in which, say, satisfiability is tractable. It will also provide a better insight into the sources of hardness by explicitly stating the combinations of temporal and propositional operators that lead to NP- or PSPACE-hard fragments. In addition, the "sources of hardness" will be identified whenever a proof technique is not transferable from an easy to a hard fragment.

The effect of propositional restrictions on the complexity of the satisfiability problem was first considered by Lewis for the case of classical propositional logic in [Lew79]. He established a dichotomy — depending on the set of propositional operators, satisfiability is either NP-complete or decidable in polynomial time. In the case of modal propositional logic, a trichotomy has been achieved in [BHSS06]: modal satisfiability is PSPACE-complete, coNP-complete, or in P. That complete classification in terms of restriction on the propositional operators follows the structure of Post's lattice of closed sets of Boolean functions [Pos41].

This paper analyzes the same restrictions for LTL and combines them with restrictions on the temporal operators. Using Post's lattice, we examine the satisfiability problem for every combination of temporal *and* propositional operators. We determine the computational complexity of these problems, except for one case — the one in which only propositional operators based on the binary *xor* function (and, perhaps, constants) are allowed. We show that all remaining cases are either PSPACE-complete, NP-complete, or in P.

It is not the aim of this paper to focus on particular propositional restrictions that are motivated by certain applications. We prefer to give a classification as complete as possible which allows to choose a fragment that is appropriate, in terms of expressivity and tractability, for any given application.

Among our results, we exhibit cases with non-trivial tractability as well as the smallest possible sets of propositional and temporal operators that already lead to NP-completeness or PSPACE-completeness, respectively. Examples for the first group are cases in which only the unary *not* function, or only monotone functions are allowed, but there is no restriction on the temporal operators. As for the second group, if only the binary function f with $f(x, y) = (x \wedge \overline{y})$ is permitted, then satisfiability is NP-complete already in the case of propositional logic [Lew79]. Our results show that the presence of the same function f separates the tractable languages from the NP-complete and PSPACE-complete ones, depending on the set of temporal operators used. According to this, minimal sets of temporal operators leading to PSPACE-completeness together with f are, for example, $\{\mathsf{U}\}$ and $\{\mathsf{F}, \mathsf{X}\}$.

The technically most involved proof is that of PSPACE-hardness for the language with only the temporal operator S and the boolean operator f (Theorem 3.3). The difficulty lies in simulating the quantifier tree of a Quantified Boolean Formula (QBF) in a linear structure.

Our results are summarized in Table 1. The first column contains the propositional restrictions in terms of closed sets of Boolean functions (clones) whose terminology is introduced in the following section. The second column shows the classification of classical propositional logic as known from [Lew79] and [Coo71]. The last line in column 3 and 4 is largely due to [SC85]. All other entries are the main results of this paper. The only open case appears in the third line and is discussed in the Conclusion. Note that the case distinction also covers all clones which are not mentioned in the present paper.

Table 1. Complexity results for satisfiability. The entries "trivial" denote cases in which a given formula is always satisfiable. The abbreviation "c." stands for "complete." Question marks stand for open questions.

temporal operators / function class (propositional operators)	$\emptyset$	{F}, {G}, {F, G}, {X}	any other combination
below R_1 or below D	trivial	trivial	trivial
below M or below N	in P	in P	in P
L_0, L	in P	?	?
above S_1	NP-c.	NP-c.	PSPACE-c.
BF (all Boolean functions)	NP-c.	NP-c.	PSPACE-c.

2 Preliminaries

A *Boolean function* or *Boolean operator* is a function $f : \{0,1\}^n \rightarrow \{0,1\}$. We can identify an n-ary propositional connector c with the n-ary Boolean operator f defined by: $f(a_1, \ldots, a_n) = 1$ if and only if the formula $c(x_1, \ldots, x_n)$ becomes true when assigning a_i to x_i for all $1 \leq i \leq n$. Additionally to propositional connectors we use the unary temporal operators X (next-time), F (eventually), G (invariantly) and the binary temporal operators U (until), and S (since).

Let B be a finite set of Boolean functions and M be a set of temporal operators. A *temporal B-formula over M* is a formula φ that is built from variables, propositional connectors from B, and temporal operators from M. More formally, a temporal B-formula over M is either a propositional variable or of the form $f(\varphi_1, \ldots, \varphi_n)$ or $g(\varphi_1, \ldots, \varphi_m)$, where φ_i are temporal B-formulae over M, f is an n-ary propositional operator from B and g is an m-ary temporal operator from M. In [SC85], complexity results for formulae using the temporal operators F, G, X (unary), and U, S (binary) were presented. We extend these results to temporal B-formulae over subsets of those temporal operators. The set of variables appearing in φ is denoted with V_φ. If $M = \{\mathsf{X}, \mathsf{F}, \mathsf{G}, \mathsf{U}, \mathsf{S}\}$ we call

φ a *temporal B-formula*, and if $M = \emptyset$ we call φ a *propositional B-formula* or simply a *B-formula*. The set of all temporal B-formulae over M is denoted with $\mathrm{L}(M, B)$.

A model in linear temporal logic is a linear structure of states, which intuitively can be seen as different points of time, with propositional assignments. Formally a *structure* $S = (s, V, \xi)$ consists of an infinite sequence $s = (s_i)_{i\in\mathbb{N}}$ of distinct states, a set of variables V, and a function $\xi : \{s_i \mid i \in \mathbb{N}\} \to 2^V$ which induces a propositional assignment of V for each state. For a temporal $\{\wedge, \neg\}$-formula over $\{\mathsf{X}, \mathsf{U}, \mathsf{S}\}$ with variables from V we define what it means that S *satisfies* φ *in* s_i ($S, s_i \vDash \varphi$): let φ_1 and φ_2 be temporal $\{\wedge, \neg\}$-formulae over $\{\mathsf{X}, \mathsf{U}, \mathsf{S}\}$ and $x \in V$ a variable.

$$\begin{array}{ll}
S, s_i \vDash x & \text{if and only if } x \in \xi(s_i), \\
S, s_i \vDash \varphi_1 \wedge \varphi_2 & \text{if and only if } S, s_i \vDash \varphi_1 \text{ and } S, s_i \vDash \varphi_2, \\
S, s_i \vDash \neg\varphi_1 & \text{if and only if } S, s_i \nvDash \varphi_1, \\
S, s_i \vDash \mathsf{X}\varphi_1 & \text{if and only if } S, s_{i+1} \vDash \varphi_1, \\
S, s_i \vDash \varphi_1 \mathsf{U} \varphi_2 & \text{if and only if there is a } k \geq i \text{ such that } S, s_k \vDash \varphi_2, \\
 & \qquad\qquad\text{and for every } i \leq j < k,\ \ S, s_j \vDash \varphi_1, \\
S, s_i \vDash \varphi_1 \mathsf{S} \varphi_2 & \text{if and only if there is a } k \leq i \text{ such that } S, s_k \vDash \varphi_2, \\
 & \qquad\qquad\text{and for every } k < j \leq i,\ \ S, s_j \vDash \varphi_1.
\end{array}$$

The remaining temporal operators are interpreted as abbreviations: $\mathsf{F}\varphi = \mathit{true}\,\mathsf{U}\varphi$ and $\mathsf{G}\varphi = \neg\mathsf{F}\neg\varphi$. Therefore and since every Boolean operator can be composed from $\wedge$ and $\neg$, the above definition generalizes to temporal B-formulae for arbitrary sets B of Boolean operators.

A temporal B-formula φ over M is *satisfiable* if there exists a structure S such that $S, s_i \vDash \varphi$ for some state s_i from S. That allows us to define the problems we want to look at in this paper: Let B be a finite set of Boolean functions and M a set of temporal operators. Then $\mathrm{SAT}(M, B)$ is the problem to decide whether a given temporal B-formula over M is satisfiable. In the literature, another notion of satisfiability is sometimes considered, where we ask if a formula can be satisfied at the first state in a structure. It is easy to see that, in terms of computational complexity, this does not make a difference for our problems as long as we do not have the temporal operator S in our language. For this paper, we only study the satisfiability problem as defined above.

Sistla and Clarke analyzed the satisfiability problem for temporal $\{\wedge, \vee, \neg\}$-formulae over some sets of temporal operators.

Theorem 2.1 ([SC85])

(1) $\mathrm{SAT}(\{\mathsf{F}\}, \{\wedge, \vee, \neg\})$ *is* NP*-complete.*
(2) $\mathrm{SAT}(\{\mathsf{F}, \mathsf{X}\}, \{\wedge, \vee, \neg\})$, $\mathrm{SAT}(\{\mathsf{U}\}, \{\wedge, \vee, \neg\})$, *and* $\mathrm{SAT}(\{\mathsf{U}, \mathsf{S}, \mathsf{X}\}, \{\wedge, \vee, \neg\})$ *are* PSPACE*-complete.*

Since there are infinitely many finite sets of Boolean functions, we introduce some algebraic tools to classify the complexity of the infinitely many arising satisfiability problems. We denote with id_k^n the n-ary projection to the k-th variable, i.e., $\mathrm{id}_k^n(x_1, \dots, x_n) = x_k$, and with c_a^n the n-ary constant function defined by $c_a^n(x_1, \dots, x_n) = a$. For $c_1^1(x)$ and $c_0^1(x)$ we simply write 1 and 0. A set C of Boolean functions is called a *clone* if it is closed under superposition, which means C contains all projections and C is closed under arbitrary composition [Pip97]. For a set B of Boolean functions we denote with $[B]$ the smallest clone containing B and call B a *base* for $[B]$. In [Pos41] Post classified the lattice of all clones and found a finite base for each clone.

With $\oplus$ we denote the binary exclusive or. Let f be an n-ary Boolean function. We define some properties for f:

- f is 1-*reproducing* if $f(1, \dots, 1) = 1$.
- f is *monotone* if $a_1 \leq b_1, \dots, a_n \leq b_n$ implies $f(a_1, \dots, a_n) \leq f(b_1, \dots, b_n)$.
- f is 1-*separating* if there exists an $i \in \{1, \dots, n\}$ such that $f(a_1, \dots, a_n) = 1$ implies $a_i = 1$.
- f is *self-dual* if $f \equiv \mathrm{dual}(f)$, where $\mathrm{dual}(f)(x_1, \dots, x_n) = \neg f(\neg x_1, \dots, \neg x_n)$.
- f is *linear* if $f \equiv x_1 \oplus \dots \oplus x_n \oplus c$ for a constant $c \in \{0, 1\}$ and variables $x_1, \dots, x_n$.

In Table 2 we define those clones that are essential for this paper plus four basic ones, and give Post's bases [Pos41] for them. The inclusions between them are given in Figure 1. The definitions of all clones as well as the full inclusion graph can be found, for example, in [BCRV03].

Table 2. List of some closed classes of Boolean functions with bases

Name	Definition	Base
BF	All Boolean functions	$\{\vee, \wedge, \neg\}$
R_1	$\{f \in \mathrm{BF} \mid f$ is 1-reproducing $\}$	$\{\vee, \leftrightarrow\}$
M	$\{f \in \mathrm{BF} \mid f$ is monotone $\}$	$\{\vee, \wedge, 0, 1\}$
S_1	$\{f \in \mathrm{BF} \mid f$ is 1-separating $\}$	$\{x \wedge \overline{y}\}$
D	$\{f \mid f$ is self-dual$\}$	$\{x\overline{y} \vee x\overline{z} \vee (\overline{y} \wedge \overline{z})\}$
L	$\{f \mid f$ is linear$\}$	$\{\oplus, 1\}$
L_0	$[\{\oplus\}]$	$\{\oplus\}$
V	$\{f \mid$ There is a formula of the form $c_0 \vee c_1x_1 \vee \dots \vee c_nx_n$ such that c_i are constants for $1 \leq i \leq n$ that describes $f\}$	$\{\vee, 1, 0\}$
E	$\{f \mid$ There is a formula of the form $c_0 \wedge (c_1 \vee x_1) \wedge \dots \wedge (c_n \vee x_n)$ such that c_i are constants for $1 \leq i \leq n$ that describes $f\}$	$\{\wedge, 1, 0\}$
N	$\{f \mid f$ depends on at most one variable$\}$	$\{\neg, 1, 0\}$
I	$\{f \mid f$ is a projection or constant$\}$	$\{0, 1\}$
I_2	$\{f \mid f$ is a projection$\}$	$\emptyset$

There is a strong connection between propositional formulae and Post's lattice. If we interpret propositional formulae as Boolean functions, it is obvious that $[B]$ includes exactly those functions that can be represented by B-formulae. This connection has been used various times to classify the complexity of problems related to propositional formulae: For example, Lewis presented a dichotomy for the satisfiability problem for propositional B-formulae: $\mathrm{SAT}(\emptyset, B)$ is NP-complete if $\mathrm{S}_1 \subseteq [B]$, and solvable in P otherwise [Lew79].

Post's lattice was applied for the equivalence problem [Rei01], counting [RW05] and finding minimal [RV03] solutions, and learnability [Dal00] for Boolean formulae. The technique has been used in non-classical logic as well: Bauland et al. achieved a trichotomy in the context of modal logic, which says that the satisfiability problem for modal formulae is, depending on the allowed propositional connectives, PSPACE-complete, coNP-complete, or solvable in P [BHSS06]. For the inference problem for propositional circumscription, Nordh presented another trichotomy theorem [Nor05].

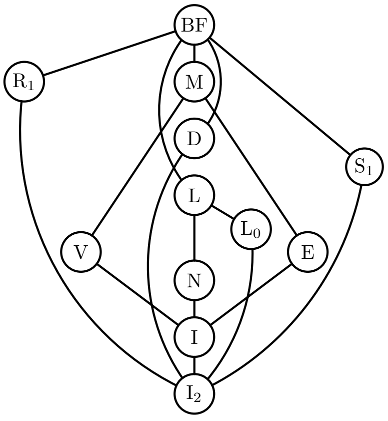


Fig. 1. Graph of some closed classes of Boolean functions

An important tool in restricting the length of the resulting formula in many of our reductions is the following lemma. It shows that for certain sets B, there are always short formulae representing the functions *and*, *or*, or *not*, respectively. Point (2) and (3) follow directly from the proofs in [Lew79], point (1) is Lemma 3.3 from [Sch05].

Lemma 2.2

(1) Let B be a finite set of Boolean functions such that $V \subseteq [B] \subseteq \mathrm{M}$ ($E \subseteq [B] \subseteq \mathrm{M}$, resp.). Then there exists a B-formula $f(x, y)$ such that f represents $x \vee y$ ($x \wedge y$, resp.) and each of the variables x and y occurs exactly once in $f(x, y)$.

(2) Let B be a finite set of Boolean functions such that $[B] = \mathrm{BF}$. Then there are B-formulae $f(x, y)$ and $g(x, y)$ such that f represents $x \vee y$, g represents $x \wedge y$, and both variables occur in each of these formulae exactly once.

(3) Let B be a finite set of Boolean functions such that $\mathrm{N} \subseteq [B]$. Then there is a B-formula $f(x)$ such that f represents $\neg x$ and the variable x occurs in f only once.

3 Results

3.1 Hard Cases

The following lemma gives our general upper bounds for various combinations of temporal operators. The proof of part (1) and (2) is a variation of the proof for Theorem 3.4 in [BHSS06], where, using a similar reduction, an analogous result for circuits was proved.

Lemma 3.1. *Let B be a finite set of Boolean functions. Then the following holds:*

(1) If $M \subseteq \{\mathsf{F}, \mathsf{G}, \mathsf{U}, \mathsf{S}, \mathsf{X}\}$, then $\mathrm{SAT}(M, B)$ is in PSPACE,
(2) if $M \subseteq \{\mathsf{F}, \mathsf{G}\}$, then $\mathrm{SAT}(M, B)$ is in NP, and
(3) if $M \subseteq \{\mathsf{X}\}$, then $\mathrm{SAT}(M, B)$ is also in NP.

Proof. For (1), we will show that $\mathrm{SAT}(M,B) \leq_m^{\log} \mathrm{SAT}(\{\mathsf{U},\mathsf{S},\mathsf{X}\},\{\wedge,\vee,\neg\})$, and for (2), we will show that $\mathrm{SAT}(M,B) \leq_m^{\log} \mathrm{SAT}(\{\mathsf{F}\},\{\wedge,\vee,\neg\})$. The complexity result for these cases then follows from Theorem 2.1. The proof for case (3) is omitted and given in [BSS+06].

The construction for (1) and (2) is nearly identical: Let φ be a formula with arbitrary temporal operators and Boolean functions from B. We recursively transform the formula to a new formula using only the Boolean operators $\wedge$, $\vee$, and $\neg$, and the temporal operators U, S, and X for the first case and the temporal operator F for the second case. For this we construct several formulae, which will be connected via conjunction. Let k be the number of subformulae of φ. Accordingly let $\varphi_1,\ldots,\varphi_k$ be those subformulae with $\varphi=\varphi_1$. Let $x_1,\ldots,x_k$ be new variables, i.e., distinct from the input variables of φ. For all i from 1 to k we make the following case distinction:

– If $\varphi_i = y$ for a variable y, then let $f_i(\varphi) = x_i \leftrightarrow y$.
– If $\varphi_i = \mathsf{X}\varphi_j$, then let $f_i(\varphi) = x_i \leftrightarrow \mathsf{X}x_j$.
– If $\varphi_i = \mathsf{F}\varphi_j$, then let $f_i(\varphi) = x_i \leftrightarrow \mathsf{F}x_j$.
– If $\varphi_i = \mathsf{G}\varphi_j$, then let $f_i(\varphi) = x_i \leftrightarrow \mathsf{G}x_j$.
– If $\varphi_i = \varphi_j\mathsf{U}\varphi_\ell$, then let $f_i(\varphi) = x_i \leftrightarrow x_j\mathsf{U}x_\ell$.
– If $\varphi_i = \varphi_j\mathsf{S}\varphi_\ell$, then let $f_i(\varphi) = x_i \leftrightarrow x_j\mathsf{S}x_\ell$.
– If $\varphi_i = g(\varphi_{i_1},\ldots,\varphi_{i_n})$ for some $g\in B$, then let $f_i(\varphi) = x_i \leftrightarrow h(x_{i_1},\ldots,x_{i_n})$, where h is a formula using only $\wedge$, $\vee$, and $\neg$, representing the function g.

Such a formula h always exists with constant length, because the set B is fixed and does not depend on the input. Now let $f(\varphi) = x_1 \wedge \bigwedge_{i=1}^{k}(\mathsf{G}f_i(\varphi) \wedge \neg(\mathit{true}\,\mathsf{S}\neg f_i(\varphi)))$ for case (1) and $f(\varphi) = x_1 \wedge \bigwedge_{i=1}^{k}\mathsf{G}f_i(\varphi)$ for case (2). The part $\mathsf{G}f_i(\varphi)$ makes sure that $f_i(\varphi)$ holds in every future state of the structure and $\neg(\mathit{true}\,\mathsf{S}\neg f_i(\varphi)))$ does the same for the past states of the structure. Additionally we consider $x \leftrightarrow y$ as a shorthand for $(x\wedge y)\vee(\neg x\wedge\neg y)$. For case (1) we consider $\mathsf{F}x$ as a shorthand for $\mathit{true}\,\mathsf{U}x$ and $\mathsf{G}x$ as a shorthand for $\neg(\mathit{true}\,\mathsf{U}\neg x)$, and for case (2) we consider $\mathsf{G}x$ as a shorthand for $\neg\mathsf{F}\neg x$. Thus we have that $f(\varphi)$ is from $\mathrm{L}(\{\mathsf{U},\mathsf{S},\mathsf{X}\},\{\wedge,\vee,\neg\})$ in case (1) and from $\mathrm{L}(\{\mathsf{F}\},\{\wedge,\vee,\neg\})$ in case (2). Furthermore f is computable in logarithmic space, because the length of f_i is polynomial and neither $\leftrightarrow$ nor the formulae h occur nested. In order to show that f is the reduction we are looking for, we still need to prove that φ is satisfiable if and only if $f(\varphi)$ is satisfiable. Assume an arbitrary structure S, such that $S,s_i \vDash f(\varphi)$ for some s_i. We first prove by induction on the structure of the formula that x_i holds if and only if φ_i holds in every state s of S (for (1)) respectively in every state which lies in the future of s_i (for (2)). Therefore for (1) let s be an arbitrary state and for (2) let s be an arbitrary state in the future of s_i. Thus by construction of $f(\varphi)$ the formulae $f_p(\varphi)$ hold at s for all $1\leq p\leq k$. Then the following holds:

– If $\varphi_p = y$ for a variable y, then $f_p(\varphi) = x_p \leftrightarrow y$ and trivially $S,s\vDash x_p$ iff $S,s\vDash y$.
– If $\varphi_p = \mathsf{X}\varphi_j$, then $f_p(\varphi) = x_p \leftrightarrow \mathsf{X}x_j$. Thus $S,s\vDash x_p$ iff for the successor state s' of s, we have $S,s'\vDash x_j$. By induction this is equivalent to $S,s'\vDash\varphi_j$ and therefore $S,s\vDash\varphi_p$ iff $S,s\vDash x_p$.

– The cases for the temporal operator F or G work analogously.
– If $\varphi_p = \varphi_j \mathsf{U} \varphi_\ell$, then $f_p(\varphi) = x_p \leftrightarrow x_j \mathsf{U} x_\ell$. Thus $S, s \vDash x_p$ iff there exists a state s' in the future of s, such that $S, s' \vDash x_\ell$ and in all states s_m in between (including s) $S, s_m \vDash x_j$. By induction this is equivalent to $S, s' \vDash \varphi_\ell$ and for all states in between $S, s_m \vDash \varphi_j$ and therefore $S, s \vDash \varphi_p$ iff $S, s \vDash x_p$.
– The case for the temporal operator S works analogously to U.
– If $\varphi_p = g(\varphi_{i_1}, \dots, \varphi_{i_n})$, then $f_p(\varphi) = x_p \leftrightarrow h(x_{i_1}, \dots, x_{i_n})$, where h is a formula using only $\wedge$, $\vee$, and $\neg$, representing the function g. Thus $S, s \vDash x_p$ iff $S, s \vDash h(x_{i_1}, \dots, x_{i_n})$. Let I be the subset of $I^n = \{i_1, \dots, i_n\}$, such that $S, s \vDash x_m$ for all $m \in I$ and $S, s \vDash \neg x_m$ for all $m \in I^n \setminus I$. By induction $S, s \vDash \varphi_m$ for all $m \in I$ and $S, s \vDash \neg\varphi_m$ for all $m \in I^n \setminus I$ and therefore $S, s \vDash h(\varphi_{i_1}, \dots, \varphi_{i_n})$. Since h represents the function g, we have that $S, s \vDash \varphi_p$ iff $S, s \vDash x_p$.

Now, assume that $f(\varphi)$ is satisfiable. Then there exists a structure $S, s_i \vDash f(\varphi)$ and thus $S, s_i \vDash x_1$. Since in every state x_j holds if and only if φ_j holds, we have that $S, s_i \vDash \varphi = \varphi_1$. For the other direction, assume that φ is satisfiable. Then there exists a structure $S, s_i \vDash \varphi = \varphi_1$. Now we can extend S by adding new variables $x_1, \dots, x_k$ in such a way, that x_j holds in a state s from S if and only if φ_j holds in that state. Call this new structure S'. Then by construction of $f(\varphi)$, we have $S', s_i \vDash f(\varphi)$, since in every state x_j holds if and only if φ_j holds. $\square$

The following two theorems show that the case in which our Boolean operators are able to express the function $x \wedge \overline{y}$, leads to PSPACE-complete problems in the same cases as for the full set of Boolean operators. This function already played an important role in the classification result from [Lew79], where it also marked the "jump" in complexity from polynomial time to NP-complete.

Theorem 3.2. *Let B be a finite set of Boolean functions such that $\mathrm{S}_1 \subseteq [B]$. Then $\mathrm{SAT}(\{\mathsf{G}, \mathsf{X}\}, B)$ and $\mathrm{SAT}(\{\mathsf{F}, \mathsf{X}\}, B)$ are PSPACE-complete.*

Proof. Since we can express F using G and negation, Theorem 2.1 implies that $\mathrm{SAT}(\{\mathsf{G}, \mathsf{X}\}, \{\wedge, \vee, \neg\})$ and $\mathrm{SAT}(\{\mathsf{F}, \mathsf{X}\}, \{\wedge, \vee, \neg\})$ are PSPACE-hard. Now, let φ be a formula in which only temporal operators G and X, or F and X, and the Boolean connectives $\wedge, \vee$, and $\neg$ appear. Let $B' = B \cup \{1\}$. The complete structure of Post's lattice [BCRV03] shows that $[B'] = \mathrm{BF}$. Now we can rewrite φ as a B'-formula with the same temporal operators appearing. Due to Lemma 2.2, we can express the crucial operators $\wedge, \vee, \neg$ with short B'-formulae, i.e., formulae in which every relevant variable occurs only once. Therefore, this transformation can be performed in polynomial time. Now, in the B'-representation of φ, we exchange every occurrence of 1 with a new variable t, and call the result φ', which is a B-formula. It is obvious that φ is satisfiable if and only if the B-formula $\varphi' \wedge t \wedge \mathsf{G}t$ is. Since $B \supseteq \mathrm{S}_1$, we can express the occurring conjunctions using operators from B (since these are a constant number of conjunctions, we do not need to worry about needing long B-formulae to express conjunction). This finishes the proof for $\mathrm{SAT}(\{\mathsf{G}, \mathsf{X}\}, B)$. For the problem $\mathrm{SAT}(\{\mathsf{F}, \mathsf{X}\}, B)$, observe that the function $g(x, y) = x \wedge \overline{y}$ generates the clone S_1, and therefore there is some B-formula equivalent to g. Now observe that the formula $t \wedge \overline{\mathsf{F}(t \wedge \overline{\mathsf{X}t})} = g(t, \mathsf{F}(g(t, \mathsf{X}t)))$ is equivalent to $\mathsf{G}t$. Since this formula is independent of the input formula φ, this can be computed in polynomial time, and therefore this formula can be used to express $\varphi' \wedge t \wedge \mathsf{G}t$ in the same way as in the first case. Additionally, observe that if the operator F appears in the original formula φ, then a subformula $\mathsf{F}\psi$ can be expressed as $(1\mathsf{U}\psi)$. Hence we conclude from Theorem (2) that $\mathrm{SAT}(\{\mathsf{U}, \mathsf{X}\}, \mathrm{BF})$ is PSPACE-complete. $\square$

The construction in the proof of Theorem 3.2 does not seem to be applicable to the languages with U and/or S, as it requires a way to express $\mathsf{G}t$ using these operators. Hence, proving the desired completeness result requires significantly more work.

Theorem 3.3

(1) Let B be a finite set of Boolean functions with $[B] = \mathrm{BF}$. Then $\mathrm{SAT}(\{\mathsf{S}\}, B)$ is PSPACE-complete.
(2) Let B be a finite set of Boolean functions with $\mathrm{S}_1 \subseteq [B]$. Then $\mathrm{SAT}(\{\mathsf{S}\}, B)$ and $\mathrm{SAT}(\{\mathsf{U}\}, B)$ are PSPACE-complete.

Proof. Since the membership for PSPACE is shown in Lemma 3.1 we only need to show hardness.

(1) We first prove an auxiliary proposition.

Claim. Let $\varphi_1, \dots, \varphi_n$ be satisfiable propositional formulae such that $\varphi_i \to \neg\varphi_j$ is true for all $i, j \in \{1, \dots, n\}$ with $i \neq j$. Then the formula

$$\varphi = \varphi_1 \wedge (\varphi_1 \mathsf{S}(\varphi_2 \mathsf{S}(\dots \mathsf{S}(\varphi_{n-1} \mathsf{S} \varphi_n) \dots))) \wedge ((\dots((\varphi_1 \mathsf{S} \varphi_2) \mathsf{S} \varphi_3) \mathsf{S} \dots) \mathsf{S} \varphi_n)$$

is satisfiable and every structure S that satisfies φ in a state s_m fulfills the following property: there exist natural numbers $0 = a_0 < a_1 < \dots < a_n \leq m+1$ such that $m - a_i < j \leq m - a_{i-1}$ implies $S, s_j \vDash \varphi_i$ for every $i \in \{1 \dots, n\}$.

Proof. Clearly φ is satisfiable: since all formulae φ_i are satisfiable we can find a structure S such that $S, s_0 \vDash \varphi_n, S, s_1 \vDash \varphi_{n-1}, \dots, S, s_{n-1} \vDash \varphi_1$. One can verify that S satisfies φ in s_{n-1}.

Let S be a structure that satisfies φ in a state s_m. Since $\varphi_i \to \neg\varphi_j$ is true for all $i, j \in \{1, \dots, n\}$ with $i \neq j$, in every state only one of the formulae φ_i can be satisfied by S. Therefore and since $S, s_m \vDash \varphi_1 \mathsf{S}(\varphi_2 \mathsf{S}(\dots \mathsf{S}(\varphi_{n-1} \mathsf{S} \varphi_n) \dots))$ holds, there are natural numbers $0 = a_0 \leq a_1 \leq \dots \leq a_{n-1} < a_n \leq m+1$ such that $m - a_i < l \leq m - a_{i-1}$ implies $S, s_l \vDash \varphi_i$ for every $i \in \{1 \dots, n\}$. Since $S, s_m \vDash \varphi_1$, it holds that $a_1 > 0$. Because $S, s_m \vDash (\dots((\varphi_1 \mathsf{S} \varphi_2) \mathsf{S} \varphi_3) \mathsf{S} \dots) \mathsf{S} \varphi_n$ we conclude that $a_1 < \dots < a_{n-1}$, which proves the claim. $\square$

To show hardness for PSPACE, we reduce QBF, which is PSPACE-complete due to [Sto77], to $\mathrm{SAT}(\{\mathsf{S}\}, B)$. Let $\psi = Q_1 x_1 \dots Q_n x_n \varphi$ for some propositional $\{\wedge, \vee, \neg\}$-formula φ with variables $x_1, \dots, x_n$ and for quantifiers $Q_1, \dots, Q_n \in \{\forall, \exists\}$.

Let $I_\forall = \{p_1, \dots, p_k\} = \{i \in \{1, \dots, n\} \mid Q_i = \forall\}$ and $I_\exists = \{q_1, \dots, q_l\} = \{i \in \{1, \dots, n\} \mid Q_i = \exists\}$ such that $p_1 < \dots < p_k$ and $q_1 < \dots < q_l$.

We construct a temporal formula $\psi' \in \mathrm{L}(\{\mathsf{S}\}, B)$ such that ψ is valid if and only if ψ' is satisfiable. Let $t_0, \dots, t_n, u_0, \dots, u_n$ be new variables. We construct subformulae of ψ' which we will combine afterwards.

$$\alpha = u_0 \wedge \overline{t_0} \wedge (u_0 \wedge \overline{t_0})\mathsf{S}((\overline{u_0} \wedge \overline{t_0})\mathsf{S}(\overline{u_0} \wedge t_0))) \wedge (((u_0 \wedge \overline{t_0})\mathsf{S}(\overline{u_0} \wedge \overline{t_0}))\mathsf{S}(\overline{u_0} \wedge t_0))$$

$$\begin{array}{l}
\beta^1[i] = \\
\quad (u_{i-1} \wedge \overline{t_{i-1}} \wedge u_i \wedge \overline{t_i} \wedge \overline{x_i})\mathsf{S} \\
\quad\quad ((\overline{u_{i-1}} \wedge \overline{t_{i-1}} \wedge \overline{u_i} \wedge \overline{t_i} \wedge \overline{x_i})\mathsf{S} \\
\quad\quad\quad ((\overline{u_{i-1}} \wedge \overline{t_{i-1}} \wedge \overline{u_i} \wedge t_i \wedge \overline{x_i})\mathsf{S} \\
\quad\quad\quad\quad ((\overline{u_{i-1}} \wedge \overline{t_{i-1}} \wedge u_i \wedge \overline{t_i} \wedge x_i)\mathsf{S} \\
\quad\quad\quad\quad\quad ((\overline{u_{i-1}} \wedge \overline{t_{i-1}} \wedge \overline{u_i} \wedge \overline{t_i} \wedge x_i)\mathsf{S} \\
\quad\quad\quad\quad\quad\quad (\overline{u_{i-1}} \wedge t_{i-1} \wedge \overline{u_i} \wedge t_i \wedge x_i)))))
\end{array}
\qquad
\begin{array}{l}
\beta^2[i] = \\
\quad (((((u_{i-1} \wedge \overline{t_{i-1}} \wedge u_i \wedge \overline{t_i} \wedge \overline{x_i}) \\
\quad\quad \mathsf{S}(\overline{u_{i-1}} \wedge \overline{t_{i-1}} \wedge \overline{u_i} \wedge \overline{t_i} \wedge \overline{x_i})) \\
\quad\quad\quad \mathsf{S}(\overline{u_{i-1}} \wedge \overline{t_{i-1}} \wedge \overline{u_i} \wedge t_i \wedge \overline{x_i})) \\
\quad\quad\quad\quad \mathsf{S}(\overline{u_{i-1}} \wedge \overline{t_{i-1}} \wedge u_i \wedge \overline{t_i} \wedge x_i)) \\
\quad\quad\quad\quad\quad \mathsf{S}(\overline{u_{i-1}} \wedge \overline{t_{i-1}} \wedge \overline{u_i} \wedge \overline{t_i} \wedge x_i)) \\
\quad\quad\quad\quad\quad\quad \mathsf{S}(\overline{u_{i-1}} \wedge t_{i-1} \wedge \overline{u_i} \wedge t_i \wedge x_i)
\end{array}$$

$$\begin{array}{l}
\gamma^1[i] = (u_{i-1} \wedge \overline{t_{i-1}} \wedge u_i \wedge \overline{t_i} \wedge \overline{x_i})\mathsf{S} \\
\quad\quad ((\overline{u_{i-1}} \wedge \overline{t_{i-1}} \wedge \overline{u_i} \wedge \overline{t_i} \wedge \overline{x_i})\mathsf{S} \\
\quad\quad\quad ((\overline{u_{i-1}} \wedge t_{i-1} \wedge \overline{u_i} \wedge t_i \wedge \overline{x_i})))
\end{array}
\qquad
\begin{array}{l}
\gamma^2[i] = (u_{i-1} \wedge \overline{t_{i-1}} \wedge u_i \wedge \overline{t_i} \wedge x_i)\mathsf{S} \\
\quad\quad ((\overline{u_{i-1}} \wedge \overline{t_{i-1}} \wedge \overline{u_i} \wedge \overline{t_i} \wedge x_i)\mathsf{S} \\
\quad\quad\quad ((\overline{u_{i-1}} \wedge t_{i-1} \wedge \overline{u_i} \wedge t_i \wedge x_i)))
\end{array}$$

Since $[B] = \mathrm{BF}$ and due to Lemma 2.2, there exist short B-representations for $\wedge, \vee$ and $\neg$. Let φ' be a copy of φ that uses these representations instead of $\wedge, \vee$ and $\neg$. Due to the short representations, φ' can be computed in polynomial time. We now define the formula ψ', which constitutes the reduction.

$$\psi' = \alpha \ \wedge \ \bigwedge_{i \in I_\forall} ((\beta^1[i] \wedge \beta^2[i])\mathsf{S}\, t_0) \ \wedge \ \bigwedge_{i = \in I_\exists} ((\gamma^1[i] \vee \gamma^2[i])\mathsf{S}\, t_0) \ \wedge \ (\varphi' \mathsf{S}\, t_0)$$

Since the operators $\wedge, \vee$, and $\neg$ are nested only in constant depth we can use their B-representations without increasing the size of ψ' significantly.

Assume that S is a structure that satisfies ψ' in a state s_m. We prove by induction over n that there are natural numbers $0 = a_0 < \dots < a_{3(2^k)} \leq m + 1$ and for every $q \in I_\exists$ a function $\sigma_q : \{0, 1\}^{q-1} \to \{0, 1\}$ such that S satisfies the following property: if $m - a_i < j \leq m - a_{i-1}$, then

1. $S, s_j \vDash x_{p_h}$ iff $\lceil \frac{i}{3(2^{k-h})} \rceil$ is even
2. $S, s_j \vDash x_{q_h}$ iff $\sigma_{q_h}(a_1 \dots, a_{q_h-1}) = 1$ where -$a_d = 1$ if $x_d \in \xi(s_j)$ and $a_d = 0$ otherwise
3. $S, s_j \vDash t_0$ iff $i = 3(2^k)$
4. $S, s_j \vDash t_{p_h}$ iff $i = c \cdot 3(2^{k-h})$ for some $c \in \mathbb{N}$
5. $S, s_j \vDash t_{q_h}$ iff $S, s_j \vDash t_{p_h-1}$
6. $S, s_j \vDash u_0$ iff $i = 1$
7. $S, s_j \vDash u_{p_h}$ iff $i = c \cdot 3(2^{k-h}) + 1$ for some $c \in \mathbb{N}$
8. $S, s_j \vDash u_{q_h}$ iff $S, s_j \vDash u_{p_h-1}$

Note that due to point 1 for every possible assignment π to $\{x_{p_1}, \dots, x_{p_k}\}$ there is a $j \in \{m - a_{3(2^k)} + 1, \dots, m\}$ such that $S, s_j \vDash x_{p_i}$ if and only if $\pi(x_{p_i}) = 1$. This is the main feature of the construction. The other variables t_i and u_i are necessary to ensure this condition.

For $n = 0$ it holds that $\psi' = \alpha \wedge (\varphi' \mathsf{S}\, t_0)$. Since α satisfies the prerequisites of the auxiliary proposition, there exist natural numbers $0 = a_0 < a_1 < a_2 < a_3 \leq m + 1$ such that

- $m - a_1 < j \leq m - a_0$ implies $S, s_j \vDash u_0 \wedge \overline{t_0}$
- $m - a_2 < j \leq m - a_1$ implies $S, s_j \vDash \overline{u_0} \wedge \overline{t_0}$
- $m - a_3 < j \leq m - a_2$ implies $S, s_j \vDash \overline{u_0} \wedge t_0$

The only occurring variables are u_0 and t_0 and it is easy to see that the above property holds for both.

For the induction step assume that $n > 1$ and the claim holds for $n - 1$. There are two cases to consider:

Case 1: $Q_n = \forall$. That means

$$\psi' = \alpha \wedge \bigwedge_{i \in I_\forall \setminus \{n\}} ((\beta^1[i] \wedge \beta^2[i]) \mathsf{S}\, t_0) \wedge \bigwedge_{i \in I_\exists} ((\gamma^1[i] \vee \gamma^2[i]) \mathsf{S}\, t_0) \wedge (\varphi' \mathsf{S}\, t_0)$$
$$\wedge ((\beta^1[n] \wedge \beta^2[n]) \mathsf{S}\, t_0)$$

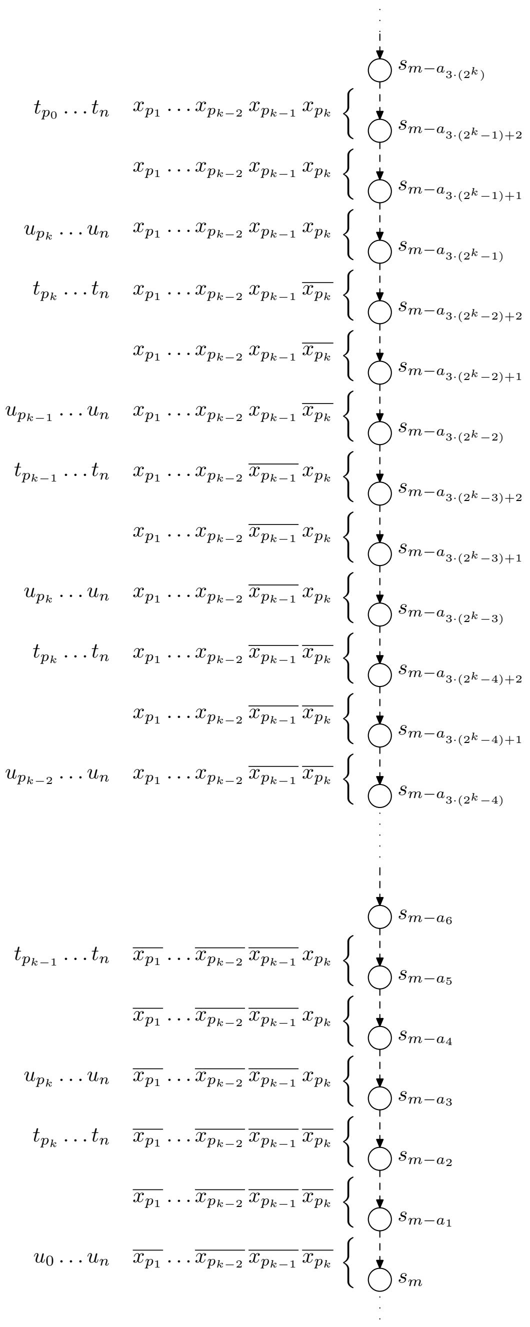
It follows that there are natural numbers $0 = a_0 < \cdots < a_{3(2^{k-1})} \leq m + 1$ and for every $q \in I_\exists$ a function $\sigma_q : \{0,1\}^{q-1} \to \{0,1\}$ such that S fulfills the properties of the claim (note that the subformula $(\psi' \mathsf{S}\, t_0)$ is not necessary for our argument). Since $S, s_m \vDash (\beta^1[n] \wedge \beta^2[n]) \mathsf{S}\, t_0$ and for $m - a_{3(2^{k-1})} < j \leq m$ it holds that $S, s_j \vDash t_0$ if and only if $j \leq m - a_{3(2^{k-1})-1}$, we have $S, s_j \vDash \beta^1[n] \wedge \beta^2[n]$ for every $m - a_{3(2^{k-1})-1} < j \leq m$. Let $i = c \cdot 3$ for some $c \in \mathbb{N}$, then it holds that $m - a_{i+1} < j \leq m - a_i$ implies $S, s_j \vDash u_{n-1}$ which means that for these states s_j it holds that $S, s_j \vDash u_{n-1} \wedge \overline{t_{n-1}} \wedge u_n \wedge \overline{t_n} \wedge x_n$. Due to our proposition there are natural numbers $0 = b_0^i < b_1^i < \cdots < b_6^i \leq a_i + 1$ such that

- $a_i - b_1^i < j \leq a_i - b_0^i$ implies $S, s_j \vDash u_{n-1} \wedge \overline{t_{n-1}} \wedge u_n \wedge \overline{t_n} \wedge \overline{x_n}$
- $a_i - b_2^i < j \leq a_i - b_1^i$ implies $S, s_j \vDash \overline{u_{n-1}} \wedge \overline{t_{n-1}} \wedge \overline{u_n} \wedge \overline{t_n} \wedge \overline{x_n}$
- $a_i - b_3^i < j \leq a_i - b_2^i$ implies $S, s_j \vDash \overline{u_{n-1}} \wedge \overline{t_{n-1}} \wedge \overline{u_n} \wedge t_n \wedge \overline{x_n}$
- $a_i - b_4^i < j \leq a_i - b_3^i$ implies $S, s_j \vDash \overline{u_{n-1}} \wedge \overline{t_{n-1}} \wedge u_n \wedge \overline{t_n} \wedge x_n$
- $a_i - b_5^i < j \leq a_i - b_4^i$ implies $S, s_j \vDash \overline{u_{n-1}} \wedge \overline{t_{n-1}} \wedge \overline{u_n} \wedge \overline{t_n} \wedge x_n$
- $a_i - b_6^i < j \leq a_i - b_5^i$ implies $S, s_j \vDash \overline{u_{n-1}} \wedge t_{n-1} \wedge \overline{u_n} \wedge t_n \wedge x_n$

The nearest state before s_{m-a_i} that satisfies $\overline{u_{n-1}}$ is $s_{m-a_{i+1}}$ and the nearest state before s_{m-a_i} that satisfies t_{n-1} is $s_{m-a_{i+2}}$, therefore it holds that $b_1^i = a_{i+1} - a_i$ and $b_5^i = a_{i+2} - a_i$. By denoting $b_j^i + a_i$ with c_{2i+j} we define natural numbers $c_0, \dots, c_{3(2^k)}$ for which it can be verified that they fulfill the claim.

Case 2: $Q_n = \exists$. In this case we have

$$\psi' = \alpha \wedge \bigwedge_{i \in I_\forall} ((\beta^1[i] \wedge \beta^2[i]) \mathsf{S}\, t_0) \wedge \bigwedge_{i \in I_\exists \setminus \{n\}} ((\gamma^1[i] \vee \gamma^2[i]) \mathsf{S}\, t_0) \wedge (\varphi' \mathsf{S}\, t_0)$$
$$\wedge ((\gamma^1[n] \vee \gamma^2[n]) \mathsf{S}\, t_0).$$



$s_{m-a_{3\cdot(2^k)}}$
$t_{p_0}\dots t_n$ $x_{p_1}\dots x_{p_{k-2}}\,x_{p_{k-1}}\,x_{p_k}$ $s_{m-a_{3\cdot(2^k-1)+2}}$
$x_{p_1}\dots x_{p_{k-2}}\,x_{p_{k-1}}\,x_{p_k}$ $s_{m-a_{3\cdot(2^k-1)+1}}$
$u_{p_k}\dots u_n$ $x_{p_1}\dots x_{p_{k-2}}\,x_{p_{k-1}}\,x_{p_k}$ $s_{m-a_{3\cdot(2^k-1)}}$
$t_{p_k}\dots t_n$ $x_{p_1}\dots x_{p_{k-2}}\,x_{p_{k-1}}\,\overline{x_{p_k}}$ $s_{m-a_{3\cdot(2^k-2)+2}}$
$x_{p_1}\dots x_{p_{k-2}}\,x_{p_{k-1}}\,\overline{x_{p_k}}$ $s_{m-a_{3\cdot(2^k-2)+1}}$
$u_{p_{k-1}}\dots u_n$ $x_{p_1}\dots x_{p_{k-2}}\,x_{p_{k-1}}\,\overline{x_{p_k}}$ $s_{m-a_{3\cdot(2^k-2)}}$
$t_{p_{k-1}}\dots t_n$ $x_{p_1}\dots x_{p_{k-2}}\,\overline{x_{p_{k-1}}}\,x_{p_k}$ $s_{m-a_{3\cdot(2^k-3)+2}}$
$x_{p_1}\dots x_{p_{k-2}}\,\overline{x_{p_{k-1}}}\,x_{p_k}$ $s_{m-a_{3\cdot(2^k-3)+1}}$
$u_{p_k}\dots u_n$ $x_{p_1}\dots x_{p_{k-2}}\,\overline{x_{p_{k-1}}}\,x_{p_k}$ $s_{m-a_{3\cdot(2^k-3)}}$
$t_{p_k}\dots t_n$ $x_{p_1}\dots x_{p_{k-2}}\,\overline{x_{p_{k-1}}}\,\overline{x_{p_k}}$ $s_{m-a_{3\cdot(2^k-4)+2}}$
$x_{p_1}\dots x_{p_{k-2}}\,\overline{x_{p_{k-1}}}\,\overline{x_{p_k}}$ $s_{m-a_{3\cdot(2^k-4)+1}}$
$u_{p_{k-2}}\dots u_n$ $x_{p_1}\dots x_{p_{k-2}}\,\overline{x_{p_{k-1}}}\,\overline{x_{p_k}}$ $s_{m-a_{3\cdot(2^k-4)}}$
s_{m-a_6}
$t_{p_{k-1}}\dots t_n$ $\overline{x_{p_1}}\dots\overline{x_{p_{k-2}}}\,\overline{x_{p_{k-1}}}\,x_{p_k}$ s_{m-a_5}
$\overline{x_{p_1}}\dots\overline{x_{p_{k-2}}}\,\overline{x_{p_{k-1}}}\,x_{p_k}$ s_{m-a_4}
$u_{p_k}\dots u_n$ $\overline{x_{p_1}}\dots\overline{x_{p_{k-2}}}\,\overline{x_{p_{k-1}}}\,x_{p_k}$ s_{m-a_3}
$t_{p_k}\dots t_n$ $\overline{x_{p_1}}\dots\overline{x_{p_{k-2}}}\,\overline{x_{p_{k-1}}}\,\overline{x_{p_k}}$ s_{m-a_2}
$\overline{x_{p_1}}\dots\overline{x_{p_{k-2}}}\,\overline{x_{p_{k-1}}}\,\overline{x_{p_k}}$ s_{m-a_1}
$u_0\dots u_n$ $\overline{x_{p_1}}\dots\overline{x_{p_{k-2}}}\,\overline{x_{p_{k-1}}}\,\overline{x_{p_k}}$ s_m

Fig. 2. Structure for the proof of Theorem 3.3

Because of the induction hypothesis there are natural numbers $0 = a_0 < a_1 < \cdots < a_{3(2^k)} \leq m+1$ such that the required properties are satisfied. Analogously to the first case $S, s_j \vDash \gamma^1[i] \vee \gamma^2[i]$ is true for every $m - a_{3(2^k)} < j \leq m$. Let $i = c \cdot 3$, then for $m - a_{i+1} < j \leq m - a_i$ it holds that $S, s_j \vDash u_{n-1} \wedge \overline{t_{n-1}} \wedge u_n \wedge \overline{t_n} \wedge x_n$ or $S, s_j \vDash u_{n-1} \wedge \overline{t_{n-1}} \wedge u_n \wedge \overline{t_n} \wedge \overline{x_n}$, because $S, s_j \vDash u_{n-1}$. For $m - a_{i+2} < j \leq m - a_{i+1}$ we have that $S, s_j \vDash \overline{u_{n-1}} \wedge \overline{t_{n-1}} \wedge \overline{u_n} \wedge \overline{t_n} \wedge x_n$ or $S, s_j \vDash \overline{u_{i-n}} \wedge \overline{t_{i-n}} \wedge \overline{u_n} \wedge \overline{t_n} \wedge \overline{x_n}$ and for $m - a_{i+3} < j \leq m - a_{i+2}$ it must hold $S, s_j \vDash \overline{u_{n-1}} \wedge t_{n-1} \wedge \overline{u_n} \wedge t_n \wedge x_n$ or $S, s_j \vDash \overline{u_{n-1}} \wedge t_{n-1} \wedge \overline{u_n} \wedge t_n \wedge \overline{x_n}$. If $S, s_{a_i} \vDash \gamma^1[n]$, then in all these states $\overline{x_n}$ is satisfied; if $S, s_{a_i} \vDash \gamma^2[n]$, then x_n is. Therefore with σ_n defined by $\sigma_n(d_1, \ldots, d_{n-1}) = 1$ if and only if $S, s_{3(d_1 2^{n-2} + \cdots + d_{n-1} 2^0)} \vDash \gamma^2[n]$, the induction is complete, because the binary numbers correspond to the assignments to the $\forall$-quantified variables.

Note that for a structure that satisfies ψ' with the above notation, $S, s_j \vDash \varphi$ holds for every $m - a_{3(2^k)} < j \leq m$, since $\varphi' \mathsf{S}\, t_0$ is a conjunct of ψ'.

Now assume that ψ' is satisfiable in a state s_m of a structure S. This is if and only if for every $q \in I_\exists$ there is a function $\sigma_q : \{0,1\}^{q-1} \to \{0,1\}$ such that S fulfills the above property. Hence each possible assignment J to the $\forall$-quantified variables $\{x_{p_1}, \ldots, x_{p_k}\}$ can be extended to an assignment to $\{x_1, \ldots, x_n\}$ by $J(x_{q_i}) = \sigma_{q_i}(J(x_1), \ldots, J(x_{q_i - 1}))$ which is equivalent to the validity of ψ.

(2) The above reduction can be modified using ideas from the proof of Theorem 3.2. The details are omitted and given in [BSS⁺06]. We can prove PSPACE-hardness for $\mathrm{SAT}(\{\mathsf{U}\}, B)$ with an analogous construction. $\square$

The following proposition follows immediately from a result of Lewis's [Lew79] and the previously established upper bounds.

Proposition 3.4. *Let B be a finite set of Boolean functions with $\mathrm{S}_1 \subseteq [B]$. Then $\mathrm{SAT}(\{\mathsf{F}\}, B)$, $\mathrm{SAT}(\{\mathsf{G}\}, B)$, $\mathrm{SAT}(\{\mathsf{F}, \mathsf{G}\}, B)$, and $\mathrm{SAT}(\{\mathsf{X}\}, B)$ are NP-complete.*

3.2 Polynomial Time Results

This subsection lists all cases for which LTL satisfiability can be decided in polynomial time. Due to the limitations of space, the proofs are omitted and can be found in the report [BSS⁺06].

As Theorem 3.5 shows, for some sets B of Boolean functions, there is a satisfying model for every temporal B-formula over any set of temporal operators.

Theorem 3.5. *Let B be a finite subset of R_1 or D. Then every formula φ from $\mathrm{L}(\{\mathsf{F}, \mathsf{G}, \mathsf{X}, \mathsf{U}, \mathsf{S}\}, B)$ is satisfiable.*

Due to Theorem 3.6, satisfiability for formulae with any combination of modal operators, but only very restricted Boolean operators is always easy to decide.

Theorem 3.6. *Let B be a finite subset of N or M. Then $\mathrm{SAT}(\{\mathsf{F}, \mathsf{G}, \mathsf{X}, \mathsf{U}, \mathsf{S}\}, B)$ can be decided in polynomial time.*

Finally, satisfiability for formulae that have X as a modal operator and the *xor* function $\oplus$ as a propositional operator is in P. This is true because functions described by these formulae have a high degree of symmetry.

Theorem 3.7. *Let B be a finite subset of* L. *Then* $\mathrm{SAT}(\{\mathsf{X}\}, B)$ *can be decided in polynomial time.*

4 Conclusion

We have almost completely classified the computational complexity of satisfiability for LTL with respect to the sets of propositional and temporal operators permitted. The only case left open is the one in which only propositional operators constructed from the binary *xor* function (and, perhaps, constants) are allowed. This case has already turned out to be difficult to handle — and hence was left open — in [BHSS06] for modal satisfiability under *restricted* frames classes. The difficulty here and in [BHSS06] is reflexivity, i. e., the property that the formula $\mathsf{F}\varphi$ is satisfied at some state if φ is satisfied at *the same* state. This does not allow for a separate treatment of the propositional part (without temporal operators) and the remainder of a given formula.

Our results bear an interesting resemblance to the classifications obtained in [Lew79] and in [BHSS06]. In all of these cases (except for one of the several classifications obtained in the latter), it turns out that sets of Boolean functions B which generate a clone above S_1 give rise to computationally hard problems, while other cases seem to be solvable in polynomial time. Therefore, in a precise sense, it is the function represented by the formula $x \wedge \overline{y}$ which turns problems in this context computationally intractable. These hardness results seem to indicate that $x \wedge \overline{y}$ and other functions which generate clones above S_1 have properties that make computational problems hard, and this notion of hardness is to a large extent independent of the actual problem considered.

It is worth knowing whether our results are transferable to what is called "determination of truth" in [SC85] — the model checking problem. In the case of LTL with no restrictions on the propositional operators, model checking has the same complexity as satisfiability [SC85]. We have done first steps towards a similar classification of this problem. The first partial results suggest that the behavior of model checking is not quite the same as that of satisfiability.

The results from this paper leave two open questions. Besides the unsolved *xor* case, it would be interesting to further classify the polynomial-time solvable cases. Further work could also examine related specification languages, such as CTL, CTL*, or hybrid temporal languages.

Acknowledgments

We thank Martin Mundhenk and the anonymous referees for their helpful comments and suggestions.

References

[BCRV03] E. Böhler, N. Creignou, S. Reith, and H. Vollmer. Playing with Boolean blocks, part I: Post's lattice with applications to complexity theory. *SIGACT News*, 34(4):38–52, 2003.

[BHSS06] M. Bauland, E. Hemaspaandra, H. Schnoor, and I. Schnoor. Generalized modal satisfiability. In B. Durand and W. Thomas, editors, *STACS*, volume 3884 of *Lecture Notes in Computer Science*, pages 500–511. Springer, 2006.

[BSS$^+$06] M. Bauland, H. Schnoor, I. Schnoor, T. Schneider, and H. Vollmer. The complexity of generalized satisfiability for linear temporal logic. Technical Report TR06-153, Electronic Colloquium on Computational Complexity, 2006.

[Coo71] S. A. Cook. The complexity of theorem proving procedures. In *Proceedings 3rd Symposium on Theory of Computing*, pages 151–158. ACM Press, 1971.

[Dal00] V. Dalmau. *Computational Complexity of Problems over Generalized Formulas.* PhD thesis, Department de Llenguatges i Sistemes Informàtica, Universitat Politécnica de Catalunya, 2000.

[Lew79] H. Lewis. Satisfiability problems for propositional calculi. *Mathematical Systems Theory*, 13:45–53, 1979.

[Nor05] G. Nordh. A trichotomy in the complexity of propositional circumscription. In *Proceedings of the 11th International Conference on Logic for Programming*, volume 3452 of *Lecture Notes in Computer Science*, pages 257–269. Springer Verlag, 2005.

[Pip97] N. Pippenger. *Theories of Computability.* Cambridge University Press, Cambridge, 1997.

[Pnu77] A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE, 1977.

[Pos41] E. Post. The two-valued iterative systems of mathematical logic. *Annals of Mathematical Studies*, 5:1–122, 1941.

[Rei01] S. Reith. *Generalized Satisfiability Problems.* PhD thesis, Fachbereich Mathematik und Informatik, Universität Würzburg, 2001.

[RV03] S. Reith and H. Vollmer. Optimal satisfiability for propositional calculi and constraint satisfaction problems. *Information and Computation*, 186(1):1–19, 2003.

[RW05] S. Reith and K. W. Wagner. The complexity of problems defined by Boolean circuits. In *Proceedings International Conference Mathematical Foundation of Informatics, (MFI99); World Science Publishing*, 2005.

[SC85] A. Sistla and E. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[Sch05] H. Schnoor. The complexity of the Boolean formula value problem. Technical report, Theoretical Computer Science, University of Hannover, 2005.

[Sto77] L. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.

Formalising the π-Calculus Using Nominal Logic

Jesper Bengtson and Joachim Parrow

Department of Information Technology, University of Uppsala, Sweden

Abstract. We formalise the pi-calculus using the nominal datatype package, a package based on ideas from the nominal logic by Pitts et al., and demonstrate an implementation in Isabelle/HOL. The purpose is to derive powerful induction rules for the semantics in order to conduct machine checkable proofs, closely following the intuitive arguments found in manual proofs. In this way we have covered many of the standard theorems of bisimulation equivalence and congruence, both late and early, and both strong and weak in a unison manner. We thus provide one of the most extensive formalisations of a process calculus ever done inside a theorem prover.

A significant gain in our formulation is that agents are identified up to alpha-equivalence, thereby greatly reducing the arguments about bound names. This is a normal strategy for manual proofs about the pi-calculus, but that kind of hand waving has previously been difficult to incorporate smoothly in an interactive theorem prover. We show how the nominal logic formalism and its support in Isabelle accomplishes this and thus significantly reduces the tedium of conducting completely formal proofs. This improves on previous work using weak higher order abstract syntax since we do not need extra assumptions to filter out exotic terms and can keep all arguments within a familiar first-order logic.

1 Introduction

1.1 Motivation

As the complexity of software systems increases, the need is growing to ensure their correct operation. One way forward is to create particular theories or frameworks geared towards particular application areas. These frameworks have the right kind of abstractions built in from the beginning, meaning that proofs can be conducted at a high level. The drawback is that different areas need different such frameworks, resulting in a proliferation and even abundance of theories. A prime example can be found in the field of process calculi. It originated in work by Milner in the late 1970s [8] and was intended to provide an abstract way to reason about parallel and communicating processes. Today there are many different strands of calculi addressing specific issues. Each of them embodies a certain kind of abstraction suitable for a particular area of application.

For each such calculus a certain amount of theoretical groundwork must be laid down. Typical examples include definitions of the semantics, establishing substitutive properties, structures for inductive proof strategies etc. This groundwork must naturally be correct beyond doubt (if there is an error in it then all proofs conducted in that calculus will be incorrect). The idea to use formal verification of the groundwork itself is therefore natural. In this paper we shall present an improved method to accomplish this.

1.2 Approach

The goal of our project is to provide a library in an automated theorem prover, Isabelle/HOL [11], which allows users to do machine checked proofs on the groundwork of process calculi. The guiding principle is that the proofs should correspond very closely to the traditional manual proofs present in the literature. This means that for a person who has completed these proofs manually very little extra effort should be required in order to let Isabelle check them. Today those proofs are reasonably well understood, but capturing them in a theorem prover has until now been a daunting task. The reason is mainly related to bound names and the desire to abstract away from α-equivalence [1].

In the literature it is not uncommon to find statements such as: "henceforth we shall not distinguish between α-equivalent terms" or "we assume bound names to always be fresh", even though it is left unsaid exactly what this means. This kind of reasoning does not necessarily imply that proofs conducted in this manner are incorrect, but for a full formal formalisation of a system involving binders, a solid mathematical foundation where α-equivalence is clearly defined has to be created.

Our approach is to formulate the π-calculus using ideas from nominal logic developed by (Pitts et al. [13,5,17]). This is a first order logic designed to work with calculi using binders. It maintains all the properties of a first order logic and introduces an explicit notion of freshness of names in the terms. Gabbay's thesis [3] uses it to introduce FM set theory, this is the standard ZF set theory but with an extra axiom for freshness of names. Recent work by Urban and Tasson [19] extends the work done by Pitts and Gabbay and solves the problem with freshness without introducing new axioms. The techniques have been implemented into the theorem prover Isabelle/HOL, in a nominal datatype package, so that when defining nominal datatypes, Isabelle will automatically generate a type which models the datatype up to α-equivalence as well as induction principles and a recursion combinator which allows the user to create functions on nominal datatypes.

1.3 Results

Our contribution is to use this nominal package in Isabelle to describe the π-calculus. We have proved substantial portions of [9] using these techniques. More specifically, we have proven that strong equivalence and weak congruence are congruence relations for both late and early operational semantics, that all structurally congruent terms are bisimilar and that late strong equivalence, weak bisimulation and weak congruence are included in their early counterparts. To our knowledge, properties about weak equivalences of the π-calculus have never before been formally derived inside a theorem prover. Our proof method is to lift the strong operational semantics to a weak one, enabling us to port our proofs between the two semantics. Moreover, our proofs follow their pen-and-paper equivalents very closely inside a first-order environment. In other words, the extra effort to have proofs checked by a machine is not prohibitive.

1.4 Exposition

In the next section we explain some basic concepts of the nominal datatype package. We do not give a full account of it, only enough that a reader may follow the rest of our paper. In Sections 3–7 we demonstrate how the π-calculus syntax, semantics, and bisimulation equivalences are represented in our framework. In the concluding section we compare our efforts to related work and comment on planned further work. An extended version of this paper together with the Isabelle source files can be found at http://www.it.uu.se/katalog/jesperb/pi.

2 The Nominal Datatype Package

For a more thorough presentation of the nominal datatype package the reader is referred to [19], but enough basic definitions will be covered here for the reader to understand the rest of this paper. A *nominal datatype* definition is like an ordinary data type but it explicitly tags the binding occurrences of names. For example, a data type for λ-calculus terms would in this way tag the name in the abstraction. The point is that the nominal package in Isabelle automatically generates induction rules where α-equivalent terms are identified, thus saving the user much tedium in large proofs.

At the heart of nominal logic is the notion of *name swapping.* If T is any term of permutation type (a term which supports permutations of its names) and a and b are names then $(a\ b) \bullet T$ denotes the term where all instances of a in T become b and vice versa. All names (even the binding and bound occurrences) are swapped in this way. A *permutation* p is a finite sequence of swappings. If $p = (a_1\ b_1) \cdots (a_n\ b_n)$ then $p \bullet T$ means applying all swappings in p to T, beginning with the last element $(a_n\ b_n)$.

Permutations are mathematically well behaved. They very rarely change the properties of a term. Most importantly, α-equivalence is preserved by permutations. The property of being preserved by permutations is often called *equivariance.* We shall mainly use equivariance on binary relations, where the definition is:

Definition 1. *Equivariance*
$$\text{eqvt}\ \mathcal{R} \overset{def}{=} \forall p\ T\ U.\ (T,\ U) \in \mathcal{R} \Longrightarrow (p \bullet T,\ p \bullet U) \in \mathcal{R}$$

Another key concept is the notion of *support.* The definition, in general, is that the support supp T of a term T is the set of names which can affect T in permutations. In other words, if p is a permutation only involving names outside the support of T then $p \bullet T = T$. Remembering that α-equivalent terms are identified we see that the support corresponds to the *free names* in calculi like the λ-calculus.

A crucial property is that the support of a term is finite. This implies that for any term it is always possible to find a name outside its support. One says that a name a is *fresh* for a term T, written $a \sharp T$, if a is not in the support of T.

Permutations can be used to capture α-equivalence. Let $[x].P$ stand for any operator that binds x in T.

Proposition 1. $[x].T = [y].U \Longrightarrow$
$(x = y \wedge T = U) \vee (x \neq y \wedge x \sharp U \wedge T = (x\ y) \bullet U)$

If $[x].T = [y].U$ then either x and y are equal and T and U are α-equivalent or x is not equal to y and fresh in U and T is α-equivalent to U with all occurrences of x swapped with y and vice versa. Another way to capture α-equivalence is the following:

Proposition 2. $c \sharp (x, y, T, U) \wedge [x].T = [y].U \Longrightarrow (x\ c) \bullet T = (y\ c) \bullet U$

Here and in the rest of the paper we use the word "proposition" for something that Isabelle generates automatically.

3 Defining the π-Calculus

We present a version of the monadic π-calculus [9]. We assume that the reader is familiar with the basic ideas of its syntax and semantics.

Definition 2. *Defining the π-calculus in Isabelle*

```
nominal_datatype pi = PiNil
                    | Tau pi
                    | Input name "<<name>> pi"
                    | Output name name pi
                    | Match name name pi
                    | Sum pi pi
                    | Par pi pi
                    | Res "<<name>> pi"
                    | Bang pi
```

This definition is an example of Isabelle syntax. The notation $\ll$*name*$\gg$ *pi* indicates that *name* is bound in *pi*. For the rest of the paper we shall use the traditional syntax for π-calculus terms, e.g. writing inputs as $a(x)$ and restrictions as (νx).

The nominal datatype package automatically generates lemmas for reasoning about α-equivalence between processes – the ones generated from Prop. 1 can be found in the following proposition.

Proposition 3. *The most commonly used α-equivalence rules for the* Input- *and the* Restriction case.

Input: $a(x).P = b(y).Q \Longrightarrow a = b \wedge ((x = y \wedge P = Q) \vee (x \neq y \wedge x \sharp Q \wedge P = (x\ y) \bullet Q))$

Restriction: $(\nu x)P = (\nu y)Q \Longrightarrow (x = y \wedge P = Q) \vee (x \neq y \wedge x \sharp Q \wedge P = (x\ y) \bullet Q)$

Most modern theorem provers automatically generate induction rules for defined datatypes. The nominal datatype package does the same for nominal datatypes but with one addition: bound names which occur in the inductive cases can be assumed to be disjoint from any finite set of names. This greatly reduces the amount of manual α-conversions.

Functions over nominal datatypes have one restriction – they may not depend on the bound names in their arguments. Since nominal types are equal up to α-equivalence two equal terms may have different bound names.

The most commonly used function is substitution where $P\{a/b\}$ (which can be read P with a for b) is the substitution of all occurrences of b in P with a.

4 Operational Semantics

4.1 Definitions

We use the standard operational semantics [9]. Here transitions are of the form $P \xrightarrow{\alpha} P'$, where α is an action. A first attempt, which works well for simpler calculi like CCS, is to inductively define a set of tuples containing three elements: a process P, an action α and the α-derivative of P [2].

However, in the π-calculus the action α may bind a name, and the scope of this binding extends into P'. In particular we shall sometimes need to α-convert the action together with the derivative P'. For this purpose, we create a *residual*-datatype which is a nominal datatype. It binds the bound names of an action also in the derivative.

Definition 3. *The residual datatype*

```
datatype subject = Input name
                 | BoundOutput name

datatype freeRes = Output name name
                 | Tau

nominal_datatype residual = BoundR subject "<<name>> pi"
                          | FreeR freeRes pi
```

We introduce a notation for an arbitrary action with a bound name, i. e., an *Input*- or a *Bound Output* action.

Definition 4
(i) $P \xrightarrow{a\ll x\gg} P'$ denotes a transition with the bound name x in the action. Note that a is of type `subject`.

(ii) $P \xrightarrow{\alpha} P'$ denotes a transition without bound names. Note that α is of type `freeRes`.

We can now define our operational semantics using inductively defined sets, and the set will contain tuples of two elements – one process and one residual.

As previously mentioned, functions over nominal datatypes cannot depend on bound names. This poses a slight problem, since traditionally some of the operational rules have conditions on the bound names like $x \notin \mathrm{bn}(\alpha)$, i.e. x is not in the bound names of α. A function such as bn does not exist in nominal logic and thus cannot be created using the nominal datatype package. An easy solution is to split the operational rules which have these types of conditions into two rules — one for the transitions with bound names, and one for the ones without. Doing this does not create extra proof obligations as most proofs have to consider bound and free transitions separately anyway. The operational semantics, including the split rules for **Par** and **Res** can be found in Fig. 1.

$$\frac{}{a(x).P \xrightarrow{a(x)} P}\ \textbf{Input} \qquad \frac{}{\bar{a}b.P \xrightarrow{\bar{a}b} .P}\ \textbf{Output} \qquad \frac{}{\tau.P \xrightarrow{\tau} P}\ \textbf{Tau}$$

$$\frac{P \xrightarrow{\alpha} P'}{[a=a]P \xrightarrow{\alpha} P'}\ \textbf{Match} \qquad \frac{P \xrightarrow{\bar{a}b} P' \quad a \neq b}{(\nu b)P \xrightarrow{\bar{a}(b)} P'}\ \textbf{Open} \qquad \frac{P \xrightarrow{\alpha} P'}{P\ +\ Q \xrightarrow{\alpha} P'}\ \textbf{Sum}$$

$$\frac{P \xrightarrow{a\ll x\gg} P' \qquad x \sharp Q}{P \mid Q \xrightarrow{a\ll x\gg} P' \mid Q}\ \textbf{ParB} \qquad \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \textbf{ParF}$$

$$\frac{P \xrightarrow{a(x)} P' \quad Q \xrightarrow{\bar{a}b} Q'}{P \mid Q \xrightarrow{\tau} P'\{b/x\} \mid Q'}\ \textbf{Comm} \qquad \frac{P \xrightarrow{a(x)} P' \quad Q \xrightarrow{\bar{a}(y)} Q' \quad y \sharp P}{P \mid Q \xrightarrow{\tau} (\nu y)(P'\{y/x\} \mid Q')}\ \textbf{Close}$$

$$\frac{P \xrightarrow{a\ll x\gg} P' \qquad y \sharp (a,\ x)}{(\nu y)P \xrightarrow{a\ll x\gg} (\nu y)P'}\ \textbf{ResB} \qquad \frac{P \xrightarrow{\alpha} P' \qquad y \sharp \alpha}{(\nu y)P \xrightarrow{\alpha} (\nu y)P'}\ \textbf{ResF}$$

$$\frac{P \mid !P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'}\ \textbf{Replication}$$

Fig. 1. The *Par*- and the *Res*-rule in the operational semantics of the π-calculus have been split. Symmetric versions have been elided.

4.2 Induction and Case Analysis Rules

Isabelle will automatically create rules for both induction and case analysis of the semantics. These rules, however, assume that the equivalence relation used is syntactic equivalence and not α-equivalence. While the nominal datatype package automatically creates induction rules for nominal datatypes there is no such automation for the kind of inductively defined sets that we use in the semantics. The rules generated by Isabelle for our operational semantics suffer from two problems, which we now address in turn.

The first problem is that some semantic rules generate bound names. When the rule is applied in the context of a proof, there is no a priori guarantee that these names are fresh in this larger context. We therefore derive rules for induction and case analysis which are parameterized on a finite set of names, the "context names", which the user can provide when applying the rule. The bound names generated by the rules are guaranteed to be fresh from the context names (just as is guaranteed automatically for nominal data types, and for the same reason: avoiding name clashes and α-conversions later in the proof). This idea stems from [19] which has been developed independently of our work in [18]. The logical framework has also been covered in [14].

As an example a derived rule for case analysis of the parallel operator is shown in the following proposition where the parameter $\mathcal{C}$ is a set of context names:

Lemma 1. *The derived case analysis rule for the parallel operator with no bound names in the transition.*

$$\frac{\begin{array}{l} P \mid Q \xrightarrow{\alpha} R' \\ \forall P'. P \xrightarrow{\alpha} P' \wedge R' = P' \mid Q \Longrightarrow Prop \\ \forall P'\ Q'\ a\ x\ b.\ P \xrightarrow{a(x)} P' \wedge Q \xrightarrow{\bar{a}b} Q' \wedge \alpha = \tau \wedge \\ \qquad\qquad\qquad R' = P'\{b/x\} \mid Q' \wedge x \sharp \mathcal{C} \Longrightarrow Prop \\ \forall P'\ Q'\ a\ x\ y.\ P \xrightarrow{a(x)} P' \wedge Q \xrightarrow{\bar{a}(y)} Q' \wedge y \sharp P \wedge \alpha = \tau\ \wedge \\ \qquad\qquad\qquad R' = (\nu y)(P'\{y/x\} \mid Q') \wedge x \sharp \mathcal{C} \wedge y \sharp \mathcal{C} \Longrightarrow Prop \end{array}}{Prop}$$

The two semantic rules which introduce bound names are the *Comm*- and the *Close* rules. The rule can be instantiated with an arbitrary finite set of context names $\mathcal{C}$ and these bound names will be set fresh for that set.

The second problem is that in case analysis, equivalence checks between terms always appear. If these terms contain bound names, such as $(\nu x)P = (\nu y)Q$, then normal unification is not possible. As seen in Prop. 1 and 2, every such equivalence check produces either two cases which both have to be proven or one case with several permutation and freshness conditions. As an example, a rule for case analysis on the ν-operator with no bound names in the action can be found in the following proposition:

Proposition 4. *The automatically generated case analysis rule for the ν-operator, based on Prop. 1, where no bound name occurs in the action.*

$$\frac{\begin{array}{c} (\nu x)P \xrightarrow{\alpha} P' \\ \forall Q\ Q'\ \beta\ y.\ Q \xrightarrow{\beta} Q' \wedge y \sharp \beta \wedge (\nu x)P = (\nu y)Q \wedge \alpha = \beta \wedge P' = (\nu y)Q' \Longrightarrow Prop \end{array}}{Prop}$$

The conjunct $(\nu x)P = (\nu y)Q$ poses a problem as we have to show *Prop* for all cases such that the equivalence holds. We can reason about this equality using either Prop. 1 or Prop. 2 but neither of these rules are convenient to work with. Prop. 1 causes a case explosion which forces us to prove the same thing several times for different permutations on terms and Prop. 2 introduces extra permutations which makes the proof more cumbersome to work with. We therefore use the following derived lemma in place of the original case analysis rule:

Lemma 2. *Case analysis rule derived from Prop. 4.*

$$\frac{\begin{array}{c} (\nu x)P \xrightarrow{\alpha} P' \\ \forall P''.\ P \xrightarrow{\alpha} P'' \wedge x \sharp \alpha \wedge P' = (\nu x)P'' \Longrightarrow Prop \end{array}}{Prop}$$

The main idea of the proof is to find a P'' which suitably depends on the universally quantified terms in the second assumption of the original proposition.

In this way, all checks for α-equivalence between agents have been abstracted away and we are left with one very simple case to work with. Similar rules have been derived from all generated induction rules, on the depth of inference as well as case analysis of all operators. The only operator which requires an induction rule rather than a case analysis rule is the !-operator as it is the only operator which occurs in the premise of its inference rule, as can be seen in Fig. 1. There is also an induction rule over all possible transitions.

5 Strong Bisimulation

Intuitively, two processes are said to be bisimilar if they can mimic each other step by step. Traditionally, a bisimulation is a symmetric binary relation $\mathcal{R}$ such that for all processes P and Q in $\mathcal{R}$, if P can do an action, then Q can mimic that action and their corresponding derivatives are in $\mathcal{R}$.

When defining bisimulation between two processes in the π-calculus, extra care has to be taken with respect to bound names in actions. Consider the following processes:

$$P \overset{\mathrm{def}}{=} a(u).(\nu b)\bar{b}x.0$$
$$Q \overset{\mathrm{def}}{=} a(x).0$$

Clearly P and Q should be bisimilar since they both can do only one input action along a channel a and then nothing more. But since x occurs free in P, P cannot be α-converted into $a(x).(\nu b)\bar{b}x$. However, since processes have finite support, there exists a name w which is fresh in both P and Q and after α-converting both processes, bisimulation is possible. Hence, when reasoning about bisimulation, we must restrict attention to the bound names of actions which are fresh for both P and Q. One of our main contributions is how this is achieved without running into a multitude of α-conversions.

Our formal definition of bisimulation equivalence uses the following notion, where $\mathcal{R}$ is a binary relation on agents.

Definition 5. *The agent P can simulate the agent Q preserving $\mathcal{R}$, written $P \leadsto_{\mathcal{R}} Q$, if*

$$(\forall a\ x\ Q'.\ Q \xrightarrow{a\ll x\gg} Q' \wedge x \sharp P \Longrightarrow$$
$$\exists P'.\ P \xrightarrow{a\ll x\gg} P' \wedge \mathit{derivative}(a,\ x,\ P',\ Q',\ \mathcal{R})) \wedge$$
$$(\forall \alpha\ Q'.\ Q \xrightarrow{\alpha} Q' \Longrightarrow \exists P'.\ P \xrightarrow{\alpha} P' \wedge (P', Q') \in \mathcal{R})$$

$$\mathit{derivative}(a,\ x,\ P',\ Q',\ \mathcal{R}) \overset{\mathit{def}}{=}$$
$$\quad \mathit{case}\ a\ \mathit{of}\ \mathit{Input}\ \_ \Rightarrow \forall u.\ (P'\{u/x\},\ Q'\{u/x\}) \in \mathcal{R}$$
$$\quad \mid \mathit{BoundOutput}\ \_ \Rightarrow (P',\ Q') \in \mathcal{R}$$

Note that the argument a in derivative is of type `subject` as described in Def. 3. Thus, the requirement is that if Q has an action then P has the same action, and the derivatives P' and Q' are in $\mathcal{R}$.

The traditional way to define strong bisimulation equivalence is to say that $\mathcal{R}$ is a *bisimulation* if it is symmetric and that for all agents P, Q it holds that $(P, Q) \in \mathcal{R} \rightarrow P \leadsto_{\mathcal{R}} Q$; the strong bisimulation equivalence is then the union of all strong bisimulations. As we shall see in a moment, an alternative definition using direct coinduction, similar to the approach in [6], yields shorter proofs. Our main improvement, however, is in the treatment of the bound name x. It is by definition ensured not to be among the free names in P, but when we use it within a complex proof we will run into a massive case analysis on whether x is equal to other names used in the proof. In the same way as in Lemma 1 we bypass this tedium and derive the following introduction rule for an arbitrary finite set of context names $\mathcal{C}$. This set is provided by the user to ensure that the bound name is distinct from any name occurring so far in the proof.

Lemma 3. *An introduction rule for simulation avoiding name clashes.*

$$\frac{\begin{array}{l}\text{eqvt } \mathcal{R} \\ (\forall a\ x\ Q'.\ Q \overset{a\ll x\gg}{\longrightarrow} Q' \wedge x \sharp \mathcal{C} \Longrightarrow \\ \qquad\qquad \exists P'.\ P \overset{a\ll x\gg}{\longrightarrow} P' \wedge \mathit{derivative}(a,\ x,\ P',\ Q',\ \mathcal{R})) \\ (\forall \alpha\ Q'.\ Q \xrightarrow{\alpha} Q' \Longrightarrow \exists P'.\ P \xrightarrow{\alpha} P' \wedge (P', Q') \in \mathcal{R})\end{array}}{P \leadsto_{\mathcal{R}} Q}$$

This is used extensively in our proofs. We can in this way make sure that whenever bound names appear in our proof context, these bound names do not clash with other names which would force us to do α-conversions. The amount of α-conversions we have to do manually is reduced to the instances where they would be required in a manual proof.

Note that we need an extra requirement that our simulation relation is equivariant. The reason is that if the relation is not closed under permutations, we cannot α-convert our processes. Fortunately, all relations of interest turn out to be equivariant and the proof trivial.

Bisimulation equivalence can now be described using coinduction, or as the greatest fixed point derived from a monotonic function.

Definition 6. *Bisimulation equivalence, $\sim$, is a* coinductive *definition.*

$$P \sim Q \overset{def}{=} P \leadsto_{\sim} Q \ \wedge\ Q \leadsto_{\sim} P$$

Note that we do not need to define what a bisimulation is; our coinductive definition uses $P \leadsto_{\mathcal{R}} Q$ directly. This defines $\sim$ to be the largest relation such that related agents can simulate each other preserving $\sim$. Conducting proofs on bisimulation equivalence often boils down to proving the same thing twice – once for each direction. With our formulation it is often easy to just prove one direction and let the other be inferred automatically.

When checking whether or not two processes are bisimilar, one picks a set $\mathcal{X}$ which contains the processes and which represents what it is we are trying to prove. It then suffices to show that all members of $\mathcal{X}$ are simulated preserving $\mathcal{X} \cup \sim$. The coinduction rule is automatically generated by Isabelle.

Strong bisimulation is not a congruence as it is not preserved by the input-prefix. We write $\mathcal{R}^s$ for the closure of the relation $\mathcal{R}$ under all substitutions.

Definition 7. *$P \mathcal{R}^s Q \overset{def}{=} \forall\sigma.\ P\sigma\ \mathcal{R}\ Q\sigma$ where σ is a chain of substitutions.*

From this we can define strong equivalence as the largest bisimulation relation closed under substitution. One of our main results is proving in Isabelle that strong equivalence is a congruence.

Theorem 1. *$\sim^s$ is a congruence.*

6 Weak Bisimulation

Weak bisimulation equivalence is often called observation equivalence. The intuition is that τ-transitions are considered internal and hence invisible to the outside environment. For two processes to be observation equivalent, they only need to mimic the visible actions of each other. More formally, we reason about a τ-chain $P \overset{\hat{\tau}}{\Longrightarrow} P'$ as the reflexive transitive closure of τ-actions, i.e. $P \overset{\hat{\tau}}{\Longrightarrow} P' \overset{\text{def}}{=} P \xrightarrow{\tau}^* P'$. A weak transition is then said to be an action preceded and succeeded by a τ-chain.

In the simulation of an input, weak late simulation is complicated. It requires substitutions made as a result of the input to be applied immediately to the input derivative before the succeeding τ-chain is executed, and that one such derivative can continue to simulate for all possible received names, see e.g. [12]. Therefore the weak late semantics needs to carry additional information in the labels as follows.

Definition 8. *Weak late transitions*

$$P \overset{\alpha}{\Longrightarrow} P' \overset{def}{=} \exists P''\ P'''.\ P \overset{\hat{\tau}}{\Longrightarrow} P''' \wedge P''' \xrightarrow{\alpha} P'' \wedge P'' \overset{\hat{\tau}}{\Longrightarrow} P'$$
$$P \overset{\bar{a}(x)}{\Longrightarrow} P' \overset{def}{=} \exists P''\ P'''.\ P \overset{\hat{\tau}}{\Longrightarrow} P''' \wedge P''' \xrightarrow{\bar{a}(x)} P'' \wedge P'' \overset{\hat{\tau}}{\Longrightarrow} P'$$
$$P \overset{u:a(x)@P''}{\Longrightarrow} P' \overset{def}{=} \exists P'''.\ P \overset{\hat{\tau}}{\Longrightarrow} P''' \wedge P''' \xrightarrow{a(x)} P'' \wedge P''\{u/x\} \overset{\hat{\tau}}{\Longrightarrow} P'$$

As with our previous transitions, we let α range over free actions with no bound names. Note that the bound name x in the bound output case is bound in P' and normal α-conversions can be applied. Also, even though we are modeling a late semantics, the name x is *not* bound in P' in the input-transition as it is substituted for u before the τ-chain. We can still do α-conversions through the following lemma:

Lemma 4. *if $P \overset{u:a(x)@P''}{\Longrightarrow} P'$ and $y \sharp P$ then $P \overset{u:a(y)@(x\ y)\bullet P''}{\Longrightarrow} P'$*

We also need to weaken the transitions in the standard way:

Definition 9. $P \overset{\hat{\alpha}}{\Longrightarrow} P' \overset{def}{=} P \overset{\hat{\tau}}{\Longrightarrow} P'$ *if* $\alpha = \tau$
$P \overset{\alpha}{\Longrightarrow} P'$ *otherwise*

We can now define weak late simulation.

Definition 10. *The agent P can weakly late simulate the agent Q preserving $\mathcal{R}$, written $P \gtrsim_{\mathcal{R}} Q$, if*

$$\begin{aligned}
&(\forall a\ x\ Q'.\ Q \xrightarrow{\bar{a}(x)} Q' \wedge x \sharp P \Longrightarrow\\
&\qquad\qquad \exists P'.\ P \overset{\bar{a}(x)}{\Longrightarrow} P' \wedge (P',\ Q') \in \mathcal{R}) \wedge\\
&(\forall a\ x\ Q'.\ Q \xrightarrow{a(x)} Q' \wedge x \sharp P \Longrightarrow\\
&\qquad\qquad \exists P''.\ \forall u.\ \exists P'.\ P \overset{u:a(x)@P''}{\Longrightarrow} P' \wedge (P',\ Q'\{u/x\}) \in \mathcal{R}) \wedge\\
&(\forall \alpha\ Q'.\ Q \xrightarrow{\alpha} Q' \Longrightarrow \exists P'.\ P \overset{\hat{\alpha}}{\Longrightarrow} P' \wedge (P',\ Q') \in \mathcal{R})
\end{aligned}$$

The important aspect of weak late simulation is the fact mentioned above – that an input-action $a(x)$ must be matched by a weak transition with the same input derivative P'' for *all* possible instantiations u of the bound name. From our definition, we can derive an introduction rule for weak simulation similar to the one done for strong simulation in Lemma 3.

Weak bisimulation equivalence is defined using coinduction in exactly the same way as strong bisimulation.

Definition 11. *Weak bisimulation equivalence, $\approx$, is a* coinductive *definition.*
$P \approx Q \overset{def}{=} P \ggtrsim_{\approx} Q\ \wedge\ Q \ggtrsim_{\approx} P$

Weak bisimulation is not a congruence since it is neither preserved by the +-operator nor by the input-prefix, but it is preserved by all other operators. The first step in in this proof is to lift the operational semantics to a weak context, that is, for every operational semantics rule we derive a weak counterpart. This means that the proof strategies for strong equivalence carry over to weak equivalence. As an example Lemma 5 derives the weak operational semantics for the |-operator.

Lemma 5

$$\frac{P \overset{u:a(x)@P''}{\Longrightarrow} P' \qquad x \sharp Q}{P \mid Q \overset{u:a(x)@P''|Q}{\Longrightarrow} P' \mid Q}\ \textbf{ParIn}$$

$$\frac{P \overset{\bar{a}(x)}{\Longrightarrow} P' \qquad x \sharp Q}{P \mid Q \overset{\bar{a}(x)}{\Longrightarrow} P' \mid Q}\ \textbf{ParBO} \qquad\qquad \frac{P \overset{\hat{\alpha}}{\Longrightarrow} P'}{P \mid Q \overset{\hat{\alpha}}{\Longrightarrow} P' \mid Q}\ \textbf{ParF}$$

$$\frac{P \overset{b:a(x)@P''}{\Longrightarrow} P' \qquad Q \overset{\bar{a}b}{\Longrightarrow} Q'}{P \mid Q \overset{\tau}{\Longrightarrow} P' \mid Q'}\ \textbf{Comm} \quad \frac{P \overset{y:a(x)@P''}{\Longrightarrow} P' \qquad Q \overset{\bar{a}(y)}{\Longrightarrow} Q' \qquad y \sharp P}{P \mid Q \overset{\tau}{\Longrightarrow} (\nu y)(P' \mid Q')}\ \textbf{Close}$$

All operational rules cannot be lifted in this manner. The rules from Fig. 1 where we cannot do this are *Match*, *Sum* and *Replication* in the case where $\alpha = \tau$ and $P = P'$. In order to prove preservation of *Match* and *Replication*, we have to prove that $P \approx [a = a]P$ and $P \mid\ !P \approx\ !P$.

To obtain a congruence we follow the standard procedure. We define weak congruence simulations, $\ggtrsim$, in the same way as weak simulations, Def. 10, but for the initial free transitions, we replace $\overset{\hat{\alpha}}{\Longrightarrow}$ with $\overset{\alpha}{\Longrightarrow}$. In other words, the simulating process must match at least the first action from the other process, even invisible ones. We have proven that it is possible to lift all the operational rules from Fig 1 to $\overset{\alpha}{\Longrightarrow}$.

Definition 12. $P \cong Q \overset{def}{=} P \mathrel{\overset{\bullet}{\Rightarrow}}_{\approx} Q \;\wedge\; Q \mathrel{\overset{\bullet}{\Rightarrow}}_{\approx} P$.

Note that this is not a recursive definition since it refers to $\approx$. The proof that $\cong$ is preserved by all operators except input-prefix corresponds closely to our corresponding proof for $\sim$. The proof that $\cong^s$ is a congruence follows in the same manner.

Theorem 2. *$\cong^s$ is a congruence.*

7 Early Semantics

In the early semantics the input action carries the name received rather than a bound name, so we have $a(x).P \xrightarrow{au} P\{u/x\}$ for all u. We have created transition systems for both early and late operational semantics. The proof strategies involved for dealing with the two different approaches are nearly identical, even though the actual theories are disjoint. The connection we have proved between them is that every early τ-transition has a corresponding late τ-transition and vice versa.

The derived early and late weak semantics are much more similar to each other then their strong counterparts. The reason for this is that in the weak late operational semantics, the instantiations of input bound names occurs inside the transition before the succeeding τ-chain. This becomes apparent when we look at our lifted rule for input-actions:

Lemma 6. $a(x).P \overset{u:a(x)@P}{\Longrightarrow} P\{u/x\}$.

Even in our late semantics this looks like an early transition since it contains the name u received in the input. The difference between weak early and late semantics is not so much in the transition system, but in the definition of simulation.

All proofs that we have done for the late semantics, simulation- and bisimulation relations have been done also for the early semantics. We have also proven that all late bisimulation relations that we have considered are included in their early counterparts.

8 Results and Conclusions

8.1 Current Status

We have used the new nominal datatype package in Isabelle to model the π-calculus and our results are very encouraging. We have proved a substantial part of [9], in particular preservation properties of strong and weak bisimulation, and both late and early. Other results include that all all late τ-transitions have a corresponding early one and vice versa and that all late bisimulation relations have an early counterport. Moreover, we have proven that all structurally congruent terms are bisimilar using both early and late semantics. We have created a substantial library concerning the fundamental mechanism in the π-calculus, such as substitution and transitions. One of our main contributions is that the proofs resemble the ones on paper very closely, since we make precise the traditional "hand waving" with respect to bound names. Since we are using Isabelle, we can write our proofs in a very readable form using *Isar* [21]. We believe this to be the most extensive formalisation of a process calculus ever to be done inside a theorem prover.

The nominal package is still work in progress and it is constantly being updated. One very recent addition allows for users to define functions on their nominal datatypes using an automatically generated recursion combinator [16]. At the moment we only use substitution as a function (both single and sequential).

8.2 Related Work

The π-calculus has been subject for many attempts at formalisations. Early sketches in HOL include [10,7]. Later attempts have also been made using de-Bruijn indices where names are encoded using natural numbers. More recent work by Gabbay utilised FM set theory [4], the precursor of nominal logic, although this attempt was later abandoned. The most extensively used approach is higher order abstract syntax (HOAS) where weak HOAS is the technique most similar to ours.

de Bruijn indices are heavily used in software which reasons about terms with binders; an example for the π-calculus is the Mobility Workbench [20]. They work well in these environments as they have very nice algorithmic properties. However, these properties do not provide an intuitive mathematical framework.

Fraenkel Mostowski set theory was one of the first serious attempts to fomalise nominal logic. It is standard ZF set theory but with an extra freshness axiom added. In [4], Gabbay formalises a portion of the π-calculus in FM. Unfortunately, this early version of nominal logic was incompatible with the axiom of choice and had to be used in Isabelle/PURE – a bare boned set of theories without much support for anything. This choice of framework was necessary since Isabelle/HOL contains the axiom of choice which is inconsistent with FM.

Higher order abstract syntax (HOAS) is the approach most similar to ours. It has been used to model the π-calculus in both Coq [6], by Honsell et. al., and in Isabelle by Röckl and Hirschkoff [15]. When using HOAS terms, binders are represented as functions of type `name->term`. However, if these functions range over the entire function space they may produce exotic terms, so the formalisations need to ensure that those are avoided. In [15], a special well-formedness predicate is used to filter out the exotic terms. Another problem is that since abstraction is handled by the meta-logic of the theorem prover, reasoning about binders at the object level can become problematic. In [6] we can read:

> The main drawback in HOAS is the difficulty of dealing with metatheoretic issues concerning names in process contexts, *i.e.* terms of type `name->proc`. As a consequence, some metatheoretic properties involving substitution and freshness of names inside proofs and processes, cannot be proved inside the framework and instead have to be postulated.

Our approach is completely free from any extra axioms, and since nominal logic is a first order approach we do not have to worry about exotic terms. Moreover, freshness conditions are part of the nominal infrastructure and all such conditions are explicitly known at the object level and do not have to be postulated, thus no extra infrastructure for choosing particular names is needed.

8.3 Impact and Further Work

Theorem provers suffer from a somewhat well-deserved reputation of being hard to use for the uninitiated. However, having theories formalised by a computer has significant advantages and making theorem provers easy to use for the general engineer is a high priority. We believe that our work helps in this venture. The challenging part has been to create inductive rules and easy-to-use definitions for simulation and bisimulation. With this done the actual proofs done in the theorem prover are not much harder than the ones done on paper.

Our next goal will be to provide support for model- and bisimulation checking on actual protocols such as ad-hoc routing. Particularly processes with infinite state space are of interest as these cannot be handled by automatic tools like the Mobility Workbench.

There are several variants of the π-calculus, polyadic π-calculus and higher order π-calculus just to name two. We believe that our definitions for simulation and bisimulation can easily be transfered to many other calculi.

Acknowledgements. We would like to thank Stefan Berghofer for his generous help with the understanding of the inner workings of Isabelle, Christian Urban for developing the nominal datatype package and providing extensive support and insights and Lars-Henrik Eriksson for discussions on theorem provers.

References

1. Brian E. Aydemir, Aaron Bohannon, Matthew Fairbairn, Nathan J. Foster, Benjamin C. Pierce, Peter Sewell, Dimitrios Vytiniotis, Geoffrey Washburn, Stephanie Weirich, and Steve Zdancewic. Mechanized metatheory for the masses: The POPLmark challenge. In *International Conference on Theorem Proving in Higher Order Logics (TPHOLs)*, August 2005.
2. Jesper Bengtson. Generic implementations of process calculi in Isabelle. In *The 16th Nordic Workshop on Programming Theory (NWPT'04)*, pages 74–78, 2004.
3. M. J. Gabbay. A theory of inductive definitions with α-equivalence, PhD thesis, University of Cambridge, 2000.
4. M. J. Gabbay. The π-calculus in FM. In Fairouz Kamareddine, editor, *Thirty-five years of Automath*. Kluwer, 2003.
5. M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2001.
6. Furio Honsell, Marino Miculan, and Ivan Scagnetto. π-calculus in (co)inductive type theory. *Theoretical Computer Science*, 253(2):239–285, 2001.
7. Thomas F. Melham. A mechanized theory of the pi-calculus in HOL. *Nordic Journal of Computing*, 1(1):50–76, 1994.

8. R. Milner. *A Calculus of Communicating Systems.* Number 92 in LNCS. Springer-Verlag, 1980.
9. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I/II. *Inf. Comput.*, 100(1):1–77, 1992.
10. Otmane Aït Mohamed. Mechanizing a pi-calculus equivalence in HOL. In *Proceedings of the 8th International Workshop on Higher Order Logic Theorem Proving and Its Applications*, pages 1–16, London, UK, 1995. Springer-Verlag.
11. T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: a proof assistant for higher-order logic.* Springer-Verlag, 2002.
12. Joachim Parrow. An introduction to the pi-calculus. In *Handbook of Process Algebra*, pages 479–543. Elsevier, 2001.
13. A. M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186:165–193, 2003.
14. A. M. Pitts. Alpha-structural recursion and induction. *Journal of the ACM*, 53:459–506, 2006.
15. Christine Röckl and Daniel Hirschkoff. A fully adequate shallow embedding of the π-calculus in Isabelle/HOL with mechanized syntax analysis. *J. Funct. Program.*, 13(2):415–451, 2003.
16. C. Urban and S. Berghoffer. A recursion combinator for nominal datatypes implemented in Isabelle/HOL, Accepted to IJCAR 2006.
17. C. Urban, A. M. Pitts, and M. J. Gabbay. Nominal unification. *Theoretical Computer Science*, 323:473–497, 2004.
18. Christian Urban and Michael Norrish. A formal treatment of the barendregt variable convention in rule inductions. In *MERLIN '05: Proceedings of the 3rd ACM SIGPLAN workshop on Mechanized reasoning about languages with variable binding*, pages 25–32, New York, NY, USA, 2005. ACM Press.
19. Christian Urban and Christine Tasson. Nominal techniques in Isabelle/HOL. In *CADE*, pages 38–53, 2005.
20. Björn Victor and Faron Moller. The Mobility Workbench — a tool for the π-calculus. In David Dill, editor, *CAV'94: Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 428–440. Springer-Verlag, 1994.
21. Markus Wenzel. Isar - a generic interpretative approach to readable formal proof documents. In *Theorem Proving in Higher Order Logics*, pages 167–184, 1999.

The Rewriting Calculus as a Combinatory Reduction System

Clara Bertolissi[1] and Claude Kirchner[2]

[1]LIF-CMI, Université de Provence, Marseille, France
[2]INRIA & LORIA*, Nancy, France
`first.last@loria.fr, clara.bertolissi@lif.univ-mrs.fr`

Abstract. The last few years have seen the development of the *rewriting calculus* (also called rho-calculus or ρ-calculus) that uniformly integrates first-order term rewriting and λ-calculus. The combination of these two latter formalisms has been already handled either by enriching first-order rewriting with higher-order capabilities, like in the *Combinatory Reduction Systems* (CRS), or by adding to λ-calculus algebraic features.

In a previous work, the authors showed how the semantics of CRS can be expressed in terms of the ρ-calculus. The converse issue is adressed here: rewriting calculus derivations are simulated by Combinatory Reduction Systems derivations. As a consequence of this result, important properties, like standardisation, are deduced for the rewriting calculus.

Introduction

Lambda calculus and term rewriting are two foundational frameworks that had a deep influence on the development of computation and deduction. Starting from Klop's ground-breaking work on higher-order rewriting [15], and because of their complementarity, many frameworks have been designed with a view to integrate these two formalisms.

Introduced in the late nineties (see e.g. [7,8]), the rewriting calculus, also denoted ρ-calculus, combines uniformly the two paradigms and allows us to write $\lambda x.t$ to abstract, like in the λ-calculus, over the variable x, but also $\lambda p.t$ to abstract over an elaborated pattern p. Indeed, this last ρ-term is also written $p \twoheadrightarrow t$, emphasizing the rewriting aspect. This general abstract mechanism has been shown to be quite expressive and useful. For instance, it is well adapted to describe the semantics of imperative languages and object calculi [17,9] and the ρ-calculus has been used to model the execution of rewrite rules and strategies in rule-based languages like ELAN [10]. The logical aspects of the ρ-calculus are also quite appealing and provide the foundation for the design of a new class of proof assistants where computation and deduction can be adapted to the user's needs and understandings [22,4,12].

We are interested here in a better understanding of the behavior of the rewriting calculus by analyzing its derivation space. To this aim, we present an encoding of the ρ-calculus into CRSs, since for this kind of higher-order systems a

* UMR 7503 CNRS-INPL-INRIA-Nancy2-UHP.

well-developed meta-theory already exists. Studies on the comparison between different higher-order formalisms has already been conducted between *Combinatory Reduction Systems* and *Higher-order Rewrite Systems* in [21]. Moreover, an encoding of CRSs into the rewriting calculus has already been proposed by the authors in [3]. This paper is concerned with the analysis of the converse relation, *i.e.* the study of derivations performed in rewriting calculus and their equivalent in higher-order rewriting, in particular in Combinatory Reduction Systems. We define a translation of the components of the ρ-calculus into the analogous notions in a CRS. Using this translation, we show that every derivation of a ρ-term has a corresponding derivation in the CRS and we prove the soundness and completeness of this encoding. We conclude by deriving some important properties concerning rewriting calculus reductions, as confluence, finiteness of developments and standardization, by using the well-known corresponding results in the CRSs.

The paper is structured as follows: in Section 1 we briefly present the ρ-calculus through its components. Section 2 provides a description of CRSs and some examples. In Section 3 we present a translation from ρ-terms and evaluation rules into CRS-terms and CRS-rewrite rules respectively, and we prove the completeness and soundness of the translation. Section 4 concludes the paper with some additional remarks and perspectives.

1 The Rewriting Calculus

We briefly present in what follows the syntax and the semantics of the basic ρ-calculus. For a more detailed presentation the reader can refer to [8].

In this paper, the symbols $t, u, \ldots$ range over the set $\mathcal{T}$ of terms, the symbols $x, y, z \ldots$ range over the infinite set $\mathcal{X}$ of variables and the symbols $f, g, \ldots$ of fixed arity range over the infinite set $\mathcal{F}$. Finally, the symbols p, q range over the set of patterns $\mathcal{P} \subseteq \mathcal{T}$. All symbols can be indexed. Syntactic equality is denoted by $\equiv$. We consider the meta-symbols "$\lambda \_ . \_$" (abstraction operator), and "$\_ \wr \_$" (structure operator), and the (hidden) application operator. The set of ρ-terms is then defined as follows:

$$\begin{aligned} \mathcal{T} &::= \mathcal{X} \mid \mathcal{F} \mid \lambda\mathcal{P}.\mathcal{T} \mid \mathcal{T}\;\mathcal{T} \mid \mathcal{T} \wr \mathcal{T} \\ \mathcal{P} &::= \mathcal{X} \mid \mathcal{F} \mid \mathcal{F}\,\mathcal{P} \ldots \mathcal{P} \end{aligned}$$

A term of the form $\lambda p.t$ is an *abstraction* with pattern p and body t. The term $t_1 \wr t_2$ is a *structure* consisting of the two terms t_1 and t_2. The set of patterns $\mathcal{P}$ is a parameter of the calculus and in full generality it could be as large as the set of all terms $\mathcal{T}$. We call *algebraic* the patterns used in this version of the calculus and we usually denote a term of the form $(\ldots((f\;\; t_1)\;\; t_2)\ldots)\;\; t_n$ with $f \in \mathcal{K}$ of arity n by $f(t_1, t_2, \ldots, t_n)$. A *linear* pattern is a pattern where every variable occurs at most once. In the rest of the paper we will consider only *well-formed* terms, *i.e.* terms where functional symbols are provided with the correct number of arguments, according to their arity. Moreover, we will restrict to linear patterns, a standard restriction that will allow us to reuse the properties of CRSsthat are indeed, in general, assuming left linearity.

We assume that the application operator associates to the left, while the other operators associate to the right. The priority of the application is higher than that of "$\lambda\_.\_$" which is, in turn, of higher priority than the "$\_ \wr \_$".

Let $\Box$ be a fresh symbol, called a *hole*. A term with one or more occurrences of $\Box$ is called a *context* and denoted by $\mathsf{Ctx}_{\lceil\ \rceil}$. An *algebraic* context is an algebraic term with one or more occurrences of $\Box$. A *ground* context is a context containing no variables. The term obtained by replacing from left to right in a context $\mathsf{Ctx}_{\lceil\ \rceil}$ the n holes $\Box$ by the terms $t_1, \ldots, t_n$, $n \geq 1$, is denoted by $\mathsf{Ctx}_{\lceil t_1,\ldots,t_n \rceil}$.

Similarly as in the λ-calculus, the "$\lambda\_.\_$" operator is a binder of the calculus, *i.e.* in the term $\lambda p.t$ the free variables of p are bound in t. Formally:

Definition 1 (Free variables). *The set of free variables of a ρ-term t, denoted $\mathcal{FV}(t)$ is inductively defined as follows:*

$$\begin{array}{ll} \mathcal{FV}(f) = \{\ \} & \mathcal{FV}(t_1\, t_2) = \mathcal{FV}(t_1) \cup \mathcal{FV}(t_2) \\ \mathcal{FV}(x) = \{x\} & \mathcal{FV}(t_1 \wr t_2) = \mathcal{FV}(t_1) \cup \mathcal{FV}(t_2) \\ \mathcal{FV}(\lambda p.t) = \mathcal{FV}(t) \setminus \mathcal{FV}(p) & \end{array}$$

The set $\mathcal{BV}$ of bound variables of a term is the complementary of the set of free variables w.r.t. the set of variables of the respective term. A term is called closed *if all its variables are bound.*

Example 1 (ρ-terms)

1. $(\lambda x.x\, x)\, (\lambda x.x\, x)$ is the ρ-term corresponding to the λ-term $\omega\, \omega$;
2. The ρ-term $(\lambda plus(x, 0).x)\ plus(n, 0)$ encodes the application of the rewrite rule $x + 0 \rightarrow x$ to the term $n + 0$;
3. The ρ-term $(\lambda f(a).a \wr \lambda f(a).b)$ represents the rewrite system consisting of the two rules $f(a) \rightarrow a$ and $f(a) \rightarrow b$.

The classical notion of simultaneous substitution used in higher-order calculi, like the λ-calculus, can be adapted to the ρ-calculus.

Definition 2 (Substitution). *A substitution σ is a mapping from the set of variables to the set of terms. A finite substitution has the form $\sigma = \{x_1/t_1 \ldots x_m/t_m\}$, also denoted $\sigma = \{\overline{x}/\overline{t}\}$, where $\mathcal{D}om(\sigma) = \{x_1, \ldots, x_m\}$. Applying a substitution σ to a term t, denoted by $\sigma(t)$ or $t\sigma$, is defined as follows:*

$$\begin{array}{ll} \sigma(f) = f & \sigma(\lambda p.t) = \lambda p.\sigma(t) \\ \sigma(x_i) = \begin{cases} t_i & \text{if } x_i \in Dom(\sigma) \\ x_i & \text{otherwise} \end{cases} & \begin{array}{l} \sigma(t_1\, t_2) = \sigma(t_1)\ \sigma(t_2) \\ \sigma(t_1 \wr t_2) = \sigma(t_1) \wr \sigma(t_2) \end{array} \end{array}$$

We point out that we work modulo α*-convention*: when applying a substitution to an abstraction, we know that the free variables of the corresponding abstracted pattern do not belong to the domain of the substitution.

The evaluation mechanism of the calculus relies on the fundamental operation of *matching* that allows us to instantiate variables by their current values. We can use different matching theories for computing the matching substitutions like, for example, an empty theory, an equational theory or even more elaborated (higher-order matching) theories [9]. In this paper, we will restrict to syntactic matching problems, which have at most one solution and are known to be decidable [13].

Definition 3 (Syntactic matching). *A (syntactic) matching problem is a formula of the form $p \prec\!\!\prec t$, where p is a pattern and t is a term. A substitution σ is solution of the matching problem $p \prec\!\!\prec t$, denoted by $Sol(p \prec\!\!\prec t)$, if $\sigma(p) \equiv t$.*

The small-step reduction semantics of the ρ-calculus is defined by the following reduction rules:

$$\begin{array}{llll} (\rho) & (\lambda p.t_2)t_3 & \to_\rho & \sigma(t_2) \qquad \textit{where } \sigma = Sol(p \prec\!\!\prec t_3) \\ (\delta) & (t_1 \wr t_2)\, t_3 & \to_\delta & t_1\, t_3 \wr t_2\, t_3 \end{array}$$

The (ρ)-rule can be applied if (and only if) a substitution of the matching problem $p \prec\!\!\prec t_3$ exists. In this case, the result of the (ρ)-rule is the application of this substitution to the term t_2. If such a substitution does not exist, then the (ρ)-rule does not apply and the term is left as it is. Nevertheless, further reductions or instantiations are likely to modify t_3 so that the appropriate substitution can be found and the rule can be fired. The (δ)-rule right-distributes the application over the structures. This gives the possibility, for example, to apply in parallel two distinct pattern abstractions to a given term.

As usual, we introduce the classical notions of one-step, many-steps, and congruence with respect to the relation $\to_{\rho\delta}$ induced by the top-level rules of ρ-calculus. The one-step evaluation $\mapsto_{\rho\delta}$ is the contextual closure of $\to_{\rho\delta}$; if we want to specify the position ω at which the rewrite steps occurs, we write $\mapsto^{\omega}_{\rho\delta}$. The many-step evaluation $\mapsto\!\!\!\!\to_{\rho\delta}$ is defined as the reflexive and transitive closure of $\mapsto_{\rho\delta}$.

Example 2 (Reductions). We consider the ρ-terms of Example 1 and we show their respective reductions.

1. $(\lambda x.x\ x)\ (\lambda x.x\ x) \mapsto_\rho (\lambda x.x\ x)\ (\lambda x.x\ x) \mapsto_\rho \ldots$ is the infinite ρ-reduction corresponding to the reduction of the λ-term $\omega\,\omega$;
2. Since $Sol(plus(x,0) \prec\!\!\prec plus(n,0)) = \{n/x\}$, we have the reduction $(\lambda plus(x,0).x)\, plus(n,0) \mapsto_\rho n$;
3. $(\lambda f(a).a \wr \lambda f(a).b)\ f(a) \mapsto_\delta (\lambda f(a).a)\ f(a) \wr (\lambda f(a).b)\ f(a) \mapsto\!\!\!\!\to_\rho a \wr b$ is a ρ-reduction capturing the non-determinism of first-order term rewriting.

2 The Combinatory Reduction Systems

The *Combinatory Reduction Systems* (CRSs), introduced by J.W. Klop in 1980 [15], are a generalization of first-order term rewrite systems with a mechanism of bound variables like in the λ-calculus. The definitions of this section are based on the presentation of CRSs given in [16].

In what follows the symbols $A, L, R, \ldots$ range over the set $\mathcal{MT}$ of so called *meta-terms*, $t, u, \ldots$ range over the set $\mathcal{T}_{\text{CRS}}$ of *terms*, $x, y, z, \ldots$ range over the set

$\mathcal{X}$ of *variables*, $X, Z \ldots$ range over the set $\mathcal{Z}$ of *meta-variables* of fixed arity and $f, g, \ldots$ range over the set $\mathcal{F}$ of *functional symbols* of fixed arity. We denote by $\mathcal{F}_i \subset \mathcal{F}$ ($\mathcal{Z}_i \subset \mathcal{Z}$) the subset of symbols (meta-variables) of arity i. All symbols can be indexed. The set of CRS-meta-terms is defined as follows:

$$\mathcal{MT} ::= \mathcal{X} \mid \mathcal{F}_n(\mathcal{MT}_1, \ldots, \mathcal{MT}_n) \mid \mathcal{Z}_n(\mathcal{MT}_1, \ldots, \mathcal{MT}_n) \mid [\mathcal{X}]\mathcal{MT}$$

The set $\mathcal{T}_{\text{CRS}} \subset \mathcal{MT}$ of CRS-terms is composed of all the meta-terms without meta-variables. We should point out that all meta-terms are *well-formed*, *i.e.* the functional symbols and meta-variables take exactly as many arguments as their arity. A CRS context is defined similarly as in the ρ-calculus.

The operator $[\_]\_$ denotes an abstraction similar to the abstraction of the λ-calculus such that in $[x]t$ the variable x is bound in t. In a meta-term of the form $[x]A$ we call A the scope of $[x]$. A variable x occurs *free* in a meta-term if it is not in the scope of an occurrence of $[x]$. A variable x occurs *bound* otherwise. The set of free variables of a meta-term A is written $\mathcal{FV}(A)$.

Meta-variables (in the CRS rewrite rules defined below) behave as (free) variables of first-order rewrite systems. The set of meta-variables of a meta-term A is written $\mathcal{MV}(A)$. As for the ρ-calculus, we work modulo the α-conversion.

Example 3 (Terms and Metaterms). Some examples of terms and metaterms:

- $f([x]g(x,a)) \in \mathcal{T}_{\text{CRS}}$ with $f \in \mathcal{F}_1$, $g \in \mathcal{F}_2$, $a \in \mathcal{F}_0$.
- $Z_1(Z_2) \in \mathcal{MT}$ with $Z_1 \in \mathcal{Z}_1$, $Z_2 \in \mathcal{Z}_0$.
- $f([x]Z(x,y)) \in \mathcal{MT}$ with $f \in \mathcal{F}_1$, $Z \in \mathcal{Z}_2$.

The application of substitutions is defined at the meta-level of the calculus and uses $\underline{\lambda}$-calculus as meta-language (underlined just for distinguishing it from classical λ-calculus). Unintended bindings of variables by the $\underline{\lambda}$-abstractor operator are avoided using α-conversion. To simplify the notation we denote $\underline{\lambda}x_1 \ldots \underline{\lambda}x_n.t$ by $\underline{\lambda}x_1 \ldots x_n.t$. The reduction of $\underline{\lambda}$-redexes is performed by the $\underline{\beta}$-rule of the $\underline{\lambda}$-calculus. The $\underline{\beta}$-normal form of a term t is denoted by $\downarrow_{\underline{\beta}}$. We should point out that a CRS-(meta)term is necessarily in $\underline{\beta}$-normal form.

Performing a substitution in a CRS corresponds to applying an *assignment* (and consequently a set of substitutes) to a CRS-meta-term.

Definition 4 (Substitute, assignment)
An n-ary substitute *is an expression of the form* $\xi = \underline{\lambda}x_1 \ldots x_n.u$ *where* $x_1, \ldots, x_n$ *are distinct variables and* u *is a* CRS*-term. Its application to an n-tuple of* CRS*-terms* $t_1, \ldots, t_n$ *yields the simultaneous substitution of* x_i *by* t_i *in* u, $i = 1 \ldots n$, *denoted* $(\underline{\lambda}x_1 \ldots x_n.u)(t_1, \ldots, t_n)\downarrow_{\underline{\beta}} = u\{x_1/t_1, \ldots, x_n/t_n\}$.

An assignment $\sigma = \{(Z_1, \xi_1), \ldots, (Z_n, \xi_n)\}$, *is a finite set of pairs (metavariable, substitute) such that arity(Z_i) = arity(ξ_i)* $\forall i \in \{1, \ldots, n\}$ *(Dom(σ) = $\{Z_1, \ldots, Z_m\}$). The application of an assignment σ to a* CRS*-meta-term A, denoted $\sigma(A)$ or $A\sigma$, is inductively defined by:*

$$\begin{array}{lll}
\sigma(x) = x & & \sigma([x]A) = [x]\sigma(A) \\
\sigma(Z_i) = \xi_i & \textit{if } (Z_i, \xi_i) \in \sigma & \sigma(f(A_1, \ldots, A_n) = f(\sigma(A_1), \ldots, \sigma(A_n)) \\
\sigma(Z_i) = Z_i & \textit{if } Z_i \notin \textit{Dom}(\sigma) & \sigma(Z_i(A_1, \ldots, A_n)) = \sigma(Z_i)(\sigma(A_1), \ldots, \sigma(A_n))\downarrow_{\underline{\beta}}
\end{array}$$

Notice that the assignments have no effect on variables since they can instantiate only meta-variables. Since we work modulo the α-convention, unintended bindings of free variables are avoided by renaming bound variables.

A CRS rewrite rule is a pair of metaterms. We consider as left-hand side of the rules only the CRS-meta-terms satisfying the CRS-pattern definition:

Definition 5 (CRS-pattern). *A* CRS-*metaterm P is a* CRS-pattern *if any of its metavariables Z appears in a sub-metaterm of P of the form $Z(x_1, \ldots, x_n)$ where the variables $x_1, \ldots, x_n$, $n \geq 0$, are distinct and all bound in P.*

In this paper we only consider rewrite rules with CRS-patterns as left-hand sides and satisfying the usual conditions imposed in first-order rewriting:

Definition 6 (Rewrite rules). *A set of* CRS *rewrite rules consists of rules of the form $L \to R$ satisfying the following conditions:*

- *L and R are closed metaterms ($\mathcal{FV}(L) = \mathcal{FV}(R) = \emptyset$);*
- *L has the form $f(A_1, \ldots, A_n)$ with $A_1, \ldots, A_n$ metaterms and $f \in \mathcal{F}_n$;*
- *$\mathcal{MV}(L) \supseteq \mathcal{MV}(R)$;*
- *L is a* CRS-*pattern.*

The last condition ensures the decidability and the uniqueness of the solution of the matching inherent to the application of the CRS-rules [3]. Moreover, the additional condition of linearity can be required for the metaterm L, meaning that L contains no multiple occurrences of the same metavariable.

Example 4 (β-rule in CRS$_s$*).* The β-rule of λ-calculus $(\lambda x.t)u \to_\beta t\{x/u\}$ can be expressed as the rewrite rule: $App(Ab([x]Z(x)), Z_1) \to Z(Z_1)$. In this rule, called β_{CRS} in this paper, $App \in \mathcal{F}_2$ and $Ab \in \mathcal{F}_1$ are the encodings for the λ-calculus application and abstraction operators respectively.

Given a rewrite rule $L \to R$ and a substitution σ, we have $\sigma(L) \to_{L \to R} \sigma(R)$ if $\sigma(L), \sigma(R) \in \mathcal{T}_{\text{CRS}}$. The term $\sigma(L)$ is called a *redex*. The left-hand side and the right-hand side of a CRS rewrite rule are metaterms, but the rewrite relation induced by the rule is a relation on terms.

Given a set of CRS rewrite rules $\mathcal{R}$, the corresponding one-step relation $\mapsto_{\mathcal{R}}$ (denoted also $\mapsto_{L \to R}$ if we want to specify the applied rule) is the context closure of the relation induced (as above) by the rules in $\mathcal{R}$. The multi-step evaluation $\mapsto\!\!\!\!\to_{\mathcal{R}}$ is defined as the reflexive and transitive closure of $\mapsto_{\mathcal{R}}$.

Example 5. Let us consider the CRS-term $f(App(Ab([x]f(x)), a))$. We apply to the sub-term $App(Ab([x]f(x)), a)$ the β_{CRS} rule (Example 4) using the assignment $\sigma = \{(Z, \underline{\lambda} y.f\ y), (Z_1, a)\}$. As result, we obtain the instantiation by σ of the right-hand side R of the rule β_{CRS}: $\sigma(R) = \sigma(Z(Z_1)) = (\sigma(Z))(\sigma(Z_1)) = (\underline{\lambda} y.f\ y)(a){\downarrow_{\underline{\beta}}} = f(a)$. Therefore we have $App(Ab([x]f(x)), a) \mapsto_{\beta_{CRS}} f(a)$ and thus $f(App(Ab([x]f(x)), a)) \mapsto_{\beta_{CRS}} f(f(a))$.

One can notice that there are two binding mechanisms in the formalism presented in this section. The first one is explicit in the syntax and denoted $[x]t$. The second one that is implicit concerns the metavariables and comes from the rewriting mechanism.

CRSs as defined so far are quite general and do not satisfy important properties like, for example, confluence. If we restrict to the class of orthogonal CRSs, confluence and other interesting properties are satisfied [15].

Definition 7 (Orthogonality). *Let $\mathcal{R} = \{L_i \to R_i | i \in I\}$ be a set of* CRS *rewrite rules.*

1. $\mathcal{R}$ *is* non-overlapping *if the following holds:*
   Let r_i be the redex $\sigma(L_i)$ and let $Z_1, \dots Z_n$ be all the distinct metavariables of the metaterm L_i. Then if r_i contains another redex $r_j = \sigma(L_j)$, this redex r_j must be already present in $\sigma(Z_p(x_{i_1} \dots x_{i_{k_p}}))$, for some subterm $Z_p(x_{i_1} \dots x_{i_{k_p}})$ of L_i.
2. $\mathcal{R}$ *is* left-linear *if all the metaterms L_i are left-linear.*
3. $\mathcal{R}$ *is* orthogonal *if it is non-overlapping and left-linear.*
4. *A* CRS *is orthogonal if it has an orthogonal set of rewrite rules.*

Similarly to the λ-calculus, CRSs can be equipped with simple types. All types are generated from base types $\tau_0, \dots, \tau_n$ in the usual way. Variables and constants have a (unique) base type, an abstraction $[x]A$ has type $[x]A : \tau_0 \to \tau_1$ if $x : \tau_0$ and $A : \tau_1$, a metaterm $g(A_1, \dots, A_n)$ with $g \in \mathcal{F}_n$ has type τ_0 if $g : \tau_1 \dots \to \tau_n \to \tau_0$ and $A_i : \tau_i$, for $i = 1 \dots n$. Similarly for a metaterm $Z(A_1, \dots, A_n)$.

3 Translating Rewriting Calculus into CRS

The two systems introduced in the previous sections can be seen as two formats of higher-order rewriting which, in spite of their differences in the presentation, use similar mechanisms for performing computations.

We propose in this section an analysis of the two calculi in order to point out their similarities. In particular, we define a translation function from the ρ-calculus to simply-typed CRS and we prove the completeness and soundness of the translation. The results are obtained using an equivalence between the rewrite relations of the two systems, making it possible to transfer the properties holding for one system to the other, as discussed in Section 3.4.

3.1 The Translation

In the following we choose a CRS having as variables the set of variables $\mathcal{X}$ of the ρ-calculus and having as functional symbols the set of constants $\mathcal{K}$ of the ρ-calculus plus four distinguished symbols: the binary symbols $App, Dis, rule$ and the unary symbol Ab. The types of the CRS-terms are built from only one base type that we denote τ.

Definition 8 (Translation of terms)
The translation, denoted $\overline{t}$, of a ρ-term t into a simply typed CRS*-term, is defined as follows:*

$$\begin{array}{ll}
\overline{x} & = x \textit{ with } x \textit{ of type } \tau \\
\overline{f(t_1,\ldots,t_n)} & = f(\overline{t_1},\ldots,\overline{t_n}) \textit{ with } f : \tau \to \ldots \to \tau \ (n \textit{ type arrows}) \\
\overline{t_1\,t_2} & = App(\overline{t_1},\overline{t_2}) \textit{ with } App : \tau \to \tau \to \tau \\
\overline{t_1 \wr t_2} & = Dis(\overline{t_1},\overline{t_2}) \textit{ with } Dis : \tau \to \tau \to \tau \\
\overline{\lambda p.t} & = Ab([x_1]\ldots[x_n].rule(\overline{p},\overline{t})) \textit{ where } \{x_1 \ldots x_1\} = \mathcal{FV}(p) \\
 & \quad \textit{with } Ab : (\tau \to \ldots \to \tau) \to (\tau \to \tau) \to \tau \\
 & \quad \textit{and } rule : \tau \to \tau \to (\tau \to \tau)
\end{array}$$

Given a context $\mathsf{Ctx}_{\lceil\ \rceil}$*, a hole* $\square$ *is of base type.*

The ρ-calculus functional application is translated into a functional symbol with associated arguments and corresponding type. The application and the structure operator are translated by the two special symbols *App* and *Dis* of arity two. Pattern abstractions need a more subtle translation. Since the CRS operator $[\_]\_$ abstracts on single variables, we use an intermediary distinguished symbol *rule* which takes as arguments the pattern and the right-hand side of the rule, and then we abstract on the variables of the pattern. Since we want the result to have a base type, we enclose the resulting CRS-term with a symbol *Ab* meant to collapse a functional type.

We can immediately notice that the CRS-terms obtained as translation of some ρ-terms have good structural properties.

Proposition 1 (Properties of the translation)

i) For any well-formed ρ-term t, we have $\overline{t} : \tau$.

ii) For any subterm $s : \tau$ of a CRS*-term $\overline{t}$, there exists a ρ-term s' such that $\overline{s'} = s$.*

Proof. By structural induction on the ρ-term t and the CRS-term $\overline{t}$, respectively.

These properties will be useful later on for proving the soundness of the translation.

The set of rewrite rules of the CRS is obtained by translating the evaluation rules of the ρ-calculus.

Definition 9 (Evaluation rules encoding). *Let $p_{\lceil\ \rceil}$ denote a ground algebraic* CRS *context with n holes. The translation of the evaluation rules of the ρ-calculus into an (infinite) set $\mathcal{R}$ of* CRS *rewrite rules (schematic in $p_{\lceil\ \rceil}$) is then defined as follows:*

$$\begin{array}{l}
(\rho_p)\ App(Ab([x_1]\ldots[x_n].rule(p_{\lceil x_1,\ldots,x_n\rceil}, Z(x_1,\ldots,x_n))), p_{\lceil Z_1,\ldots,Z_n\rceil}) \\
\quad \to \quad Z(Z_1,\ldots,Z_n)
\end{array}$$

$$(\delta_C)\ App(Dis(Z_1,Z_2),Z_3) \quad \to \quad Dis(App(Z_1,Z_3),App(Z_2,Z_3))$$

The δ_C-rule is a direct encoding of the (δ) evaluation rule of the rewriting calculus, where the structure operator and the application operator have been replaced by the corresponding functional symbols in the CRS. The first CRS-rewrite rule, the ρ_p-rule, is used to reduce the CRS-redexes corresponding to an abstraction application in the ρ-calculus. To any ρ-reduction $t_0 = (\lambda p.t_1)\ t_3 \mapsto_{\rho} \ldots$ we associate a CRS-reduction $\overline{t_0} \mapsto_{\rho_p} \ldots$. The context $p_{\lceil\ \rceil}$ in the corresponding ρ_p-rule is obtained by translating the ρ-pattern p into the CRS and replacing its n variables by n holes. Moreover, the translation implicitly defines a relation between the free variables of the ρ-pattern p, say $y_1, \ldots, y_n$, and the metavariables $Z_1, \ldots, Z_n$ in the ρ_p-rule. We can formalise this relation using an injective function $\zeta : \mathcal{X} \mapsto \mathcal{Z}$ such that $\zeta(y_i) = Z_i \in \mathcal{Z}_0$ for all $i = 1 \ldots n$.

The side condition of the rule (ρ), *i.e.* $\sigma = Sol(p \prec\!\!\prec t_3)$, is encoded directly inside the CRS-rule ρ_p, by choosing the same structure of the context $p_{\lceil\ \rceil}$ in the subterm $rule(p_{\lceil x_1,\ldots,x_n \rceil}, \ldots)$, encoding the ρ-abstraction, and in the metaterm $p_{\lceil Z_1,\ldots,Z_n \rceil}$ to which the abstraction is applied. This corresponds to the encoding of the rule (ρ) with syntactic matching.

Observe that the rule schema ρ_p represents an infinity of rewrite rules, one for each $p_{\lceil\ \rceil}$. In principle, we suppose to have a rule ρ_p for each algebraic linear ρ-pattern p. In practice, for the properties we are interested in, we will only need a finite number of rules ρ_p, as discussed in Section 3.4.

In the following we will often denote the left-hand side of the rules δ_C and ρ_p by L_C and L_p, respectively. It is not difficult to see that L_C and L_p are CRS-patterns:

Proposition 2. *The metaterms L_C and L_p of the rewrite rules δ_C and ρ_p are* CRS-*patterns, i.e. verify Definition 5, for any context $p_{\lceil\ \rceil}$.*

We can remark that this would not be the case without the assumption of linearity on ρ-patterns. As a consequence of L_C and L_p being CRS-patterns, matching in the obtained CRS is decidable and unitary [3].

Example 6. In the rule ρ_p, consider the empty context for $p_{\lceil\ \rceil}$ and $n = 1$ (so that we abstract on a single variable). We obtain in this way the following encoding in the CRS of the β-rule of the λ-calculus:

$$App(Ab([x].rule(x, Z(x))), Z_1) \quad \to \quad Z(Z_1)$$

which is slightly different from the usual encoding, presented in Example 4.

Example 7. Consider the ρ-term $t = (\lambda g(x, y).f(x))\ g(a, b)$. In the ρ-calculus we have the reduction $(\lambda g(x, y).f(x))\ g(a, b) \ \mapsto_{\rho}\ f(a)$.

The translation of the term t into a CRS-term is $\overline{t} = App(Ab([x][y].rule(g(x, y), f(x))), g(a, b))$. We can apply to $\overline{t}$ the CRS-rewrite rule $\rho_{g(x_1,x_2)}$: $App(Ab([x_1][x_2].rule(g(x_1, x_2), Z(x_1, x_2))), g(Z_1, Z_2)) \quad \to \quad Z(Z_1, Z_2)$ using the assignment $\sigma = \{(Z/\underline{\lambda} z_1 z_2.f(z_1), Z_1/a, Z_2/b\}$. We obtain

$$\begin{aligned} &App(Ab([x][y].rule(g(x, y), f(x))), g(a)) \\ \mapsto_{\rho_P}\ &\sigma(Z(Z_1, Z_2)) = \sigma(Z)(\sigma(Z_1), \sigma(Z_2)) = (\underline{\lambda} z_1 z_2.f(z_1))(a, b){\downarrow_{\underline{\beta}}} = f(a). \end{aligned}$$

3.2 Completeness of the Translation

The aim of this section is to show that for every ρ-reduction there exists a corresponding CRS-reduction. We start by proving a lemma expressing the interaction between the formation of contexts and substitutions and the translation. We then show the simulation of one-step ρ-reductions in the CRS and we conclude the section by a theorem stating the completeness of the translation.

Lemma 1 (Context stability). *Let $s, s_1, \dots s_n$ be ρ-terms and $\mathsf{Ctx}_{\lceil\ \rceil}$ be a context. Then we have*

i) $\overline{\mathsf{Ctx}_{\lceil s \rceil}} = \overline{\mathsf{Ctx}}_{\lceil \overline{s} \rceil}$
ii) $\overline{\mathsf{Ctx}_{\lceil s_1,\dots,s_n \rceil}} = \overline{\mathsf{Ctx}}_{\lceil \overline{s_1},\dots,\overline{s_n} \rceil}$
iii) $\overline{s\{x_1/s_1,\dots,x_n/s_n\}} = \overline{s}\{x_1/\overline{s_1},\dots,x_n/\overline{s_n}\}$

Proof. By induction on the structure of the context $\mathsf{Ctx}_{\lceil\ \rceil}$.

The previous lemma is used to show that rewrite steps are naturally preserved by the translation.

Lemma 2 (One-step simulation). *Given a ρ-term t such that $t \mapsto_{\rho\delta} t_1$, then in the corresponding* CRS *we have $\overline{t} \mapsto_{\mathcal{R}} \overline{t_1}$.*

Proof. Without loss of generality, we suppose the redex being at the head position in the ρ-term t.

– $t \mapsto_{\delta} t_1$ then $t = (t_1 \wr t_2)\, t_3$ and $t' = t_1\, t_3 \wr t_2\, t_3$. We have $\overline{t} = App(Dis(\overline{t_1},\overline{t_2}),\overline{t_3})$, we apply the rule (δ_C) using the assignment $\sigma = \{Z_1/\overline{t_1}, Z_2/\overline{t_2}, Z_3/\overline{t_3}\}$ and we obtain $\sigma(Dis(App(Z_1,Z_3), App(Z_2,Z_3))) = Dis(App(\overline{t_1},\overline{t_3}), App(\overline{t_2},\overline{t_3})) = \overline{t_1}$.

– $t \mapsto_{\rho} t_1$ then $t = (\lambda p.v)\ u$ and $t_1 = \sigma(v)$ with $\mathcal{FV}(p) = \{x_1,\dots,x_n\}$ and $\sigma = Sol(p \lll u) = \{x_1/u_1,\dots,x_n/u_n\}$.
In the CRS, we have $\overline{t} = App(Ab([x_1]\dots[x_n].rule(\overline{p},\overline{v})),\overline{u})$. We can apply the corresponding CRS-rewrite rule (ρ_p) using the assignment $\sigma' = \{Z/\underline{\lambda} z_1 \dots z_n.\overline{v}', Z_1/\overline{u_1},\dots,Z_n/\overline{u_n}\}$ where $\overline{v}'$ is the term $\overline{v}$ to which a renaming of the variables x_i into z_i, for $i = 1\dots n$, has been applied. By applying the converse renaming and by Lemma [1], we obtain $\sigma'(Z(Z_1,\dots,Z_n)) = (\underline{\lambda} z_1 \dots z_n.\overline{v}')(\overline{u_1},\dots,\overline{u_n}) \downarrow_{\underline{\beta}} = \overline{v}'\{z_1/\overline{u_1},\dots,z_n/\overline{u_n}\}$ $= \overline{v}\{x_1/\overline{u_1},\dots,x_n/\overline{u_n}\} = \overline{v\{x_1/u_1,\dots,x_n/u_n\}} = \overline{\sigma(v)} = \overline{t_1}$.

The generalisation to derivations of arbitrary length follows easily:

Theorem 1 (Completeness). *Given a ρ-term t such that $t \mapsto\!\!\!\!\rightarrow_{\rho\delta} t_n$, then in the corresponding* CRS *we have $\overline{t} \mapsto\!\!\!\!\rightarrow_{\mathcal{R}} \overline{t_n}$.*

Proof. By induction on the length of the reduction, using Lemma [2].

We can notice that there is a one-to-one correspondence in the derivations, *i.e.* for every step in the rewriting calculus a corresponding step is performed in the CRS. The substitution application is performed at the meta-level in both calculi, by underlined beta-reductions in the CRS and by simultaneous variable substitutions in the ρ-calculus, and therefore does not affect the length of the reductions.

3.3 Soundness of the Translation

Completeness proves that for every rewrite step in ρ-calculus, a rewrite step in the associated CRS can be performed. We will show now that a rewrite step in a translated term must originate from a rewrite step in the ρ-calculus. We state first a precise relation between a CRS assignment and the corresponding ρ-calculus substitution.

Lemma 3. *Let $t = (\lambda p.v)\ u$ be a ρ-term with $\mathcal{FV}(p) = \{y_1, \dots y_n\}$ and L_p be the left hand side of the* CRS*-rewrite rule ρ_p with $Z_i = \zeta(y_i)$ for all $i = 1 \dots n$. Let σ be an assignment such that $\sigma(L_p) = \overline{t}$. Then*

1. *the assignment σ is of the form $\sigma = \{Z/\underline{\lambda} z_1 \dots z_n.\overline{v}, Z_1/\overline{u_1}, \dots, Z_n/\overline{u_n}\}$.*
2. *in the ρ-calculus the substitution $\sigma' = \{y_1/u_1, \dots, y_n/u_n\}$ is such that $\sigma'(p) = u$.*

Proof. 1. We show that the given assignment σ is a solution of the matching of the CRS metaterm L_p to the CRS-term $\overline{t}$. Since the solution of a CRS pattern matching problem is unique [3], this concludes the proof.

2. By Lemma 1 and the injectivity of the translation function.

We can show now that a rewrite step in the translation of the ρ-calculus is related with a rewrite step in the ρ-calculus itself.

Lemma 4. *If $\overline{t} \mapsto_{\mathcal{R}} t_1$ in the* CRS*, then we have $t \mapsto_{\rho\delta} t'$ with $\overline{t'} = t_1$.*

Proof. By induction on the depth of the redex position in the term $\overline{t}$.
Base case: the redex is at the head position in the term $\overline{t}$. Then we have:

- $\overline{t} \mapsto_{\delta_C} t_1$ with $\overline{t} = App(Dis(\overline{t_1}, \overline{t_2}), \overline{t_3})$ and $t_1 = Dis(App(\overline{t_1}, \overline{t_3}), App(\overline{t_2}, \overline{t_3}))$ using the assignment $\sigma = \{Z_1/\overline{t_1}, Z_2/\overline{t_2}, Z_3/\overline{t_3}\}$. In the ρ-calculus we have $t = (t_1 \wr t_2)\, t_3 \mapsto_\delta t' = t_1\, t_3 \wr t_2\, t_3$ and thus it is easy to see that $\overline{t'} = t_1$.
- $\overline{t} \mapsto_{\rho_P} t_1$. We have $\overline{t} = App(Ab([x_1] \dots [x_n].rule(\overline{p}, \overline{v})), \overline{u})$ which reduces to $t_1 = \overline{v}'\{z_1/\overline{u_1}, \dots, z_n/\overline{u_n}\}$ using the assignment $\overline{\sigma} = \{Z/\underline{\lambda} z_1 \dots z_n.\overline{v}', Z_1/\overline{u_1}, \dots, Z_n/\overline{u_n}\}$, where $\overline{v}'$ is the term $\overline{v}$ to which a renaming of the variables x_i into z_i, for $i = 1 \dots n$, has been applied. In the ρ-calculus we have $t = (\lambda p.v)\ u$ with $\mathcal{FV}(p) = \{x_1, \dots x_n\}$. By Lemma 3 there exists a substitution $\sigma = \{x_1/u_1, \dots, x_n/u_n\}$ solution of the matching problem $p \prec\!\!\prec u$. Thus we obtain $t \mapsto_\rho t' = \sigma(v) = v\{x_1/u_1, \dots, x_n/u_n\}$. Using the converse renaming of z_i into x_i, for $i = 1 \dots n$, by Lemma 1, we conclude $\overline{t'} = t_1$.

Induction: If the redex is not at the head position in the term $\overline{t}$. Then we have $\overline{t} = \mathsf{Ctx}_{\lceil \sigma(L) \rceil} \mapsto_R \mathsf{Ctx}_{\lceil \sigma(R) \rceil}$, for some context $\mathsf{Ctx}_{\lceil\ \rceil}$. Since $\overline{t}$ and $\sigma(L)$ are of base type by Proposition 1 we have a ρ-context $\mathsf{Ctx}'_{\lceil\ \rceil}$ such that $\overline{\mathsf{Ctx}'_{\lceil\ \rceil}} = \mathsf{Ctx}_{\lceil\ \rceil}$ and a ρ-term s such that $\overline{s} = \sigma(L)$. Thus, by induction $t = \mathsf{Ctx}'_{\lceil s \rceil} \mapsto_{\rho\delta} \mathsf{Ctx}'_{\lceil s' \rceil} = t'$ with $\overline{s'} = \sigma(R)$. Hence, using Lemma 1, $\overline{t'} = \overline{\mathsf{Ctx}'_{\lceil s' \rceil}} = \overline{\mathsf{Ctx}'}_{\lceil \overline{s'} \rceil} = \mathsf{Ctx}_{\lceil \sigma(R) \rceil} = t_1$

The previous lemma is generalised to arbitrary rewrite sequences in the following theorem.

Theorem 2 (Soundness). *If $\overline{t} \mapsto\!\!\!\rightarrow_{\mathcal{R}} t_n$ in the* CRS*, then we have $t \mapsto\!\!\!\rightarrow_{\rho\delta} t'$ with $\overline{t'} = t_n$.*

Proof. By induction on the length of the reduction, using Lemma 4.

Soundness and completeness say that the fact that a term rewrites to another is preserved and reflected in the two systems. This can be seen as a measure of the neatness of the translation. Indeed, other important properties, as discussed in the following section, can be reflected from the CRS into the ρ-calculus.

3.4 Properties

The obtained results allow us to deduce important properties for the rewriting calculus via the properties of the corresponding CRS, avoiding the development of *ad hoc* proofs for the ρ-calculus.

In particular, we are interested in properties as confluence, finiteness of developments and standardisation, that analyse the derivation space starting from an initial ρ-term. In the version of the rewriting calculus considered here (and contrarily to the dynamic patterns used in [1]), all variables in a ρ-pattern p are bound, therefore no new patterns can be created during the reduction. Therefore, we can consider a CRS having a finite number of rewrite rules, that is the rewrite rule (δ_C) and for any pattern p present in the initial ρ-term, a corresponding rule (ρ_P).

First of all, we can notice that any CRS obtained from the translation belongs to the class of orthogonal CRSs, since it is left-linear and non-overlapping:

Lemma 5 (Orthogonality). *The* CRS *obtained as result of the translation of the ρ-calculus is an orthogonal* CRS*.*

Proof. First, it is clear that the CRS-patterns L_p and L_C are linear, for any context $p_{\lceil\ \rceil}$. We show next that the rules (δ_C) and (ρ_P) are non-overlapping.

Let t, t' be two CRS-terms. We show that if there exists an assignment σ such that $\sigma(L_p) = t$ and $t \mapsto\!\!\!\rightarrow_{\mathcal{R}} t'$, then there exists an assignment σ' such that $\sigma'(L_p) = t'$. Suppose $t = \mathsf{Ctx}_{\lceil \sigma_2(L)\rceil} \mapsto\!\!\!\rightarrow_{\mathcal{R}} \mathsf{Ctx}_{\lceil \sigma_2(R)\rceil} = t'$. The fact that the assignment σ exists implies that the occurrence of the redex $\sigma_2(L)$ in t, say ω, corresponds to a metavariables position in L_p, say the position of the metavariable Z_i. Thus σ of the form $\{Z/s_0, Z_1/s_1, \ldots, Z_i/\sigma_2(L), \ldots, Z_n/s_n\}$ is such that $\sigma(L_p) = t$. The contraction of the redex $\sigma_2(L)$ in t affects only the subterm of t headed in ω, therefore the substitution $\sigma' = \{Z/s_0, Z_1/s_1, \ldots, Z_i/\sigma_2(R), \ldots, Z_n/s_n\}$ is such that $\sigma'(L_p) = t'$. A similar reasoning can be done for the left-hand side L_C of the CRS rewrite rule δ_C.

As a consequence of orthogonality, we can deduce immediately the confluence property for the rewriting calculus. This property has already been proved for various versions of the calculus, in particular in [5]. A confluence proof is also available for the typed version of the left linear calculus called *PPTS* [1]. Here we obtain the confluence property for the ρ-calculus exploiting the well-known results for orthogonal CRSs:

Corollary 1 (Confluence). *The ρ-calculus with linear algebraic patterns and syntactic matching is confluent.*

Proof. By the encoding of Section 3.1, Lemma 5 and the confluence results for orthogonal CRSs (see for example [20,21]).

Similarly, we can deduce in the ρ-calculus interesting properties for a special kind of reductions, called developments. Intuitively, a development corresponds to the computation of a chosen set of reducible expressions in a term. The theory of developments, originally developed for the λ-calculus, has been successfully adapted to several other computational paradigms, like first- and higher-order term rewrite system. Interestingly, the notion of superdevelopments allows to derive a new second-order matching algorithm [11].

The general defintion of developments for the rewriting calculus can be found in [2]. The main desirable results on developments are the fact that the complete development of a finite set of redexes always terminates (FD) and the fact that, for a given initial term, all complete developments of a fixed set of redexes end with the same term (FD!):

Corollary 2 (Finite developments)

– *Developments in the ρ-calculus are always finite.*
– *All developments of a ρ-term t end on the same final term.*

Proof. Follows from the results on developments proved for CRSs (see [15,20]).

These properties of developments are a key hypothesis to achieve a standardisation result. We know from the two theorems FD and FD! on developments that the result of this kind of computations is unique and does not depend on the choice of a particular reduction. We still lack of information about the way to perform the computation in order to reach this result, when it exists. Standardisation ensures that, for any derivation, the reduction steps can always be reordered to obtain a derivation in a canonical form, called standard:

Corollary 3 (Standardisation). *For any two ρ-terms t_1 and t_2 such that t_1 rewrites to t_2, there exists a standard derivation leading from t_1 to t_2.*

Proof. Follows from the results on standardisation proved for CRSs (see *e.g.* [18]).

4 Conclusions

The ρ-calculus is a powerful framework for specifying and reasoning about computation and deduction. It is therefore of main interest to study the relationship of the calculus with similar ones, in particular to understand its capabilities and properties.

We have presented in this paper the simulation of the rewriting calculus into *Combinatory Reduction systems*, the converse simulation having already been addressed in [3]. We have shown the encoding of the ρ-calculus into an appropriate CRS having as set of terms the encoding of ρ-terms and as set of rewrite rules the encoding of the evaluation rules of the ρ-calculus. We have then proved the soundness and completeness of this translation. These results allow us to adapt the well-developed CRSs meta-theory to the rewriting calculus. In particular, we can derive the properties of confluence, finite developments and standardisation for the ρ-calculus. Similar approaches can be applied to related calculi like the lambda calculus with patterns [19] or the pattern calculus [14].

The natural encoding we have presented in the paper leads to a simulation step-by-step of ρ-derivations into CRS-derivations. The translation the other way round was not so neat, since "walking through the context" is done implicitly in CRSs and thus additional ρ-terms needed to be inserted to direct the reduction in the ρ-calculus. Here instead, the explicit application operator of the ρ-calculus and the definition of rewrite rules at the object level in the ρ-calculus are encoded into CRS-terms using appropriate functional symbols. This allows to maintain, in the translated terms, the control on the position to which the rewrite rule is applied, which is a typical ingredient of the ρ-calculus.

We have considered in this paper the ρ-calculus with syntactic matching and the two evaluation rules (δ) and (ρ). The obtained results can be easily generalised to the version of the ρ-calculus with explicit delayed matching constraints. This version of the ρ-calculus introduces the matching problems as part of the ρ-calculus syntax and represents a first step towards an explicit handling of the matching related computations [6]. Basically, a ternary symbol $[\_ \ll \_]\_$, representing a term constrained by a matching, is added to the syntax of the ρ-calculus and the set of evaluation rules is adapted accordingly. This new symbol can be encoded in a CRS-term using additional functional symbols in the CRS-signature, similarly as for the other ρ-calculus operators. The encoding of other versions of the ρ-calculus into CRSs is matter of further study.

Other higher-order rewrite systems have already been compared. In particular, it has been shown that CRSs and HRSs have the same expressive power and therefore they can be considered equivalent [21]. Using this comparison, we can have an indirect representation of the ρ-calculus into HRSs that is based on the translation from the CRSs to HRSs and the translation from ρ-calculus to CRSs we have defined in this paper.

Acknowledgements. Many thanks to Vincent van Oostrom for fruitful suggestions on the most natural encoding of the ρ-calculus as a CRS and to the anonymous referees for their valuable remarks and suggestions.

References

1. G. Barthe, H. Cirstea, C. Kirchner, and L. Liquori. Pure Patterns Type Systems. In *Principles of Programming Languages - POPL'03*, pages 250–261. ACM, 2003.
2. C. Bertolissi. Developments in the rewriting calculus. Research Report http://hal.inria.fr/inria-00121212, INRIA & LORIA, 2006.

3. C. Bertolissi, H. Cirstea, and C. Kirchner. Expressing combinatory reduction systems derivations in the rewriting calculus. *Higher-Order and Symbolic Computation*, 19(4):345–376, dec 2006.
4. P. Brauner. Un calcul des séquents extensible. Master thesis, LORIA, 2006.
5. H. Cirstea. *Calcul de réécriture : fondements et applications.* Thèse de Doctorat d'Université, Université Henri Poincaré - Nancy I, 2000.
6. H. Cirstea, G. Faure, and C. Kirchner. A rho-calculus of explicit constraint application. In *Proc. of WRLA'2004*, volume 117 of *ENTCS*, 2004.
7. H. Cirstea and C. Kirchner. The rewriting calculus as a semantics of ELAN. In J. Hsiang and A. Ohori, editors, *4th Asian Computing Science Conference*, volume 1538 of *LNCS*, pages 8–10. Springer-Verlag, 1998.
8. H. Cirstea and C. Kirchner. The rewriting calculus — Part I *and* II. *Logic Journal of the Interest Group in Pure and Applied Logics*, 9(3):427–498, 2001.
9. H. Cirstea, C. Kirchner, and L. Liquori. Matching Power. In *Proc. of RTA*, volume 2051 of *LNCS*, pages 77–92. Springer-Verlag, 2001.
10. H. Cirstea, C. Kirchner, L. Liquori, and B. Wack. Rewrite strategies in the rewriting calculus. In B. Gramlich and S. Lucas, editors, *Proceedings of WRS'03*. ENTCS, June 2003.
11. G. Faure. Matching modulo superdeveloppements. application to second-order matching. In *Logic for Programming Artificial Intelligence and Reasoning*, Phnom Penh, 2006.
12. C. Houtmann. Cohérence de la déduction surnaturelle. Master thesis, LORIA, 2006.
13. G. Huet. *Résolution d'équations dans les langages d'ordre 1,2, ...,ω.* Thèse de Doctorat d'Etat, Université de Paris 7 (France), 1976.
14. B. Jay and D. Kesner. Pure pattern calculus. In P. Sestoft, editor, *15th European Symposium on Programming Languages and Systems*, volume 3924 of *Lecture Notes in Computer Science*, pages 100–114, Vienna (Austria), mar 2006. Springer Verlag.
15. J. W. Klop. *Combinatory Reduction Systems.* PhD thesis, CWI, 1980.
16. J. W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory Reduction Systems: Introduction and Survey. *Theoretical Computer Science*, 121:279–308, 1993.
17. L. Liquori and B. Serpette. iRho: an Imperative Rewriting Calculus. In *Proc. of PPDP'04*, pages 167–178. The ACM Press, 2004.
18. P.-A. Melliès. *Description Abstraite des Systèmes de Réécriture.* PhD thesis, Université Paris 7, 1996.
19. V. van Oostrom. Lambda calculus with patterns. Technical report, Vrije Universiteit, Amsterdam, Nov. 1990.
20. V. van Oostrom. *Confluence for Abstract and Higher-Order Rewriting.* PhD thesis, Vrije Universiteit, Amsterdam, 1994.
21. V. van Oostrom and F. van Raamsdonk. Comparing combinatory reduction systems and higher-order rewrite systems. In *Higher-Order Algebra, Logic and Term Rewriting (HOA)*, pages 276–304, 1993.
22. B. Wack. *Typage et déduction dans le calcul de réécriture.* Thèse de doctorat, Université Henri Poincaré - Nancy I, Oct 2005.

Relational Parametricity and Separation Logic*

Lars Birkedal[1] and Hongseok Yang[2]

[1] IT University of Copenhagen, Denmark
[2] Queen Mary, University of London, UK

Abstract. Separation logic is a recent extension of Hoare logic for reasoning about programs with references to shared mutable data structures. In this paper, we provide a new interpretation of the logic for a programming language with higher types. Our interpretation is based on Reynolds's relational parametricity, and it provides a formal connection between separation logic and data abstraction.

1 Introduction

Separation logic [16,11,6] is a Hoare-style program logic, and variants of it have been applied to prove correct interesting pointer algorithms such as copying a dag, disposing a graph, the Schorr-Waite graph algorithm, and Cheney's copying garbage collector. The main advantage of separation logic compared to ordinary Hoare logic is that it facilitates *local reasoning*, formalized via the so-called *frame rule* using a connective called *separating conjunction.* The development of separation logic initially focused on *low-level* languages with heaps and pointers, although in recent work [12,7] it was shown how to extend separation logic first to languages with a simple kind of procedures [12] and then to languages also with higher-types [7]. Moreover, in [12] a second-order frame rule was proved sound and in [7] a whole range of higher-order frame rules were proved sound for a separation-logic type system.

In [12] and [7] it was explained how second and higher-order frame rules can be used to reason about static imperative modules. The idea is roughly as follows. Suppose that we prove a specification for a client c, depending on a module k,

$$\{P_1\}\, k\, \{Q_1\} \vdash \{P\}\, c(k)\, \{Q\}.$$

The proof of the client depends only on the "abstract specification" of the module k, which describes the external behavior of k. Suppose further that an actual implementation m of the module satisfies

$$\{P_1 * R\}\, m\, \{Q_1 * R\}.$$

Here R is the internal resource invariant of the module m, describing the internal heap storage used by the module m to implement the abstract specification. We can then employ a frame rule on the specification for the client to get

* This work was supported by FUR (FIRST). Yang was supported also by EPSRC. We would like to thank Nick Benton for his insightful comments.

$$\{P_1 * R\}\, k\, \{Q_1 * R\} \vdash \{P * R\}\, c(k)\, \{Q * R\},$$

and combine it with the specification for m to obtain

$$\{P * R\}\, c(m)\, \{Q * R\}.$$

A key advantage of this approach to modularity is that it facilitates so-called "ownership transfer." For example, if the module is a queue, then the ownership of cells transfers from the client to module upon insertion into the queue. Moreover, the discipline allows clients to maintain pointers into cells that have changed ownership to the module. See [12] for examples and more explanations of these facts.

Note that the higher-order frame rules in essence provide implicit quantification over internal resource invariants. In [4] it is shown how one can employ a higher-order version of separation logic, with explicit quantification of assertion predicates to reason about dynamic modularity (where there can be several instances of the same abstract data type implemented by an imperative module), see also [13]. The idea is to existentially quantify over the internal resource invariants in a module, so that in the above example, c would depend on a specification for k of the form

$$\exists R. \{P_1 * R\}\, k\, \{Q_1 * R\}.$$

As emphasized in the papers mentioned above, note that, both in the case of implicit quantification over internal resource invariants (higher-order frame rules) and in the case of explicit quantification over internal resource invariants (existentials over assertion predicates), reasoning about a client does not depend on the internal resource invariant of possible module implementations. Thus the methodology allows us to formally reason about *mutable abstract data types*, aka. *imperative modules*. However, the models in the papers mentioned above do not allow us to make all the conclusions we would expect from reasoning about mutable abstract data types. In particular, we would expect that *clients should behave parametrically in the internal resource invariants*: When a client is applied to two different implementations of a mutable abstract data type, it should be the case that the client preserves relations between the internal resource invariants of the two implementations. This is analogous to Reynolds's style relational parametricity for abstract data types with quantification over type variables [15].

In this paper we provide a new parametric model of separation logic, which captures that clients behave parametrically in internal resource invariants of mutable abstract data types. For the purposes of the present paper, we have decided to focus on the implicit approach to quantification over internal resource invariants via higher-order frame rules, since it is technically simpler than the explicit approach.[1] Our model validates a whole range of higher-order frame

[1] The reason is that the implicit quantification of separation logic uses quantification in a very disciplined way so that the usual reading of assertions as sets of heaps can be maintained; if we use quantification without any restrictions, as in [2], it appears that we cannot have the usual reading of assertions as sets of heaps because, then, the rule of consequence is not sound.

rules, as in [7], but here we achieve that for a more standard presentation of separation logic and not only for a separation-logic type system as in [7].

Technically, it has proven to be a very non-trivial problem to define a parametric model for separation logic. We describe the challenges and give an overview of the main ideas in our approach in the following section. In Section 3 we describe the programming and the assertion language we consider and in Section 4 we define our version of separation logic. In Section 5 we define the semantics of our programming language in the category of FM-cpos and we define our relational interpretation of separation logic in Section 6. Section 7 relates our relational interpretation to the standard interpretation of separation logic, and in Section 8 we present the abstraction theorem that our parametric model validates. We briefly describe an example in Section 9 and finally we conclude and discuss future work in Section 10. For reasons of space most proofs have been omitted; they can be found in the full version of the paper.[2]

2 Challenges and Main Ideas

One of the main technical challenges in developing a relationally parametric model of separation logic, even for a simple first-order language, is that the standard models of separation logic allow the identity of locations to be observed in the model. This means in particular that allocation of new heap cells is not parametric because the identity of the location of the allocated cell can be observed in the model. (We made this observation in earlier unpublished joint work with Noah Torp-Smith, see [18, Ch. 6].)

This problem of non-parametric memory allocation has also been noticed by recent work on data refinement for heap storage, which exploits semantic ideas from separation logic [8,9]. However, the work on data refinement does not provide a satisfactory solution. Either it avoids the problem by assuming that clients do not allocate cells [8], or its solution has difficulties for handling higher-order procedures and formalizing (observational) equivalences, not refinements, between two implementations of a mutable abstract data type [9].

Our solution to this challenge is to define a more refined semantics of the programming language using FM domain theory, in the style of Benton and Leperchey [3], in which one can name locations but not observe the identity of locations because of the built-in use of permutation of locations. Part of the trick of *loc. cit.* is to define the semantics in a continuation-passing style so that one can ensure that new locations are suitably fresh with respect to the remainder of the computation. (See Section 5 for more details.) Benton and Leperchey used the FM domain-theoretic model to reason about contextual equivalence and here we extend the approach to give a semantics of separation logic in a continuation-passing style. We relate this new interpretation to the standard direct-style interpretation of separation logic via the so-called observation closure $(-)^{\perp\perp}$ of a relation, see Section 7.

[2] The full version is available at the following URL:
`http://www.dcs.qmul.ac.uk/~hyang/paper/fossacs07-full.pdf`

The other main technical challenge in developing a relationally parametric model of separation logic for reasoning about mutable abstract data types is to devise a model which validates a wide range of higher-order frame rules. Our solution to this challenge is to define an intuitionistic interpretation of the specification logic over a Kripke structure, whose ordering relation intuitively captures the framing-in of resources. Technically, the intuitionistic interpretation, in particular the associated Kripke monotonicity, is used to validate a generalized frame rule. Further, to show that the semantics of the logic does indeed satisfy Kripke monotonicity for the base case of triples, we interpret triples using a universal quantifier, which intuitively quantifies over resources that can possibly be framed in. In the earlier non-parametric model of higher-order frame rules for separation-logic typing in [7] we also made use of a Kripke structure. The difference is that in the present work the elements of the Kripke structure are *relations* on heaps rather than predicates on heaps because we build a *relationally* parametric model.

3 Programs and Assertions

In this paper, we consider a higher-order language with immutable stack variables. The types and terms of the languages are defined as follows:

$$
\begin{array}{ll}
\text{Types } \tau ::= \mathsf{com} \mid \mathsf{ref} \to \tau \mid \tau \to \tau & \text{Expressions } E ::= i \mid \mathsf{nil} \\
\multicolumn{2}{l}{\text{Terms } M ::= x \mid \lambda i.\, M \mid M\ E \mid \lambda x{:}\,\tau.\, M \mid M\, M \mid \mathsf{fix}\, M \mid \mathsf{if}\ (E{=}E)\ M\ M \mid M; M} \\
\multicolumn{2}{l}{\qquad\quad \mid\ \mathsf{let}\ i{=}\mathsf{new}\ \mathsf{in}\ M \mid \mathsf{free}(E) \mid \mathsf{let}\ i{=}[E]\ \mathsf{in}\ M \mid [E]{:=}E}
\end{array}
$$

The language separates expressions E from terms M. Expressions denote heap-independent reference values, and they are bound to *stack variables* i, j. On the other hand, terms denote possibly heap-dependent computations, and they are bound to *identifiers* x, y. The syntax of the language ensures that expressions always terminate, while terms can diverge. The types are used to classify terms only. com denotes commands, $\mathsf{ref} \to \tau$ means functions that take an expression parameter, and $\tau \to \tau'$ denotes functions that takes a term parameter. Note that to support two different function types, the language includes two kinds of abstraction and application, one for expression parameters and the other for term parameters. We assume that term parameters are passed by name, and expression parameters are passed by value.

To simplify the presentation, we take a simple storage model where each heap cell has only one field for references. Command $\mathsf{let}\ i{=}\mathsf{new}\ \mathsf{in}\ M$ allocates such a unary heap cell, binds the address of the cell to i, and runs M under this binding. The content of this newly allocated cell at address i is read by $\mathsf{let}\ j = [i]\ \mathsf{in}\ N$ and updated by $[i] := E$. The cell i is deallocated by $\mathsf{free}(i)$.

The language uses typing judgments of the form $\Delta \vdash E(:\mathsf{ref})$ and $\Delta \,|\, \Gamma \vdash M : \tau$, where Δ is a finite set of stack variables and Γ is a standard type environment for identifiers x. The typing rules for expressions and terms are shown in Figure 1.

$$\frac{}{\Delta, i \vdash i} \qquad \frac{}{\Delta \vdash \mathsf{nil}}$$

$$\frac{}{\Delta \mid \Gamma, x:\tau \vdash x:\tau} \quad \frac{\Delta, i \mid \Gamma \vdash M : \tau}{\Delta \mid \Gamma \vdash \lambda i.\, M : \mathsf{ref} \to \tau} \quad \frac{\Delta \mid \Gamma \vdash M : \mathsf{ref} \to \tau \quad \Delta \vdash E}{\Delta \mid \Gamma \vdash M\ E : \tau}$$

$$\frac{\Delta \mid \Gamma, x:\tau \vdash M : \tau'}{\Delta \mid \Gamma \vdash \lambda x:\tau.\, M : \tau \to \tau'} \quad \frac{\Delta \mid \Gamma \vdash M : \tau' \to \tau \quad \Delta \mid \Gamma \vdash N : \tau'}{\Delta \mid \Gamma \vdash M\ N : \tau} \quad \frac{\Delta \mid \Gamma \vdash M : \tau \to \tau}{\Delta \mid \Gamma \vdash \mathsf{fix}\ M : \tau}$$

$$\frac{\Delta \vdash E \quad \Delta \vdash F \quad \Delta \mid \Gamma \vdash M : \mathsf{com} \quad \Delta \mid \Gamma \vdash N : \mathsf{com}}{\Delta \mid \Gamma \vdash \mathsf{if}\ (E{=}F)\ M\ N : \mathsf{com}}$$

$$\frac{\Delta \mid \Gamma \vdash M : \mathsf{com} \quad \Delta \mid \Gamma \vdash N : \mathsf{com}}{\Delta \mid \Gamma \vdash M; N : \mathsf{com}} \quad \frac{\Delta, i \mid \Gamma \vdash M : \mathsf{com}}{\Delta \mid \Gamma \vdash \mathsf{let}\ i{=}\mathsf{new}\ \mathsf{in}\ M : \mathsf{com}}$$

$$\frac{\Delta \vdash E}{\Delta \mid \Gamma \vdash \mathsf{free}(E) : \mathsf{com}} \quad \frac{\Delta, i \mid \Gamma \vdash M : \mathsf{com} \quad \Delta \vdash E}{\Delta \mid \Gamma \vdash \mathsf{let}\ i{=}[E]\ \mathsf{in}\ M : \mathsf{com}} \quad \frac{\Delta \vdash E \quad \Delta \vdash F}{\Delta \mid \Gamma \vdash E := F : \mathsf{com}}$$

Fig. 1. Typing Rules for Expressions and Terms

We use the standard assertions from separation logic to describe properties of the heap:[3] $P ::= E = E \mid E \leq E \mid E \mapsto E \mid \mathsf{emp} \mid P{*}P \mid P{\wedge}P \mid \neg P \mid \exists i.\, P$. The points-to predicate $E \mapsto E'$ means that the current heap has only one cell at address E and that the content of the cell is E'. The emp predicate denotes the empty heap, and the separating conjunction $P * Q$ means that the current heap can be split into two parts so that P holds for the one and Q holds for the other. The other connectives have the usual meaning from classical logic. All the missing connectives from classical logic are defined as usual.

Assertions only depend on stack variables i, j, not identifiers x, y. Thus assertions are typed by a judgment $\Delta \vdash P : \mathsf{Assertion}$. The typing rules for this judgment are completely standard, and thus omitted from this paper.

4 Separation Logic

Our version of separation logic is the first-order *intuitionistic* logic extended with Hoare triples and invariant extension. The formulas in the logic are called *specifications*, and they are defined by the following grammar:

$$\begin{aligned}\varphi ::= &\ \{P\}M\{Q\} \mid \varphi \otimes P \mid E = E \mid M = M \\ \mid &\ \ \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \Rightarrow \varphi \mid \forall x{:}\,\tau.\varphi \mid \exists x{:}\,\tau.\varphi \mid \forall i.\varphi \mid \exists i.\varphi\end{aligned}$$

The formula $\varphi \otimes P$ means the extension of φ by the invariant P. It can be viewed as a syntactic transformation of φ that inserts $P * -$ into the pre and post conditions of all triples in φ. For instance, $(\{P\}x\{Q\} \Rightarrow \{P'\}M(x)\{Q'\}) \otimes P_0$ is equivalent to $\{P * P_0\}x\{Q * P_0\} \Rightarrow \{P' * P_0\}M(x)\{Q' * P_0\}$. We write Specs for the set of all specifications.

[3] We omit separating implication $-\!\!*$ to simplify presentation.

Proof Rules for Hoare Triples

$$
\begin{array}{rcl}
(\forall i.\{P\}M\{Q\}) & \Rightarrow & \{\exists i.\,P\}M\{\exists i.\,Q\} \quad (\text{where } i \notin \mathsf{fv}(M)) \\
(\{P\}M\{Q\} \vee \{P'\}M\{Q'\}) & \Rightarrow & \{P \vee P'\}M\{Q \vee Q'\} \\
\{P \wedge E{=}E\}M\{Q\} \wedge \{P \wedge E{\neq}F\}N\{Q\} & \Rightarrow & \{P\}\mathsf{if}\,(E{=}F)\,M\,N\{Q\} \\
\{P\}M\{P_0\} \wedge \{P_0\}N\{Q\} & \Rightarrow & \{P\}M;N\{Q\} \\
(\forall i.\,\{P * i \mapsto \mathsf{nil}\}M\{Q\}) & \Rightarrow & \{P\}\mathsf{let}\ i{=}\mathsf{new}\ \mathsf{in}\ M\{Q\}\ (\text{where } i{\notin}\mathsf{fv}(P,Q)) \\
(\forall i.\,\{P * E \mapsto i\}M\{Q\}) & \Rightarrow & \{\exists i.\,P * E \mapsto i\}\mathsf{let}\ i{=}[E]\ \mathsf{in}\ M\{Q\} \\
 & & \hfill (\text{where } i{\notin}\mathsf{fv}(Q))
\end{array}
$$

$$\{E \mapsto F\}\mathsf{free}(E)\{\mathsf{emp}\} \qquad \{E \mapsto E'\}[E] := F\{E \mapsto F\}$$

$$\frac{[\![P]\!]_\rho \subseteq [\![P']\!]_\rho \text{ and } [\![Q']\!]_\rho \subseteq [\![Q]\!]_\rho \text{ for all } \rho \in [\![\Delta]\!]}{\Delta \mid \Gamma \vdash \{P'\}M\{Q'\} \Rightarrow \{P\}M\{Q\}}$$

Proof Rules for Invariant Extension $-\otimes P$

$$
\begin{array}{rclrcl}
\varphi & \Rightarrow & \varphi \otimes P & \{P\}C\{P'\} \otimes Q & \Leftrightarrow & \{P * Q\}C\{P' * Q\} \\
(E = F) \otimes Q & \Leftrightarrow & E = F & (M = N) \otimes Q & \Leftrightarrow & (M = N) \\
(\varphi \otimes P) \otimes Q & \Leftrightarrow & \varphi \otimes (P * Q) & (\varphi \oplus \psi) \otimes P & \Leftrightarrow & (\varphi \otimes P) \oplus (\psi \otimes P) \\
 & & & & & (\text{where } \oplus \in \{\Rightarrow, \wedge, \vee\}) \\
(\kappa x{:}\,\tau.\,\varphi) \otimes P & \Leftrightarrow & \kappa x{:}\,\tau.\,\varphi \otimes P & (\kappa i.\,\varphi) \otimes P & \Leftrightarrow & \kappa i.\,\varphi \otimes P \\
 & & (\text{where } \kappa \in \{\forall, \exists\}) & & & (\text{where } \kappa \in \{\forall, \exists\} \text{ and } i \notin \mathsf{fv}(P))
\end{array}
$$

Rule for Fixed-Point Induction

$$C ::= [\,] \mid \lambda i.C \mid C\,E \mid \lambda x{:}\,\tau.C \mid C\,M \mid \mathsf{fix}\,C \mid C;M \qquad \gamma ::= \{P\}C\{Q\} \mid \gamma{\wedge}\gamma \mid \forall x{:}\,\tau.\gamma \mid \forall i.\gamma$$

$$(\forall x.\ \gamma(x) \Rightarrow \gamma(M\ x)) \ \Rightarrow\ \gamma(\mathsf{fix}\ M)$$

where $\gamma(N)$ is a capture-avoiding insertion of N into the hole $[-]$ in γ.

Fig. 2. Sample Proof Rules

Specifications are typed by the judgment $\Delta \mid \Gamma \vdash \varphi : \mathsf{Specs}$, where we overloaded Specs to mean the type for specifications.

The logic includes all the usual proof rules from first-order intuitionistic logic with equality, and a rule for fixed-point induction. In addition, it contains proof rules from separation logic, and *higher-order frame rules*, expressed in terms of rules for invariant introduction and distribution. Figure 2 shows some of these additional rules and a rule for fixed-point induction. In the figure, we often omit contexts $\Delta \mid \Gamma$ for specifications and also conditions about typing.

The rules for Hoare triples are the standard proof rules of separation logic adapted to our language. Note that in the rule of consequence, we use the standard semantics of assertions P, P', Q, Q', in order to express semantic implications between those assertions. The rules for invariant extension formalize higher-order frame rules, extending the idea in [7]. The generalized higher-order frame rule $\varphi \Rightarrow \varphi \otimes P$ adds an invariant P to specification φ, and the other rules distribute this added invariant all the way down to the triples. The last rule is for fixed-point induction, and it relies on the restriction that a specification is of the form $\gamma(\mathsf{fix}\ M)$. The grammar for γ guarantees that $\gamma(x)$ defines an admissible predicate for x, thus ensuring the soundness of fixed-point induction. Moreover, it also guarantees that $\gamma(x)$ holds when M means $\bot$, so allowing us to omit a usual base case, "$\gamma(\bot)$," from the rule.

Note that the rules do *not* include the so-called conjunction rule:

$$(\{P\}M\{Q\} \wedge \{P'\}M\{Q'\}) \;\Rightarrow\; \{P \wedge P'\}M\{Q \wedge Q'\}$$

The omission of this rule is crucial, since our parametricity interpretation does not validate the rule. We discuss the conjunction rule further in Section 10.

5 Semantics of Programming Language

Let *Loc* be a countably infinite set of locations. The programming language is interpreted in the category of FM-cpos on *Loc*.

We remind the reader of the basics of FM domain theory. Call a bijection π on *Loc* a *permutation* when $\pi(l) \neq l$ only for finitely many l, and let perm be the set of all permutations. An FM-set is a pair of a set A and a function $\cdot$ of type $\mathsf{perm} \times A \to A$, such that (1) $\mathsf{id} \cdot a = a$ and $\pi \cdot (\pi' \cdot a) = (\pi \circ \pi') \cdot a$, and (2) every $a \in A$ is *supported* by some finite subset L of *Loc*, i.e.,

$$\forall \pi \in \mathsf{perm}.\ (\forall l \in L.\ \pi(l) = l) \Longrightarrow \pi \cdot a = a.$$

It is known that every element a in an FM-set A has a smallest set L that supports a. This smallest set is denoted $\mathsf{supp}(a)$. An FM function f from an FM-set A to an FM-set B is a function from A to B such that $f(\pi \cdot a) = \pi \cdot (f(a))$ for all a, π.

An FM-poset is an FM-set A with a partial order $\sqsubseteq$ on A such that $a \sqsubseteq b \Longrightarrow \pi \cdot a \sqsubseteq \pi \cdot b$ for all π, a, b. We say that a (ω-)chain $\{a_i\}_i$ in FM-poset A is *finitely supported* iff there is a finite subset L of *Loc* that supports all elements in the chain. Finally, an FM-cpo is an FM-poset $(A, \sqsubseteq)$ for which every finitely-supported chain $\{a_i\}_i$ has a least upper bound, and an FM continuous function f from an FM-cpo A to an FM-cpo B is an FM function from A to B that preserves the least upper bounds of all finitely supported chains.

Types are interpreted as pointed FM-cpos, using the categorical structure of the category of FM-cpos, see Figure 3. In the figure, we use the FM-cpo *ref* of references defined by: $\mathit{ref} \overset{def}{=} \mathit{Loc} + \{\mathit{nil}\}$ with $\pi \cdot v \overset{def}{=} \mathsf{if}\ (v = \mathit{nil})\ \mathsf{then}\ \mathit{nil}\ \mathsf{else}\ \pi(v)$. The only nonstandard part is the semantics of the command type com, which we define in the continuation passing style following [17,3]:

$$O \overset{def}{=} \{\mathit{normal}, \mathit{err}\}_\bot \text{ (with } \pi \cdot o = o) \quad \mathit{Heap} \overset{def}{=} \mathit{Loc} \rightharpoonup_{\mathsf{fin}} \mathit{ref}$$
$$\mathit{cont} \overset{def}{=} (\mathit{Heap} \to O) \qquad [\![\mathsf{com}]\!] \overset{def}{=} (\mathit{Heap} \times \mathit{cont} \to O).$$

Here $A \times B$ and $A \to B$ are cartesian product and exponential in the category of FM-cpos. And $A \rightharpoonup_{\mathsf{fin}} B$ is the FM-cpo of the finite partial functions from A to B whose order and permutation action are defined below:

$$\begin{array}{lll} \mathit{ref} \overset{def}{=} \mathit{Loc} + \{\mathit{nil}\} & [\![\mathsf{ref} \to \tau]\!] \overset{def}{=} \mathit{ref} \to [\![\tau]\!] & [\![\tau \to \tau']\!] \overset{def}{=} [\![\tau]\!] \to [\![\tau']\!] \\ [\![\mathsf{com}]\!] \overset{def}{=} \mathit{Heap} \times \mathit{cont} \to O & \multicolumn{2}{l}{(\text{where } O = \{\mathit{normal}, \mathit{err}\}_\perp \text{ and } \mathit{cont} = \mathit{Heap} \to O)} \\ [\![\Delta]\!] \overset{def}{=} \prod_{i \in \Delta} \mathit{ref} & [\![\Gamma]\!] \overset{def}{=} \prod_{x:\tau \in \Gamma} [\![\tau]\!]. & \end{array}$$

Fig. 3. Interpretation of Types and Typing Contexts

$$[\![\Delta \vdash E]\!] : [\![\Delta]\!] \to \mathit{ref} \qquad [\![\Delta, i \vdash i]\!]_\rho \overset{def}{=} \rho(i) \qquad [\![\Delta \vdash \mathsf{nil}]\!]_\rho \overset{def}{=} \mathit{nil}$$

Fig. 4. Interpretation of Expressions

1. $f \sqsubseteq g \overset{def}{\Longleftrightarrow} \mathsf{dom}(f) = \mathsf{dom}(g)$ and $f(a) \sqsubseteq g(a)$ for all $a \in \mathsf{dom}(f)$,
2. $(\pi \cdot f)(a) \overset{def}{=} \mathsf{if}\ (a \in \pi(\mathsf{dom}(f)))\ \mathsf{then}\ (\pi \cdot ((f \circ \pi^{-1})(a)))\ \mathsf{else}$ undefined.

The first FM-cpo O specifies all possible observations, which are normal termination *normal*, erroneous termination *err* or divergence $\perp$. The next FM-cpo *Heap* denotes the set of heaps. It formalizes that a heap contains only finitely many allocated cells and each cell in the heap contains a reference. The third FM-cpo *cont* represents the set of continuations that consume heaps. Finally, $[\![\mathsf{com}]\!]$ is the set of cps-style commands. Those commands take a current heap h and a continuation k, and compute an observation in O (often by computing a final heap h', and calling the given continuation k with h').

Note that *Heap* has the usual heap disjointness predicate $h \# h'$, which denotes the disjointness of $\mathsf{dom}(h)$ and $\mathsf{dom}(h')$, and the usual partial heap combining operator $\bullet$, which takes the union of (the graphs of) two disjoint heaps. The # predicate and $\bullet$ operator fit well with FM domain theory, because they preserve all permutations: $h \# h' \iff (\pi \cdot h) \# (\pi \cdot h')$ and $\pi \cdot (h \bullet h') = (\pi \cdot h) \bullet (\pi \cdot h')$.

The semantics of typing contexts Δ and Γ is given by cartesian products: $[\![\Delta]\!] \overset{def}{=} \prod_{i \in \Delta} \mathit{ref}$ and $[\![\Gamma]\!] \overset{def}{=} \prod_{x:\tau \in \Gamma} [\![\tau]\!]$. The products here are taken over finite families, so they give well-defined FM-cpos.[4] We will use symbols ρ and η to denote environments in $[\![\Delta]\!]$ and $[\![\Gamma]\!]$, respectively.

The semantics of expressions and terms is shown in Figures 4 and 5. It is standard, except for the case of allocation, where we make use of the underlying FM domain theory: The interpretation picks a location that is fresh with respect to currently known values (i.e., $\mathsf{supp}(h, \eta, \rho)$) as well as those that will be used by the continuation (i.e., $\mathsf{supp}(k)$). The cps-style interpretation gives us an explicit handle on which locations are used by the continuation, and the FM domain theory ensures that $\mathsf{supp}(h, \eta, \rho, k)$ is finite (so a new location l can be chosen) and that the choice of l does not matter, as long as l is not in $\mathsf{supp}(h, \eta, \rho, k)$. We borrowed this interpretation from Benton and Leperchey [3].

[4] An infinite product of FM-cpos is not necessarily an FM-cpo.

$$\llbracket \Delta \mid \Gamma \vdash M{:}\,\tau \rrbracket \;:\; \llbracket \Delta \rrbracket \times \llbracket \Gamma \rrbracket \to \llbracket \tau \rrbracket$$

$$\begin{aligned}
\llbracket \Delta \mid \Gamma, x{:}\,\tau \vdash x{:}\,\tau \rrbracket_{\rho,\eta} &\overset{def}{=} \eta(x) \\
\llbracket \Delta \mid \Gamma \vdash \lambda i.\, M{:}\, \mathsf{ref} \to \tau \rrbracket_{\rho,\eta} &\overset{def}{=} \lambda v{:}\, \mathit{ref}.\, \llbracket \Delta, i \mid \Gamma \vdash M{:}\,\tau \rrbracket_{\rho[i\to v],\eta} \\
\llbracket \Delta \mid \Gamma \vdash M\ E{:}\,\tau \rrbracket_{\rho,\eta} &\overset{def}{=} (\llbracket \Delta \mid \Gamma \vdash M{:}\, \mathsf{ref} \to \tau \rrbracket_{\rho,\eta})\ \llbracket E \rrbracket_{\rho} \\
\llbracket \Delta \mid \Gamma \vdash \lambda x{:}\,\tau'.\, M{:}\,\tau' \to \tau \rrbracket_{\rho,\eta} &\overset{def}{=} \lambda m{:}\, \llbracket \tau' \rrbracket.\, \llbracket \Delta \mid \Gamma, x{:}\,\tau' \vdash M{:}\,\tau \rrbracket_{\rho,\eta[x\to m]} \\
\llbracket \Delta \mid \Gamma \vdash M\ N{:}\,\tau \rrbracket_{\rho,\eta} &\overset{def}{=} (\llbracket \Delta \mid \Gamma \vdash M{:}\,\tau' \to \tau \rrbracket_{\rho,\eta})\ \llbracket \Delta \mid \Gamma \vdash N{:}\,\tau' \rrbracket_{\rho,\eta} \\
\llbracket \Delta \mid \Gamma \vdash \mathsf{fix}\ M{:}\,\tau \rrbracket_{\rho,\eta} &\overset{def}{=} \mathit{leastfix}\ \llbracket \Delta \mid \Gamma \vdash M{:}\,\tau \to \tau \rrbracket_{\rho,\eta} \\
\llbracket \Delta \mid \Gamma \vdash \mathsf{if}\ (E{=}F)\ M\ N{:}\, \mathsf{com} \rrbracket_{\rho,\eta} &\overset{def}{=} \mathsf{if}\ \llbracket E \rrbracket_{\rho} {=} \llbracket F \rrbracket_{\rho}\ \mathsf{then}\ \llbracket \Delta \mid \Gamma \vdash M{:}\, \mathsf{com} \rrbracket_{\rho,\eta} \\
&\quad\ \mathsf{else}\ \llbracket \Delta \mid \Gamma \vdash N{:}\, \mathsf{com} \rrbracket_{\rho,\eta} \\
\llbracket \Delta \mid \Gamma \vdash M; N{:}\, \mathsf{com} \rrbracket_{\rho,\eta}(h,k) &\overset{def}{=} \mathsf{let}\ k'\ \mathsf{be}\ \lambda h'.\, \llbracket \Delta \mid \Gamma \vdash N{:}\, \mathsf{com} \rrbracket_{\rho,\eta}(h',k) \\
&\quad\ \mathsf{in}\ \llbracket \Delta \mid \Gamma \vdash M{:}\, \mathsf{com} \rrbracket_{\rho,\eta}(h,k') \\
\llbracket \Delta \mid \Gamma \vdash \mathsf{let}\ i{=}\mathsf{new}\ \mathsf{in}\ M{:}\, \mathsf{com} \rrbracket_{\rho,\eta}(h,k) &\overset{def}{=} \llbracket \Delta, i \mid \Gamma \vdash M{:}\, \mathsf{com} \rrbracket_{\rho[i\to l],\eta}(h \bullet [l{\to}\mathit{nil}], k) \\
&\qquad\qquad (\text{where } l \in (\mathit{Loc} {-} \mathsf{supp}(h,\rho,\eta,k))) \\
\llbracket \Delta \mid \Gamma \vdash \mathsf{free}(E){:}\, \mathsf{com} \rrbracket_{\rho,\eta}(h,k) &\overset{def}{=} \mathsf{if}\ \llbracket E \rrbracket_{\rho} {\notin} \mathsf{dom}(h)\ \mathsf{then}\ \mathit{err} \\
&\quad\ \mathsf{else}\ (k(h')\ \text{for}\ h'\ \text{s.t.}\ h' \bullet [\llbracket E \rrbracket_{\rho} {\to} h(\llbracket E \rrbracket_{\rho})] = h) \\
\llbracket \Delta \mid \Gamma \vdash \mathsf{let}\ i{=}[E]\ \mathsf{in}\ M{:}\, \mathsf{com} \rrbracket_{\rho,\eta}(h,k) &\overset{def}{=} \mathsf{if}\ \llbracket E \rrbracket_{\rho} {\notin} \mathsf{dom}(h)\ \mathsf{then}\ \mathit{err} \\
&\quad\ \mathsf{else}\ \llbracket \Delta, i \mid \Gamma \vdash M{:}\, \mathsf{com} \rrbracket_{\rho[i\to h(\llbracket E \rrbracket_{\rho})],\eta}(h,k) \\
\llbracket \Delta \mid \Gamma \vdash [E]{=}F{:}\, \mathsf{com} \rrbracket_{\rho,\eta}(h,k) &\overset{def}{=} \mathsf{if}\ \llbracket E \rrbracket_{\rho} {\notin} \mathsf{dom}(h)\ \mathsf{then}\ \mathit{err}\ \mathsf{else}\ k(h[\llbracket E \rrbracket_{\rho} {\to} \llbracket F \rrbracket_{\rho}])
\end{aligned}$$

Fig. 5. Interpretation of Terms

6 Relational Interpretation of Separation Logic

We now present the main result of this paper, a relational interpretation of separation logic. In this interpretation, a specification means a relation on terms, rather than a set of terms "satisfying" the specification. This relational reading formalizes the intuitive claim that proof rules in separation logic ensure parametricity with respect to the heap.

Our interpretation has two important components that ensure parametricity. The first is a Kripke structure $\mathcal{R}$. The possible worlds of $\mathcal{R}$ are finitely supported binary relations r on heaps,[5] and the accessibility relation is the preorder defined by the separating conjunction for relations:

$$\begin{aligned}
h_0[r * s]h_1 &\overset{def}{\Leftrightarrow} \text{there exist splittings } n_0 \bullet m_0 = h_0 \text{ and } n_1 \bullet m_1 = h_1 \text{ such that} \\
&\qquad n_0[r]n_1 \text{ and } m_0[s]m_1, \\
r \sqsubseteq r' &\overset{def}{\Leftrightarrow} \text{there exists } s \text{ such that } r * s = r'.
\end{aligned}$$

Intuitively, $r \sqsubseteq r'$ means that r' is a $*$-extension of r by some s. The Kripke structure $\mathcal{R}$ parameterizes our interpretation, and it guarantees that all the logical connectives behave parametrically wrt. relations between internal resource invariants.

[5] A relation r is finitely supported iff there is $L \subseteq_{\mathsf{fin}} \mathit{Loc}$ s.t. for every permutation π, if $\pi(l) = l$ for all $l \in L$, then $\forall h_0, h_1.\, h_0[r]h_1 \iff (\pi \cdot h_0)[r](\pi \cdot h_1)$.

The second is *semantic quadruples*, which describe the relationship between two commands. We use the semantic quadruples to interpret Hoare triples relationally. Consider $c_0, c_1 \in [\![\mathsf{com}]\!]$ and $r, s \in \mathcal{R}$. For each subset D_0 of an FM-cpo D, define $\mathsf{eq}(D_0)$ to be the partial identity relation on D that equates only the elements in D_0. A *semantic quadruple* $[r](c_0, c_1)[s]$ holds iff

$$\begin{array}{l}\forall r' \in \mathcal{R}. \forall h_0, h_1 \in \mathit{Heap}. \forall k_0, k_1 \in \mathit{cont}. \\ \quad (h_0[r * r']h_1 \wedge k_0[s * r' \to \mathsf{eq}(G)]k_1) \Longrightarrow (c_0(h_0, k_0)[\mathsf{eq}(G)]c_1(h_1, k_1)),\end{array}$$

where G is the set $O - \{\mathit{err}\} = \{\mathit{normal}, \bot\}$ of good observations. The above condition indirectly expresses that if the input heaps h_0, h_1 are $r * r'$-related, then the output heaps are related by $s * r'$. Note that the definition quantifies over relations r' for new heaps, thus implementing relational parametricity. In Section 7, we show how semantic quadruples are related to a more direct way of relating two commands and we also show that the parametricity in the definition of semantic quadruples implies the locality condition in separation logic [16].

The semantics of the logic is defined by the satisfaction relation $\models_{\Delta|\Gamma}$ between $[\![\Delta]\!] \times [\![\Gamma]\!]^2 \times \mathcal{R}$ and Specs, such that $\models_{\Delta|\Gamma}$ satisfies Kripke monotonicity:

$$(\rho, \eta_0, \eta_1, r \models_{\Delta|\Gamma} \varphi) \wedge (r \sqsubseteq r') \Longrightarrow (\rho, \eta_1, \eta_2, r' \models_{\Delta|\Gamma} \varphi).$$

One way to understand the satisfaction relation is to assume two machines that execute terms in the context of one specific module. Intuitively, the $(\rho, \eta_0, \eta_1, r)$ parameter of $\models$ specifies the configurations of those machines: one machine uses (ρ, η_0) to bind free stack variables and identifiers of terms, and the other machine uses (ρ, η_1) for the same purposes; and the internal resource invariants of the modules in those machines are related by r. The judgment $(\rho, \eta_0, \eta_1, r)$ means that if two machines are configured by $(\rho, \eta_0, \eta_1, r)$, then the meanings of the terms in two machines are φ-related. Note that we allow different environments for the Γ context only, not for the Δ context. This is because we are mainly concerned with parametricity with respect to the heap and only Γ entities, not Δ entities, depend on the heap.

Figure 6 shows the detailed interpretation of specifications. In the figure, we make use of the standard semantics of assertions [16]. We now explain three cases in the definition of $\models$.

The first case is implication. Our interpretation of implication exploits the specific notion of accessibility in $\mathcal{R}$. It is equivalent to the standard Kripke semantics of implication:

$$\text{for all } r' \in \mathcal{R}, \text{ if } r \sqsubseteq r' \text{ and } (\rho, \eta_0, \eta_1, r') \models \varphi, \text{ then } (\rho, \eta_0, \eta_1, r') \models \psi,$$

because $r \sqsubseteq r'$ iff $r' = r * s$ for some s.

The second case is quantification. If a stack variable i is quantified, we consider one semantic value, but if an identifier x is quantified, we consider two semantic values. This is again to reflect that in our relational interpretation, we are mainly concerned with heap-dependent entities. Thus, we only read quantifiers for heap-dependent entities x relationally.

For all environments $\rho \in [\![\Delta]\!]$ and $\eta_0, \eta_1 \in [\![\Gamma]\!]$ and all worlds $r \in \mathcal{R}$,

$$\begin{array}{lcl}
(\rho,\eta_0,\eta_1,r) \models \{P\}M\{Q\} & \overset{def}{\iff} & [\mathsf{eq}([\![P]\!]_\rho) * r]([\![M]\!]_{\rho,\eta_0}, [\![M]\!]_{\rho,\eta_1})[\mathsf{eq}([\![Q]\!]_\rho) * r] \\
(\rho,\eta_0,\eta_1,r) \models \varphi \otimes P & \overset{def}{\iff} & (\rho,\eta_0,\eta_1, r * \mathsf{eq}([\![P]\!]_\rho)) \models \varphi \\
(\rho,\eta_0,\eta_1,r) \models E = F & \overset{def}{\iff} & [\![E]\!]_\rho = [\![F]\!]_\rho \\
(\rho,\eta_0,\eta_1,r) \models M = N & \overset{def}{\iff} & [\![M]\!]_{\rho,\eta_0} = [\![N]\!]_{\rho,\eta_0} \text{ and } [\![M]\!]_{\rho,\eta_1} = [\![N]\!]_{\rho,\eta_1} \\
(\rho,\eta_0,\eta_1,r) \models \varphi \Rightarrow \psi & \overset{def}{\iff} & \text{for all } s \in \mathcal{R}, \text{ if } (\rho,\eta_0,\eta_1,r*s) \models \varphi, \\
 & & \quad \text{then } (\rho,\eta_0,\eta_1,r*s) \models \psi \\
(\rho,\eta_0,\eta_1,r) \models \forall i.\, \varphi & \overset{def}{\iff} & \text{for all } v \in \mathit{ref},\ (\rho[i\to v],\eta_0,\eta_1,r) \models \varphi \\
(\rho,\eta_0,\eta_1,r) \models \exists i.\, \varphi & \overset{def}{\iff} & \text{there exists } v \in \mathit{ref} \text{ s.t. } (\rho[i\to v],\eta_0,\eta_1,r) \models \varphi \\
(\rho,\eta_0,\eta_1,r) \models \forall x{:}\tau.\, \varphi & \overset{def}{\iff} & \text{for all } m,n \in [\![\tau]\!],\ (\rho,\eta_0[x\to m],\eta_1[x\to n],r) \models \varphi \\
(\rho,\eta_0,\eta_1,r) \models \exists x{:}\tau.\, \varphi & \overset{def}{\iff} & \text{there exist } m,n \in [\![\tau]\!] \text{ s.t. } (\rho,\eta_0[x\to m],\eta_1[x\to n],r) \models \varphi \\
(\rho,\eta_0,\eta_1,r) \models \varphi \wedge \psi & \overset{def}{\iff} & (\rho,\eta_0,\eta_1,r) \models \varphi \text{ and } (\rho,\eta_0,\eta_1,r) \models \psi \\
(\rho,\eta_0,\eta_1,r) \models \varphi \vee \psi & \overset{def}{\iff} & (\rho,\eta_0,\eta_1,r) \models \varphi \text{ or } (\rho,\eta_0,\eta_1,r) \models \psi
\end{array}$$

Fig. 6. Relational Interpretation of Separation Logic

The last case is invariant extension $\varphi \otimes P$. Mathematically, it says that if we extend the r parameter by the partial equality for predicate P, specification φ holds. Intuitively, this means that some heap cells not appearing in a specification φ satisfy the invariant P.

A specification $\Delta \mid \Gamma \vdash \varphi$ is *valid* iff $(\rho,\eta_0,\eta_1,r) \models \varphi$ holds for all (ρ,η_0,η_1,r). A proof rule is *sound* when it is a valid axiom or an inference rule that concludes a valid specification from valid premises.

Theorem 1. *All the proof rules in our logic are sound.*

7 Properties of Semantic Quadruples

In this section, we prove two properties of semantic quadruples. The first clarifies the connection between our new interpretation of Hoare triples and the standard interpretation, and the second shows how our cps-style semantic quadruples are related to a more direct way of relating two commands.

First, we consider the relation between semantic quadruples and Hoare triples. Define an operator cps that cps-transforms a state transformer semantically:

$$\begin{array}{rcl}
\mathsf{cps}_D & : & (\mathit{Heap} \to (\mathit{Heap} + \{\mathit{err}\})_\bot) \ \to\ (\mathit{Heap} \times \mathit{cont} \to O) \\
\mathsf{cps}_D(c) & \overset{def}{=} & \lambda(h,k).\ \mathsf{if}\ (c(h) \notin \{\bot, \mathit{err}\})\ \mathsf{then}\ k(c(h))\ \mathsf{else}\ c(h).
\end{array}$$

Proposition 1. *For all $p, q \subseteq \mathit{Heap}$ and all $c \in \mathit{Heap} \to (\mathit{Heap} \times D + \{\mathit{err}\})_\bot$, quadruple $[\mathsf{eq}(p)](\mathsf{cps}(c), \mathsf{cps}(c))[\mathsf{eq}(q)]$ holds iff the below two conditions hold:*

1. *for every h in p, either $c(h) = \bot$ or $c(h) \in q$, hence $c(h)$ cannot be err;*
2. *for every h in p and h_1 such that $h\#h_1$,*
   (a) *if $c(h) = \bot$, then $c(h \bullet h_1) = \bot$,*
   (b) *if $c(h) \neq \bot$, then $c(h) \bullet h_1$ is defined and equal to $c(h \bullet h_1)$.*

Note that the first condition is the usual meaning of Hoare triples, and the second the locality condition of commands in separation logic restricted to heaps in p [16]. Since the locality condition merely expresses the parametricity of commands with respect to new heaps, the proposition indicates that our interpretation of triples is the usual one enhanced by an additional parametricity requirement.

Next, we relate our cps-style notion of semantic quadruples to the direct-style alternative. The notion underlying this relationship is the observation closure, denoted $(-)^{\bot\bot}$. For each FM-cpo D and relation $r \subseteq D \times D$, we define two relations, $r^{\bot}$ on $[D \to O]$ and $r^{\bot\bot}$ on D, as follows:

$$k_1[r^{\bot}]k_2 \overset{def}{\iff} \forall d_1, d_2 \in D.\ (d_1[r]d_2 \Longrightarrow k_1(d_1)[\mathsf{eq}(G)]k_2(d_2)),$$
$$d_1[r^{\bot\bot}]d_2 \overset{def}{\iff} \forall k_1, k_2 \in [D \to O].\ (k_1[r^{\bot}]k_2 \Longrightarrow k_1(d_1)[\mathsf{eq}(G)]k_2(d_2)).$$

Operator $(-)^{\bot}$ dualizes a relation on D to one on observations on D, and $(-)^{\bot\bot}$ closes a given relation r under observations.

Proposition 2. *Let r, s be relations in $\mathcal{R}$, and let c_1, c_2 be functions of type $Heap \to (Heap + \{err\})_{\bot}$. A quadruple $[r](\mathsf{cps}(c_1), \mathsf{cps}(c_2))[s]$ holds, iff*

$$\forall(r', h_1, h_2).\ h_1[r * r']h_2 \Longrightarrow (c_1(h_1){=}c_2(h_2){=}\bot \vee c_1(h_1)[(s * r')^{\bot\bot}]c_2(h_2)).$$

This proposition shows that our semantic quadruples are close to what one might expect at first for relating two commands parametrically. The only difference is that our quadruple always closes the post-relation $s * r'$ under observations.

8 Abstraction Theorem

The abstraction theorem below formalizes that well-specified programs (specified in separation logic with implicit quantification over internal resource invariants by frame rules) behave relationally parametrically in internal resource invariants. The easiest way to understand this intuition may be from the corollary following the theorem.

Some readers might feel that it is too much to call the abstraction theorem a "theorem" since it really is a trivial corollary of the soundness theorem — but that is just as it should be: the semantics was defined to achieve that.

Theorem 2 (Abstraction Theorem). *If $\Delta \mid \Gamma \vdash \varphi$ is provable in the logic, then for all $(\rho, \eta_0, \eta_1, r) \in [\![\Delta]\!] \times [\![\Gamma]\!]^2 \times \mathcal{R}$, we have that $(\rho, \eta_0, \eta_1, r) \models \varphi$.*

Proof. By Theorem 1, we get that $\Delta \mid \Gamma \vdash \varphi$ is valid, which is just what the conclusion expresses. □

Corollary 1. *Suppose that $\Delta \mid x{:}\,\mathsf{com} \vdash \{P_1\}x\{Q_1\} \Rightarrow \{P\}M\{Q\}$ is provable in the logic. Then for all (ρ, c_0, c_1, r), if $[\mathsf{eq}([\![P_1]\!]_\rho) * r](c_0, c_1)[\mathsf{eq}([\![Q_1]\!]_\rho) * r]$ holds, then $[\mathsf{eq}([\![P]\!]_\rho) * r]([\![M]\!]_{[x \to c_0]}, [\![M]\!]_{[x \to c_1]})[\mathsf{eq}([\![Q]\!]_\rho) * r]$ holds as well.*

$$\mathsf{put}_1 \equiv (\lambda i.\,\mathsf{let}\ j = [i]\ \mathsf{in}\ (\mathsf{free}(i); [k] := j) \qquad \mathsf{get}_1 \equiv (\lambda i.\,\mathsf{let}\ j = [k]\ \mathsf{in}\ [i] := j)$$

$$\{i \mapsto j * k \mapsto \text{-}\}\mathsf{put}_1(i)\{k \mapsto \text{-}\} \qquad \{i \mapsto \text{-} * k \mapsto \text{-}\}\mathsf{get}_1(i)\{i \mapsto \text{-} * k \mapsto \text{-}\}$$

$$\mathsf{put}_2 \equiv (\lambda i.\,\mathsf{let}\ k'{=}[k]\ \mathsf{in}\ (\mathsf{free}(k'); [k]{:=}i)) \quad \mathsf{get}_2 \equiv (\lambda i.\,\mathsf{let}\ k'{=}[k]\ \mathsf{in}\ \mathsf{let}\ j{=}[k']\ \mathsf{in}\ [i]{:=}j)$$

$$\{i \mapsto j * \exists k'.k \mapsto k' * k' \mapsto \text{-}\}\mathsf{put}_2(i)\{\exists k'.k \mapsto k' * k' \mapsto \text{-}\}$$

$$\{i \mapsto \text{-} * \exists k'.k \mapsto k' * k' \mapsto \text{-}\}\mathsf{get}_2(i)\{i \mapsto \text{-} * \exists k'.k \mapsto k' * k' \mapsto \text{-}\}$$

$$c \equiv (\mathsf{let}\ i''{=}\mathsf{new}\ \mathsf{in}\ [i'']{:=}i'; \mathsf{put}(i''); \mathsf{get}(i'))$$

$$\Delta \mid \Gamma \vdash (\forall i.\{P_1\}\mathsf{put}(i)\{Q_1\} \wedge \{P_2\}\mathsf{get}(i)\{Q_2\}) \Rightarrow \{i' \mapsto \text{-}\}c\{i' \mapsto \text{-}\}$$

$$\text{(where } \Delta = \{i', k\} \text{ and } \Gamma = \{\mathsf{put}\colon \mathsf{ref} \to \mathsf{com}, \mathsf{get}\colon \mathsf{ref} \to \mathsf{com}\})$$

Fig. 7. Two Implementations of a Buffer and a Simple Client

Intuitively, x corresponds to a module with a single operation, and M a client of the module. This corollary says that if we prove a property of the client M, assuming only an abstract external specification $\{P_1\}x\{Q_1\}$ of the module, the client cannot tell apart two different implementations c_0, c_1 of the module, as long as c_0, c_1 have identical external behavior. The four instances of eq in the proposition formalize that the external behaviors of c_0, c_1 are identical and that the client M behaves the same externally regardless of whether it is used with c_0 or c_1. The relation r is a simulation relation for internal resource invariants of c_0 and c_1.

Proof. Define environments η_0, η_1 and heap sets p, p_1, q, q_1 as follows:

$$\eta_0 = [x{\to}c_0],\ \eta_1 = [x{\to}c_1],\ \text{and } (p_1, q_1, p, q) = ([\![P_1]\!]_\rho, [\![Q_1]\!]_\rho, [\![P]\!]_\rho, [\![Q]\!]_\rho).$$

By Theorem 2, we have, for any r, that $(\rho, \eta_0, \eta_1, r) \models \{P_1\}x\{Q_1\} \Rightarrow \{P\}M\{Q\}$. From this, we derive the conclusion of the proposition:

$$\begin{aligned}
& (\rho, \eta_0, \eta_1, r) \models \{P_1\}x\{Q_1\} \Rightarrow \{P\}M\{Q\} \\
\Longrightarrow\ & (\forall s \in \mathcal{R}.\ (\rho, \eta_0, \eta_1, r * s) \models \{P_1\}x\{Q_1\} \Longrightarrow (\rho, \eta_0, \eta_1, r * s) \models \{P\}M\{Q\}) \\
\Longrightarrow\ & ((\rho, \eta_0, \eta_1, r) \models \{P_1\}x\{Q_1\} \Longrightarrow (\rho, \eta_0, \eta_1, r) \models \{P\}M\{Q\}) \\
\Longrightarrow\ & ([\mathsf{eq}(p_1) * r](c_0, c_1)[\mathsf{eq}(q_1) * r] \Longrightarrow [\mathsf{eq}(p) * r]([\![M]\!]_{\eta_0}, [\![M]\!]_{\eta_1})[\mathsf{eq}(q) * r]). \quad \square
\end{aligned}$$

9 Example

For reasons of space we only include one very simple example (but at least it does involve ownership transfer).

We will consider a mutable abstract data type that is a buffer of size one. It has operations put and get. Intuitively, $\mathsf{put}(i)$ stores the value found at i in the buffer and $\mathsf{get}(i)$ retrieves the value stored in the buffer and stores it at i. Let $P_1 \equiv i \mapsto j$, and $Q_1 \equiv \mathsf{emp}$, and $P_2 \equiv i \mapsto \text{-}$, and $Q_2 \equiv i \mapsto \text{-}$, where - denotes existentially quantified variables. We assume the following abstract specifications of this mutable abstract data type: $\{P_1\}\mathsf{put}(i)\{Q_1\}$ and $\{P_2\}\mathsf{get}(i)\{Q_2\}$.

Figure 7 shows two implementations of the buffer and a client. The figure also includes the concrete specifications for the implementation and a specification for the buffer. Note that the first implementation just uses one cell for the buffer and that the implementation follows the intuitive description given above. The second implementation uses two cells for the buffer. The additional cell is used to hold the cell pointed to by i itself. Note that this additional cell is transferred from the caller of $\mathsf{put}_2(i)$, i.e., a client of the buffer. Finally, the specification of the client describes the safety property of c, assuming the abstract specification for the buffer.

Pick $\rho \in [\![\{i', k\}]\!]$, and define $f_1, f_2, g_1, g_2, c_1, c_2$ as follows:

$$f_i \overset{def}{=} [\![\mathsf{put}_i]\!]_{\rho,[]}, \quad g_i \overset{def}{=} [\![\mathsf{get}_i]\!]_{\rho,[]}, \quad c_i \overset{def}{=} [\![c]\!]_{\rho,[\mathsf{put}\to f_i,\mathsf{get}\to g_i]}.$$

Now, by the Abstraction Theorem, we get that, for all r,

$$\begin{aligned}&\big(\forall v \in \mathit{ref}.\ [\mathsf{eq}([\![P_1]\!]_{\rho[i\to v]}) * r](f_1(v), f_2(v))[\mathsf{eq}([\![Q_1]\!]_{\rho[i\to v]}) * r] \wedge \\ &\qquad [\mathsf{eq}([\![P_2]\!]_{\rho[i\to v]}) * r](g_1(v), g_2(v))[\mathsf{eq}([\![Q_2]\!]_{\rho[i\to v]}) * r]\big) \\ &\Rightarrow [\mathsf{eq}([\![i' \mapsto \text{-},\text{-}]\!]_\rho) * r](c_1, c_2)[\mathsf{eq}([\![i' \mapsto \text{-},\text{-}]\!]_\rho) * r].\end{aligned} \tag{1}$$

We now sketch a consequence of this result; for brevity we allow ourselves to be a bit informal. Fix location k and let r be the following simulation relation between the two implementations: $r = \{(h_1, h_2) \mid \exists j.\, h_1 = [k{\to}j] \wedge \exists k'.\, h_2 = [k{\to}k'] \bullet [k'{\to}j]\}$. Then one can verify that the antecedent of the implication in (1) holds, and thus conclude that $[\mathsf{eq}([\![i' \mapsto \text{-}]\!]_\rho) * r](c_1, c_2)[\mathsf{eq}([\![i' \mapsto \text{-}]\!]_\rho) * r]$ holds. Take $(h_1, h_2) \in \mathsf{eq}([\![i' \mapsto \text{-}]\!]_\rho) * r$, and denote the result of running c_1 on h_1 by h_1', and the result of running c_2 on h_2 by h_2'. We then conclude that h_1' will be of the form $h_{11}' \bullet h_{12}'$ and that h_2' will be of the form $h_{21}' \bullet h_{22}'$ with $(h_{12}', h_{22}') \in r$ and with $(h_{11}', h_{21}') \in \mathsf{eq}([\![i' \mapsto \text{-}]\!]_\rho)$.

Thus the relation between the internal resource invariants is maintained and, for the visible part, c_1 and c_2 both produce the *same heap* with exactly one cell.

10 Conclusion and Future Work

We have succeeded in defining the first relationally parametric model of separation logic. The model captures the informal idea that well-specified clients of mutable abstract data types should behave parametrically in the internal resource invariants of the abstract data type.

We see our work as a first step towards devising a logic for reasoning about mutable abstract data types, similar in spirit to Abadi and Plotkin's logic for parametricity [14,5]. To this end, we also expect to make use of the ideas of relational separation logic in [19] for reasoning about relations between different programs syntactically. The logic should include a link between separation logic and relational separation logic so that one could get a syntactic representation of the semantic Abstraction Theorem and its corollary presented above.

One can also think of our work as akin to the O'Hearn-Reynolds model for idealized algol based on translation into a relationally parametric polymorphic linear lambda calculus [10]. In *loc. cit.* O'Hearn and Reynolds show how to provide a better model of stack variables for idealized algol by making a formal connection to parametricity. Here we provide a better model for the more unwieldy world of heap storage by making a formal connection to parametricity.

As mentioned in Section 4, the conjunction rule is not sound in our model. This is a consequence of our cps-style interpretation. We don't know whether it is possible to develop a parametric model in which the conjunction rule is sound.

Future work further includes developing a parametric model for the higher-order version of separation logic with explicit quantification over internal resource invariants. Finally, we hope that ideas similar to those presented here can be used to develop parametric models for other recent approaches to mutable abstract data types (e.g., [1]).

References

1. M. Barnett and D. Naumann. Towards imperative modules: Reasoning about invariants and sharing of mutable state. In *Proc. of LICS'04*, 2004.
2. N. Benton. Abstracting Allocation:The New new Thing. In *Proc. of CSL'06*, 2006.
3. N. Benton and B. Leperchey. Relational reasoning in a nominal semantics for storage. In *Proc. of TLCA'05*, pages 88–101, Nara, Japan, 2005.
4. B. Biering, L. Birkedal, and N. Torp-Smith. BI-hyperdoctrines and higher order separation logic. In *Proc. of ESOP'05*, pages 233–247, Edinburgh, UK, 2005.
5. L. Birkedal and R. Møgelberg. Categorical models for Abadi-Plotkin's logic for parametricity. *Mathematical Structures in Computer Science*, 15:709–772, 2005.
6. L. Birkedal, N. Torp-Smith, and J. C. Reynolds. Local reasoning about a copying garbage collector. In *Proc. of POPL'04*, pages 220–231, Venice, Italy, 2004.
7. L. Birkedal, N. Torp-Smith, and H. Yang. Semantics of separation-logic typing and higher-order frame rules. In *Proc. of LICS'05*, pages 260–269, 2005.
8. I. Mijajlović, N. Torp-Smith, and P. O'Hearn. Refinement and separation context. In *Proc. of FSTTCS'04*, pages 421–433, Chennai, India, 2004.
9. I. Mijajlović and H. Yang. Data refinements with low-level pointer operations. In *Proc. of APLAS'05*, pages 19–36, Tsukuba, Japan, 2005.
10. P. O'Hearn and J. Reynolds. From Algol to polymorphic linear lambda-calculus. *Journal of the ACM*, 47(1):167–223, 2000.
11. P. W. O'Hearn, H. Yang, and J. C. Reynolds. Local reasoning about programs that alter data structures. In *Proc. of CSL'01*, pages 1–19, Paris, France, 2001.
12. P. W. O'Hearn, H. Yang, and J. C. Reynolds. Separation and information hiding. In *Proc. of POPL'04*, pages 268–280, Venice, Italy, 2004.
13. M. Parkinson and G. Bierman. Separation logic and abstraction. In *Proc. of POPL'05*, pages 247–258, Long Beach, CA, USA, 2005.
14. G. Plotkin and M. Abadi. A logic for parametric polymorphism. In *Proc. of TLCA'93*, pages 361–375, Utrecht, Netherlands, 1993.
15. J. Reynolds. Types, abstraction, and parametric polymorphism. *Information Processing*, 83:513–523, 1983.
16. J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proc. of LICS'02*, pages 55–74, Copenhagen, Denmark, 2002.
17. M. R. Shinwell and A. M. Pitts. On a monadic semantics for freshness. *Theoretical Computer Science*, 342:28–55, 2005.
18. N. Torp-Smith. *Advances in Separation Logic — A Study of Logics for Reasoning about Stateful Programs.* PhD thesis, IT University of Copenhagen, 2005.
19. H. Yang. Relational separation logic. *Theoretical Comput. Sci.*, 2005. (to appear).

Model-Checking One-Clock Priced Timed Automata

Patricia Bouyer[1,⋆], Kim G. Larsen[2,⋆⋆], and Nicolas Markey[1,⋆]

[1] LSV, CNRS & ENS de Cachan, France
{bouyer,markey}@lsv.ens-cachan.fr
[2] Aalborg University, Denmark
kgl@cs.aau.dk

Abstract. We consider the model of priced (a.k.a. weighted) timed automata, an extension of timed automata with cost information on both locations and transitions. We prove that model-checking this class of models against the logic WCTL, CTL with cost-constrained modalities, is PSPACE-complete under the "single-clock" assumption. In contrast, it has been recently proved that the model-checking problem is undecidable for this model as soon as the system has three clocks. We also prove that the model-checking of WCTL* becomes undecidable, even under this "single-clock" assumption.

1 Introduction

An interesting direction of real-time model-checking that has recently received substantial attention is the extension and re-targeting of timed automata technology towards optimal scheduling and controller synthesis [1,18,7]. In particular, as part of this effort, the notion of priced (or weighted) timed automata [4,3] has been promoted as a useful extension of the classical model of timed automata allowing continuous consumption of resources (e.g. energy) to be modelled and analyzed.

A number of optimization problems have been shown decidable for priced timed automata including minimum-cost reachability [4,3], optimal (minimum and maximum cost) reachability in multi-priced settings [17] and cost-optimal infinite schedules [6,7].

Unfortunately, the addition of cost comes with a price: certain problems become undecidable for priced timed automata. In fact, in [11] it has recently been shown that the problem of determining cost-optimal winning strategies for priced timed games is not computable. Also, by the same authors, it has been shown that the model-checking problem for priced timed automata w.r.t. WCTL —CTL with cost-constrained modalities— is undecidable [10]. In [5] it has been shown that these negative results hold even for priced timed (game) automata with no more than three clocks.

⋆ Partly supported by ACI "Sécurité & Informatique" CORTOS.
⋆⋆ Partly supported by an invited professorship from ENS Cachan.

However, when restricting to the setting of priced timed game automata with a single clock, the most recent work in [9] shows that the optimal cost of winning and (almost-) optimal strategies are computable problems. In this paper we focus on model-checking problems for priced timed automata with a single clock. In particular we show that the model-checking problem with respect to WCTL is PSPACE-complete under the "single clock" assumption. This is rather surprising as model-checking TCTL (the only cost variable is the time elapsed) under the same assumption is also PSPACE-complete [15]. We also prove that the model-checking of WCTL* becomes undecidable, even under this "single clock" assumption.

The paper is organized as follows: In Section 2, we present the model of priced timed automata, the logic WCTL and develop an example. In Section 3, we state the main result of the paper. In Section 4, we study the granularity which is required for model-checking the logic WCTL. In Section 5, we first propose an EXPTIME algorithm for model-checking one-clock priced timed automata against WCTL formulas, then refine it to get a PSPACE algorithm, and finally give an example. In Section 6, we prove that model-checking one-clock priced timed automata against WCTL* formulas is undecidable.

2 Preliminaries

2.1 Priced Timed Automata

Let $\mathcal{X}$ be a set of clock variables. The set of clock constraints (or guards) over $\mathcal{X}$ is defined by the grammar "$g ::= x \sim c \mid g \wedge g$" where $x \in \mathcal{X}$, $c \in \mathbb{N}$ and $\sim \in \{<, \leq, =, \geq, >\}$. The set of all clock constraints is denoted $\mathcal{B}(\mathcal{X})$. When a valuation $v : \mathcal{X} \to \mathbb{R}_+$ satisfies a clock constraint g is defined in a natural way (v satisfies $x \sim c$ whenever $v(x) \sim c$), and we then write $v \models g$. We denote by v_0 the valuation that assigns zero to all clock variables, by $v + t$ ($t \in \mathbb{R}_+$) the valuation that assigns $v(x) + t$ to all $x \in \mathcal{X}$, and for $R \subseteq \mathcal{X}$ we write $v[R \to 0]$ to denote the valuation that assigns zero to all variables in R and agrees with v for all $\mathcal{X} \setminus R$.

Definition 1. *A* priced timed automaton *(PTA for short) is a tuple* $\mathcal{A} = (Q, q_0, \mathcal{X}, T, \eta, (P_i)_{1 \leq i \leq p})$ *where* Q *is a finite set of* locations*,* $q_0 \in Q$ *is the* initial location*,* $\mathcal{X}$ *is a set of* clocks*,* $T \subseteq Q \times \mathcal{B}(\mathcal{X}) \times 2^{\mathcal{X}} \times Q$ *is the set of* transitions*,* $\eta \colon Q \to \mathcal{B}(\mathcal{X})$ *defines the* invariants *of each location, and* $P_i \colon Q \cup T \to \mathbb{N}$ *is a* cost (or price) function*.*

The semantics of a PTA $\mathcal{A}$ is given as a labeled timed transition system $\mathcal{T} = (S, s_0, \to)$ where $S \subseteq Q \times \mathbb{R}_+^{\mathcal{X}}$ is the set of states, $s_0 = (q_0, v_0)$[1] is the initial state, and the transition relation $\to \subseteq S \times \mathbb{R}_+^p \times S$ is defined as:

1. *(discrete transition)* $(q, v) \xrightarrow{c} (q', v')$ if there exists $(q, g, R, q') \in E$ s.t. $v \models g$, $v' = [R \leftarrow 0]v$, $v' \models \eta(q')$, and $c_i = P_i(q, g, R, q')$ for every $1 \leq i \leq p$;

[1] v_0 assigns zero to each clock.

2. *(delay transition)* $(q,v) \xrightarrow{c} (q, v+t)$ if $\forall 0 \leq t' \leq t$, $v + t' \models \eta(q)$, and $c_i = t \cdot P_i(q)$ for every $1 \leq i \leq p$.

A *run* of a PTA is a path in the underlying transition system. Given a run $\varrho = s_0 \xrightarrow{c^0} s_1 \xrightarrow{c^1} \cdots \xrightarrow{c^{n-1}} s_n$, its ith-cost is $P_i(\varrho) = \sum_{j=0}^{n-1} c_i^j$. A *position* along a run ϱ is an occurrence of a state (q,v) along ϱ. Let π be such a position, then $\varrho[\pi]$ denotes the corresponding state, whereas $\varrho_{\leq\pi}$ denotes the finite prefix of ϱ ending at position π.

Remark 1. In the model of priced timed automata, the cost variables are *observers*: the values of these variables don't constrain the behaviour of the system (the behaviours of a priced timed automaton are those of the underlying timed automaton), but can be used as evaluation functions. For instance, problems such as "optimal reachability" [4,3], "optimal infinite schedules" [6] or "optimal reachability timed games" [2,8,11,5] have recently been investigated. The problem we consider in this paper is closely related to these kinds of problems: we will use WCTL as a language for evaluating the performances of a system.

2.2 The Logic WCTL

Let AP be a set of atomic propositions. The logic WCTL[2] [10] extends CTL with cost constraints. Its syntax is given by the following grammar:

$$\text{WCTL} \ni \phi \ ::= \ \texttt{true} \ \mid \ a \ \mid \ \neg\phi \ \mid \ \phi \vee \phi \ \mid \ \mathsf{E}\,\phi \mathsf{U}_{P\sim c}\phi \ \mid \ \mathsf{A}\,\phi \mathsf{U}_{P\sim c}\phi$$

where $a \in \mathsf{AP}$, P is a cost function, c ranges over $\mathbb{N}$, and $\sim \in \{<, \leq, =, \geq, >\}$.

We interpret formulas of WCTL over labeled PTA, *i.e.* PTA having a labeling function ℓ which associates with every location q a subset of AP.

Definition 2. *Let $\mathcal{A}$ be a labeled PTA. The satisfaction relation of WCTL is defined over configurations (q,v) of $\mathcal{A}$ as follows:*

$$\begin{aligned}
(q,v) &\models \texttt{true} && \\
(q,v) &\models p && \Leftrightarrow \quad a \in \ell(q) \\
(q,v) &\models \neg\phi && \Leftrightarrow \quad (q,v) \not\models \phi \\
(q,v) &\models \phi_1 \vee \phi_2 && \Leftrightarrow \quad (q,v) \models \phi_1 \textit{ or } (q,v) \models \phi_2 \\
(q,v) &\models \mathsf{E}\,\phi_1 \mathsf{U}_{P\sim c}\phi_2 && \Leftrightarrow \quad \textit{there is an infinite run } \varrho \textit{ in } \mathcal{A} \\
& && \qquad \textit{from } (q,v) \textit{ s.t. } \varrho \models \phi_1 \mathsf{U}_{P\sim c}\phi_2 \\
(q,v) &\models \mathsf{A}\,\phi_1 \mathsf{U}_{P\sim c}\phi_2 && \Leftrightarrow \quad \textit{any infinite run } \varrho \textit{ in } \mathcal{A} \textit{ from } (q,v) \\
& && \qquad \textit{satisfies } \varrho \models \phi_1 \mathsf{U}_{P\sim c}\phi_2 \\
\varrho &\models \phi_1 \mathsf{U}_{P\sim c}\phi_2 && \Leftrightarrow \quad \textit{there exists } \pi > 0 \textit{ position along } \varrho \textit{ s.t.} \\
& && \qquad \varrho[\pi] \models \phi_2, \textit{ for all position } \pi' > 0 \\
& && \qquad \textit{before } \pi \textit{ on } \varrho,\ \varrho[\pi'] \models \phi_1, \\
& && \qquad \textit{and } P(\varrho_{\leq\pi}) \sim c
\end{aligned}$$

[2] WCTL stands for "Weighted CTL", following [10] terminology. It would have been more natural to call it "Priced CTL" (PCTL) in our setting, but this would have been confusing with "Probabilistic CTL" [13].

If $\mathcal{A}$ is not clear from the context, we may write $(q, v), \mathcal{A} \models \phi$ instead of simply $(q, v) \models \phi$.

As usual, we will use shortcuts as $\mathsf{E}\,\mathsf{F}_{P\sim c}\phi \equiv \mathsf{E}\,\mathtt{true}\,\mathsf{U}_{P\sim c}\phi$, or $\mathsf{A}\,\mathsf{G}_{P\sim c}\phi \equiv \neg\mathsf{E}\,\mathsf{F}_{P\sim c}\neg\phi$. Moreover, if the cost function P is unique or clear from the context, we may write $\phi\mathsf{U}_{\sim c}\psi$ instead of $\phi\mathsf{U}_{P\sim c}\psi$.

We write WCTL* for the extension of WCTL similar to the extension CTL* of CTL [12]: temporal modality $\mathsf{U}_{\sim c}$ can then be nested independently of path quantifiers.

2.3 Example

The 1PTA of Fig. 1 models a never-ending process of repairing problems, which are bound to occur repeatedly with a certain frequency. The repair of a problem has a certain cost, captured in the model by the cost variable c. As soon as a problem occurs (modeled by the Problem location) the value of c grows with rate 3, until actual repair is taking place in one of the locations Cheap (rate 2) or Expensive (rate 4). At most 20 time units after the occurrence of a problem it will have been repaired one way or another. In this setting we are interested in properties concerning the cost of repairs as stated by the following WCTL formulas (all satisfied by the model):

$$\begin{array}{l}\mathsf{A}\,\mathsf{G}\big(\mathsf{Problem} \implies \mathsf{E}\,\mathsf{F}_{c\leq 47}\mathsf{OK}\big)\\ \mathsf{A}\,\mathsf{G}\big(\mathsf{Problem} \implies \mathsf{A}\,\mathsf{F}_{c\leq 56}\mathsf{OK}\big)\\ \mathsf{A}\,\mathsf{G}\big(\neg\mathsf{E}\,(\mathsf{OK}\;\mathsf{U}_{t\geq 8}(\mathsf{Problem} \wedge \neg\mathsf{E}\,\mathsf{F}_{c<30}\mathsf{OK}))\big)\end{array}$$

where t holds for the time elapsed (special cost variable with rate 1).

Here the first property claims that whenever a problem occurs it may be repaired (*i.e.* reach the location OK) within a total cost of 47. In fact Fig. 2 gives the minimum cost of repair —as well as an optimal strategy— for any state of the form $(\mathsf{Problem}, x)$ with $x \in [0, 10]$. Correspondingly, the minimum

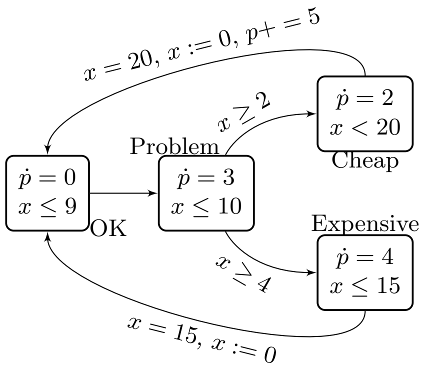


Fig. 1. Repair problem as a PTA

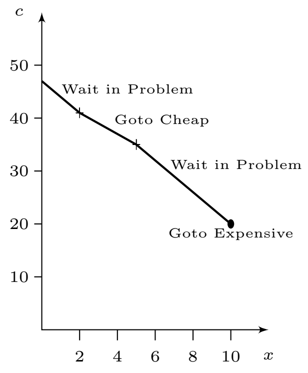


Fig. 2. Minimum cost of repair and associated strategy in location Problem

cost of reaching OK from states of the form (Cheap, x) (resp. $(\mathsf{Expensive}, x)$) is given by the expression $45 - 2x$ (resp. $60 - 4x$). The second property states that no matter which method is used for the repair, it will cost no more than 56. Finally, the third property claims that whenever the system has been OK for at least 8 time units before a problem occurs, then there must be a way of solving the problem with a total cost less than 30. In fact, as indicated in Fig. 2, any state $(\mathsf{Problem}, x)$ with $x \geq \frac{20}{3}$ satisfies the WCTL property $\mathsf{E}\,\mathsf{F}_{c \leq 30}\mathsf{OK}$.

3 Main Result

We focus on one-clock priced timed automata (1PTA for short), *i.e.* priced timed automata where $|\mathcal{X}| = 1$. The main result of this paper is the following theorem:

Theorem 3. *Model-checking WCTL on 1PTA is* PSPACE*-complete.*

The PSPACE lower bound is a consequence of the PSPACE-hardness of the model-checking of TCTL, the restriction of WCTL to time constraints, over 1PTA [15].

The PSPACE upper bound is rather involved, and will be done in two steps: *i*) first we will exhibit a set of regions which will be correct for model-checking WCTL formulas, see Section 4; *ii*) then we will use this result to propose a PSPACE algorithm for model-checking WCTL, see Section 5.

Finally, it is worth reminding here that the model-checking of WCTL over priced timed automata with three clocks is undecidable [5].

4 Sufficient Granularity for Model-Checking WCTL

The proof of Theorem 3 is rather involved and partly relies on the following proposition, which exhibits a set of *regions* on which truth of WCTL formulas is uniform.

Proposition 4. *Let Φ be a WCTL formula and let $\mathcal{A}$ be a 1PTA. Then there exist finitely many constants $0 = a_0 < a_1 < \ldots < a_n < a_{n+1} = +\infty$ s.t. for every location q of $\mathcal{A}$, for every $0 \leq i \leq n$, the truth of Φ is uniform over $\{(q, x) \mid a_i < x < a_{i+1}\}$. Moreover,*

- *$\{a_0, ..., a_n\}$ contains all the constants appearing in clock constraints of $\mathcal{A}$;*
- *the constants are integral multiples of $1/C^{\hbar(\Phi)}$ where $\hbar(\Phi)$ is the* constrained temporal height *of Φ, i.e. the maximal number of nested constrained modalities in Φ, and C is the lcm of all positive costs labeling a location of $\mathcal{A}$;*
- *a_n equals the largest constant M appearing in the guards of $\mathcal{A}$;*
- *$n \leq M \cdot C^{\hbar(\Phi)} + 1$.*

As a corollary, we recover the partial decidability result of [10], stating that the model-checking of 1PTA with a *stopwatch cost*[3] against WCTL formulas is decidable using classical one-dimensional regions of timed automata (*i.e.* with granularity 1).

[3] *I.e.* cost with rates in $\{0, 1\}$.

Proof. The proof of this proposition is by structural induction on Φ. We focus on the case when $\Phi = \mathsf{E}\,\phi\mathsf{U}_{P\sim c}\psi$ (we will simply write $\Phi = \mathsf{E}\,\phi\mathsf{U}_{\sim c}\psi$): the cases of atomic propositions, boolean combinations are straightforward, unconstrained modalities require no refinement of the granularity (a basic CTL algorithm handles this case), and the other modalities will be reduced to this main case.

Assume that the result has been proved for WCTL subformulas ϕ and ψ, and that we have merged all constants for ϕ and ψ: we thus have constants $0 = a_0 < a_1 < \ldots < a_n < a_{n+1} = +\infty$ such that for every location q of $\mathcal{A}$, for every $0 \le i \le n$, the truth of ϕ and that of ψ are both uniform over $\{(q,x) \mid a_i < x < a_{i+1}\}$. The granularity of these constants is $1/C^{\max(\hbar(\phi),\hbar(\psi))} = 1/C^{\hbar(\Phi)-1}$. We will exhibit extra constants such that the above proposition then also holds for formula $\Phi = \mathsf{E}\,\phi\mathsf{U}_{\sim c}\psi$. For the sake of simplicity, we will call *regions* all elementary intervals (a_i, a_{i+1}) and singletons $\{a_i\}$. We also assume that $\mathcal{A}$ has no discrete costs (*i.e.* $P(T) = \{0\}$). The general case would be handled in a similar way, and will be developed in the long version of this paper.

In order to compute the set of states satisfying $\mathsf{E}\,\phi\mathsf{U}_{\sim c}\psi$, we compute for every state (q,x) all costs of paths from (q,x) to some region (q',r), along which ϕ continuously holds, and such that a ψ-state can be reached immediately from (q',r). We then check whether we can achieve a cost satisfying "$\sim c$". We thus explain how we compute the set of possible costs between a state (q,x) and a region (q',r) in $\mathcal{A}$.

For each index i, we restrict the automaton $\mathcal{A}$ to transitions whose guards contain the interval (a_i, a_{i+1}), and that do not reset the clock. We denote by $\mathcal{A}_i$ this restricted automaton. Let q and q' be two locations of $\mathcal{A}_i$. As stated by the following lemma, the set of costs of paths between (q, a_i) and (q', a_{i+1}) is an interval that can be easily computed:

Lemma 5. *Let $S_i(q,q')$ be the set of locations that are reachable from (q,a_i) and co-reachable from (q',a_{i+1}) in $\mathcal{A}_i$ (assuming $a_{i+1} \neq +\infty$), and assume it is non-empty. Let $c^{i,q,q'}_{\min}$ and $c^{i,q,q'}_{\max}$ be the minimum and maximum costs among the costs of locations in $S_i(q,q')$. Then the set of all possible costs of paths going from (q,a_i) to (q',a_{i+1}) in $\mathcal{A}_i$ is an interval $\langle (a_{i+1}-a_i)\cdot c^{i,q,q'}_{\min} ; (a_{i+1}-a_i)\cdot c^{i,q,q'}_{\max} \rangle$. The interval is left-closed iff there exist two locations r and s (with possibly $r = s$) in $S_i(q,q')$ with cost $c^{i,q,q'}_{\min}$ such that*[4] *$(q,a_i) \rightsquigarrow^*_{\mathcal{A}_i} (r,a_i)$, $(r,a_i) \rightsquigarrow^*_{\mathcal{A}_i} (s,a_{i+1})$, and $(s,a_{i+1}) \rightsquigarrow^*_{\mathcal{A}_i} (q',a_{i+1})$. The interval is right-closed iff there exists two locations r and s in $S_i(q,q')$ with cost $c^{i,q,q'}_{\max}$ such that $(q,a_i) \rightsquigarrow^*_{\mathcal{A}_i} (r,a_i)$, $(r,a_i) \rightsquigarrow^*_{\mathcal{A}_i} (s,a_{i+1})$, and $(s,a_{i+1}) \rightsquigarrow^*_{\mathcal{A}_i} (q',a_{i+1})$.*

The conditions on left/right-closures characterize the fact that it is possible to instantaneously reach/leave a location with minimal/maximal cost, or if a small positive delay has to be waited (due to a strict guard).

Proof. Obviously the costs of all paths in $\mathcal{A}_i$ belong to the interval $(a_{i+1} - a_i) \cdot [c^{i,q,q'}_{\min}, c^{i,q,q'}_{\max}]$. We will now prove that the set of costs is an interval containing $(a_{i+1} - a_i) \cdot (c^{i,q,q'}_{\min} ; c^{i,q,q'}_{\max})$.

[4] The notation $\alpha \rightsquigarrow^*_{\mathcal{A}_i} \alpha'$ means that there is a path in $\mathcal{A}_i$ from α to α'.

Let $\tau_{\min}$ (resp. $\tau_{\max}$) be a sequence of transitions in $\mathcal{A}_i$ leading from (q, a_i) to (q', a_{i+1}) and going through a location with minimal (resp. maximal) cost. Easily enough, the possible costs of the paths following $\tau_{\min}$ (resp. $\tau_{\max}$) form an interval whose left (resp. right) bound is $c_{\min}^{i,q,q'} \cdot (a_{i+1} - a_i)$ (resp. $c_{\max}^{i,q,q'} \cdot (a_{i+1} - a_i)$).

Now, if c and c' are the respective costs of q and q', then $\frac{1}{2} \cdot (c + c') \cdot (a_{i+1} - a_i)$ is in both intervals. Indeed, the path following $\tau_{\min}$ (resp. $\tau_{\max}$) which delays $\frac{1}{2} \cdot (a_{i+1} - a_i)$ time units in q, then directly goes to q' and waits there for the remaining $\frac{1}{2} \cdot (a_{i+1} - a_i)$ time units achieves the above-mentioned cost. This implies that the set of all possible costs is an interval.

The bound $c_{\min}^{i,q,q'} \cdot (a_{i+1} - a_i)$ is reached iff there is a path from (q, a_i) to (q', a_{i+1}) which delays only in locations with cost $c_{\min}^{i,q,q'}$. This is precisely the condition expressed in the lemma. The same holds for the upper bound $c_{\max}^{i,q,q'} \cdot (a_{i+1} - a_i)$. □

Similar results clearly hold for other kinds of regions:

– between a state (q, a_i) and a region $(q', (a_i, a_{i+1}))$ with $a_{i+1} \neq +\infty$, the set of possible costs is an interval $\langle 0; c_{\max}^{i,q,q'} \cdot (a_{i+1} - a_i)\rangle$, where 0 can be reached iff it is possible to go from (q, a_i) to some state (q'', a_i) with $P(q'') = 0$.
– between a state (q, x), with $x \in (a_1, a_{i+1})$, and (q', a_{i+1}), the set of costs is $(a_{i+1} - x) \cdot \langle c_{\min}^{i,q,q'}; c_{\max}^{i,q,q'}\rangle$, with similar conditions as above for the bounds of the interval.
– between a state (q, x), with $x \in (a_1, a_{i+1})$, and region $(q', (a_i, a_{i+1}))$ (assuming $a_{i+1} \neq +\infty$), the set of possible costs is $[0, c_{\max}^{i,q,q'} \cdot (a_{i+1} - x))$;
– between a state (q, a_n) and a region $(q', (a_n, a_{n+1}))$ (with $a_{n+1} = +\infty$), the set of possible costs is either $[0, 0]$, if no positive cost rate is reachable and co-reachable, or $\langle 0, +\infty)$ otherwise. If the latter case, 0 can be achieved iff it is possible to reach a state (q'', a_n) with $P(q'') = 0$;
– between a state (q, x), with $x \in (a_n, a_{n+1})$ and $a_{n+1} = +\infty$, and a region $(q', (a_n, a_{n+1}))$, the set of costs is either $[0, 0]$ or $[0, +\infty)$, with the same conditions as previously.

We use these computations and build a graph G labeled by intervals which will store all possible costs between symbolic states (*i.e.* pairs (q, r), where q is a location and r a region) in $\mathcal{A}$. Vertices of G are pairs $(q, \{a_i\})$ and $(q, (a_i, a_{i+1}))$, and tuples $(q, x, \{a_i\})$ and $(q, x, (a_i, a_{i+1}))$, where q is a location of $\mathcal{A}$. Their roles are as follows: vertices of the form (q, x, r) are used to initiate a computation, they represent a state (q, x) with $x \in r$. States $(q, \{a_i\})$ are "regular" steps in the computation, while states $(q, (a_i, a_{i+1}))$ are used either for finishing a computation, or just before resetting the clock (there will be no edge from $(q, (a_i, a_{i+1}))$ to any $(q', \{a_{i+1}\})$).

Edges of G are defined as follows:

– $(q, \{a_i\}) \rightarrow (q', \{a_{i+1}\})$ if there is a path from (q, a_i) to (q', a_{i+1}). This edge is then labeled with an interval $\langle (a_{i+1} - a_i) \cdot c_{\min}^{i,q,q'}; (a_{i+1} - a_i) \cdot c_{\max}^{i,q,q'}\rangle$, the nature of the interval (left-closed and/or right-closed) depending on the criteria exposed in Lemma 5.

- $(q, \{a_i\}) \to (q', \{a_i\})$ if there is an instantaneous path from (q, a_i) to (q', a_i) in $\mathcal{A}$, the edge is then labeled with the interval $[0, 0]$ (remember that we assumed there are no discrete costs on transitions of $\mathcal{A}$).
- $(q, \{a_i\}) \to (q', \{a_0\})$ if there is a transition in $\mathcal{A}$ enabled when the value of the clock is a_i and resetting the clock. It is labeled with $[0, 0]$.
- $(q, (a_i, a_{i+1})) \to (q', \{a_0\})$ if there is a transition in $\mathcal{A}$ enabled when the value of the clock is in (a_i, a_{i+1}) and resetting the clock. It is labeled with $[0, 0]$.
- $(q, \{a_i\}) \to (q', (a_i, a_{i+1}))$ if there is a path from (q, a_i) to some (q', α) with $a_i < \alpha < a_{i+1}$. This edge is labeled with the interval $\langle 0; (a_{i+1} - a_i) \cdot c_{\max}^{i,q,q'})$.
- $(q, x, \{a_i\}) \to (q, \{a_i\})$ labeled with $[0, 0]$.
- $(q, x, (a_i, a_{i+1})) \to (q', \{a_{i+1}\})$ if there is a path from some (q, α) with $a_i < \alpha < a_{i+1}$ to (q', a_{i+1}). This edge is labeled with $(a_{i+1} - x) \cdot \langle c_{\min}^{i,q,q'}; c_{\max}^{i,q,q'}\rangle$.
- $(q, x, (a_i, a_{i+1})) \to (q', (a_i, a_{i+1}))$ labeled with $[0, (a_{i+1} - x) \cdot c_{\max}^{i,q,q'})$.

Figure 3 represents one part of this graph. Note that each path π of this graph is naturally associated with an interval $\iota(\pi)$ (possibly depending on variable x if we start from a node $(q, x, (a_i, a_{i+1})))$ by summing up all intervals labeling transitions of π.

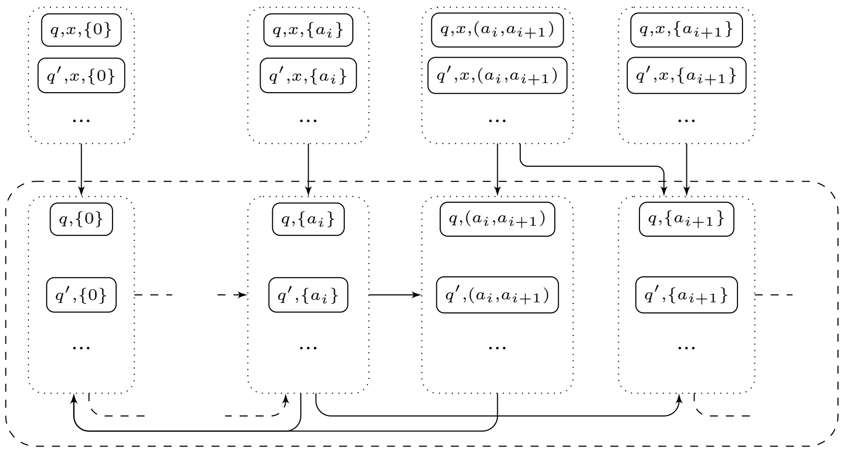


Fig. 3. (Schematic) representation of the graph G (intervals omitted)

The correctness of graph G w.r.t. costs is stated by the following lemma, which is a direct consequence of the previous investigations.

Lemma 6. *Let q and q' be two locations of $\mathcal{A}$. Let r and r' be two regions, and let $\alpha \in r$. Let $d \in \mathbb{R}^+$. There exists a path π in G from a state (q, x, r) to (q', r') with cost $d \in \iota(\pi)(\alpha)$ if, and only if, there is a path in $\mathcal{A}$ with total cost d, and going from (q, α) to some (q', β) with $\beta \in r'$.*

Corollary 7. *Fix two regions r and r'. Then the set of possible costs of paths in G from (q, x, r) to (q', r') is of the form*

$$\bigcup_{m\in\mathbb{N}} \langle \alpha_m - \beta_m \cdot x; \alpha'_m - \beta'_m \cdot x\rangle$$

(possibly with β_m and/or $\beta'_m = 0$, and/or $\alpha'_m = +\infty$). Moreover,

- *all constants α_m and α'_m are either integral multiples of $1/C^{\max(\hbar(\phi),\hbar(\psi))}$ or $+\infty$, and constants β_m and β'_m are either costs of the automaton or 0;*
- *if $r = (a_n, +\infty)$, then $\beta_m = \beta'_m = 0$ for all m.*

Proof (Sketch). The set of possible costs can be computed by guessing the Parikh image of a possible path. Then the set of possible costs along that path has the form given in the statement. And as the set of possible Parikh images is countable, we obtain the (possibly infinite) union of intervals of the corollary. □

Lemma 8. *For every location q, the set of clock values x such that (q, x) satisfies $\mathsf{E}\,\phi \mathsf{U}_{\sim c}\psi$ is a finite union of intervals. Moreover,*

- *the bounds of those intervals are integral multiples of $1/C^{\hbar(\Phi)}$;*
- *the largest finite bound of those intervals is at most the maximal constant appearing in the guards of the automaton.*

Proof (Sketch). It is possible to prove that the (possibly infinite) union of intervals of the previous corollary can be reduced, for checking formula $\mathsf{E}\,\phi \mathsf{U}_{\sim c}\psi$, to a finite union of such intervals.

Then, new constants α we need to consider for checking $\mathsf{E}\,\phi \mathsf{U}_{\sim c}\psi$ are such that $\alpha_m - \beta_m \cdot \alpha = c$, *i.e.* $\alpha = (\alpha_m - c)/\beta_m$. Thus α is an integral multiple of $1/C^{\hbar(\Phi)}$. □

This concludes the induction step for formula $\mathsf{E}\,\phi \mathsf{U}_{\sim c}\psi$ when the automaton has no discrete cost. Extending this result to other modalities and to automata with discrete cost is a rather technical matter that gives no new insights on the model-checking problem; we thus postpone the proofs of these two extensions to the full version of this paper. ■

Remark 2. The exponential number of constants a_i's is unavoidable in general. Indeed, consider the 1PTA $\mathcal{A}$ displayed on Fig. 4. Using a WCTL formula, we will require that the cost is exactly 4 between a and b. That way, if clock x equals $x_0.x_1x_2x_3\ldots x_n\ldots$ (this is the binary representation of a real in the interval $(0, 2)$) when leaving a, then it will be equal to $x_1.x_2x_3\ldots x_n\ldots$ in b. We consider the WCTL formula $\phi(X) = \mathsf{E}\left((a \vee b)\mathsf{U}_{=0}(\neg a \wedge \mathsf{E}\,(\neg b \mathsf{U}_{=4}(b \wedge X)))\right)$, where X is a formula we will specify. Then formula $\phi(\mathsf{E}\,\mathsf{F}_{=0}c)$ states that we can go from a to b with cost 4, and that $x = 0$ when arriving in b (since we can fire the transition leading to c). From the remark above, this can only be true if $x = 0$ or $x = 1$ in a. Now, consider formula $\phi(\mathsf{E}\,\mathsf{F}_{=0}c \vee \phi(\mathsf{E}\,\mathsf{F}_{=0}c))$. If it holds in state a, then state c can be reached after exactly one or two rounds in the automaton, *i.e.*, if the value of x is in $\{0, 1/2, 1, 3/2\}$. Clearly enough, nesting ϕ n times characterizes values of the clocks of the form $p/2^{n-1}$ where p is an integer strictly less than 2^n.

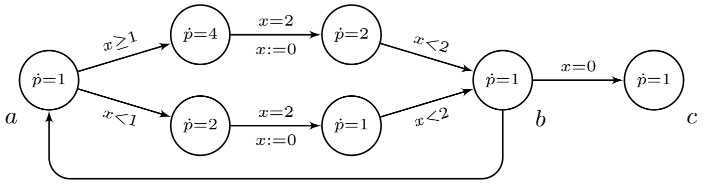


Fig. 4. The 1PTA $\mathcal{A}$

5 Algorithms and Complexity

In this section, we provide two algorithms for model-checking WCTL on 1PTA. The first algorithm runs in EXPTIME, whereas the second one runs in PSPACE, thus matching the PSPACE lower bound. However, it is easier to first explain the first algorithm, and then reuse part of it in the second algorithm. Finally, we will pursue the example of Subsection 2.3 for illustrating our PSPACE algorithm.

5.1 An EXPTIME Algorithm

The correctness of the algorithm we propose for model-checking 1PTA against WCTL properties relies on the properties we have proved in the previous section: if $\mathcal{A}$ is an automaton with maximal constant M, writing C for the l.c.m. of all costs labeling a location, and if Φ is a WCTL formula of size n, then the satisfaction of Φ is uniform on the regions $(m/C^n; (m{+}1)/C^n)$ with $m < M{\cdot}C^n$, and also on $(M; +\infty)$. The idea is thus to test the satisfaction of Φ for each state of the form $(q, k/2C^n)$ for $0 \leq k \leq (M \cdot 2C^n) + 1$ (*i.e.* at the bounds and in the middle of each region).

To check the truth of $\Phi = \mathsf{E}\,\phi \mathsf{U}_{P\sim c}\psi$ in state (q, x) with $x = k/2C^n$, we will use the graph G that we have defined in Section 4. From the state (q, x, r) of G, where r is the region containing $k/2C^n$, we check if $\mathsf{E}\,\phi\mathsf{U}_{\sim c}\psi$ (say) holds by non-deterministically discovering a witness. This requires the following lemma:

Lemma 9. *Let s be the smallest positive cost in $\mathcal{A}$, and C be the lcm of all positive costs of $\mathcal{A}$. Let q be a location of $\mathcal{A}$, and $x \in \mathbb{R}^+$. Let $\Phi = \mathsf{E}\,\phi\mathsf{U}_{\sim c}\psi$ be a WCTL formula of size n. Then $(q, x) \models \Phi$ iff there exists a trajectory in $\mathcal{A}$, from (q, x) and satisfying $\phi\mathsf{U}_{\sim c}\psi$, and whose projection in G visits at most $N = \lfloor c \cdot C^n/s\rfloor + 2$ times each state of G.*

Proof (Sketch). Let τ be a trajectory in $\mathcal{A}$, starting from (q, x) and satisfying $\phi\mathsf{U}_{\sim c}\psi$. To that trajectory corresponds a trajectory ρ in G, starting in (q, x, r). Consider a cycle in that trajectory ρ: either it has a global cost interval $[0, 0]$, in which case it can be removed and still yields a witnessing trajectory; or it has a global cost interval of the form $\langle a, b\rangle$ with $b > 0$. In that case, letting s be the smallest positive cost of the automaton, we know that $b \geq s/C^n$. Now, if some state of G is visited (strictly) more than $N = \lfloor c \cdot C^n/s\rfloor + 2$ times along ρ, we build a trajectory ρ' from ρ by removing extraneous cycles, in such a way that each state of G is visited at most N times along ρ (and that ρ starts and ends in the same states). Since we assumed that ρ does not contain cycles with cost interval $[0;0]$, we know that the upper bound of the accumulated cost along ρ' is above c. Also, the lower bound of the accumulated costs along ρ' is less than that of ρ. Since ρ "contains" a trajectory witnessing $\phi \mathsf{U}_{\sim c}\psi$, the cost interval of ρ contains a value satisfying $\sim c$, thus so does the cost interval of ρ'. In other words, ρ' still contains a trajectory witnessing $\phi \mathsf{U}_{\sim c}\psi$. $\square$

We now describe our algorithm: assuming we have computed, for each state q of $\mathcal{A}$, the intervals of values of x where ϕ (resp. ψ) holds, we non-deterministically guess the successive states of a trajectory in G. At each step, we also have to guess the intermediary states that are visited (between $(q, \{a_i\})$ and $(q', \{a_{i+1}\})$), and check that they satisfy ϕ when x is in (a_i, a_{i+1}). This verification can be achieved in PSPACE. Moreover, at each step of this algorithm for checking that $(q, x) \models \mathsf{E}\, \phi \mathsf{U}_{\sim c}\psi$, we only need to store a polynomial amount of information: the current position in G, the number of steps so far, and the interval of costs accumulated so far. At each point, the algorithm may non-deterministically decide to go to a ψ-state, and will check that the cost constraint is satisfied. In that case, it returns `yes`. Otherwise, when the number of steps reaches $|G| \cdot (\lfloor c \cdot C^n/s \rfloor + 2)$ (which is exponential), the procedure stops and returns `no`.

Thus, our procedure for checking that $(q, x) \models \mathsf{E}\, \phi \mathsf{U}_{\sim c}\psi$ is in PSPACE. Still, since we store all the intervals for each location of the automaton and each subformula, the whole algorithm requires an exponential amount of space, but it runs in exponential time.

The other existential modalities are handled by reducing to the case of $\mathsf{E}\, \mathsf{U}_{\sim c}$, as explained in Section 4. We assume that no universal modality appears in the formula by replacing them with negated existential ones.

5.2 A PSPACE Algorithm

The PSPACE algorithm will reuse some parts of the previous algorithm, but it will improve on space performance by storing only the minimal information required, preferring to spend time on reconstructing model-checking information rather than to spend space on storing it. Our method is thus similar in spirit to the space-efficient, on-the-fly algorithm for TCTL presented in [14].

We will then need, while guessing a witness for $\mathsf{E}\, \phi \mathsf{U}_{P \sim c}\psi$, to check that all intermediary states satisfy formula ϕ. As ϕ might be itself a WCTL formula with several nested modalities, we will fork a new computation of our algorithm on formula ϕ from each intermediary state. The maximal number of threads running simultaneaously is at most the depth of the parsing tree of formula Φ. When a thread is preempted we only need to store a polynomial amount of information in order to be able to resume it. Indeed, it is sufficient to store for each preempted thread a triple (α, K, I) where α is a node a graph G, K is the value of a counter bounded by $|G| \cdot (\lfloor c \cdot C^n/s \rfloor + 2)$ counting the number of steps of the path we are guessing (we know that a witness can be bounded by this constant), and I is an interval corresponding to the accumulated cost along the path being guessed.

The algorithm thus runs as follows: we start by labeling the root of the tree by $\alpha = (q, x, r)$, $K = 0$ and $I = [0; 0]$. Then we guess a path in G starting from (q, x, r), and when a new state (q', r') is added, we increment the value of K, update the value of the interval, as described in the previous section. Then, either we choose to verify that the state satisfies ϕ, or the constraint $P \sim c$ can be satisfied by the new interval and we verify in addition that the new state satisfies ψ. Moreover, we need to prove that all intermediary states (see the EXPTIME algorithm) also satisfy ϕ (it is of course sufficient to check intermediary with clock values of the form $h/2C^n$). All these verifications of ϕ or ψ are done by starting a new thread in the computation, and a new guess of path can start for a subformula of the original one... when all these computations are finished, we can continue guessing the original path for formula Φ, and so on.

The number of nested guesses can be bounded by the depth of the parsing tree of Φ, because when a new thread starts, it starts from a node which is a child of the previous node. Thus, the memory which is needed in this algorithm is the parsing tree of formula Φ with each node labeled by a tuple which can be stored in polynomial space, which leads to a globally PSPACE algorithm.

Example 1. We illustrate our PSPACE algorithm on our initial example, with formula $\Phi = \neg \mathsf{E}\,(\mathsf{OK}\ \mathsf{U}_{t\leq 8}(\mathsf{Problem} \wedge \neg \mathsf{E}\,\mathsf{F}_{c<30}\mathsf{OK}))$. We write $g = 1/C^2$ for the resulting granularity as defined in Prop. 4, and consider a starting state, e.g. $(\mathsf{OK}, x = mg)$.

Fig. 5 show three steps of our algorithm. The first step represents the first iteration, where subformula OK is satisfied at the beginning of the trajectory. At step 2, the execution goes to $(\mathsf{OK}, x + g)$: we check that the left-hand-side formula still holds in $(\mathsf{OK}, x + g)$ (as depicted), but also in intermediary states. The third figure corresponds to k steps later, when the algorithm decides to go to the right-hand-part of $\mathsf{E}\,\mathsf{U}_{t\leq 8}$. In that case, of course, it is checked that $kg \leq 8$, and then goes on verifying the second until subformula.

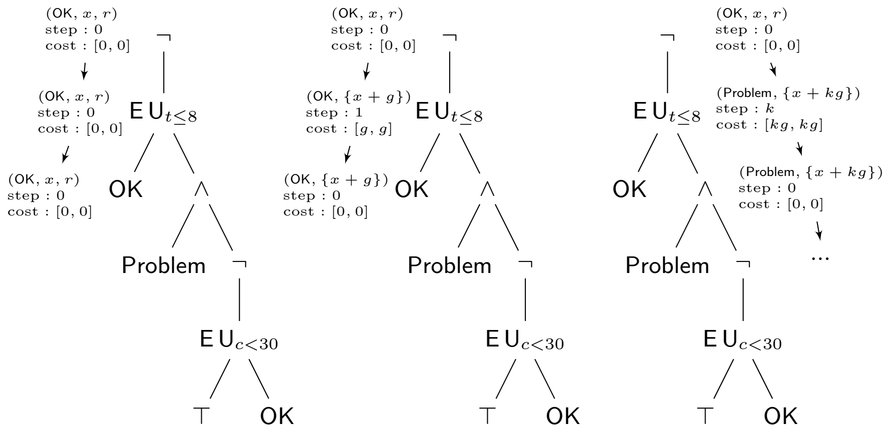


Fig. 5. Execution of our PSPACE algorithm on the initial example

6 Undecidability of WCTL* Model-Checking

The logic WCTL* is an extension of WCTL that allows nesting of modalities without existential or universal quantifications. We prove that it is undecidable on 1PTAs. To our knowledge, the complexity of TCTL* model-checking has not been studied on one-clock timed automata. However, it is in EXPSPACE on durational Kripke structures, a discrete-time extension of Kripke structures [16].

Theorem 10. *Model-checking WCTL* over 1PTA is undecidable.*

Proof (Sketch). We encode the halting problem for a two-counter machine $\mathcal{M}$ as a model-checking problem for WCTL* over 1PTA. The counters c_1 and c_2 are encoded by clock x being equal to $1/(2^{c_1} \cdot 3^{c_2})$.

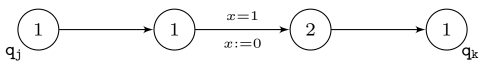


Fig. 6. Incrementing a counter

We first explain how we encode an instruction incrementing counter c_1, say "`q_j: c_1:=c_1+1; goto q_k`". Such an instruction is encoded by the automaton displayed on Fig. 6 (where costs are written in locations). We will require that the price between the date at which we enter (or equivalently exit) $\mathsf{q_j}$ and the date at which we enter $\mathsf{q_k}$ is exactly 1. This is enforced by checking the following path formula (with nested until modalities) when entering q_j:

$$\varphi_{\text{incr1}} = \mathsf{q_j} \mathsf{U}_{=0} (\neg \mathsf{q_j} \wedge (\neg \mathsf{q_k} \mathsf{U}_{=1} \mathsf{q_k}))$$

This ensures that clock x has been divided by 2, *i.e.*, that counter c_1 has been incremented. Decrementation can be handled in a similar way by setting the cost of the second (resp. third) location to 2 (resp. 1) and enforcing global cost along that module to be 2. Those operations easily adapt to counter c_2.

Testing if counter c_1 equals 0 reduces to checking that the value of clock x is of the form $1/3^{c_2}$, thus to multiplying clock x by 3 until it possibly equals 1. Consider the following instruction: "`q_k: if (c_1==0) goto q_l`". We encode this instruction with the automaton of Fig. 7.

Multiplying clock x by 3 is achieved by one pass through the loop with cost exactly 3. Consider the following formula:

$$\varphi_{\text{mult}} = \mathsf{E} \Big(m \Rightarrow \big(m \mathsf{U}_{=0} z \vee m \mathsf{U}_{=0} (\neg m \wedge \neg m \mathsf{U}_{=3} m) \big) \Big) \mathsf{U} z$$

It precisely expresses that it is possible to reach z after a finite number of passes through the loop, each pass having total cost 3. This holds iff the original value

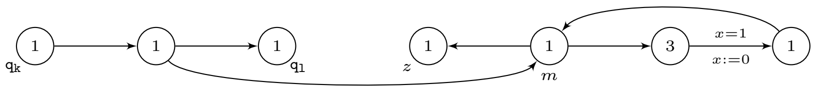


Fig. 7. Testing a counter to 0

of clock x when entering the module was of the form $1/3^i$, *i.e.*, iff counter c_1 was equal to 0. Now, from $\mathsf{q_k}$, we simply have to ensure the following property:

$$\varphi_{\text{test1}} = \mathsf{q_k}\mathsf{U}_{=0}\Big(\neg\mathsf{q_k} \wedge \mathsf{E}\left(\neg m\mathsf{U}_{=0}(m \wedge \varphi_{\text{mult}})\right) \wedge \left(\neg\mathsf{q_1}\mathsf{U}_{=0}\mathsf{q_1}\right)\Big)$$

Now, the global reduction consists in building a larger automaton, with one state $\mathsf{q_j}$ per instruction of the two-counter machine, and the intermediary states required by the above modules. The following formula expresses that the halting state can be reached after a finite number of executions of the instructions:

$$\mathsf{E}\Big(\bigwedge_j(\mathsf{q_j} \rightarrow \varphi_{\text{type}(\mathsf{q_j})})\Big)\mathsf{U}\mathsf{q}_{\text{Halt}}$$

where type($\mathsf{q_j}$) is the type of instruction $\mathsf{q_j}$ (*i.e.*, "incr1" if $\mathsf{q_j}$ is an incrementation of counter c_1, "test1" is it is a test of counter c_1, and so on). State $\mathsf{q_0}$ satisfies this property iff there exists a computation of the two-counter machine that ends up in state q_{Halt}. □

7 Conclusion

In this paper we have proved that the model-checking of one-clock priced timed automata against WCTL properties is PSPACE-complete. This is rather surprising as model-checking TCTL over one-clock timed automata has the same complexity, though it allows much less features. For proving this result, we have exhibited a sufficient granularity such that truth of formulas over regions defined with this granularity is uniform. Based on this result, we developed a space-efficient algorithm which computes satisfaction of subformulas on-the-fly. This result has to be contrasted with the undecidability result of [5] which establishes that model-checking priced timed automata with three clocks and more against WCTL properties is undecidable.

There are several natural research directions: the decidability of WCTL model-checking for two-clocks priced timed automata is not known, we just know that these models have an infinite bisimulation [10]; another interesting extension is multi-constrained modalities, e.g. $\mathsf{E}\,\phi\mathsf{U}_{P_1\leq 5, P_2>3}\phi$?

References

1. Y. Abdeddaïm, E. Asarin, and O. Maler. Scheduling with timed automata. *Theor. Comp. Science*, 354(2):272–300, 2006.
2. R. Alur, M. Bernadsky, and P. Madhusudan. Optimal reachability in weighted timed games. In *Proc. 31st Intl. Coll. Automata, Languages and Programming (ICALP'04)*, LNCS 3142, p. 122–133. Springer, 2004.
3. R. Alur, S. La Torre, and G. J. Pappas. Optimal paths in weighted timed automata. In *Proc. 4th Intl. Workshop Hybrid Systems: Computation and Control (HSCC'01)*, LNCS 2034, p. 49–62. Springer, 2001.

4. G. Behrmann, A. Fehnker, Th. Hune, K. G. Larsen, P. Pettersson, J. Romijn, and F. Vaandrager. Minimum-cost reachability for priced timed automata. In *Proc. 4th Intl. Workshop Hybrid Systems: Computation and Control (HSCC'01)*, LNCS 2034, p. 147–161. Springer, 2001.
5. P. Bouyer, Th. Brihaye, and N. Markey. Improved undecidability results on weighted timed automata. *Inf. Proc. Letters*, 98(5):188–194, 2006.
6. P. Bouyer, E. Brinksma, and K. G. Larsen. Staying alive as cheaply as possible. In *Proc. 7th Intl. Workshop Hybrid Systems: Computation and Control (HSCC'04)*, LNCS 2993, p. 203–218. Springer, 2004.
7. P. Bouyer, E. Brinksma, and K. G. Larsen. Optimal infinite scheduling for multi-priced timed automata. *Form. Meth. in Syst. Design*, 2006. To appear.
8. P. Bouyer, F. Cassez, E. Fleury, and K. G. Larsen. Optimal strategies in priced timed game automata. In *Proc. 24th Conf. Found. Softw. Tech. & Theor. Comp. Science (FST&TCS'04)*, LNCS 3328, p. 148–160. Springer, 2004.
9. P. Bouyer, K. G. Larsen, N. Markey, and J. I. Rasmussen. Almost optimal strategies in one-clock priced timed automata. In *Proc. 26th Conf. Found. Softw. Tech. & Theor. Comp. Science (FST&TCS'06)*, LNCS 4337, p. 346–357. Springer, 2006.
10. Th. Brihaye, V. Bruyère, and J.-F. Raskin. Model-checking for weighted timed automata. In *Proc. Joint Conf. Formal Modelling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault Tolerant System (FORMATS+FTRTFT'04)*, LNCS 3253, p. 277–292. Springer, 2004.
11. Th. Brihaye, V. Bruyère, and J.-F. Raskin. On optimal timed strategies. In *Proc. 3rd Intl. Conf. Formal Modeling and Analysis of Timed Systems (FORMATS'05)*, LNCS 3821, p. 49–64. Springer, 2005.
12. E. A. Emerson and J. Y. Halpern. "Sometimes" and "not never" revisited: On branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, 1986.
13. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.
14. T. A. Henzinger, O. Kupferman, and M. Y. Vardi. A space-efficient on-the-fly algorithm for real-time model checking. In *Proc. 7th Intl. Conf. Concurrency Theory (CONCUR'96)*, LNCS 1119, p. 514–529. Springer, 1996.
15. F. Laroussinie, N. Markey, and Ph. Schnoebelen. Model checking timed automata with one or two clocks. In *Proc. 15th Intl. Conf. Concurrency Theory (CONCUR'04)*, LNCS 3170, p. 387–401. Springer, 2004.
16. F. Laroussinie, N. Markey, and Ph. Schnoebelen. Efficient timed model checking for discrete-time systems. *Theor. Comp. Science*, 353(1-3):249–271, 2006.
17. K. G. Larsen and J. I. Rassmussen. Optimal conditional reachability for multi-priced timed automata. In *Proc. 8th Intl. Conf. Found. Softw. Science and Computation Structures (FoSSaCS'05)*, LNCS 3441, p. 234–249. Springer, 2005.
18. J. I. Rasmussen, K. G. Larsen, and K. Subramani. Resource-optimal scheduling using priced timed automata. In *Proc. 10th Intl. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04)*, LNCS 2988, p. 220–235. Springer, 2004.

Approximating a Behavioural Pseudometric Without Discount for Probabilistic Systems

Franck van Breugel[1], Babita Sharma[1], and James Worrell[2,⋆]

[1] York University
4700 Keele Street, Toronto, M3J 1P3, Canada
[2] Oxford University Computing Laboratory
Parks Road, Oxford, OX1 3QD, England

Abstract. Desharnais, Gupta, Jagadeesan and Panangaden introduced a family of behavioural pseudometrics for probabilistic transition systems. These pseudometrics are a quantitative analogue of probabilistic bisimilarity. Distance zero captures probabilistic bisimilarity. Each pseudometric has a discount factor, a real number in the interval $(0, 1]$. The smaller the discount factor, the more the future is discounted. If the discount factor is one, then the future is not discounted at all. Desharnais et al. showed that the behavioural distances can be calculated up to any desired degree of accuracy if the discount factor is smaller than one. In this paper, we show that the distances can also be approximated if the future is not discounted. A key ingredient of our algorithm is Tarski's decision procedure for the first order theory over real closed fields. By exploiting the Kantorovich-Rubinstein duality theorem we can restrict to the existential fragment for which more efficient decision procedures exist.

1 Introduction

For systems that contain quantitative information, like, for example, probabilities, time and costs, several *behavioural pseudometrics* (and closely related notions) have been introduced (see, for example, [4,6,8,12,13,16,17,18,19,26,31]). In this paper, we focus on *probabilistic transition systems*, which are a variant of Markov chains. Desharnais, Gupta, Jagadeesan and Panangaden [16] introduced a family of behavioural pseudometrics for these systems. These pseudometrics assign a distance, a real number in the interval $[0, 1]$, to each pair of states of the probabilistic transition system. The distance captures the behavioural similarity of the states. The smaller the distance, the more alike the states behave. The distance is zero if and only if the states are *probabilistic bisimilar*, a behavioural equivalence introduced by Larsen and Skou [24].

The pseudometrics of Desharnais et al. are defined via real-valued interpretations of Larsen and Skou's probabilistic modal logic. Formulae assume truth values in the interval $[0, 1]$. Conjunction and disjunction are interpreted using

⋆ The first and second author were supported by NSERC.

the lattice structure of the unit interval. The modality $\langle a \rangle$ is interpreted arithmetically by integration. The behavioural distance between states s_1 and s_2 is then defined as the supremum over all formulae φ of the difference in the truth value of φ in s_1 and in s_2.[1]

The definition of the behavioural pseudometrics of Desharnais et al. is parametrized by a *discount factor* δ, a real number in the interval $(0, 1]$. The smaller the discount factor, the more (behavioural differences in) the future are discounted. In the case that δ equals one, the future is not discounted. All differences in behaviour, whether in the near or far future, contribute alike to the distance. For systems that (in principle) run forever, we may be interested in all these differences and, hence, in the pseudometric that does not discount the future.

In [14], Desharnais et al. presented an *algorithm* to *approximate* the behavioural distances for δ smaller than one. The first and third author [5] presented also an approximation algorithm for δ smaller than one.

There is a fundamental difference between pseudometrics that discount the future and the one that does not. This is, for example, reflected by the fact that all pseudometrics that discount the future give rise to the same topology, whereas the pseudometric that does not discount the future gives rise to a different topology (see, for example, [16, page 350]). As a consequence, it may not be surprising that neither approximation algorithm mentioned in the previous paragraph can be modified in an obvious way to handle the case that δ equals one.

The main contribution of this paper is an algorithm that approximates behavioural distances in case the discount factor δ equals one. Starting from the *logical* definition of the pseudometric by Desharnais et al., we first give a characterisation of the pseudometric as the greatest (post-)fixed point of a functional on a complete lattice $[0, 1]^S$, where S is the set of states of the probabilistic transition system in question. This functional is closely related to the Kantorovich metric [22] on probability measures. Next, we dualize this characterization exploiting the Kantorovich-Rubinstein duality theorem [23]. Subsequently, we show, exploiting the dual characterization, that a pseudometric being a post-fixed point can be expressed in the existential fragment of the first order theory over real closed fields. Based on the fact that this first order theory is decidable, a result due to Tarski [29], we show how to approximate the behavioural distances. Finally, we discuss an implementation of our algorithm in Mathematica.

Exploiting the techniques put forward in this paper, we have also developed an algorithm to approximate the behavioural pseudometric that is presented in [3]. Due to lack of space, we cannot present this algorithm here. That other algorithm and also the proofs of the results in this paper can be found in [28].

[1] More generally, de Alfaro [11] and McIver and Morgan [25] have given real-valued interpretations to the modal mu-calculus following this pattern. Moreover, de Alfaro has shown that the behavioural pseudometrics induced by mu-calculus formulae agree with those of [16].

2 Systems and Pseudometrics

Some basic notions that will play a role in the rest of this paper are presented below. First we introduce the systems of interest: probabilistic transition systems.

Definition 1. *A probabilistic transition system is a tuple $\langle S, \pi \rangle$ consisting of*

– *a finite set S of states and*
– *a function $\pi : S \times S \to [0, 1] \cap \mathbb{Q}$ satisfying $\sum_{s' \in S} \pi(s, s') \in \{0, 1\}$.*

We write $s \to$ if $\sum_{s' \in S} \pi(s, s') = 1$ and $s \not\to$ if $\sum_{s' \in S} \pi(s, s') = 0$.

For states s and s', $\pi(s, s')$ is the probability of making a transition to state s' given that the system is in state s. Each state s either has no outgoing transitions ($s \not\to$) or a transition is taken with probability 1 ($s \to$). To simplify the presentation, we do not consider the case that a state s may refuse to make a transition with some probability, that is, $\sum_{s' \in S} \pi(s, s') \in (0, 1)$. However, all our results can easily be generalized to handle that case as well (see [28]). We also do not consider transitions that are labelled with actions. All our results can also easily be modified to handle labelled transitions (see [28]). In the labelled case, the definition of probabilistic transition system is a mild generalisation of the notion of Markov chain.

In the rest of this paper, we will use the following probabilistic transition system as our running example.

Example 1. We consider a probabilistic transition system with five states: s_1, s_2, s_3, s_4 and s_5. The following table contains the transition probabilities and, hence, captures π. The probabilistic transition system be depicted as the following graph.

	s_1	s_2	s_3	s_4	s_5
s_1	0	$\frac{2}{5}$	$\frac{3}{5}$	0	0
s_2	$\frac{7}{10}$	0	0	$\frac{1}{5}$	$\frac{1}{10}$
s_3	0	0	1	0	0
s_4	0	0	0	0	0
s_5	0	0	0	0	1

$$\begin{tikzcd}
s_1 \arrow[r, bend left, "\frac{2}{5}"] \arrow[d, "\frac{3}{5}"'] & s_2 \arrow[l, bend left, "\frac{7}{10}"] \arrow[d, "\frac{1}{5}"'] \arrow[dr, "\frac{1}{10}"] & \\
s_3 \arrow[loop left, "1"] & s_4 & s_5 \arrow[loop right, "1"]
\end{tikzcd}$$

We consider states of a probabilistic transition system behaviourally equivalent if they are probabilistic bisimilar [24].

Definition 2. *Let $\langle S, \pi \rangle$ be a probabilistic transition system. An equivalence relation $\mathcal{R}$ on the set of states S is a probabilistic bisimulation if $s_1 \, \mathcal{R} \, s_2$ implies $\sum_{s \in E} \pi(s_1, s) = \sum_{s \in E} \pi(s_2, s)$ for all $\mathcal{R}$-equivalence classes E. States s_1 and s_2 are probabilistic bisimilar, denoted $s_1 \sim s_2$, if $s_1 \, \mathcal{R} \, s_2$ for some probabilistic bisimulation $\mathcal{R}$.*

Note that probabilistic bisimilar states s_1 and s_2 have the same probability of transitioning to an equivalence class E of probabilistic bisimilar states.

Example 2. Consider the probabilistic transition system of Example 1. The smallest equivalence relation containing (s_3, s_5) is a probabilistic bisimulation. Hence, the states s_3 and s_5 are probabilistic bisimilar.

The behavioural pseudometrics that we study in this paper yield pseudometric spaces on the state space of probabilistic transition systems.

Definition 3. *A 1-bounded pseudometric space is a pair (X, d_X) consisting of a set X and a distance function $d_X : X \times X \to [0, 1]$ satisfying*

1. *for all $x \in X$, $d_X(x, x) = 0$,*
2. *for all x, $y \in X$, $d_X(x, y) = d_X(y, x)$, and*
3. *for all x, y, $z \in X$, $d_X(x, z) \leq d_X(x, y) + d_X(y, z)$.*

Instead of (X, d_X) we often write X and we denote the distance function of a metric space X by d_X.

A (1-bounded) pseudometric space differs from a (1-bounded) metric space in that different points may have distance zero in the former and not in the latter. Since different states of a system may behave the same, such states will have distance zero in our behavioural pseudometrics.

In the characterization of a behavioural pseudometric in Section 4 nonexpansive functions play a key role.

Definition 4. *Let X be a 1-bounded pseudometric space. A function $f : X \to [0, 1]$ is nonexpansive if for all x_1, $x_2 \in X$,*

$$|f(x_1) - f(x_2)| \leq d_X(x_1, x_2).$$

The set of nonexpansive functions from X to $[0, 1]$ is denoted by $X \xrightarrow{\,1\,} [0, 1]$.

3 Behavioural Pseudometrics

Desharnais, Gupta, Jagadeesan and Panangaden [16] introduced a family of behavioural pseudometrics for probabilistic transitions systems. Below, we will briefly review the key ingredients of their definition.

To define their behavioural pseudometrics, Desharnais et al. defined a real-valued semantics of a variant of Larsen and Skou's probabilistic modal logic [24]. We describe this variant, adapted to the case of unlabelled transition systems, in Definition 5.

Definition 5. *The logic $\mathcal{L}$ is defined by*

$$\varphi ::= \text{true} \mid \Diamond\varphi \mid \varphi \wedge \varphi \mid \neg\varphi \mid \varphi \ominus q$$

where $q \in [0, 1] \cap \mathbb{Q}$.

The main difference between the above logic and the one of Larsen and Skou is that we have $\Diamond\varphi$ and $\varphi \ominus q$ whereas they combine the operators $\Diamond$ and $\ominus q$ into one. Since they consider labelled transitions, they use the notation $\langle a \rangle_q$ for this combined operator.

Desharnais et al. provided a family of real-valued interpretations of the logic. That is, given a probabilistic transition system and a discount factor δ, the interpretation gives a quantitative measure of the validity of a formula φ of the logic in a state s of the system. The interpretation $[\![\varphi]\!]_\delta(s)$ is a real number in the interval $[0, 1]$. It measures the validity of the formula φ in the state s. This real number can roughly be thought of as the probability that φ is true in s.

Definition 6. *Given a probabilistic transition system $\langle S, \pi \rangle$ and a discount factor $\delta \in (0, 1]$, for each $\varphi \in \mathcal{L}$, the function $[\![\varphi]\!]_\delta : S \to [0, 1]$ is defined by*

$$\begin{aligned}
[\![\mathrm{true}]\!]_\delta(s) &= 1 \\
[\![\Diamond\varphi]\!]_\delta(s) &= \delta \textstyle\sum_{s' \in S} \pi(s, s')[\![\varphi]\!]_\delta(s') \\
[\![\varphi \wedge \psi]\!]_\delta(s) &= \min\{[\![\varphi]\!]_\delta(s), [\![\psi]\!]_\delta(s)\} \\
[\![\neg\varphi]\!]_\delta(s) &= 1 - [\![\varphi]\!]_\delta(s) \\
[\![\varphi \ominus q]\!]_\delta(s) &= \max\{[\![\varphi]\!]_\delta(s) - q, 0\}
\end{aligned}$$

Example 3. Consider the probabilistic transition system of Example 1. For this system, $[\![\Diamond\mathrm{true}]\!]_\delta(s_3) = \delta$ and $[\![\Diamond\mathrm{true}]\!]_\delta(s_4) = 0$.

Given a discount factor $\delta \in (0, 1]$, the behavioural pseudometric d_δ assigns a distance, a real number in the interval $[0, 1]$, to every pair of states of a probabilistic transition system. The distance is defined in terms of the logical formulae and their interpretation. Roughly speaking, the distance is captured by the logical formula that distinguishes the states the most.

Definition 7. *Given a probabilistic transition system $\langle S, \pi \rangle$ and a discount factor $\delta \in (0, 1]$, the distance function $d_\delta : S \times S \to [0, 1]$ is defined by*

$$d_\delta(s_1, s_2) = \sup_{\varphi \in \mathcal{L}} [\![\varphi]\!]_\delta(s_1) - [\![\varphi]\!]_\delta(s_2).$$

Example 4. Consider the probabilistic transition system of Example 1. For example, the states s_3 and s_4 are δ apart. This distance is witnessed by the formula $\Diamond$true. The distances[2] are collected in the following table. Since a distance function is symmetric and the distance from a state to itself is zero, we do not give all the entries.

	s_1	s_2	s_3	s_4
s_2	$\frac{25\delta^2-2\delta^4}{125-25\delta-35\delta^2+7\delta^3}$			
s_3	$\frac{2\delta^3}{25-7\delta^2}$	$\frac{5\delta^2}{25-7\delta^2}$		
s_4	δ	δ	δ	
s_5	$\frac{2\delta^3}{25-7\delta^2}$	$\frac{5\delta^2}{25-7\delta^2}$	0	δ

[2] These distances were obtained "by hand" and checked for numerous different discount factors using the algorithm described in [5].

Proposition 1 ([16, Theorem 5.2]). *d_δ is a 1-bounded pseudometric space.*

Each behavioural pseudometric d_δ is a quantitative analogue of probabilistic bisimilarity. This behavioural equivalence is exactly captured by those states that have distance zero.

Proposition 2 ([16, Theorem 4.10]). *Given a probabilistic transition system $\langle S, \pi \rangle$ and a discount factor $\delta \in (0, 1]$, for all s_1, $s_2 \in S$,*

$$d_\delta(s_1, s_2) = 0 \textit{ if and only if } s_1 \sim s_2.$$

In [14], Desharnais et al. present a decision procedure for the behavioural pseudometric d_δ when δ is smaller than one. Let us briefly sketch their algorithm. They define the depth of a logical formula as follows.

$$\begin{aligned} \mathrm{depth}(\mathrm{true}) &= 0 \\ \mathrm{depth}(\Diamond\varphi) &= \mathrm{depth}(\varphi) + 1 \\ \mathrm{depth}(\varphi \wedge \psi) &= \max\{\mathrm{depth}(\varphi), \mathrm{depth}(\psi)\} \\ \mathrm{depth}(\neg\varphi) &= \mathrm{depth}(\varphi) \\ \mathrm{depth}(\varphi \ominus q) &= \mathrm{depth}(\varphi) \end{aligned}$$

One can easily verify that $[\![\varphi]\!]_\delta(s_1) - [\![\varphi]\!]_\delta(s_2) \leq \delta^{\mathrm{depth}(\varphi)}$ for each $\varphi \in \mathcal{L}$. This suggests that one can compute d_δ to any desired degree of accuracy by restricting attention to formulae φ of a fixed modal depth. Clearly, there exist infinitely many formulae of each fixed modal depth. Nevertheless Desharnais et al. show how to construct a finite subset $\mathcal{F}_n$ of the logical formulae of at most depth n such that

$$d_\delta(s_1, s_2) - \sup_{\varphi \in \mathcal{F}_n} [\![\varphi]\!]_\delta(s_1) - [\![\varphi]\!]_\delta(s_2) \leq \delta^n.$$

In this way, $d_\delta(s_1, s_2)$ can be approximated up to arbitrary accuracy *provided* δ is smaller than one.

4 A Fixed Point Characterization and Its Dual

For the rest of this paper, we focus on the behavioural pseudometric that does not discount the future. That is, we concentrate on the pseudometric d_1. Below, we present an alternative characterization of this pseudometric. In particular, we characterize d_1 as the greatest (post-)fixed point of a function from a complete lattice to itself. This characterization can be viewed as a quantitative analogue of the greatest fixed point characterization of bisimilarity [27].

We also dualize the definition of Δ exploiting the Kantorovich-Rubinstein duality theorem [23]. As we will see in Section 5, this dual characterization will allow us to define Δ as the solution to a minimization problem rather than a maximization problem, as above. In turn this will allow us to capture the fact that a pseudometric is a post-fixed point of Δ in the existential fragment of the first order theory over real closed fields.

For the rest of this paper, we fix a probabilistic transition system $\langle S, \pi \rangle$. We endow the set of pseudometrics on S with the following order.

Definition 8. *The relation $\sqsubseteq$ on 1-bounded pseudometrics on S is defined by*

$$d_1 \sqsubseteq d_2 \text{ if } d_1(s_1, s_2) \geq d_2(s_1, s_2) \text{ for all } s_1,\ s_2 \in S.$$

Note the reverse direction of $\sqsubseteq$ and $\geq$ in the above definition. We decided to make this reversal so that d_1 is a greatest fixed point, in analogy with the characterization of bisimilarity, rather than a least fixed point. This choice has no impact on any results in this paper.

Proposition 3 ([15], Lemma 3.2]). *The set of 1-bounded pseudometrics on S endowed with the order $\sqsubseteq$ forms a complete lattice.*

Next, we introduce a function from this complete lattice to itself of which the behavioural pseudometric d_1 is the greatest fixed point.

Definition 9. *Let d be a 1-bounded pseudometric on S. The distance function $\Delta(d) : S \times S \to [0, 1]$ is defined by*

$$\Delta(d)(s_1, s_2) = \max \left\{ \sum_{s \in S} f(s)(\pi(s_1, s) - \pi(s_2, s)) \;\middle|\; f \in (S, d) \xrightarrow{\nless} [0, 1] \right\}$$

if $s_1 \to$ and $s_2 \to$, and $\Delta(d)(s_1, s_2) = \begin{cases} 0 \text{ if } s_1 \not\to \text{ and } s_2 \not\to \\ 1 \text{ otherwise.} \end{cases}$

The functional Δ is closely related to the Kantorovich metric [22] on probability measures. In the definition of that metric, nonexpansive functions play a key role.[3]

Since $\Delta(d)$ is a 1-bounded pseudometric on S and Δ is order-preserving, we can conclude from Tarski's fixed point theorem [30, Theorem 1] that Δ has a greatest fixed point. We denote the greatest fixed point of Δ by $\text{gfp}(\Delta)$. This greatest fixed point of Δ is also the greatest post-fixed point of Δ (see, for example, [10, Theorem 4.11][4]).

Theorem 1. $d_1 = \text{gfp}(\Delta)$.

The greatest fixed point of an order-preserving function on a complete lattice can be obtained by iteration (see, for example, [10, Exercise 4.13]).

Definition 10. *For each ordinal α, the 1-bounded pseudometric d^α on S is defined by*

$$\begin{aligned} d^0 &= \top \\ d^{\alpha+1} &= \Delta(d^\alpha) \\ d^\beta &= \bigsqcap_{\alpha \in \beta} d^\alpha \text{ if } \beta \text{ is a limit ordinal} \end{aligned}$$

[3] The Kantorovich metric is the smallest distance function on probability measures for which integration of nonexpansive functions is nonexpansive.

[4] d is a *post-fixed point* of Δ if $d \sqsubseteq \Delta(d)$. In [10, page 94], such a d is called a pre-fixpoint.

For Δ, we need to iterate (at most) ω times before reaching the greatest fixed point. For the system of Example 1 we need ω iterations.

Proposition 4. $\mathrm{gfp}(\Delta) = d^{\omega}$.

Let us recall (a minor variation of) the Kantorovich-Rubinstein duality theorem. Let X be a 1-bounded compact pseudometric space. Let μ_1 and μ_2 be Borel probability measures on X. We denote the set of Borel probability measures on the product space with marginals μ_1 and μ_2, that is, the Borel probability measures μ on X^2 such that for all Borel subsets B of X,

$$\mu(B \times X) = \mu_1(B) \text{ and } \mu(X \times B) = \mu_2(B),$$

by $\mu_1 \otimes \mu_2$. The Kantorovich-Rubinstein duality theorem tells us

$$\max\left\{\int_X f d\mu_1 - \int_X f d\mu_2 \;\middle|\; f \in X \not\rightarrow [0,1]\right\} = \min\left\{\int_{X^2} d_X d\mu \;\middle|\; \mu \in \mu_1 \otimes \mu_2\right\}.$$

The following proposition, which is a consequence of the Kantorovich-Rubinstein duality theorem, defines $\Delta(d)$ as a minimum as opposed to the maximum in Definition 9.

Proposition 5. *Let d be a 1-bounded pseudometric on S. Let s_1, $s_2 \in S$ such that $s_1 \rightarrow$ and $s_2 \rightarrow$. Then*

$$\Delta(d)(s_1, s_2) = \min\left\{\sum_{(s_i,s_j)\in S^2} d(s_i, s_j)\mu(s_i, s_j) \;\middle|\; \mu \in \pi(s_1,\cdot) \otimes \pi(s_2,\cdot)\right\}$$

where $\mu \in \pi(s_1,\cdot) \otimes \pi(s_2,\cdot)$ if

$$\forall s_j \in S \sum_{s_i\in S} \mu(s_i, s_j) = \pi(s_1, s_j) \wedge \forall s_i \in S \sum_{s_j\in S} \mu(s_i, s_j) = \pi(s_2, s_i).$$

5 The Algorithm

Before we present our algorithm, we first show that the fact that a pseudometric is a post-fixed point of Δ can be expressed in (the existential fragment of) the first order theory over real closed fields. This will allow us to exploit Tarski's decision procedure to approximate the behavioural pseudometric.

For the rest of this paper, we assume that the probabilistic transition system $\langle S, \pi\rangle$ has N states s_1, s_2, ..., s_N. Instead of $\pi(s_i, s_j)$ we will write π_{ij}. We represent a 1-bounded pseudometric on the set S of states of the probabilistic transition system, as (the values of) a collection of real valued variables d_{ij}.

The fact that d is a 1-bounded pseudometric can now be captured as follows.

Definition 11. *The predicate* $\mathrm{pseudo}(d)$ *is defined by*

$$\begin{aligned}\mathrm{pseudo}(d) \equiv & \bigwedge_{1\le i,j\le N} d_{ij} \ge 0 \wedge d_{ij} \le 1 \wedge \\ & \bigwedge_{1\le i\le N} d_{ii} = 0 \wedge \bigwedge_{1\le i,j\le N} d_{ij} = d_{ji} \wedge \bigwedge_{1\le h,i,j\le N} d_{hj} \le d_{hi} + d_{ij}\end{aligned}$$

Furthermore, the fact that d is a post-fixed point of Δ can be captured as follows.

Definition 12. *The predicate* post-fixed(d) *is defined by*

$$\begin{aligned}&\text{post-fixed}(d)\\ &\equiv \bigwedge_{1 \leq i_0, j_0 \leq N} \text{post-fixed}_1(d, i_0, j_0) \vee \text{post-fixed}_2(d, i_0, j_0) \vee \text{post-fixed}_3(d, i_0, j_0)\end{aligned}$$

where

$$\begin{aligned}\text{post-fixed}_1(d, i_0, j_0) \equiv\; & \sum_{1 \leq i \leq N} \pi_{i_0 i} > 0 \wedge \sum_{1 \leq j \leq N} \pi_{j_0 j} > 0 \wedge \\ & \exists (\mu_{ij})_{1 \leq i, j \leq N} \bigwedge_{1 \leq i,j \leq N} \mu_{ij} \geq 0 \wedge \mu_{ij} \leq 1 \\ & \qquad \bigwedge_{1 \leq j \leq N} \sum_{1 \leq i \leq N} \mu_{ij} = \pi_{i_0 j} \wedge \\ & \qquad \bigwedge_{1 \leq i \leq N} \sum_{1 \leq j \leq N} \mu_{ij} = \pi_{j_0 i} \wedge \\ & \qquad \sum_{1 \leq i,j \leq N} d_{ij} \mu_{ij} \leq d_{i_0 j_0} \\ \text{post-fixed}_2(d, i_0, j_0) \equiv\; & \sum_{1 \leq i \leq N} \pi_{i_0 i} = 0 \wedge \sum_{1 \leq j \leq N} \pi_{j_0 j} = 0 \wedge 0 \leq d_{i_0 j_0} \\ \text{post-fixed}_3(d, i_0, j_0) \equiv\; & \left(\left(\sum_{1 \leq i \leq N} \pi_{i_0 i} > 0 \wedge \sum_{1 \leq j \leq N} \pi_{j_0 j} = 0 \right) \vee \right. \\ & \left. \left(\sum_{1 \leq i \leq N} \pi_{i_0 i} = 0 \wedge \sum_{1 \leq j \leq N} \pi_{j_0 j} > 0 \right) \right) \wedge \\ & 1 \leq d_{i_0 j_0}\end{aligned}$$

Now we are ready to present our algorithm. Consider the states s_{i_0} and s_{j_0}. We restrict our attention to the case that $s_{i_0} \rightarrow$ and $s_{j_0} \rightarrow$. In the other cases the computation of the distance is trivial.

In our algorithm, we use the algorithm `tarski` that takes as input a sentence of the first order theory of real closed fields and decides the truth or falsity of the given sentence. The fact that there exists such an algorithm was first proved by Tarski [29].

Let ϵ be the desired accuracy. That is, we want to find an interval $[\ell_0, u_0] \subseteq [0, 1]$ such that $u_0 - \ell_0 \leq \epsilon$ and $d_1(s_{i_0}, s_{j_0}) \in [\ell_0, u_0]$. The algorithm `approximate` takes as input an interval $[\ell, u] \subseteq [0, 1]$ such that $d_1(s_{i_0}, s_{j_0}) \in [\ell, u]$ and returns the desired result. As a consequence, `approximate`(0, 1) returns an approximation of $d_1(s_{i_0}, s_{j_0})$ with accuracy ϵ.

```
approximate($\ell$, $u$):
   if $u - \ell \leq \epsilon$
      return $[\ell, u]$
   else
      $m = \frac{\ell+u}{2}$
      if tarski($\exists d\, \text{pseudo}(d) \wedge \text{post-fixed}(d) \wedge d_{i_0 j_0} \leq m$)
         return approximate($\ell$, $m$)
      else
         return approximate($m$, $u$)
```

Note that the argument of `tarski` is a sentence that is part of the existential fragment of the first order theory over real closed fields. For this fragment there are more efficient decision procedures than for the general theory (see, for example, [2]).

Let us sketch a correctness proof of our algorithm. Assume that $d_1(s_{i_0}, s_{j_0}) \in [\ell, u]$. We distinguish the following three cases.

– If $u - \ell \leq \epsilon$, then the algorithm obviously returns the desired result.
– Assume that $u - \ell > \epsilon$ and suppose that `tarski` returns true. Then there exists a 1-bounded pseudometric d that is a post-fixed point of Δ and $d(s_{i_0}, s_{j_0}) \leq m$. Since d_1 is the greatest post-fixed point of Δ, we have that $d \sqsubseteq d_1$. Hence, $d_1(s_{i_0}, s_{j_0}) \leq d(s_{i_0}, s_{j_0}) \leq m$. Therefore, $d_1(s_{i_0}, s_{j_0}) \in [\ell, m]$.
– Assume that $u - \ell > \epsilon$ and suppose that `tarski` returns false. Then $d(s_{i_0}, s_{j_0}) > m$ for every 1-bounded pseudometric d that is a post-fixed point of Δ. Since d_1 is a post-fixed point of Δ, we have that $d_1(s_{i_0}, s_{j_0}) > m$. Therefore, $d_1(s_{i_0}, s_{j_0}) \in [m, u]$.

Obviously, the algorithm terminates.

6 An Implementation in Mathematica

A decision procedure for the first order theory of real closed fields based on quantifier elimination was first given by Tarski [29]. A number of algorithms have been developed thereafter for the theory (see, for example, [2,9,21]). Collin's algorithm is implemented in the tool Mathematica and can be used for solving our formulae. However, it works for very small examples and therefore it is essential to simplify the formula and reduce its size to make it solvable. To simplify the formula, we first compute some of the distances using the following results.

Proposition 6

– *If* $s_1 \not\rightarrow$ *and* $s_2 \not\rightarrow$ *then* $d_1(s_1, s_2) = 0$.
– *If* $s_1 \not\rightarrow$ *and* $s_2 \rightarrow$, *or* $s_1 \rightarrow$ *and* $s_2 \not\rightarrow$ *then* $d_1(s_1, s_2) = 1$.

Example 5. Consider the probabilistic transition system of Example 1. State s_4 has distance one to all other states.

Next, we present a simple characterization of the distance between a state that never terminates (that is, the probability of reaching a state with no outgoing transitions is zero) and another state.

Given a state s and $n \in \omega + 1$, $\tau_n(s)$ is the probability of terminating in less than n transitions when started in s.

Definition 13. *For each $n \in \omega + 1$, the function $\tau_n : S \to [0, 1]$ is defined by*

$$\begin{aligned}\tau_0(s) &= 0\\ \tau_{n+1}(s) &= \begin{cases} 1 & \textit{if } s \not\to\\ \sum_{s' \in S} \pi(s, s')\tau_n(s') & \textit{otherwise}\end{cases}\\ \tau_\omega(s) &= \sup_{n\in\omega} \tau_n(s)\end{aligned}$$

Example 6. Consider the probabilistic transition system of Example 1. Then we have that $\tau_\omega(s_1) = \frac{1}{9}$, $\tau_\omega(s_2) = \frac{5}{18}$, $\tau_\omega(s_3) = 0$, $\tau_\omega(s_4) = 1$ and $\tau_\omega(s_5) = 0$.

Obviously, for a state s without outgoing transitions, we have that $\tau_\omega(s) = 1$. For a state s that cannot reach any state without outgoing transitions, we have that $\tau_\omega(s) = 0$. For the remaining states, we can compute the probability of termination using standard techniques as described in, for example, [20, Section 11.2].

Proposition 7. *If $\tau_\omega(s_2) = 0$ then $d_1(s_1, s_2) = \tau_\omega(s_1)$.*

Example 7. Consider the probabilistic transition system of Example 1. From Proposition 7 we can conclude that $d_1(s_1, s_3) = \frac{1}{9}$, $d_1(s_2, s_3) = \frac{5}{18}$, $d_1(s_4, s_3) = 1$ and $d_1(s_5, s_3) = 0$.

Given a probabilistic bisimulation $\mathcal{R}$, we can quotient the probabilistic transition system $\langle S, \pi\rangle$ as follows.

Definition 14. *Let $\mathcal{R}$ be a probabilistic bisimulation. The probabilistic transition system $\langle S_\mathcal{R}, \pi_\mathcal{R}\rangle$ consists of*

- *the set $S_\mathcal{R} = \{\, [s] \mid s \in S \,\}$ of $\mathcal{R}$-equivalence classes and*
- *the function $\pi_\mathcal{R} : S_\mathcal{R} \times S_\mathcal{R} \to [0, 1]$ defined by*

$$\pi_\mathcal{R}([s], [s']) = \sum_{s'' \mathcal{R} s'} \pi(s, s'').$$

Note that the function $\pi_\mathcal{R}$ is well-defined since $\mathcal{R}$ is a probabilistic bisimulation. We will apply the above quotient construction for probabilistic bisimilarity (which can be computed in polynomial time [1]).

Example 8. Consider the probabilistic transition system of Example 1. The smallest equivalence relation containing $\{\langle s_3, s_5\rangle\}$ is a probabilistic bisimulation. The resulting quotient can be depicted as

$$\begin{tikzcd}
{[s_1]} \arrow[r, bend left, "\frac{2}{5}"] \arrow[d, "\frac{3}{5}"'] & {[s_2]} \arrow[l, bend left, "\frac{7}{10}"] \arrow[dl, "\frac{1}{10}"] \arrow[d, "\frac{1}{5}"] \\
{[s_3]} \arrow[loop left, "1"] & {[s_4]}
\end{tikzcd}$$

By quotienting, the number of states that need to be considered and, hence, the number of variables in the formula may be reduced. However, we still have to check that the quotiented system gives rise to the same distances. Next we relate the behavioural pseudometric d_1 of the original system $\langle S, \pi\rangle$ with the behavioural pseudometric $d_{\mathcal{R}}$ of the quotiented system $\langle S_{\mathcal{R}}, \pi_{\mathcal{R}}\rangle$.

Proposition 8. *For all s_1, $s_2 \in S$, $d_{\mathcal{R}}([s_1], [s_2]) = d_1(s_1, s_2)$.*

To simplify the formula even further, we exploit the following three observations.

- Since d is a pseudometric, $d_1(s_i, s_i) = 0$ and $d_1(s_i, s_j) = d_1(s_j, s_i)$. Therefore, in pseudo($d$) $\wedge$ post-fixed(d) we can replace all d_{ii}'s with zero and all d_{ij}'s where $i > j$ with d_{ji}'s. As a consequence, we only need to consider d_{ij}'s with $i < j$. This reduces the number of variables in the formula considerably.
- Let C be the set of pairs of states for which the distances have already been computed. Then

$$\exists d\, \text{pseudo}(d) \wedge \text{post-fixed}(d) \wedge d_{i_0 j_0} \leq m$$

  is equivalent to

$$\exists d\, \text{pseudo}(d) \wedge \text{post-fixed}(d) \wedge d_{i_0 j_0} \leq m \wedge \bigwedge_{(i,j) \in C} d_{ij} = d_1(s_i, s_j)$$

  since d_1 is the greatest post-fixed point. As a consequence, we can replace all d_{ij}'s where $(i, j) \in C$ with their already computed distances $d_1(s_i, s_j)$. Again, the number of variables may be reduced.
- If $\pi_{i_0 j} = 0$, we can infer that $\mu_{ij} = 0$ for all $1 \leq i \leq N$. As a consequence, we can replace the occurrences of all those μ_{ij}'s with 0. Symmetrically, if $\pi_{j_0 i} = 0$ we can simplify the formula similarly. Also this simplification may reduce the number of variables.

We have implemented these simplifications in the form of a Java program that takes as input the probability matrix π and that produces as output the simplified formula in a format that can be fed to Mathematica.[5]

Example 9. Consider the probabilistic transition system of Example 1. The simplified formula for this system is given below.

[5] The code and documentation is available at the URL
`www.cse.yorku.ca/~franck/research/pm2m`

```
1  Reduce[
2    Exists[d12,
3      (0 <= d12 <= 1) && (0.11112 <= d12 + 0.27778) && (d12 <= 0.38889) &&
4      Exists[{u12,u13,u32,u42,u43,u33},
5        (0 <= u12 <= 1) && (0 <= u13 <= 1) && (0 <= u32 <= 1) &&
6        (0 <= u42 <= 1) && (0 <= u43 <= 1) &&
7        (u12 + u32 + u42 == 0.4) && (u13 + u43 + u33 == 0.6) &&
8        (u12 + u13 == 0.7) && (u32 + u33 == 0.1) && (u42 + u43 == 0.2) &&
9        (d12 * u12 + 0.11112 * u13 + 0.27778 * u32 + u42 + u43 <= d12)] &&
10     Exists[{u21,u23,u24,u31,u33, u34},
11       (0 <= u21 <= 1) && (0 <= u23 <= 1) && (0 <= u24 <= 1) &&
12       (0 <= u31 <= 1) && (0 <= u34 <= 1) &&
13       (u21 + u31 == 0.7) && (u23 + u33 == 0.1) && (u24 + u34 == 0.2) &&
14       (u21 + u23 + u24 == 0.4) && (u31 + u33 + u34 == 0.6) &&
15       (d12 * u21 + 0.27778 * u23 + u24 + 0.11112 * u31 + u34 <= d12)] &&
16     (0 <= d12 <= 0.5)]]
```

Line 3 correspond to pseudo(d), line 4–9 correspond to post-fixed$_1(d, 1, 2)$ and line 10–15 correspond to post-fixed$_1(d, 2, 1)$. The formula was reduced to true by Mathematica in 8.2 seconds on a 3GHz machine with 1GB RAM. When feeding Mathematica the formula that has not been simplified, it runs out of memory after some time.

We also attempted to solve this example with a solver called QEPCAD B [7] but the performance of Mathematica on this example was better.

7 Conclusion

This paper combines a number of ingredients, known already for a long time, including the Kantorovich-Rubinstein duality theorem of the fifties, Tarski's fixed point theorem of the fourties and Tarski's decision procedure for the first order theory of real closed fields of the thirties. We show that the behavioural pseudometric d_1, which does not discounts the future, can be approximated up to an arbitrary accuracy. While the combination of the above results into a decision procedure for the pseudometric is not technically difficult, we do solve a problem that has been open since 1999. Most of the results in Section 3 and 4 are (variations on) known results. As far as we know, the results in Section 5 and 6 are new. The techniques exploited in this paper have also been used to approximate other behavioural pseudometrics that do not discount the future like, for example, the one presented in [3]. Furthermore, our algorithm can easily be adjusted to the discounted case. As future work, we plan to apply our techniques to obtain approximation algorithms for other behavioural pseudometrics like, for example, the one for systems that combine nondeterminism and probability presented in [13]. Since the satisfiability problem for the existential fragment of the first order theory of the real closed fields is PSPACE, it is not surprising that our algorithm can only handle small examples as we have shown in Section 6. As a consequence, the quest for practical algorithms to approximate d_1 is still open. Since the closure ordinal of Δ is ω, as proved in Proposition 4, an iterative algorithm might be feasible.

Acknowledgements. The authors would like to thank Christel Baier for providing some pointers to the literature and Jeff Edmonds, Parke Godfrey and the referees for their constructive feedback.[6]

References

1. C. Baier, B. Engelen, and M. Majster-Cederbaum. Deciding bisimilarity and similarity for probabilistic processes. *Journal of Computer and System Sciences*, 60(1):187–231, 2000.
2. S. Basu, R. Pollack, and M.-F. Roy. On the combinatorial and algebraic complexity of quantifier elimination. *Journal of the ACM*, 43(6):1002–1045, 1996.
3. F. van Breugel. A behavioural pseudometric for metric labelled transition systems. In *Proceedings of the 16th International Conference on Concurrency Theory*, volume 3653 of *Lecture Notes in Computer Science*, pages 141–155. Springer-Verlag, 2005.
4. F. van Breugel and J. Worrell. A behavioural pseudometric for probabilistic transition systems. *Theoretical Computer Science*, 331(1):115–142, 2005.
5. F. van Breugel and J. Worrell. An algorithm for approximating behavioural pseudometrics for probabilistic transition systems. *Theoretical Computer Science*, 360(1/3):373–385, 2006.
6. M. Broucke. Regularity of solutions and homotopic equivalence for hybrid systems. In *Proceedings of the 37th IEEE Conference on Decision and Control*, volume 4, pages 4283–4288. IEEE, 1998.
7. C.W. Brown. An overview of QEPCAD B: a tool for real quantifier elimination and formula simplification. *Journal of Japan Society for Symbolic and Algebraic Computation*, 10(1):13–22, 2003.
8. P. Caspi and A. Benveniste. Toward an approximation theory for computerised control. In *Proceedings of the 2nd International Conference on Embedded Software*, volume 2491 of *Lecture Notes in Computer Science*, pages 294–304. Springer-Verlag, 2002.
9. G.E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Proceedings of the 2nd GI Conference on Automata Theory and Formal Languages*, volume 33 of *Lecture Notes in Computer Science*, pages 134–183. Springer-Verlag, 1975.
10. B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order*, Cambridge University Press, 1990.
11. L. de Alfaro. Quantitative verification and control via the mu-calculus. In *Proceedings of the 14th International Conference on Concurrency Theory*, volume 2761 of *Lecture Notes in Computer Science*, pages 102–126. Springer-Verlag, 2003.
12. L. de Alfaro, T.A. Henzinger, and R. Majumdar. Discounting the future in systems theory. In *Proceedings of 30th International Colloquium on Automata, Languages and Programming*, volume 2719 of *Lecture Notes in Computer Science*, pages 1022–1037. Springer-Verlag, 2003.
13. Y. Deng, T. Chothia, C. Palamidessi, and J. Pang. Metrics for action-labelled quantitative transition systems. In *Proceedings of 3rd Workshop on Quantitative Aspects of Programming Languages*, volume 153(2) of *Electronic Notes in Theoretical Computer Science*, pages 79–96. Elsevier, 2005.

[6] Due to lack of space, some suggestions could unfortunately not be implemented.

14. J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for labeled Markov systems. In *Proceedings of 10th International Conference on Concurrency Theory*, volume 1664 of *Lecture Notes in Computer Science*, pages 258–273. Springer-Verlag, 1999.
15. J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. The metric analogue of weak bisimulation for probabilistic processes. In *Proceedings of 17th Annual IEEE Symposium on Logic in Computer Science*, pages 413–422. IEEE, 2002.
16. J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for labelled Markov processes. *Theoretical Computer Science*, 318(3):323–354, 2004.
17. A. Di Pierro, C. Hankin, and H. Wiklicky. Quantitative relations and approximate process equivalences. In *Proceedings of the 14th International Conference on Concurrency Theory*, volume 2761 of *Lecture Notes in Computer Science*, pages 508–522. Springer-Verlag, 2003.
18. A. Giacalone, C.-C. Jou, and S.A. Smolka. Algebraic reasoning for probabilistic concurrent systems. In *Proceedings of the IFIP WG 2.2/2.3 Working Conference on Programming Concepts and Methods*, pages 443–458. North-Holland, 1990.
19. A. Girard and G.J. Pappas. Approximate bisimulations for nonlinear dynamical systems. In *Proceedings of the 44th IEEE Conference on Decision and Control and the European Control Conference*, pages 684–689. IEEE, 2005.
20. C.M. Grinstead and J.L. Snell. *Introduction to Probability*, AMS, 1997.
21. L. Hörmander. *The Analysis of Linear Partial Differential Operators II: Differential Operators with Constant Coefficients*, Springer-Verlag, 2005.
22. L. Kantorovich. On the transfer of masses (in Russian). *Doklady Akademii Nauk*, 37(2):227–229, 1942. Translated in *Management Science*, 5:1–4, 1959.
23. L.V. Kantorovich and G.Sh. Rubinstein. On the space of completely additive functions. *Vestnik Leningradskogo Universiteta*, 3(2):52–59, 1958. In Russian.
24. K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.
25. A.K. McIver and C.C. Morgan. Results of the quantitative μ-calculus qMμ. *ACM Transactions on Computational Logic*. To appear.
26. J.C Mitchell, A. Ramanathan, A. Scedrov, and V. Teague. A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. *Theoretical Computer Science*, 353(1/3):118–164, 2006.
27. D. Park. Concurrency and automata on infinite sequences. In *Proceedings of 5th GI-Conference on Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.
28. B. Sharma. An algorithm to quantify behavioural similarity between probabilistic systems. Master's thesis, York University, Toronto, 2006.
29. A. Tarski. *A decision method for elementary algebra and geometry*, University of California Press, 1951.
30. A. Tarski. A lattice-theoretic fixed point theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.
31. M. Ying. Bisimulation indexes and their applications. *Theoretical Computer Science*, 275(1/2):1–68, 2002.

Optimal Strategy Synthesis in Stochastic Müller Games[⋆,⋆⋆]

Krishnendu Chatterjee

EECS, University of California, Berkeley, USA
c_krish@eecs.berkeley.edu

Abstract. The theory of graph games with ω-regular winning conditions is the foundation for modeling and synthesizing reactive processes. In the case of stochastic reactive processes, the corresponding stochastic graph games have three players, two of them (System and Environment) behaving adversarially, and the third (Uncertainty) behaving probabilistically. We consider two problems for stochastic graph games: the *qualitative* problem asks for the set of states from which a player can win with probability 1 (*almost-sure winning*); and the *quantitative* problem asks for the maximal probability of winning (*optimal winning*) from each state. We consider ω-regular winning conditions formalized as Müller winning conditions. We present optimal memory bounds for *pure* almost-sure winning and optimal winning strategies in stochastic graph games with Müller winning conditions. We also present improved memory bounds for randomized almost-sure winning and optimal strategies.

1 Introduction

A stochastic graph game [6] is played on a directed graph with three kinds of states: player-1, player-2, and probabilistic states. At player-1 states, player 1 chooses a successor state; at player-2 states, player 2 chooses a successor state; and at probabilistic states, a successor state is chosen according to a given probability distribution. The result of playing the game forever is an infinite path through the graph. If there are no probabilistic states, we refer to the game as a *2-player graph game*; otherwise, as a $2^1/_2$*-player graph game*. There has been a long history of using 2-player graph games for modeling and synthesizing reactive processes [1,16]: a reactive system and its environment represent the two players, whose states and transitions are specified by the states and edges of a game graph. Consequently, $2^1/_2$-player graph games provide the theoretical foundation for modeling and synthesizing processes that are both reactive and stochastic.

For the modeling and synthesis (or "control") of reactive processes, one traditionally considers ω-regular winning conditions, which naturally express the temporal specifications and fairness assumptions of transition systems [12]. This

⋆ This research was supported in part by the the AFOSR MURI grant F49620-00-1-0327, and the NSF grant CCR-0225610.

⋆⋆ Full proofs available in [2].

paper focuses on $2^1\!/_2$-player graph games with respect to an important normal form of ω-regular winning conditions; namely *Müller* winning conditions [17].

In the case of 2-player graph games, where no randomization is involved, a fundamental determinacy result of Gurevich and Harrington [10] based on LAR (*latest appearance record*) construction ensures that, given an ω-regular winning condition, at each state, either player 1 has a strategy to ensure that the condition holds, or player 2 has a strategy to ensure that the condition does not hold. Thus, the problem of solving 2-player graph games consists in finding the set of *winning states*, from which player 1 can ensure that the condition holds. Along with the computation of the winning states, the characterization of complexity of winning strategies is a central question, since the winning strategies represent the implementation of the controller in the synthesis problem. The elegant algorithm of Zielonka [18] uses the LAR construction to compute winning sets in 2-player graph games with Müller conditions. In [7] the authors present an insightful analysis of Zielonka's algorithm to present optimal memory bounds (matching upper and lower bound) for winning strategies in 2-player graph games with Müller conditions.

In the case of $2^1\!/_2$-player graph games, where randomization is present in the transition structure, the notion of winning needs to be clarified. Player 1 is said to *win surely* if she has a strategy that guarantees to achieve the winning condition against all player-2 strategies. While this is the classical notion of winning in the 2-player case, it is less meaningful in the presence of probabilistic states, because it makes all probabilistic choices adversarial (it treats them analogously to player-2 choices). To adequately treat probabilistic choice, we consider the *probability* with which player 1 can ensure that the winning condition is met. We thus define two solution problems for $2^1\!/_2$-player graph games: the *qualitative* problem asks for the set of states from which player 1 can ensure winning with probability 1; the *quantitative* problem asks for the maximal probability with which player 1 can ensure winning from each state (this probability is called the *value* of the game at a state). Correspondingly, we define *almost-sure winning strategies*, which enable player 1 to win with probability 1 whenever possible, and *optimal strategies*, which enable player 1 to win with maximal probability. The main result of this paper is an optimal memory bound for pure (deterministic) almost-sure and optimal strategies in $2^1\!/_2$-player graph games with Müller conditions. In fact we generalize the elegant analysis of [7] to present an upper bound for optimal strategies for $2^1\!/_2$-player graph games with Müller conditions that matches the lower bound for sure winning in 2-player games. As a consequence we generalize several results known for $2^1\!/_2$-player graph games: such as existence of pure memoryless optimal strategies for *parity* conditions [5,19,14] and Rabin conditions [4]. We present the result for almost-sure strategies in Section 3; and then generalize it to optimal strategies in Section 4. We also study the memory bounds for randomized strategies. In case of randomized strategies we improve the upper bound for almost-sure and optimal strategies as compared to pure strategies (Section 5). The problem of a matching upper and lower bound for almost-sure and optimal randomized strategies remains open.

2 Definitions

We consider several classes of turn-based games, namely, two-player turn-based probabilistic games ($2\,1/2$-player games), two-player turn-based deterministic games (2-player games), and Markov decision processes ($1\,1/2$-player games).

Notation. For a finite set A, a *probability distribution* on A is a function $\delta\colon A \to [0,1]$ such that $\sum_{a\in A}\delta(a) = 1$. We denote the set of probability distributions on A by $\mathcal{D}(A)$. Given a distribution $\delta \in \mathcal{D}(A)$, we denote by $\mathrm{Supp}(\delta) = \{x \in A \mid \delta(x) > 0\}$ the *support* of δ.

Game graphs. A *turn-based probabilistic game graph* (*$2\,1/2$-player game graph*) $G = ((S,E),(S_1,S_2,S_{\bigcirc}),\delta)$ consists of a directed graph (S,E), a partition (S_1, S_2, $S_{\bigcirc}$) of the finite set S of states, and a probabilistic transition function δ: $S_{\bigcirc} \to \mathcal{D}(S)$, where $\mathcal{D}(S)$ denotes the set of probability distributions over the state space S. The states in S_1 are the *player*-1 states, where player 1 decides the successor state; the states in S_2 are the *player*-2 states, where player 2 decides the successor state; and the states in $S_{\bigcirc}$ are the *probabilistic* states, where the successor state is chosen according to the probabilistic transition function δ. We assume that for $s \in S_{\bigcirc}$ and $t \in S$, we have $(s,t) \in E$ iff $\delta(s)(t) > 0$, and we often write $\delta(s,t)$ for $\delta(s)(t)$. For technical convenience we assume that every state in the graph (S,E) has at least one outgoing edge. For a state $s \in S$, we write $E(s)$ to denote the set $\{\, t \in S \mid (s,t) \in E \,\}$ of possible successors.

A set $U \subseteq S$ of states is called *δ-closed* if for every probabilistic state $u \in U \cap S_{\bigcirc}$, if $(u,t) \in E$, then $t \in U$. The set U is called *δ-live* if for every nonprobabilistic state $s \in U \cap (S_1 \cup S_2)$, there is a state $t \in U$ such that $(s,t) \in E$. A δ-closed and δ-live subset U of S induces a *subgame graph* of G, indicated by $G \restriction U$.

The *turn-based deterministic game graphs* (*2-player game graphs*) are the special case of the $2\,1/2$-player game graphs with $S_{\bigcirc} = \emptyset$. The *Markov decision processes* (*$1\,1/2$-player game graphs*) are the special case of the $2\,1/2$-player game graphs with $S_1 = \emptyset$ or $S_2 = \emptyset$. We refer to the MDPs with $S_2 = \emptyset$ as *player*-1 *MDPs*, and to the MDPs with $S_1 = \emptyset$ as *player*-2 *MDPs*.

Plays and strategies. An infinite path, or *play*, of the game graph G is an infinite sequence $\omega = \langle s_0, s_1, s_2, \ldots\rangle$ of states such that $(s_k, s_{k+1}) \in E$ for all $k \in \mathbb{N}$. We write Ω for the set of all plays, and for a state $s \in S$, we write $\Omega_s \subseteq \Omega$ for the set of plays that start from the state s.

A *strategy* for player 1 is a function σ: $S^* \cdot S_1 \to \mathcal{D}(S)$ that assigns a probability distribution to all finite sequences $\boldsymbol{w} \in S^* \cdot S_1$ of states ending in a player-1 state (the sequence represents a prefix of a play). Player 1 follows the strategy σ if in each player-1 move, given that the current history of the game is $\boldsymbol{w} \in S^* \cdot S_1$, she chooses the next state according to the probability distribution $\sigma(\boldsymbol{w})$. A strategy must prescribe only available moves, i.e., for all $\boldsymbol{w} \in S^*$, and $s \in S_1$ we have $\mathrm{Supp}(\sigma(\boldsymbol{w} \cdot s)) \subseteq E(s)$. The strategies for player 2 are defined analogously. We denote by Σ and Π the set of all strategies for player 1 and player 2, respectively.

Once a starting state $s \in S$ and strategies $\sigma \in \Sigma$ and $\pi \in \Pi$ for the two players are fixed, the outcome of the game is a random walk $\omega_s^{\sigma,\pi}$ for which the probabilities of events are uniquely defined, where an *event* $\mathcal{A} \subseteq \Omega$ is a measurable set of paths. Given strategies σ for player 1 and π for player 2, a play $\omega = \langle s_0, s_1, s_2, \ldots \rangle$ is *feasible* if for every $k \in \mathbb{N}$ the following three conditions hold: (1) if $s_k \in S_{\bigcirc}$, then $(s_k, s_{k+1}) \in E$; (2) if $s_k \in S_1$, then $\sigma(s_0, s_1, \ldots, s_k)(s_{k+1}) > 0$; and (3) if $s_k \in S_2$ then $\pi(s_0, s_1, \ldots, s_k)(s_{k+1}) > 0$. Given two strategies $\sigma \in \Sigma$ and $\pi \in \Pi$, and a state $s \in S$, we denote by $\mathrm{Outcome}(s, \sigma, \pi) \subseteq \Omega_s$ the set of feasible plays that start from s given strategies σ and π. For a state $s \in S$ and an event $\mathcal{A} \subseteq \Omega$, we write $\mathrm{Pr}_s^{\sigma,\pi}(\mathcal{A})$ for the probability that a path belongs to $\mathcal{A}$ if the game starts from the state s and the players follow the strategies σ and π, respectively. In the context of player-1 MDPs we often omit the argument π, because Π is a singleton set.

We classify strategies according to their use of randomization and memory. The strategies that do not use randomization are called pure. A player-1 strategy σ is *pure* if for all $\boldsymbol{w} \in S^*$ and $s \in S_1$, there is a state $t \in S$ such that $\sigma(\boldsymbol{w} \cdot s)(t) = 1$. We denote by $\Sigma^P \subseteq \Sigma$ the set of pure strategies for player 1. A strategy that is not necessarily pure is called *randomized*. Let $\mathtt{M}$ be a set called *memory*, that is, $\mathtt{M}$ is a set of memory elements. A player-1 strategy σ can be described as a pair of functions $\sigma = (\sigma_u, \sigma_m)$: a *memory-update* function σ_u: $S \times \mathtt{M} \to \mathtt{M}$ and a *next-move* function σ_m: $S_1 \times \mathtt{M} \to \mathcal{D}(S)$. We can think of strategies with memory as input/output automaton computing the strategies (see [7] for details). The strategy (σ_u, σ_m) is *finite-memory* if the memory $\mathtt{M}$ is finite, and then we denote the size of the memory of the strategy σ by the size of its memory $\mathtt{M}$, i.e., $|\mathtt{M}|$. We denote by Σ^F the set of finite-memory strategies for player 1, and by Σ^{PF} the set of *pure finite-memory* strategies; that is, $\Sigma^{PF} = \Sigma^P \cap \Sigma^F$. The strategy (σ_u, σ_m) is *memoryless* if $|\mathtt{M}| = 1$; that is, the next move does not depend on the history of the play but only on the current state. A memoryless player-1 strategy can be represented as a function σ: $S_1 \to \mathcal{D}(S)$. A *pure memoryless strategy* is a pure strategy that is memoryless. A pure memoryless strategy for player 1 can be represented as a function σ: $S_1 \to S$. We denote by Σ^M the set of memoryless strategies for player 1, and by Σ^{PM} the set of pure memoryless strategies; that is, $\Sigma^{PM} = \Sigma^P \cap \Sigma^M$. Analogously we define the corresponding strategy families Π^P, Π^F, Π^{PF}, Π^M, and Π^{PM} for player 2.

Given a finite-memory strategy $\sigma \in \Sigma^F$, let G_σ be the game graph obtained from G under the constraint that player 1 follows the strategy σ. The corresponding definition G_π for a player-2 strategy $\pi \in \Pi^F$ is analogous, and we write $G_{\sigma,\pi}$ for the game graph obtained from G if both players follow the finite-memory strategies σ and π, respectively. Observe that given a $2^1\!/\!_2$-player game graph G and a finite-memory player-1 strategy σ, the result G_σ is a player-2 MDP. Similarly, for a player-1 MDP G and a finite-memory player-1 strategy σ, the result G_σ is a Markov chain. Hence, if G is a $2^1\!/\!_2$-player game graph and the two players follow finite-memory strategies σ and π, the result $G_{\sigma,\pi}$ is a Markov chain. These observations will be useful in the analysis of $2^1\!/\!_2$-player games.

Objectives. An *objective* for a player consists of an ω-regular set of *winning plays* $\Phi \subseteq \Omega$ [17]. In this paper we study zero-sum games, where the objectives of the two players are complementary; that is, if the objective of one player is Φ, then the objective of the other player is $\overline{\Phi} = \Omega \setminus \Phi$. We consider ω-regular objectives specified as Müller objectives. For a play $\omega = \langle s_0, s_1, s_2, \ldots \rangle$, let $\mathrm{Inf}(\omega)$ be the set $\{ s \in S \mid s = s_k \text{ for infinitely many } k \geq 0 \}$ of states that appear infinitely often in ω. We use colors to define objectives as in [7]. A $2^1\!/\!_2$-player game $(G, C, \chi, \mathcal{F} \subseteq \mathcal{P}(C))$ consists of a $2^1\!/\!_2$-player game graph G, a finite set C of colors, a partial function $\chi : S \rightharpoonup C$ that assigns colors to some states, and a winning condition specified by a subset $\mathcal{F}$ of the power set $\mathcal{P}(C)$ of colors. The winning condition defines subset $\Phi \subseteq \Omega$ of winning plays, defined as follows: $\text{Müller}(\mathcal{F}) = \{ \omega \in \Omega \mid \chi(\mathrm{Inf}(\omega)) \in \mathcal{F} \}$, that is the set of paths ω such that the colors appearing infinitely often in ω is in $\mathcal{F}$.

Remarks. A winning condition $\mathcal{F} \subseteq \mathcal{P}(C)$ has a *split* if there are sets $C_1, C_2 \in \mathcal{F}$ such that $C_1 \cup C_2 \notin \mathcal{F}$. A winning condition is a *Rabin winning* condition if it do not have splits, and it is a *Streett winning* condition if $\mathcal{P}(C) \setminus \mathcal{F}$ does not have a split. This notions coincide with the Rabin and Streett winning conditions usually defined in the literature (see [15,7] for details). We now define the reachability, safety, Büchi and coBüchi objectives that will be useful in proofs.

– *Reachability and safety objectives.* Given a set $T \subseteq S$ of "target" states, the reachability objective requires that some state of T be visited. The set of winning plays is thus $\mathrm{Reach}(T) = \{ \omega = \langle s_0, s_1, s_2, \ldots \rangle \in \Omega \mid s_k \in T \text{ for some } k \geq 0 \}$. Given a set $F \subseteq S$, the safety objective requires that only states of F be visited. Thus, the set of winning plays is $\mathrm{Safe}(F) = \{ \omega = \langle s_0, s_1, s_2, \ldots \rangle \in \Omega \mid s_k \in F \text{ for all } k \geq 0 \}$.
– *Büchi and coBüchi objectives.* Given a set $B \subseteq S$ of "Büchi" states, the Büchi objective requires that B is visited infinitely often. Formally, the set of winning plays is $\text{Büchi}(B) = \{ \omega \in \Omega \mid \mathrm{Inf}(\omega) \cap B \neq \emptyset \}$. Given $C \subseteq S$, the coBüchi objective requires that all states visited infinitely often are in C. Formally, the set of winning plays is $\text{coBüchi}(C) = \{ \omega \in \Omega \mid \mathrm{Inf}(\omega) \subseteq C \}$.

Sure, almost-sure, positive winning and optimality. Given a player-1 objective Φ, a strategy $\sigma \in \Sigma$ is *sure winning* for player 1 from a state $s \in S$ if for every strategy $\pi \in \Pi$ for player 2, we have $\mathrm{Outcome}(s, \sigma, \pi) \subseteq \Phi$. A strategy σ is *almost-sure winning* for player 1 from the state s for the objective Φ if for every player-2 strategy π, we have $\mathrm{Pr}_s^{\sigma,\pi}(\Phi) = 1$. A strategy σ is *positive winning* for player 1 from the state s for the objective Φ if for every player-2 strategy π, we have $\mathrm{Pr}_s^{\sigma,\pi}(\Phi) > 0$. The sure, almost-sure and positive winning strategies for player 2 are defined analogously. Given an objective Φ, the *sure winning set* $\langle\!\langle 1 \rangle\!\rangle_{sure}(\Phi)$ for player 1 is the set of states from which player 1 has a sure winning strategy. Similarly, the *almost-sure winning set* $\langle\!\langle 1 \rangle\!\rangle_{almost}(\Phi)$ and the *positive winning set* $\langle\!\langle 1 \rangle\!\rangle_{pos}(\Phi)$ for player 1 is the set of states from which player 1 has an almost-sure winning and a positive winning strategy, respectively. The sure winning set $\langle\!\langle 2 \rangle\!\rangle_{sure}(\Omega \setminus \Phi)$, the almost-sure winning set $\langle\!\langle 2 \rangle\!\rangle_{almost}(\Omega \setminus \Phi)$ and the positive winning set $\langle\!\langle 2 \rangle\!\rangle_{pos}(\Omega \setminus \Phi)$ for player 2 are defined analogously. It follows from the definitions that for all $2^1\!/\!_2$-player game graphs and all objectives Φ, we have $\langle\!\langle 1 \rangle\!\rangle_{sure}(\Phi) \subseteq \langle\!\langle 1 \rangle\!\rangle_{almost}(\Phi) \subseteq \langle\!\langle 1 \rangle\!\rangle_{pos}(\Phi)$. Computing sure, almost-sure and positive winning sets and strategies is referred to as the *qualitative* analysis of $2^1\!/\!_2$-player games [8].

Given ω-regular objectives $\Phi \subseteq \Omega$ for player 1 and $\Omega \setminus \Phi$ for player 2, we define the *value* functions $\langle\langle 1 \rangle\rangle_{val}$ and $\langle\langle 2 \rangle\rangle_{val}$ for the players 1 and 2, respectively, as the following functions from the state space S to the interval $[0, 1]$ of reals: for all states $s \in S$, let $\langle\langle 1 \rangle\rangle_{val}(\Phi)(s) = \sup_{\sigma \in \Sigma} \inf_{\pi \in \Pi} \Pr_s^{\sigma,\pi}(\Phi)$ and $\langle\langle 2 \rangle\rangle_{val}(\Omega \setminus \Phi)(s) = \sup_{\pi \in \Pi} \inf_{\sigma \in \Sigma} \Pr_s^{\sigma,\pi}(\Omega \setminus \Phi)$. In other words, the value $\langle\langle 1 \rangle\rangle_{val}(\Phi)(s)$ gives the maximal probability with which player 1 can achieve her objective Φ from state s, and analogously for player 2. The strategies that achieve the value are called optimal: a strategy σ for player 1 is *optimal* from the state s for the objective Φ if $\langle\langle 1 \rangle\rangle_{val}(\Phi)(s) = \inf_{\pi \in \Pi} \Pr_s^{\sigma,\pi}(\Phi)$. The optimal strategies for player 2 are defined analogously. Computing values and optimal strategies is referred to as the *quantitative* analysis of $2^1\!/\!_2$-player games. The set of states with value 1 is called the *limit-sure winning set* [8]. For $2^1\!/\!_2$-player game graphs with ω-regular objectives the almost-sure and limit-sure winning sets coincide [4].

Let $\mathcal{C} \in \{P, M, F, PM, PF\}$ and consider the family $\Sigma^{\mathcal{C}} \subseteq \Sigma$ of special strategies for player 1. We say that the family $\Sigma^{\mathcal{C}}$ *suffices* with respect to a player-1 objective Φ on a class $\mathcal{G}$ of game graphs for *sure winning* if for every game graph $G \in \mathcal{G}$ and state $s \in \langle\langle 1 \rangle\rangle_{sure}(\Phi)$, there is a player-1 strategy $\sigma \in \Sigma^{\mathcal{C}}$ such that for every player-2 strategy $\pi \in \Pi$, we have $\text{Outcome}(s, \sigma, \pi) \subseteq \Phi$. Similarly, the family $\Sigma^{\mathcal{C}}$ *suffices* with respect to the objective Φ on the class $\mathcal{G}$ of game graphs for (a) *almost-sure winning* if for every game graph $G \in \mathcal{G}$ and state $s \in \langle\langle 1 \rangle\rangle_{almost}(\Phi)$, there is a player-1 strategy $\sigma \in \Sigma^{\mathcal{C}}$ such that for every player-2 strategy $\pi \in \Pi$, we have $\Pr_s^{\sigma,\pi}(\Phi) = 1$; (b) *positive winning* if for every game graph $G \in \mathcal{G}$ and state $s \in \langle\langle 1 \rangle\rangle_{pos}(\Phi)$, there is a player-1 strategy $\sigma \in \Sigma^{\mathcal{C}}$ such that for every player-2 strategy $\pi \in \Pi$, we have $\Pr_s^{\sigma,\pi}(\Phi) > 0$; and (c) *optimality* if for every game graph $G \in \mathcal{G}$ and state $s \in S$, there is a player-1 strategy $\sigma \in \Sigma^{\mathcal{C}}$ such that $\langle\langle 1 \rangle\rangle_{val}(\Phi)(s) = \inf_{\pi \in \Pi} \Pr_s^{\sigma,\pi}(\Phi)$. The notion of sufficiency for size of finite-memory strategies is obtained by referring to the size of the memory $\mathtt{M}$ of the strategies. The notions of sufficiency of strategies for player 2 is defined analogously.

Determinacy. For sure winning, the $1^1\!/\!_2$-player and $2^1\!/\!_2$-player games coincide with 2-player (deterministic) games where the random player (who chooses the successor at the probabilistic states) is interpreted as an adversary, i.e., as player 2. Theorem 1 and Theorem 2 state the classical determinacy results for 2-player and $2^1\!/\!_2$-player game graphs with Müller objectives. It follows from Theorem 2 that for all Müller objectives Φ, for all $\varepsilon > 0$, there exists an ε-optimal strategy σ_ε for player 1 such that for all π and all $s \in S$ we have $\Pr_s^{\sigma,\pi}(\Phi) \geq \langle\langle 1 \rangle\rangle_{val}(\Phi)(s) - \varepsilon$.

Theorem 1 (Qualitative determinacy [10]). *For all 2-player game graphs and Müller objectives Φ, we have $\langle\langle 1 \rangle\rangle_{sure}(\Phi) \cap \langle\langle 2 \rangle\rangle_{sure}(\Omega \setminus \Phi) = \emptyset$ and $\langle\langle 1 \rangle\rangle_{sure}(\Phi) \cup \langle\langle 2 \rangle\rangle_{sure}(\Omega \setminus \Phi) = S$. Moreover, on 2-player game graphs, the family of pure finite-memory strategies suffices for sure winning with respect to Müller objectives.*

Theorem 2 (Quantitative determinacy [13]). *For all $2^1\!/\!_2$-player game graphs, for all Müller winning conditions $\mathcal{F} \subseteq \mathcal{P}(C)$, and all states s, we have $\langle\langle 1 \rangle\rangle_{val}(\text{Müller}(\mathcal{F}))(s) + \langle\langle 2 \rangle\rangle_{val}(\Omega \setminus \text{Müller}(\mathcal{F}))(s) = 1$.*

3 Memory Bound for Qualitative Winning Strategies

In this section we present optimal memory bounds for pure strategies with respect to qualitative (almost-sure and positive) winning for 2½-player game graphs with Müller winning conditions. The result is obtained by a generalization of the result of [7] and depends on the novel constructions of Zielonka [18] for 2-player games. In [7] the authors use an insightful analysis of Zielonka's construction to present an upper bound and a matching lower bound on memory of sure winning strategies in 2-player games with Müller objectives. In this section we generalize the result of [7] to show that the same upper bound holds for qualitative winning strategies in 2½-player games with Müller objectives. We now introduce some notations and the Zielonka tree of a Müller condition.

Notation. Let $\mathcal{F} \subseteq \mathcal{P}(C)$ be a winning condition. For $D \subseteq C$ we define $(\mathcal{F} \upharpoonright D) \subseteq \mathcal{P}(D)$ as the set $\{ D' \in \mathcal{F} \mid D' \subseteq D \}$. For a Müller condition $\mathcal{F} \subseteq \mathcal{P}(C)$ we denote by $\overline{\mathcal{F}}$ the complementary condition, i.e., $\overline{\mathcal{F}} = \mathcal{P}(C) \setminus \mathcal{F}$. Similarly for an objective Φ we denote by $\overline{\Phi}$ the complementary objective, i.e., $\overline{\Phi} = \Omega \setminus \Phi$.

Definition 1 (Zielonka tree of a winning condition [18]). *The* Zielonka tree *of a winning condition $\mathcal{F} \subseteq \mathcal{P}(C)$, denoted $\mathcal{Z}_{\mathcal{F},C}$, is defined inductively as follows: (a) if $C \notin \mathcal{F}$, then $\mathcal{Z}_{\mathcal{F},C} = \mathcal{Z}_{\overline{\mathcal{F}},C}$, where $\overline{\mathcal{F}} = \mathcal{P}(C) \setminus \mathcal{F}$; and (b) if $C \in \mathcal{F}$, then the root of $\mathcal{Z}_{\mathcal{F},C}$ is labeled with C. Let $C_0, C_1, \ldots, C_{k-1}$ be all the maximal sets in $\{ X \notin \mathcal{F} \mid X \subseteq C \}$. Then we attach to the root, as its subtrees, the Zielonka trees of $\mathcal{F} \upharpoonright C_i$, i.e., $\mathcal{Z}_{\mathcal{F}\upharpoonright C_i, C_i}$, for $i = 0, 1, \ldots, k-1$. Hence the Zielonka tree is a tree with nodes labeled by sets of colors. A node of $\mathcal{Z}_{\mathcal{F},C}$ is a 0-level node if it is labeled with a set from $\mathcal{F}$, otherwise it is a 1-level node. In the sequel we write $\mathcal{Z}_{\mathcal{F}}$ to denote $\mathcal{Z}_{\mathcal{F},C}$ if C is clear from the context.* ∎

Definition 2 (The number $m_{\mathcal{F}}$ of Zielonka tree). *Let $\mathcal{F} \subseteq \mathcal{P}(C)$ be a winning condition and $\mathcal{Z}_{\mathcal{F}_0,C_0}, \mathcal{Z}_{\mathcal{F}_1,C_1}, \ldots, \mathcal{Z}_{\mathcal{F}_{k-1},C_{k-1}}$ be the subtrees attached to the root of the tree $\mathcal{Z}_{\mathcal{F},C}$, where $\mathcal{F}_i = \mathcal{F} \upharpoonright C_i \subseteq \mathcal{P}(C_i)$ for $i = 0, 1, \ldots, k-1$. We define the number $m_{\mathcal{F}}$ inductively as follows*

$$m_{\mathcal{F}} = \begin{cases} 1 & \textit{if } \mathcal{Z}_{\mathcal{F},C} \textit{ does not have any subtrees,} \\ \max\{ m_{\mathcal{F}_0,}, m_{\mathcal{F}_1}, \ldots, m_{\mathcal{F}_{k-1}} \} & \textit{if } C \notin \mathcal{F}, (1\textit{-level node}) \\ \sum_{i=1}^{k-1} m_{\mathcal{F}_i} & \textit{if } C \in \mathcal{F}, (0\textit{-level node}). \end{cases}$$ ∎

Our goal is to show that for winning conditions $\mathcal{F}$ pure finite-memory qualitative winning strategies of size $m_{\mathcal{F}}$ exist in 2½-player games. This proves the upper bound. The results of [7] already established the matching lower bound for 2-player games. This establishes the optimal bound of memory of qualitative winning strategies for 2½-player games. We start with the key notion of *attractors* that will be crucial in our proofs.

Definition 3 (Attractors). *Given a 2½-player game graph G and a set $U \subseteq S$ of states, such that $G \upharpoonright U$ is a subgame, and $T \subseteq S$ we define $\mathsf{Attr}_{1,\bigcirc}(T, U)$ as follows: $T_0 = T \cap U$ and for $j \geq 0$ we define T_{j+1} from T_j as follows*

$$T_{j+1} = T_j \cup \{ s \in (S_1 \cup S_\bigcirc) \cap U \mid E(s) \cap T_j \neq \emptyset \} \cup \{ s \in S_2 \cap U \mid E(s) \cap U \subseteq T_j \}.$$

and $A = \mathsf{Attr}_{1,\bigcirc}(T,U) = \bigcup_{j\geq 0} T_j$. *We obtain* $\mathsf{Attr}_{2,\bigcirc}(T,U)$ *by exchanging the roles of player 1 and player 2. A pure memoryless attractor strategy* $\sigma^A : (A \setminus T) \cap S_1 \to S$ *for player 1 on* A *to* T *is as follows: for* $i > 0$ *and a state* $s \in (T_i \setminus T_{i-1}) \cap S_1$, *the strategy* $\sigma^A(s) \in T_{i-1}$ *chooses a successor in* T_{i-1} *(which exists by definition).* ∎

Lemma 1 (Attractor properties). *Let* G *be a* $2\frac{1}{2}$*-player game graph and* $U \subseteq S$ *be a set of states such that* $G \restriction U$ *is a subgame. For a set* $T \subseteq S$ *of states, let* $Z = \mathsf{Attr}_{1,\bigcirc}(T,U)$. *Then the following assertions hold.*

1. $G \restriction (U \setminus Z)$ *is a subgame.*
2. *Let* σ^Z *be a pure memoryless attractor strategy for player 1. For all strategies* π *for player 2 in the subgame* $G \restriction U$ *and for all states* $s \in U$ *we have*
   (a) if $\mathrm{Pr}_s^{\sigma^Z,\pi}(\mathit{Reach}(Z)) > 0$, *then* $\mathrm{Pr}_s^{\sigma^Z,\pi}(\mathit{Reach}(T)) > 0$*; and*
   (b) if $\mathrm{Pr}_s^{\sigma^Z,\pi}(\mathit{Büchi}(Z)) > 0$, *then* $\mathrm{Pr}_s^{\sigma^Z,\pi}(\mathit{Büchi}(T) \mid \mathit{Büchi}(Z)) = 1$.

We now present the main result of this section (upper bound on memory for qualitative winning strategies).

Theorem 3 (Qualitative forgetful determinacy). *Let* $(G, C, \chi, \mathcal{F})$ *be a* $2\frac{1}{2}$*-player game with Müller winning condition* $\mathcal{F}$ *for player 1. Let* $\Phi = \mathit{Müller}(\mathcal{F})$, *and consider the following sets*

$$W_1^{>0} = \langle\langle 1\rangle\rangle_{pos}(\Phi);\quad W_1 = \langle\langle 1\rangle\rangle_{almost}(\Phi);\quad W_2^{>0} = \langle\langle 2\rangle\rangle_{pos}(\overline{\Phi});\quad W_2 = \langle\langle 2\rangle\rangle_{almost}(\overline{\Phi}).$$

The following assertions hold.

1. *We have (a)* $W_1^{>0} \cup W_2 = S$ *and* $W_1^{>0} \cap W_2 = \emptyset$*; and (b)* $W_2^{>0} \cup W_1 = S$ *and* $W_2^{>0} \cap W_1 = \emptyset$.
2. *(a) Player 1 has a pure strategy* σ *with memory of size* $m_{\mathcal{F}}$ *such that for all states* $s \in W_1^{>0}$ *and for all strategies* π *for player 2 we have* $\mathrm{Pr}_s^{\sigma,\pi}(\Phi) > 0$*; and (b) player 2 has a pure strategy* π *with memory of size* $m_{\overline{\mathcal{F}}}$ *such that for all states* $s \in W_2$ *and for all strategies* σ *for player 1 we have* $\mathrm{Pr}_s^{\sigma,\pi}(\overline{\Phi}) = 1$.
3. *(a) Player 1 has a pure strategy* σ *with memory of size* $m_{\mathcal{F}}$ *such that for all states* $s \in W_1$ *and for all strategies* π *for player 2 we have* $\mathrm{Pr}_s^{\sigma,\pi}(\Phi) = 1$*; and (b) player 2 has a pure strategy* π *with memory of size* $m_{\overline{\mathcal{F}}}$ *such that for all states* $s \in W_2^{>0}$ *and for all strategies* σ *for player 1 we have* $\mathrm{Pr}_s^{\sigma,\pi}(\overline{\Phi}) > 0$.

Proof. The first part of the result is a consequence of Theorem 2. We will concentrate on the proof for the result for part 2. The last part (part 3) follows from a symmetric argument.

The proof goes by induction on the structure of the Zielonka tree $\mathcal{Z}_{\mathcal{F},C}$ of the winning condition $\mathcal{F}$. We assume that $C \notin \mathcal{F}$. The case when $C \in \mathcal{F}$ can be proved by a similar argument: if $C \in \mathcal{F}$, then we consider $\widehat{c} \notin C$ and consider the winning condition $\widehat{\mathcal{F}} = \mathcal{F} \subseteq \mathcal{P}(C \cup \{\widehat{c}\})$ with $C \cup \{\widehat{c}\} \notin \widehat{\mathcal{F}}$. Hence we consider, without loss of generality, that $C \notin \mathcal{F}$ and let $C_0, C_1, \ldots, C_{k-1}$ be the label of the subtrees attached to the root C, i.e., $C_0, C_1, \ldots, C_{k-1}$ are maximal subset of colors that appear in $\mathcal{F}$. We will define by induction a non-decreasing sequence of sets $(U_j)_{j\geq 0}$ as follows. Let $U_0 = \emptyset$ and for $j > 0$ we define U_j below:

1. $A_j = \mathsf{Attr}_{1,\bigcirc}(U_{j-1}, S)$ and $X_j = S \setminus A_j$;
2. $D_j = C \setminus C_{j \bmod k}$ and $Y_j = X_j \setminus \mathsf{Attr}_{2,\bigcirc}(\chi^{-1}(D_j), X_j)$;
3. let Z_j be the set of positive winning states for player 1 in $(G \upharpoonright Y_j, C_{j \bmod k}, \chi, \mathcal{F} \upharpoonright C_{j \bmod k})$, (i.e., $Z_j = \langle\langle 1 \rangle\rangle_{pos}(\text{Müller}(\mathcal{F} \upharpoonright C_{j \bmod k}))$ in $G \upharpoonright Y_j$); hence $(Y_j \setminus Z_j)$ is almost-sure winning for player 2 in the subgame; and
4. $U_j = A_j \cup Z_j$.

We start with an observation.

Observation 1. For all $s \in S_2 \cap Z_j$, we have $E(s) \subseteq Z_j \cup A_j$. This follows from the following case analysis.

- Since Y_j is a complement of an attractor set $\mathsf{Attr}_{2,\bigcirc}(\chi^{-1}(D_j), X_j)$, it follows that for all states $s \in S_2 \cap Y_j$ we have $E(s) \cap X_j \subseteq Y_j$. It follows that $E(s) \subseteq Y_j \cup A_j$.
- Since player 2 can win almost-surely from the set $Y_j \setminus Z_j$, if a state $s \in Y_j \cap S_2$ has an edge to $Y_j \setminus Z_j$, then $s \in Y_j \setminus Z_j$. Hence for $s \in S_2 \cap Z_j$ we have $E(s) \cap (Y_j \setminus Z_j) = \emptyset$.

We will denote by $\mathcal{F}_i$ the winning condition $\mathcal{F} \upharpoonright C_i$, for $i = 0, 1, \ldots, k-1$, and $\overline{\mathcal{F}}_i = \mathcal{P}(C_i) \setminus \mathcal{F}_i$. By induction hypothesis on $\mathcal{F}_i = \mathcal{F} \upharpoonright C_{j \bmod k}$, player 1 has a pure positive winning strategy of size $m_{\mathcal{F}_i}$ from Z_j and player 2 has a pure almost-sure winning strategy of size $m_{\overline{\mathcal{F}}_i}$ from $Y_j \setminus Z_j$. Let $W = \bigcup_{j \geq 0} U_j$. We will show in Lemma 2 that player 1 has a pure positive winning strategy of size $m_{\mathcal{F}}$ from W; and then in Lemma 3 we will show that player 2 has a pure almost-sure winning strategy of size $m_{\overline{\mathcal{F}}}$ from $S \setminus W$. This completes the proof. We now prove the Lemmas 2 and 3. ∎

Lemma 2. *Player 1 has a pure positive winning strategy of size $m_{\mathcal{F}}$ from W.*

Proof. By induction hypothesis on U_{j-1} player 1 has a pure positive winning strategy σ_{j-1}^U of size $m_{\mathcal{F}}$ from U_{j-1}. From the set $A_j = \mathsf{Attr}_{1,\bigcirc}(U_{j-1}, S)$, player 1 has a pure memoryless attractor strategy σ_j^A to bring the game to U_{j-1} with positive probability (Lemma 1(part 2.(a))), and then use σ_{j-1}^U and ensure winning with positive probability from the set A_j. Let σ_j^Z be the pure positive winning strategy for player 1 in Z_j of size $m_{\mathcal{F}_i}$, where $i = j \mod k$. We now show the combination of strategies σ_{j-1}^U, σ_j^A and σ_j^Z ensure positive probability winning for player 1 from U_j. If the play starts at a state $s \in Z_j$, then player 1 follows σ_j^Z. If the play stays in Y_j for ever, then the strategy σ_j^Z ensures that player 1 wins with positive probability. By observation 1 of Theorem 3, for all states $s \in Y_j \cap S_2$, we have $E(s) \subseteq Y_j \cup A_j$. Hence if the play leaves Y_j, then player 2 must chose an edge to A_j. In A_j player 1 can use the attractor strategy σ_j^A followed by σ_{j-1}^U to ensure positive probability win. Hence if the play is in Y_j for ever with probability 1, then σ_j^Z ensures positive probability win, and if the play reaches A_j with positive probability, then σ_j^A followed by σ_{j-1}^U ensures positive probability win.

We now formally present σ_j^U defined on U_j. Let $\sigma_j^Z = (\sigma_{j,u}^Z, \sigma_{j,m}^Z)$ be the strategy obtained from inductive hypothesis; defined on Z_j (i.e., arbitrary elsewhere) of size $m_{\mathcal{F}_i}$, where $i = j \mod k$, and ensure winning with positive probability on Z_j. Let $\sigma^Z_{j,u}$ be the memory-update function and $\sigma^Z_{j,m}$ be the next-move function of σ^Z_j. We assume the memory $M_{\mathcal{F}_i}$ of σ^Z_j to be the set $\{1, 2, \ldots, m_{\mathcal{F}_i}\}$. The strategy $\sigma^A_j : (A_j \setminus U_{j-1}) \cap S_1 \to A_j$ is a pure memoryless attractor strategy on A_j to U_{j-1}. The strategy σ^U_j is as follows: the memory-update function and the next-move function is

$$\sigma^U_{j,u}(s,m) = \begin{cases} \sigma^U_{j-1,u}(s,m) & s \in U_{j-1} \\ \sigma^Z_{j-1,u}(s,m) & s \in Z_j, m \in M_{\mathcal{F}_i} \\ 1 & \text{otherwise.} \end{cases}$$

$$\sigma^U_{j,m}(s,m) = \begin{cases} \sigma^U_{j-1,m}(s,m) & s \in U_{j-1} \cap S_1 \\ \sigma^Z_{j-1,m}(s,m) & s \in Z_j \cap S_1, m \in M_{\mathcal{F}_i} \\ \sigma^Z_{j-1,m}(s,1) & s \in Z_j \cap S_1, m \notin M_{\mathcal{F}_i} \\ \sigma^A_j(s) & s \in (A_j \setminus U_{j-1}) \cap S_1. \end{cases}$$

The strategy σ^U_j formally defines the strategy described and proves the result. ∎

Lemma 3. *Player 2 has a pure almost-sure winning strategy of size $m_{\overline{\mathcal{F}}}$ from $S \setminus W$.*

Proof. Let $\ell \in \mathbb{N}$ be such that $\ell \mod k = 0$ and $W = U_{\ell-1} = U_\ell = U_{\ell+1} = \cdots = U_{\ell+k-1}$. From the equality $W = U_{\ell-1} = U_\ell$ we have $\mathsf{Attr}_{1,\bigcirc}(W,S) = W$. Let us denote by $\overline{W} = S \setminus W$. Hence $G \upharpoonright \overline{W}$ is a subgame (by Lemma 1), and also for all $s \in \overline{W} \cap (S_1 \cup S_\bigcirc)$ we have $E(s) \subseteq \overline{W}$. The equality $U_{\ell+i-1} = U_{\ell+i}$ implies that $Z_{\ell+i} = \emptyset$. Hence for all $i = 0, 1, \ldots, k-1$, we have $Z_{\ell+i} = \emptyset$. By inductive hypothesis for all $i = 0, 1, \ldots, k-1$, player 2 has a pure almost-sure winning strategy π^i of size $m_{\overline{\mathcal{F}}_i}$ in the game $(G \upharpoonright Y_{\ell+i}, C_i, \chi, \mathcal{F} \upharpoonright C_i)$.

We now describe the construction of a pure almost-sure winning strategy π^* for player 2 in $\overline{W}$. For $D_i = C \setminus C_i$ we denote by $\widehat{D}_i = \chi^{-1}(D_i)$ the set of states with colors D_i. If the play starts in a state in $Y_{\ell+i}$, for $i = 0, 1, \ldots, k-1$, then player 2 uses the almost-sure winning strategy π^i. If the play leaves $Y_{\ell+i}$, then the play must reach $\overline{W} \setminus Y_{\ell+i} = \mathsf{Attr}_{2,\bigcirc}(\widehat{D}_i, \overline{W})$, since player 1 and random states do not have edges to W. In $\mathsf{Attr}_{2,\bigcirc}(\widehat{D}_i, \overline{W})$, player 2 plays a pure memoryless attractor strategy to reach the set $\widehat{D}_i$ with positive probability. If the set $\widehat{D}_i$ is reached, then a state in $Y_{(\ell+i+1) \mod k}$ or in $\mathsf{Attr}_{2,\bigcirc}\big(\widehat{D}_{(i+1) \mod k}, \overline{W}\big)$ is reached. If $Y_{(\ell+i+1) \mod k}$ is reached $\pi^{(i+1) \mod k}$ is followed, and otherwise the pure memoryless attractor strategy to reach the set $\widehat{D}_{(i+1) \mod k}$ with positive probability is followed. Of course, the play may leave $Y_{(\ell+i+1) \mod k}$, and reach $Y_{(\ell+i+2) \mod k}$, and then we would repeat the reasoning, and so on. Let us analyze various cases to prove that π^* is almost-sure winning for player 2.

1. If the play finally settles in some $Y_{\ell+i}$, for $i = 0, 1, \ldots, k-1$, then from this moment player 2 follows π^i and ensures that the objective $\overline{\Phi}$ is satisfied with probability 1. Formally, for all states $s \in \overline{W}$, for all strategies σ for player 1

we have $\Pr_s^{\sigma,\pi^*}(\overline{\Phi} \mid \text{coBüchi}(Y_{\ell+i})) = 1$. This holds for all $i = 0, 1, \ldots, k-1$ and hence for all states $s \in \overline{W}$, for all strategies σ for player 1 we have $\Pr_s^{\sigma,\pi^*}(\overline{\Phi} \mid \bigcup_{0 \leq i \leq k-1} \text{coBüchi}(Y_{\ell+i})) = 1$.

2. Otherwise, for all $i = 0, 1, \ldots, k-1$, the set $\overline{W} \setminus Y_{\ell+i} = \mathsf{Attr}_{2,\bigcirc}(\widehat{D}_i, \overline{W})$ is visited infinitely often. By Lemma 1, given $\mathsf{Attr}_{2,\bigcirc}(\widehat{D}_i, \overline{W})$ is visited infinitely often, then the attractor strategy ensures that the set $\widehat{D}_i$ is visited infinitely often with probability 1. Formally, for all states $s \in \overline{W}$, for all strategies σ for player 1, for all $i = 0, 1, \ldots, k-1$, we have $\Pr_s^{\sigma,\pi^*}(\text{Büchi}(\widehat{D}_i) \mid \text{Büchi}(\overline{W} \setminus Y_{\ell+i})) = 1$; and also $\Pr_s^{\sigma,\pi^*}(\text{Büchi}(\widehat{D}_i) \mid \bigcap_{0 \leq i \leq k-1} \text{Büchi}(\overline{W} \setminus Y_{\ell+i})) = 1$. It follows that for all states $s \in \overline{W}$, for all strategies σ for player 1 we have $\Pr_s^{\sigma,\pi^*}(\bigcap_{0 \leq i \leq k-1} \text{Büchi}(\widehat{D}_i) \mid \bigcap_{0 \leq i \leq k-1} \text{Büchi}(\overline{W} \setminus Y_{\ell+i})) = 1$. Hence the play visits states with colors not in C_i with probability 1. Hence the set of colors visited infinitely often is not contained in any C_i. Since $C_0, C_1, \ldots, C_{k-1}$ are all the maximal subsets of $\mathcal{F}$, we have the set of colors visited infinitely often is not in $\mathcal{F}$ with probability 1, and hence player 2 wins almost-surely.

Hence it follows that for all strategies σ and for all states $s \in (S \setminus W)$ we have $\Pr_s^{\sigma,\pi^*}(\overline{\Phi}) = 1$. To complete the proof we present precise description of the strategy π^* with memory of size $m_{\overline{\mathcal{F}}}$. Let $\pi^i = (\pi_u^i, \pi_m^i)$ be an almost-sure winning strategy for player 2 for the subgame on $Y_{\ell+i}$ with memory $M_{\overline{\mathcal{F}}_i}$. By definition we have $m_{\overline{\mathcal{F}}} = \sum_{i=0}^{k-1} m_{\overline{\mathcal{F}}_i}$. Let $M_{\overline{\mathcal{F}}} = \bigcup_{i=0}^{k-1} (M_{\overline{\mathcal{F}}_i} \times \{ i \})$. This set is not exactly the set $\{ 1, 2, \ldots, m_{\mathcal{F}} \}$, but has the same cardinality (which suffices for our purpose). We define the strategy π^* as follows:

$$\pi_u^*(s, (m, i)) = \begin{cases} \pi_u^i(s, (m, i)) & s \in Y_{\ell+i} \\ (1, i+1 \mod k) & \text{otherwise.} \end{cases}$$

$$\pi_m^*(s, (m, i)) = \begin{cases} \pi_m^i(s, (m, i)) & s \in Y_{\ell+i} \\ \pi^{L_i}(s) & s \in L_i \setminus \widehat{D}_i \\ s_i & s \in \widehat{D}_i, s_i \in E(s) \cap \overline{W}. \end{cases}$$

where $L_i = \mathsf{Attr}_{2,\bigcirc}(\widehat{D}_i, \overline{W})$; π^{L_i} is a pure memoryless attractor strategy on L_i to $\widehat{D}_i$, and s_i is a successor state of s in $\overline{W}$ (such a state exists since $\overline{W}$ induces a subgame). This formally represents π^* and the size of π^* satisfies the required bound. Observe that the disjoint sum of all $M_{\overline{\mathcal{F}}_i}$ was required since $Y_\ell, Y_{\ell+1}, \ldots, Y_{\ell+k-1}$ may not be disjoint and the strategy π^* need to know which Y_j the play is in. ∎

Lower bound. In [7] the authors show a matching lower bound for sure winning strategies in 2-player games. In 2-player games any pure almost-sure winning or any pure positive winning strategy is also a sure winning strategy. This observation along with the result of [7] gives us the following result.

Theorem 4 (Lower bound [7]). *For all Müller winning conditions $\mathcal{F} \subseteq \mathcal{P}(C)$, there is a 2-player game $(G, C, \chi, \mathcal{F})$ (with a 2-player game graph G) such* *that every pure almost-sure and positive winning strategy for player 1 requires memory of size at least $m_{\mathcal{F}}$; and every pure almost-sure and positive winning strategy for player 2 requires memory of size at least $m_{\overline{\mathcal{F}}}$.*

4 Memory Bound for Optimal Strategies

In this section we extend the sufficiency results for families of strategies from almost-sure winning to optimality with respect to all Müller objectives. In the following, we fix a $2^1\!/\!_2$-player game graph G. We first present some definitions.

Definition 4 (Value classes). *Given a Müller objective Φ, for every real $r \in [0, 1]$ the* value class *with value r is $\mathrm{VC}(\Phi, r) = \{ s \in S \mid \langle\!\langle 1 \rangle\!\rangle_{val}(\Phi)(s) = r \}$ is the set of states with value r for player 1. For $r \in [0, 1]$ we denote by $\mathrm{VC}(\Phi, > r) = \bigcup_{q>r} \mathrm{VC}(\Phi, q)$ the value classes greater than r and by $\mathrm{VC}(\Phi, < r) = \bigcup_{q<r} \mathrm{VC}(\Phi, q)$ the value classes smaller than r.* ∎

Definition 5 (Boundary probabilistic states). *Given a value class $\mathrm{VC}(\Phi, r)$, a state $s \in \mathrm{VC}(\Phi, r) \cap S_{\bigcirc}$ is a* boundary probabilistic state *if $E(s) \cap (S \setminus \mathrm{VC}(\Phi, r)) \neq \emptyset$, i.e., the probabilistic state has an edge out of the value class. For a value class $\mathrm{VC}(\Phi, r)$ we denote by $\mathsf{Bnd}(\Phi, r)$ the set of boundary probabilistic states of value class r.* ∎

Observation. For a state $s \in \mathsf{Bnd}(\Phi, r)$ we have $E(s) \cap \mathrm{VC}(\Phi, > r) \neq \emptyset$ and $E(s) \cap \mathrm{VC}(\Phi, < r) \neq \emptyset$, i.e., the boundary probabilistic states have edges to higher and lower value classes. It follows that for all Müller objectives Φ we have $\mathsf{Bnd}(\Phi, 1) = \emptyset$ and $\mathsf{Bnd}(\Phi, 0) = \emptyset$.

Reduction of a value class. Given a value class $\mathrm{VC}(\Phi, r)$, let $\mathsf{Bnd}(\Phi, r)$ be the set of boundary probabilistic states in $\mathrm{VC}(\Phi, r)$. We denote by $G_{\mathsf{Bnd}(\Phi,r)}$ the subgame where every boundary probabilistic state in $\mathsf{Bnd}(\Phi, r)$ is converted to an absorbing state (state with a self-loop). We denote by $G_{\Phi,r} = G_{\mathsf{Bnd}(\Phi,r)} \upharpoonright \mathrm{VC}(\Phi, r)$: this is a subgame since every value class is δ-live, and δ-closed as all states in $\mathsf{Bnd}(\Phi, r)$ are converted to absorbing states.

Lemma 4 (Almost-sure reduction). *Let G be a $2^1\!/\!_2$-player game graph and $\mathcal{F} \subseteq \mathcal{P}(C)$ be a Müller winning condition. Let $\Phi = \mathit{Müller}(\mathcal{F})$. For $0 < r < 1$, the following assertions hold.*

1. *Player 1 wins almost-surely for objective $\Phi \cup \mathit{Reach}(\mathsf{Bnd}(\Phi, r))$ from all states in $G_{\Phi,r}$, i.e., $\langle\!\langle 1 \rangle\!\rangle_{almost}(\Phi \cup \mathit{Reach}(\mathsf{Bnd}(\Phi, r))) = \mathrm{VC}(\Phi, r)$ in $G_{\Phi,r}$.*
2. *Player 2 wins almost-surely for objective $\overline{\Phi} \cup \mathit{Reach}(\mathsf{Bnd}(\Phi, r))$ from all states in $G_{\Phi,r}$, i.e., $\langle\!\langle 2 \rangle\!\rangle_{almost}(\overline{\Phi} \cup \mathit{Reach}(\mathsf{Bnd}(\Phi, r))) = \mathrm{VC}(\Phi, r)$ in $G_{\Phi,r}$.*

Lemma 5 (Almost-sure to optimality [4]). *Let G be a $2^1\!/\!_2$-player game graph and $\mathcal{F} \subseteq \mathcal{P}(C)$ be a Müller winning condition. Let $\Phi = \mathit{Müller}(\mathcal{F})$. Let σ be a strategy such that (a) σ is an almost-sure winning strategy from the almost-sure winning states ($\langle\!\langle 1 \rangle\!\rangle_{almost}(\Phi)$ in G); and (b) σ is an almost-sure winning strategy for objective $\Phi \cup \mathit{Reach}(\mathsf{Bnd}(\Phi, r))$ in the game $G_{\Phi,r}$, for all $0 < r < 1$. Then σ is an optimal strategy.*

Müller reduction for $G_{\Phi,r}$. Given a Müller winning condition $\mathcal{F}$ and the objective $\Phi = \text{Müller}(\mathcal{F})$, we consider the game $G_{\Phi,r}$ with the objective $\Phi \cup \text{Reach}(\mathsf{Bnd}(\Phi, r))$ for player 1. We present a simple reduction to a game with objective Φ. The reduction is achieved as follows: without loss of generality we assume $\mathcal{F} \neq \emptyset$, and let $F \in \mathcal{F}$ and $F = \{ c_1^F, c_2^F, \ldots, c_f^F \}$. We construct a game graph $\widetilde{G}_{\Phi,r}$ with objective Φ for player 1 as follows: convert every state $s_j \in \mathsf{Bnd}(\Phi, r)$ to a cycle $U_j = \{ s_1^j, s_2^j, \ldots, s_f^j \}$ with $\chi(s_i^j) = c_i^F$, i.e., once s_j is reached the cycle U_j is repeated with $\chi(U_j) \in \mathcal{F}$. An almost-sure winning strategy in $G_{\Phi,r}$ with objective $\Phi \cup \text{Reach}(\mathsf{Bnd}(\Phi, r))$, is an almost-sure winning strategy in $\widetilde{G}_{\Phi,r}$ with objective Φ; and vice-versa. The present reduction along with Lemma 4 and Lemma 5 gives us Lemma 6. Lemma 6 along with Theorem 3 gives us Theorem 5.

Lemma 6. *For all Müller winning conditions $\mathcal{F}$, the following assertions hold.*

1. *If the family of pure finite-memory strategies of size $\ell_{\mathcal{F}}^P$ suffices for almost-sure winning on $2^1\!/_2$-player game graphs, then the family of pure finite-memory strategies of size $\ell_{\mathcal{F}}^P$ suffices for optimality on $2^1\!/_2$-player game graphs.*
2. *If the family of randomized finite-memory strategies of size $\ell_{\mathcal{F}}^R$ suffices for almost-sure winning on $2^1\!/_2$-player game graphs, then the family of randomized finite-memory strategies of size $\ell_{\mathcal{F}}^R$ suffices for optimality on $2^1\!/_2$-player game graphs.*

Theorem 5. *For all Müller winning conditions $\mathcal{F}$, the family of pure finite-memory strategies of size $m_{\mathcal{F}}$ suffices for optimality on $2^1\!/_2$-player game graphs.*

5 An Improved Bound for Randomized Strategies

We now show that if a player plays randomized strategies, then the upper bound on memory for optimal strategies can be improved. We first present the notions of an upward closed restriction of a Zielonka tree. The number $m_{\mathcal{F}}^U$ of such restrictions of the Zielonka tree will be in general lower than the number $m_{\mathcal{F}}$ of Zielonka trees, and we show that randomized strategies with memory of size $m_{\mathcal{F}}^U$ suffices for optimality.

Upward closed sets. A set $\mathcal{F} \subseteq \mathcal{P}(C)$ is *upward closed* if for all $F \in \mathcal{F}$ and all $F \subseteq F_1$ we have $F_1 \in \mathcal{F}$, i.e., if a set F is in $\mathcal{F}$, then all supersets F_1 of F are in $\mathcal{F}$ as well.

Upward closed restriction of Zielonka tree. The upward closed restriction of a Zielonka tree for a Müller winning condition $\mathcal{F} \subseteq \mathcal{P}(C)$, denoted as $\mathcal{Z}_{\mathcal{F},C}^U$, is obtained by making upward closed conditions as leaves. Formally, we define $\mathcal{Z}_{\mathcal{F},C}^U$ inductively as follows:

1. if $\mathcal{F}$ is upward closed, then $\mathcal{Z}_{\mathcal{F},C}^U$ is leaf labeled $\mathcal{F}$ (i.e., it has no subtrees);
2. otherwise

(a) if $C \notin \mathcal{F}$, then $\mathcal{Z}^U_{\mathcal{F},C} = \mathcal{Z}^U_{\overline{\mathcal{F}},C}$, where $\overline{\mathcal{F}} = \mathcal{P}(C) \setminus \mathcal{F}$.

(b) if $C \in \mathcal{F}$, then the root of $\mathcal{Z}^U_{\mathcal{F},C}$ is labeled with C; and let $C_0, C_1, \ldots, C_{k-1}$ be all the maximal sets in $\{ X \notin \mathcal{F} \mid X \subseteq C \}$; then we attach to the root, as its subtrees, the Zielonka upward closed restricted trees $\mathcal{Z}^U_{\mathcal{F},C}$ of $\mathcal{F} \restriction C_i$, i.e., $\mathcal{Z}^U_{\mathcal{F}\restriction C_i, C_i}$, for $i = 0, 1, \ldots, k-1$.

The number $m^U_{\mathcal{F}}$ for $\mathcal{Z}^U_{\mathcal{F},C}$ is the number defined as the number $m_{\mathcal{F}}$ was defined for the tree $\mathcal{Z}_{\mathcal{F},C}$.

We will prove randomized strategies of size $m^U_{\mathcal{F}}$ suffices for optimality. To prove this result, we first prove that randomized strategies of size $m^U_{\mathcal{F}}$ suffices for almost-sure winning. The result then follows from Lemma 6. To prove the result for almost-sure winning we take a closer look at the proof of Theorem 3. The inductive proof characterizes that if existence of randomized memoryless strategies can be proved for $2^1\!/\!_2$-player games with Müller winning conditions that appear in the leaves of the Zielonka tree, then the inductive proof generalizes to give a bound as in Theorem 3. Hence to prove an upper bound of size $m^U_{\mathcal{F}}$ for almost-sure winning, it suffices to show that randomized memoryless strategies suffices for upward closed Müller winning conditions. In [3] it was shown that for all $2^1\!/\!_2$-player games randomized memoryless strategies suffices for almost-sure winning for upward closed objectives. This gives us Theorem 6.

Theorem 6. *For all Müller winning conditions $\mathcal{F}$, the family of randomized finite-memory strategies of size $m^U_{\mathcal{F}}$ suffices for optimality on $2^1\!/\!_2$-player game graphs.*

Remark. In general we have $m^U_{\mathcal{F}} < m_{\mathcal{F}}$. Consider for example $\mathcal{F} \subseteq \mathcal{P}(C)$, where $C = \{ c_1, c_2, \ldots, c_k \}$. For the Müller winning condition $\mathcal{F} = \{ C \}$. We have $m^U_{\mathcal{F}} = 1$, and $m_{\mathcal{F}} = |C|$.

6 Conclusion

In this work we present optimal memory bounds for pure almost-sure, positive and optimal strategies for $2^1\!/\!_2$-player games with Müller winning conditions. We also present improved memory bounds for randomized strategies. Unlike the results of [7] our results do not extend to infinite state games: for example, the results of [9] showed that even for $2^1\!/\!_2$-player pushdown games optimal strategies need not exist, and for $\varepsilon > 0$ even ε-optimal strategies may require infinite memory. For lower bound of randomized strategies the constructions of [7] do not work: in fact for the family of games used for lower bounds in [7] randomized memoryless almost-sure winning strategies exist. However, it is known that there exist Müller winning conditions $\mathcal{F} \subseteq \mathcal{P}(C)$, such that randomized almost-sure winning strategies may require memory $|C|!$ [11]. However, whether a matching lower bound of size $m^U_{\mathcal{F}}$ can be proved in general, or whether the upper bound of $m^U_{\mathcal{F}}$ can be improved and a matching lower bound can be proved for randomized strategies with memory remains open.

References

1. J.R. Büchi and L.H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the AMS*, 138:295–311, 1969.
2. K. Chatterjee The complexity of stochastic Müller games. Technical report: UC Berkeley, EECS-2006-141.
3. K. Chatterjee, L. de Alfaro, and T.A. Henzinger. Trading memory for randomness. In *QEST'04* IEEE Computer Society Press, 2004.
4. K. Chatterjee, L. de Alfaro, and T.A. Henzinger. The complexity of stochastic Rabin and Streett games. In *ICALP'05* vol. 3580 of *LNCS*, pages 878–890. Springer.
5. K. Chatterjee, M. Jurdziński, and T.A. Henzinger. Quantitative stochastic parity games. In *SODA'04*, pages 114–123. SIAM, 2004.
6. A. Condon. The complexity of stochastic games. *Information and Computation*, 96:203–224, 1992.
7. S. Dziembowski, M. Jurdziński, and I. Walukiewicz. How much memory is needed to win infinite games? In *LICS'97*, pages 99–110. IEEE, 1997.
8. L. de Alfaro and T.A. Henzinger. Concurrent ω-regular games. In *LICS'00*, pages 141–154. IEEE, 2000.
9. K. Etessami and M. Yannakakis Recursive Markov decision processes and recursive stochastic games. In *ICALP'05* vol. 3580 of *LNCS*, pages 891–903. Springer.
10. Y. Gurevich and L. Harrington. Trees, automata, and games. In *STOC'82*, pages 60–65. ACM, 1982.
11. R. Majumdar. *Symbolic algorithms for verification and control.* PhD Thesis, UC Berkeley, 2003.
12. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer, 1992.
13. D.A. Martin. The determinacy of Blackwell games. *Journal of Symbolic Logic*, 63:1565–1581, 1998.
14. A.K. McIver and C.C. Morgan. Games, probability, and the quantitative μ-calculus $qm\mu$. In *LPAR'02*, volume 2514 of *LNAI*, pages 292–310. Springer, 2002.
15. D. Niwiński Fixed-point characterization of infinite behavior of finite-state systems. In *Theoretical Computer Science*, 189(1-2): 1–69, 1997.
16. P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete-event processes. *SIAM Journal of Control and Optimization*, 25:206–230, 1987.
17. W. Thomas. Languages, automata, and logic. In *Handbook of Formal Languages*, volume 3 (*Beyond Words*), pages 389–455. Springer, 1997.
18. W. Zielonka Infinite games on finitely coloured graphs with applications to automata on infinite trees. In *Theoretical Computer Science*, 200(1-2): 135–183, 1998.
19. W. Zielonka. Perfect-information stochastic parity games. In *FoSSaCS'04*, volume 2987 of *LNCS*, pages 499–513. Springer, 2004.

Generalized Parity Games*

Krishnendu Chatterjee[1], Thomas A. Henzinger[1,2], and Nir Piterman[2]

[1] University of California, Berkeley, USA
[2] EPFL, Switzerland
c_krish@eecs.berkeley.edu, {tah,Nir.Piterman}@epfl.ch

Abstract. We consider games where the winning conditions are disjunctions (or dually, conjunctions) of parity conditions; we call them *generalized* parity games. These winning conditions, while ω-regular, arise naturally when considering fair simulation between parity automata, secure equilibria for parity conditions, and determinization of Rabin automata. We show that these games retain the computational complexity of Rabin and Streett conditions; i.e., they are NP-complete and co-NP-complete, respectively. The (co-)NP-hardness is proved for the special case of a conjunction/disjunction of two parity conditions, which is the case that arises in fair simulation and secure equilibria. However, considering these games as Rabin or Streett games is not optimal. We give an exposition of Zielonka's algorithm when specialized to this kind of games. The complexity of solving these games for k parity objectives with d priorities, n states, and m edges is $O(n^{2kd} \cdot m) \cdot \frac{(k \cdot d)!}{d!^k}$, as compared to $O(n^{2kd} \cdot m) \cdot (k \cdot d)!$ when these games are solved as Rabin/Streett games. We also extend the subexponential algorithm for solving parity games recently introduced by Jurdziński, Paterson, and Zwick to generalized parity games. The resulting complexity of solving generalized parity games is $n^{O(\sqrt{n})} \cdot \frac{(k \cdot d)!}{d!^k}$. As a corollary we obtain an improved algorithm for Rabin and Streett games with d pairs, with time complexity $n^{O(\sqrt{n})} \cdot d!$.

1 Introduction

Games offer a natural framework for reasoning about systems. For example, two-player games arise in *controller synthesis*. We consider the controller that we wish to synthesize as a player in a game against an environment. The controller has to come up with a strategy that will allow it to decide on its action given environment inputs such that regardless of environment actions some goal is satisfied [18].

A *two-player game* is a finite or infinite directed graph where the vertices are partitioned between the two players. A *play* proceeds by moving a token between the vertices of the graph. If the token is found on a vertex of player 1, she chooses an outgoing edge and moves the token along that edge. If the token is found on a vertex of player 2, she gets to choose the outgoing edge. The result

* This research was supported in part by the Swiss National Science Foundation, and by the NSF grants CCR-0225610 and CCR-0234690.

is an infinite sequence of vertices. In order to determine the winner in a play we consider the *infinity set*, the set of states occurring infinitely often in the play. There are several methods to define acceptance conditions that determine which infinity sets are winning for which player. We *solve* a game by computing the set of states from which player 1 has a *strategy* to resolve her choices so that regardless of player 2's choices the play is winning; this is called the *winning set* of player 1. In the games considered here, the winning set of player 1 and the winning set of player 2 (defined dually) form a partition of the vertices of the game [13].

The class of *Rabin* [17] and *Streett* [21] winning conditions are cannonical forms to express all ω-regular winning conditions. Both conditions are defined using a set of pairs of subsets of the vertices of the graph. In order to win the Rabin condition over $\{\langle E_1, F_1\rangle, \ldots, \langle E_k, F_k\rangle\}$, the infinity set has to intersect E_i and not intersect F_i for some i. The Streett winning condition is the dual of the Rabin condition. In order to win the Streett condition over $\{\langle E_1, F_1\rangle, \ldots, \langle E_k, F_k\rangle\}$, the infinity set has to either be disjoint from E_i or to intersect F_i for every i. Rabin and Streett games with n vertices, m edges, and k pairs can be solved in time $O(m \cdot n^k \cdot k!)$ [16].

Another general acceptance condition is the *parity* acceptance condition [7]. In the parity condition, every vertex has a priority and a play is won if the maximal priority visited infinitely often is even. The parity condition is a special case of Rabin and Streett conditions which is closed under complement. While Rabin games are NP-complete (and Streett co-NP-complete) [6], parity games are in NP $\cap$ co-NP [7]. Solving a parity game with m edges, n vertices, and $2k$ priorities can be done in time $O(m \cdot n^k)$ [11] or $n^{O(\sqrt{n})}$ [12].

In this paper, we are interested in games where the winning condition is a disjunction (dually, conjunction) of parity conditions. That is, instead of considering one function assigning priorities to vertices, we consider a set of such functions. A play is winning according to this definition if for one of the functions the maximal priority visited infinitely often is even. We call these winning conditions *generalized parity*.

Generalized parity winning conditions arise naturally in several scenarios. As mentioned, one of the main motivations for considering two-player games is controller synthesis. In the classical setting we consider the system playing against an arbitrary environment. Sometimes, it makes more sense to consider the case where the environment has a goal of its own. In such a case, we are searching for some equilibrium between the system and the environment in which both satisfy their requirements. This led to the introduction of *secure equilibria* [2]. When both players have parity winning conditions, the solution of secure equilibria requires considering a game where the winning condition is the implication between two parity conditions. As parity objectives are closed under complement, we can think about this as either the disjunction or the conjunction of two parity conditions.

Two-player games arise also in the context of simulation [14,9]. Simulation is an important precondition for language containment between automata [4,9] and is also used in the context of minimization of automata [8,20,1]. Simulation between parity automata (automata whose acceptance condition is parity) is naturally framed as a game whose winning condition is again the implication between two parity conditions. Finally, the disjunction of parity conditions also arises when considering the determinization of Rabin and parity automata. Given a Rabin automaton with one pair, we know how to create an equivalent deterministic parity automaton [19,15]. It follows that in order to determinize a Rabin automaton with k pairs, we can consider the disjunction of deterministic parity automata. The acceptance condition of such an automaton is again a disjunction of parity conditions.

As explained, parity conditions are a special case of Rabin and Streett conditions. It follows that generalized parity conditions are again a special case of Rabin and Streett conditions. Indeed, every parity condition is in particular a Rabin condition, and a disjunction of Rabin conditions is again a Rabin condition. Dually, every parity condition is a Streett condition, and a conjunction of Streett conditions is again a Streett condition. On the other hand, generalized parity conditions are also more general than Rabin and Streett conditions. This is because a Rabin condition is a disjunction of parity conditions with three priorities, and a Streett condition is a conjunction of parity conditions with three priorities. It is an interesting question whether generalized parity conditions retain the computational hardness of Rabin and Streett conditions. We would also like to devise specialized algorithms for generalized parity conditions that outperform the natural reduction to Rabin and Streett conditions. These are the two questions considered in this paper.

We show that generalized parity conditions are NP and co-NP complete, suggesting that the computational complexity of Rabin and Streett conditions is retained. Our lower bound applies already to the special case of a disjunction/conjunction of two parity conditions, which is the case that arises in secure equilibria and in fair simulation.

We give specialized algorithms that outperform the reduction of generalized parity conditions to Rabin and Streett conditions. Specifically, Zielonka's algorithm [22] when specialized to a disjunction of k parity objectives with d priorities works in time proportional to $O(m \cdot n^{2kd} \cdot \frac{(k \cdot d)!}{d!^k})$ (compared to $O(m \cdot n^{2kd} \cdot (k \cdot d)!)$ when these games are solved as Rabin or Streett games). We generalize the techniques of the subexponential algorithm for solving parity games [12] to generalized parity games. The resulting complexity of solving generalized parity games is $n^{O(\sqrt{n})} \cdot \frac{(k \cdot d)!}{d!^k}$. As a corollary we obtain an improved algorithm for Rabin and Streett games with k pairs, with time complexity $n^{O(\sqrt{n})} \cdot k!$, as compared to the previous best known algorithm with time complexity $O(m \cdot n^k \cdot k!)$ [16].

In the full version we also show how to extend the direct rank computation [11,16] to generalized parity conditions. The resulting complexity of solving generalized parity games is $O(m \cdot n^{kd} \cdot \frac{(k \cdot d)!}{d!^k})$ (as compared to $O(m \cdot n^{kd} \cdot (k \cdot d)!)$).

2 Definitions

We consider turn-based deterministic games played by two players with a conjunction / disjunction of parity objectives; we call them *generalized parity* games. We define game graphs, plays, strategies, objectives, and the notion of winning.

Game graphs. A *game graph* $G = ((S, E), (S_1, S_2))$ consists of a directed graph (S, E) with a finite state space S and a set E of edges, and a partition (S_1, S_2) of the state space S into two sets. The states in S_1 are player-1 states, and the states in S_2 are player-2 states. For a state $s \in S$, we write $E(s) = \{t \in S \mid (s, t) \in E\}$ for the set of successor states of s. We assume that every state has at least one outgoing edge, i.e., $E(s)$ is nonempty for all states $s \in S$. Given a set $U \subseteq S$, if in the subgraph induced by U every state has at least one outgoing edge, then the subgraph is called a *subgame*, denoted $G \upharpoonright U$. Formally, $G \upharpoonright U = ((S \cap U, E \cap (U \times U)), (S_1 \cap U, S_2 \cap U))$.

Plays. A game is played by two players: player 1 and player 2, who form an infinite path in the game graph by moving a token along edges. They start by placing the token on an initial state, and then they take moves indefinitely in the following way. If the token is on a state in S_1, then player 1 moves the token along one of the edges going out of the state. If the token is on a state in S_2, then player 2 does likewise. The result is an infinite path in the game graph; we refer to such infinite paths as plays. Formally, a *play* is an infinite sequence $\langle s_0, s_1, s_2, \ldots \rangle$ of states such that $(s_k, s_{k+1}) \in E$ for all $k \geq 0$. We write Ω for the set of all plays.

Strategies. A strategy for a player is a recipe that specifies how to extend plays. Formally, a *strategy* σ for player 1 is a function σ: $S^* \cdot S_1 \to S$ that, given a finite sequence of states (representing the history of the play so far) which ends in a player 1 state, chooses the next state. The strategy must choose only available successors, i.e., for all $w \in S^*$ and $s \in S_1$ we have $\sigma(w \cdot s) \in E(s)$. The strategies for player 2 are defined analogously. We write Σ and Π for the sets of all strategies for player 1 and player 2, respectively. Strategies in general require memory to remember the history of plays. An equivalent definition of strategies is as follows. Let M be a set called *memory.* A strategy with memory can be described as a pair of functions: (a) a *memory-update* function σ_u: $S \times M \to M$ that, given the memory and the current state, updates the memory; and (b) a *next-state* function σ_n: $S \times M \to S$ that, given the memory and the current state, specifies the successor state. The strategy is *finite-memory* if the memory M is finite. An important special class of strategies are the *memoryless* strategies. A strategy is *memoryless* if the memory M is a singleton set. The memoryless strategies do not depend on the history of a play, but only on the current state. Each memoryless strategy for player 1 can be specified as a function σ: $S_1 \to S$ such that $\sigma(s) \in E(s)$ for all $s \in S_1$, and analogously for memoryless player-2 strategies. Given a starting state $s \in S$, a strategy $\sigma \in \Sigma$ for player 1, and a strategy $\pi \in \Pi$ for player 2, there is a unique play, denoted $\omega(s, \sigma, \pi) = \langle s_0, s_1, s_2, \ldots \rangle$, which is defined as follows: $s_0 = s$ and for all $k \geq 0$, if $s_k \in S_1$, then $\sigma(s_0, s_1, \ldots, s_k) = s_{k+1}$, and if $s_k \in S_2$, then $\pi(s_0, s_1, \ldots, s_k) = s_{k+1}$.

Conjunction and disjunction of parity objectives. We consider game graphs with a conjunction of parity objectives for player 1, and the complementary disjunction of parity objectives for player 2. For a play $\omega = \langle s_0, s_1, s_2, \ldots \rangle$, we define $\text{Inf}(\omega) = \{s \in S \mid s_k = s \text{ for infinitely many } k \geq 0\}$ to be the set of states that occur infinitely often in ω. We also define reachability and safety objectives as they will be useful in the analysis of the algorithms.

Reachability and safety objectives. Given two sets $T, F \subseteq S$ of states, the reachability objective $\text{Reach}(T)$ requires that some state in T be visited, and dually, the safety objective $\text{Safe}(F)$ requires that only states in F be visited. Formally, the sets of winning plays are $\text{Reach}(T) = \{\langle s_0, s_1, s_2, \ldots \rangle \in \Omega \mid \exists k \geq 0.\ s_k \in T\}$ and $\text{Safe}(F) = \{\langle s_0, s_1, s_2, \ldots \rangle \in \Omega \mid \forall k \geq 0.\ s_k \in F\}$. The reachability and safety objectives are dual in the sense that $\text{Reach}(T) = \Omega \setminus \text{Safe}(S \setminus T)$.

Parity objectives; conjunctions and disjunctions. For $d \in \mathbb{N}$, we let $[d] = \{0, 1, \ldots, d\}$ and $[d]_+ = \{1, 2, \ldots, d\}$. Let $p : S \to [d]$ be a function that assigns a *priority* $p(s)$ to every state $s \in S$. The parity objective requires that the maximal priority occurring infinitely often is *even*. Formally, the set of winning plays is $\text{Parity}(p) = \{\omega \in \Omega \mid \max(\text{Inf}(\omega)) \text{ is even}\}$. For a priority function $p : S \to [d]$, we denote by $\overline{p} : S \to [d+1]_+$ the priority function $\overline{p}(s) = p(s) + 1$ for all $s \in S$. Then $\text{Parity}(\overline{p}) = \Omega \setminus \text{Parity}(p)$, i.e., parity objectives are closed under complementation. For $i = 1, 2, \ldots, k$, consider k priority functions $p_i : S \to [d_i]$. The objective $\text{ConjParity}(p_1, p_2, \ldots, p_k)$ is the conjunction of the parity objectives defined by p_i, i.e., $\text{ConjParity}(p_1, p_2, \ldots, p_k) = \bigcap_{i=1}^{k} \text{Parity}(p_i)$. Similarly, the objective $\text{DisjParity}(p_1, p_2, \ldots, p_k)$ is the disjunction of the parity objectives defined by p_i, i.e., $\text{DisjParity}(p_1, p_2, \ldots, p_k) = \bigcup_{i=1}^{k} \text{Parity}(p_i)$. The conjunction and disjunction of parity objectives are dual in the sense that $\text{ConjParity}(p_1, p_2, \ldots, p_k) = \Omega \setminus \text{DisjParity}(\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_k)$. If all priority functions have range $[d]$ and there are k priority functions, then we refer to this class of conjunctions and disjunctions of parity objectives as $(\wedge, k, [d])$ and $(\vee, k, [d])$, respectively. Similarly, if all priority functions have range $[d]_+$ and there are k priority functions, then we refer to this class of conjunctions and disjunctions of parity objectives as $(\wedge, k, [d]_+)$ and $(\vee, k, [d]_+)$, respectively. Parity objectives with priority functions with range $[1]$ are called coBüchi objectives, and with range $[2]_+$ they are called Büchi objectives.

Rabin and Streett objectives. A *Rabin specification* for the game graph G is a finite set $\mathcal{F} = \{\langle E_1, F_1 \rangle, \ldots, \langle E_d, F_d \rangle\}$ of pairs of sets of states, that is, $E_j \subseteq S$ and $F_j \subseteq S$ for all $1 \leq j \leq d$. The pairs in $\mathcal{F}$ are called Rabin pairs. The Rabin specification $\mathcal{F}$ requires that for some Rabin pair $1 \leq j \leq d$, all states in the left set E_j be visited finitely often, and some state in the right set F_j be visited infinitely often. Thus, the Rabin objective defined by $\mathcal{F}$ is the set $\text{Rabin}(\mathcal{F}) = \{\omega \in \Omega \mid \exists 1 \leq j \leq d. (\text{Inf}(\omega) \cap E_j = \emptyset \ \wedge\ \text{Inf}(\omega) \cap F_j \neq \emptyset)\}$ of winning plays. The complements of Rabin objectives are called Streett objectives. A *Streett specification* for G is likewise a set $\mathcal{F} = \{\langle E_1, F_1 \rangle, \ldots, \langle E_d, F_d \rangle\}$ of pairs of sets of states $E_j \subseteq S$ and $F_j \subseteq S$. The pairs in $\mathcal{F}$ are called Streett pairs. The Streett specification $\mathcal{F}$ requires that for all Streett pairs $1 \leq j \leq d$, if some state in the left set F_j is visited infinitely often, then some state in the right set

E_j is visited infinitely often. Formally, the Streett objective defined by $\mathcal{F}$ is the set $\text{Streett}(\mathcal{F}) = \{\omega \in \Omega \mid \forall 1 \le j \le d.(\text{Inf}(\omega) \cap E_j \neq \emptyset \ \vee \ \text{Inf}(\omega) \cap F_j = \emptyset)\}$ of winning plays. The Rabin and Streett objectives are dual in the sense that $\text{Streett}(\mathcal{F}) = \Omega \setminus \text{Rabin}(\mathcal{F})$. The parity objectives are a subclass of the Rabin objectives that is closed under complementation. It follows that every parity objective is both a Rabin objective and a Streett objective.

Relationship between objectives. It may be noted that given k priority functions $p_1, p_2, \ldots, p_k$ with ranges $[2d_1]$, $[2d_2] \ldots$, $[2d_k]$, the disjunction of the parity objectives can be expressed as a Rabin objective with $\sum_{i=1}^{k} d_i$ pairs, and the conjunction of the parity objectives can be expressed as a Streett objective with $\sum_{i=1}^{k} d_i$ pairs. Conversely, a Rabin objective $\text{Rabin}(\mathcal{F})$ with k pairs can be expressed as an objective in $(\vee, k, [3]_+)$ as follows: for each pair $\langle E_i, F_i \rangle$ consider the priority function $p_i : S \to [3]_+$ such that $p_i(s) = 3$ if $s \in E_i$, and 2 if $s \in F_i \setminus E_i$, and 1 otherwise; then $\text{DisjParity}(p_1, p_2, \ldots, p_k) = \text{Rabin}(\mathcal{F})$. Similarly, a Streett objective $\text{Streett}(\mathcal{F})$ with k pairs can be expressed as an objective in $(\wedge, k, [2])$ as follows: for each pair $\langle E_i, F_i \rangle$ consider the priority function $p_i : S \to [2]$ such that $p_i(s) = 2$ if $s \in E_i$, and 1 if $s \in F_i \setminus E_i$, and 0 otherwise; then $\text{ConjParity}(p_1, p_2, \ldots, p_k) = \text{Streett}(\mathcal{F})$.

Winning strategies and sets. Given a game graph G and an objective $\Phi \subseteq \Omega$ of winning plays for player 1, a strategy $\sigma \in \Sigma$ is a *winning strategy* for player 1 from a state $s \in S$ if for all player-2 strategies $\pi \in \Pi$, the play $\omega(s, \sigma, \pi)$ is winning, i.e., $\omega(s, \sigma, \pi) \in \Phi$. The winning strategies for player 2 are defined analogously. A state $s \in S$ is winning for player 1 with respect to the objective Φ if player 1 has a winning strategy from s. Formally, the set of *winning states* for player 1 with respect to the objective Φ in a game graph G is $W_1^G(\Phi) = \{s \in S \mid \exists \sigma \in \Sigma.\ \forall \pi \in \Pi.\ \omega(s, \sigma, \pi) \in \Phi\}$. Analogously, the set of winning states for player 2 with respect to an objective $\Psi \subseteq \Omega$ of winning plays for player 2 is $W_2^G(\Psi) = \{s \in S \mid \exists \pi \in \Pi.\ \forall \sigma \in \Sigma.\ \omega(s, \sigma, \pi) \in \Psi\}$. If the game graph is clear from the context, we drop the superscript. We say that there exists a memoryless winning strategy for player 1 with respect to the objective Φ if there exists such a strategy from all states in $W_1(\Phi)$; and similarly for player 2.

Theorem 1 (Determinacy and complexity [6])

1. *For all game graphs $G = ((S, E), (S_1, S_2))$, all Streett objectives Φ for player 1, and the complementary Rabin objective $\Psi = \Omega \setminus \Phi$ for player 2, the following assertions hold.*
   - *We have $W_1(\Phi) = S \setminus W_2(\Psi)$.*
   - *There exists a memoryless winning strategy for player 2, and a finite-memory winning strategy for player 1.*
2. *Given a game graph G, a Streett objective Φ for player 1, the complementary Rabin objective $\Psi = \Omega \setminus \Phi$ for player 2, and a state s, the problem of deciding whether $s \in W_2(\Psi)$ is NP-complete, and deciding whether $s \in W_1(\Phi)$ is co-NP-complete.*

Closed sets and attractors. Two notions that will play key roles in the analysis of the algorithms are the notions of *closed sets* and *attractors*.

Closed sets. A set $U \subseteq S$ of states is a *closed set* for player 1 if the following two conditions hold: (a) for all states $u \in (U \cap S_1)$, we have $E(u) \subseteq U$, i.e., all successors of player-1 states in U are again in U; and (b) for all $u \in (U \cap S_2)$, we have $E(u) \cap U \neq \emptyset$, i.e., every player 2 state in U has a successor in U. A player-1 closed set is also called a *trap* for player 1. The closed sets for player 2 are defined analogously. For every closed set U for player ℓ, for $\ell \in \{1, 2\}$, the game $G \upharpoonright U$ is a subgame.

Proposition 1. *Consider a game graph G, and a closed set U for player 2. For every objective Φ for player 1, we have $W_1^{G \upharpoonright U}(\Phi) \subseteq W_1^G(\Phi)$.*

Attractors. Given a game graph G, a set $U \subseteq S$ of states, and a player $\ell \in \{1, 2\}$, the set $Attr_\ell(U, G)$ contains the states from which player ℓ has a strategy to reach a state in U against all strategies of the other player; that is, $Attr_\ell(U, G) = W_\ell(\text{Reach}(U))$. The set $Attr_1(U, G)$ can be computed inductively as follows: let $R_0 = U$; let $R_{i+1} = R_i \cup \{s \in S_1 \mid E(s) \cap R_i \neq \emptyset\} \cup \{s \in S_2 \mid E(s) \subseteq R_i\}$ for all $i \geq 0$; then $Attr_1(U, G) = \bigcup_{i \geq 0} R_i$. The inductive computation of $Attr_2(U, G)$ is analogous. For all states $s \in Attr_1(U, G)$, define $rank(s) = i$ if $s \in (R_i \setminus R_{i-1})$, that is, $rank(s)$ denotes the least $i \geq 0$ such that s is included in R_i. Define a memoryless strategy $\sigma \in \Sigma$ for player 1 as follows: for each state $s \in (Attr_1(U, G) \cap S_1)$ with $rank(s) = i$, choose a successor $\sigma(s) \in (R_{i-1} \cap E(s))$ (such a successor exists by the inductive definition). It follows that for all states $s \in Attr_1(U, G)$ and all strategies $\pi \in \Pi$ for player 2, the play $\omega(s, \sigma, \pi)$ reaches U in at most $|Attr_1(U, G)|$ transitions.

Proposition 2. *For all game graphs G, all players $\ell \in \{1, 2\}$, and all sets $U \subseteq S$ of states, the set $S \setminus Attr_\ell(U, G)$ is a closed set for player ℓ.*

Notation. For a game graph $G = ((S, E), (S_1, S_2))$, a set $U \subseteq S$, and $\ell \in \{1, 2\}$, we write $G \setminus Attr_\ell(U, G)$ to denote the game graph $G \upharpoonright (S \setminus Attr_\ell(U, G))$.

3 Computational Complexity

In this section we study the computational complexity of generalized parity games. We consider $(\vee, k, [d])$ and $(\wedge, k, [d])$ objectives and present complexity results varying both k and d. Observe that if both k and d are constants, then generalized parity games can be solved in polynomial time (by reduction to Rabin and Streett objectives with a constant number of pairs). The next theorem completes the complexity analysis. Other than the last hardness result (part 5) of Theorem 2, all other results can be easily derived (see [3] for details); and part 5 of Theorem 2 is proved in Lemma 1.

Theorem 2. *Given a game graph G, the following assertions hold.*

1. *For objectives Ψ in $(\vee, k, [d])$ and Φ in $(\wedge, k, [d])$, and a state s: whether $s \in W_2(\Psi)$ and $s \in W_1(\Phi)$ can be decided in NP and co-NP, respectively.*
2. *For objectives Ψ in $(\vee, k, [3]_+)$ and Φ in $(\wedge, k, [2])$, and a state s: (a) whether $s \in W_2(\Psi)$ is NP-hard, and (b) whether $s \in W_1(\Phi)$ is co-NP-hard.*

3. *For objectives Φ in $(\vee, k, [2]_+)$ or $(\wedge, k, [2]_+)$ or $(\vee, k, [1])$ or $(\wedge, k, [1])$, and a state s: whether $s \in W_1(\Phi)$ (or $s \in W_2(\Phi)$) can be decided in PTIME.*
4. *For objectives Φ in $(\vee, 1, [d])$ or $(\wedge, 1, [d])$, and a state s: whether $s \in W_1(\Phi)$ (or $s \in W_2(\Phi)$) can be decided in NP $\cap$ co-NP.*
5. *For objectives Ψ in $(\vee, 2, [d])$ and Φ in $(\wedge, 2, [d])$, and a state s: whether $s \in W_2(\Psi)$ is NP-hard, and whether $s \in W_1(\Phi)$ is co-NP-hard.*

Lemma 1. *Given a game graph G, an objective Ψ in $(\vee, 2, [d])$, and a state s, deciding whether $s \in W_2(\Psi)$ is NP-hard.*

Proof. We present a reduction from SAT. Consider a SAT formula ψ with clauses $C_0, C_1, \ldots, C_m$ over boolean variables $x_0, x_1, \ldots, x_n$. We denote by C the set of all clauses and by X the set of all variables. A *literal* is a variable or its negation (i.e, x_i or $\neg x_i$). We denote by l a literal and by L the set of all literals. We now construct a game graph $G = ((S, E), (S_1, S_2))$ and an objective Ψ that is obtained as a disjunction of two parity objectives.

1. *State space and transitions.* We have $S_1 = \{s_0\}$; $S_2 = C \cup L$ and $E = \{(s_0, C_i) \mid C_i \in C\} \cup \{(C_i, l) \mid C_i \in C, l \text{ occurs in } C_i\} \cup \{(l, s_0) \mid l \in L\}$. Hence player 1 chooses between the clauses, and in each clause player 2 can choose a literal that makes the clause true, and from the literals the next state is the starting state s_0.
2. *Priority functions.* We specify priority functions $p_1 : S \to [2n]$ and $p_2 : S \to [2n]$ as follows:

$$p_1(s) = \begin{cases} 0 & s \in C; \text{ or } s = s_0; \\ 2k & s = x_k; \\ 2k+1 & s = \neg x_k; \end{cases} \qquad p_2(s) = \begin{cases} 0 & s \in C; \text{ or } s = s_0; \\ 2k & s = \neg x_k; \\ 2k+1 & s = x_k; \end{cases}$$

We analyze the game with objective $\Psi = \mathrm{DisjParity}(p_1, p_2)$ for player 2. Since the objective is a Rabin objective it suffices to analyze the memoryless strategies as candidate winning strategies for player 2. We analyze the following two cases.

1. *Satisfiability implies winning.* Let $A : X \to \{0, 1\}$ be a satisfying assignment for ψ. We define $\widehat{A} : X \to L$ as follows: for $x \in X$ we have $\widehat{A}(x) = x$ if $A(x) = 1$ and $\neg x$ otherwise. Fix a memoryless strategy $\pi : S_2 \to S$ for player 2, as follows: for $C_i \in C$ pick a literal l_k that appears in C_i and $\widehat{A}(x_k) = l_k$ (such a literal exists since A is a satisfying assignment), and set $\pi(C_i) = l_k$. Now consider any strategy σ for player 1. Let l_j be the maximal literal that appear infinitely often along the play $\omega(s_0, \sigma, \pi)$. Observe that both x_j and $\neg x_j$ cannot appear infinitely often. If $l_j = x_j$, then $\mathrm{Parity}(p_1)$ is satisfied, and if $l_j = \neg x_j$, then $\mathrm{Parity}(p_2)$ is satisfied. Hence, player 2 has a winning strategy.
2. *Winning implies satisfiability.* Consider a pure memoryless strategy π for player 2. If there exists C_j, C_k such that $\pi(C_j) = x_i$ and $\pi(C_k) = \neg x_i$, then we show that π is not winning for player 2; otherwise, it is easy to construct a satisfying assignment from the memoryless strategy π. Consider C_j, C_k such that $\pi(C_j) = x_i$ and $\pi(C_k) = \neg x_i$, and the strategy σ for player 1 that

alternates between C_j and C_k at s_0. Then we have $\max(p_\ell(\mathrm{Inf}(\omega(s,\sigma,\pi)))) = \max\{p_\ell(x_i), p_\ell(\neg x_i)\} = 2i{+}1$, for $\ell \in \{1,2\}$. It follows that π is not a winning strategy for player 2, contrary to our assumption. ∎

4 The Classical Algorithm

We first present the classical algorithm (Zielonka's algorithm) for games with conjunctions and disjunctions of parity objectives. We start with an informal description of the algorithm; a formal description is given as Algorithm 1. Without loss of generality we consider all priority functions to have the range $[1..(2d{+}1)]$ for some d.

Notations. We consider k priority functions $p_1 : S \to [2d_1]$, $p_2 : S \to [2d_2], \ldots, p_k : S \to [2d_k]$. The objective Φ for player 1 is the conjunction $\mathrm{ConjParity}(p_1, p_2, \ldots, p_k)$ of the parity objectives and the objective for player 2 is the complementary objective $\Psi = \mathrm{DisjParity}(\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_k)$. We use the following notation: (a) for $p_i : S \to [2d_i]$, we denote by $\mathsf{MaxEven}(p_i) = p^{-1}(2d_i)$ the set of maximal even priority states, and if we consider a subgame defined by a subset S_j of states with $p_i : S_j \to [2\widehat{d_i}]$ for $\widehat{d_i} \le d_i$, we denote by $\mathsf{MaxEven}(p_i) = p^{-1}(2\widehat{d_i})$ the maximal even priority states in the subgame; and (b) for $p_i : S \to [2d_i]$, we denote by $\mathsf{MaxOdd}(p_i) = p^{-1}(2d_i - 1)$ the set of maximal odd priority states. If we consider a subgame defined by a subset S_j of states with $p_i : S_j \to [2\widehat{d_i}]$ for $\widehat{d_i} \le d_i$, then we denote by $\mathsf{MaxOdd}(p_i) = p^{-1}(2\widehat{d_i} - 1)$ the maximal odd priority states in the subgame.

Informal description of the classical algorithm. The algorithm computes the set of states that are winning for player 2 according to the disjunction of parity conditions. If all parity conditions contain only states of priority 1, then obviously player 2 is losing. Indeed, every infinite play visits the maximal priority 1 according to all disjuncts. Suppose that no such void parity condition exists. The algorithm proceeds by choosing one of the disjuncts. Let d denote the maximal odd priority occurring in this disjunct. Then we compute the states from which player 2 wins by visiting priority d finitely often and visiting $d-1$ infinitely often, or eventually avoiding both of them and winning according to the lower priorities of this disjunct or one of the other disjuncts. In order to compute this set of states, we first compute the set of states from which player 1 can force a visit to priority d; clearly we want to avoid these states so we consider the arena without these states. We now search for a trap of player 1 that is composed of two parts: first some states with priority $d-1$ and player 2's attractor to these states, and second, some states that are winning for player 2 with the simpler winning condition. When we find such a trap, we conclude that it is winning for player 2, remove it from the arena, and continue with the rest. If we do not find such a trap for every one of the disjuncts, we conclude that player 1 wins from all the states that remain.

Correctness and time complexity. The following theorem states the correctness and complexity of Algorithm 1. The correctness proof is similar to the

Algorithm 1. Classical Algorithm for Disjunction of Parity Objectives

```
    Input: a 2-player game graph $G = ((S,E),(S_1,S_2))$ and
           priority functions $p_1 : S \to [2d_1], p_2 : S \to [2d_2], \ldots, p_k : S \to [2d_k]$.
    Output: $W_2 \subseteq S$.
    1. return DisjParityWin($G$,$\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_k$);
procedure DisjParityWin($G$,$\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_k$)
    1. if (for all $i = 1, 2, \ldots, k$ we have $d_i = 0$)
       1.1 return $\emptyset$;
    2. foreach $i = 1, 2, \ldots, k$ such that $d_i \neq 0$
       2.1 $G_1 := G \setminus Attr_1(\mathsf{MaxOdd}(\overline{p}_i), G)$;
       2.2 $H_1 := G_1 \setminus Attr_2(\mathsf{MaxEven}(\overline{p}_i), G_1)$; $j := 0$;
       2.3 repeat
           2.3.1 $j := j + 1$;
           2.3.2 $W_j :=$ DisjParityWin($H_j, \overline{p}_1, \overline{p}_2, \ldots, \overline{p}_i : H_j \to [2d_i - 1]_+, \ldots, \overline{p}_k$);
           2.3.3 $\overline{W}_j := Attr_1(H_j \setminus W_j, G_j)$;
           2.3.4 $G_{j+1} := G_j \setminus \overline{W}_j$;
           2.3.5 $H_{j+1} := G_{j+1} \setminus Attr_2(\mathsf{MaxEven}(\overline{p}_i), G_{j+1})$;
       2.4 until ($W_j = \emptyset$ or $W_j = H_j$);
       2.5 if ($W_j = H_j$)
           2.5.1 return $Attr_2(G_j, G) \cup$ DisjParityWin($G \setminus Attr_2(G_j, G)$,$\overline{p}_1, \ldots, \overline{p}_k$);
       end foreach;
    3. return $\emptyset$;
```

correctness proofs in [5,22,10]; see [3] for details. If we denote the run time of the algorithm by $T(n, d_1, d_2, \ldots, d_k)$, then the following recurrence holds: $T(n, d_1, d_2, \ldots, d_k) = O(m) + n^2 \cdot \sum_{i=1}^{k} T(n-1, d_1, d_2, \ldots, d_i - 1, \ldots, d_k)$. The bound $T(n, d_1, d_2, \ldots, d_k) \leq O(m \cdot n^{2d}) \cdot \binom{d}{d_1, d_2, \ldots, d_k}$ follows.

Theorem 3 (Correctness and run time). *Given a game graph $G = ((S,E),(S_1,S_2))$ and priority functions $p_1 : S \to [2d_1], p_2 : S \to [2d_2], \ldots, p_k : S \to [2d_k]$, the following assertions hold.*

1. *If W is the output of Algorithm 1, then $W = W_2(\text{DisjParity}(\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_k))$, and $S \setminus W = W_1(\text{ConjParity}(p_1, p_2, \ldots, p_k))$.*
2. *The run time of Algorithm 1 is $O(m \cdot n^{2d}) \cdot \binom{d}{d_1, d_2, \ldots, d_k}$, where $n = |S|$, $m = |E|$, and $d = \sum_{i=1}^{k} d_i$.*

Remark. In the case of Rabin or Streett objectives the above algorithm is identical to the one in [5,22,10]. Indeed, if every disjunct has 3 priorities, then for all i we have $d_i = 1$, and $\binom{d}{d_1, d_2, \ldots d_k}$ is $d!$. On the other hand, if we reduce ConjParity$(p_1, \ldots, p_k)$ to a Streett objective, we get $d = \Sigma_{i=1}^{k} d_i$ pairs, and the classical Streett algorithm [22] would compute in time $O(m \cdot n^{2d} \cdot d!)$.

5 A New Algorithm

In this section we present a new algorithm for games with disjunctions and conjunctions of parity objectives. The algorithm is inspired by the algorithm of [12] for parity games. The algorithm is based on the notion of *dominions*; it tries to identify small dominions cheaply. We now define dominions and study the complexity to compute nonempty dominions (if they exist).

Dominions. Given a game graph $G = ((S, E), (S_1, S_2))$ with priority functions $p_1, p_2, \ldots, p_k$, we consider the objectives $\Phi = \mathrm{ConjParity}(p_1, p_2, \ldots, p_k)$ and $\Psi = \mathrm{DisjParity}(\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_k)$ for player 1 and player 2, respectively.

1. A set $U \subseteq S$ is a *dominion* for player 1, if U is a player-2 closed set and player 1 has a winning strategy for objective Φ from all states in U in the subgame $G \upharpoonright U$;
2. A set $U \subseteq S$ is a *dominion* for player 2, if U is a player-1 closed set and player 2 has a winning strategy for objective Ψ from all states in U in the subgame $G \upharpoonright U$.

The following lemma characterizes the computation of dominions (see [3] for details).

Lemma 2. *Let G be a game graph with n states. Consider priority functions $p_1, p_2, \ldots, p_k$, and objectives $\Phi = \mathrm{ConjParity}(p_1, p_2, \ldots, p_k)$ and $\Psi = \mathrm{DisjParity}(\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_k)$ for player 1 and player 2, respectively. Let $p_i : S \to [2d_i]$ and $d = \sum_{i=1}^{k} d_i$. A dominion for player 1 or player 2 of size at most ℓ, for $\ell \geq 1$, if one exists, can be computed in time $n^{O(\ell)} \cdot O(d)$.*

We use the following notation in the sequel. Given a game graph $G = ((S, E), (S_1, S_2))$ with priority functions $p_1, p_2, \ldots, p_k$, and objectives $\Phi = \mathrm{ConjParity}(p_1, p_2, \ldots, p_k)$ and $\Psi = \mathrm{DisjParity}(\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_k)$ we denote by *DisjParityDominion*$(G, \overline{p}_1, \overline{p}_2, \ldots, \overline{p}_k, \ell)$ a procedure that returns a dominion of size at most ℓ for player 2 (if one exists) and runs in time $|S|^{O(\ell)} \cdot O(d)$; if the procedure returns empty set, then all dominions for player 2 have at least $\ell + 1$ states. Similarly, *ConjParityDominion*$(G, p_1, p_2, \ldots, p_k, \ell)$ is a procedure that returns a dominion of size at most ℓ for player 1 (if one exists) and runs in time $|S|^{O(\ell)} \cdot O(d)$; if the procedure returns the empty set, then all dominions for player 1 have at least $\ell + 1$ states.

The new algorithm. The new algorithm is based on the following simple observations about the sets obtained by the classical algorithm.

Fact 1. The set G_j obtained in Step 2.5.1 of Algorithm 1 is a player-2 dominion in the game G.

Fact 2. The set $H_j \setminus W_j$ obtained in Step 2.3.2 of Algorithm 1 is a player-1 dominion in the subgame G_j.

With the above observations we obtain the new algorithm from the classical algorithm as follows; the formal description is presented as Algorithm 2.

1. Before Step 2 of the classical algorithm (which corresponds to Step 3 of Algorithm 2) we invoke *DisjParityDominion*$(G, \overline{p}_1, \overline{p}_2, \ldots, \overline{p}_k, \ell)$ with $\ell = \left\lceil \sqrt{|S|} \right\rceil$; if a nonempty set U is obtained, then we remove U and its player-2 attractor as a subset of the player-2 winning set, and proceed on the subgame; else we proceed as the classical algorithm.

Algorithm 2. New Algorithm for Disjunction of Parity Objectives

```
    Input: a 2-player game graph $G = ((S,E),(S_1,S_2))$ and
           priority functions $p_1 : S \to [2d_1], p_2 : S \to [2d_2], \ldots, p_k : S \to [2d_k]$.
    Output: $W_2 \subseteq S$.
    1. return DisjParityWin($G$,$\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_k$);
procedure DisjParityWin($G$,$\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_k$)
    1. if (for all $i = 1, 2, \ldots, k$ we have $d_i = 0$)
       1.1 return $\emptyset$;
    2. $U$ :=$DisjParityDominion(G, \overline{p}_1, \overline{p}_2, \ldots, \overline{p}_k, \ell)$ for $\ell = \left\lceil \sqrt{|S|} \right\rceil$;
          2.1 if ($U \neq \emptyset$)
           2.1.1 return $Attr_2(U,G) \cup$ DisjParityWin($G \setminus Attr_2(U,G)$,$\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_k$);
    3. foreach $i = 1, 2, \ldots, k$ such that $d_i \neq 0$
       3.1 $G_1 := G \setminus Attr_1(\mathsf{MaxOdd}(\overline{p}_i), G)$;
       3.2 $H_1 := G_1 \setminus Attr_2(\mathsf{MaxEven}(\overline{p}_i), G_1)$; $j := 0$;
       3.3 repeat
          3.3.1 $j := j + 1$;
          3.3.2 $U$ :=$ConjParityDominion(H_j, p_1, p_2, \ldots, p_k, \ell)$ for $\ell = \left\lceil \sqrt{|S|} \right\rceil$;
             3.3.2.1 if ($U \neq \emptyset$)
                3.3.2.1.1 $\overline{W}_j := Attr_1(U, G_j)$; goto step 3.3.5;
          3.3.3 $W_j :=$ DisjParityWin($H_j$, $\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_i : H_j \to [2d_i - 1]_+, \ldots, \overline{p}_k$);
          3.3.4 $\overline{W}_j := Attr_1(H_j \setminus W_j, G_j)$;
          3.3.5 $G_{j+1} := G_j \setminus \overline{W}_j$;
          3.3.6 $H_{j+1} := G_{j+1} \setminus Attr_2(\mathsf{MaxEven}(\overline{p}_i), G_{j+1})$;
       3.4 until ($W_j = \emptyset$ or $W_j = H_j$);
       3.5 if ($W_j = H_j$)
          3.5.1 return $Attr_2(G_j, G) \cup$ DisjParityWin($G \setminus Attr_2(G_j, G)$,$\overline{p}_1, \ldots, \overline{p}_k$);
       end foreach;
    3. return $\emptyset$;
```

2. Before Step 2.3.2 of the classical algorithm (which corresponds to Step 3.3.3 of Algorithm 2), we invoke $ConjParityDominion(G, p_1, p_2, \ldots, p_k, \ell)$ with $\ell = \left\lceil \sqrt{|S|} \right\rceil$; if a nonempty set U is obtained, then we remove U and its player-1 attractor and proceed to Step 2.3.4 (Step 3.3.5 of Algorithm 2); else we proceed as the classical algorithm.

Correctness. The correctness of Algorithm 2 is immediate from the correctness of the classical algorithm and from Proposition 1.

Time complexity. We now analyze the time complexity of Algorithm 2. Let us denote by $T(n, d_1, d_2, \ldots, d_k)$ the run time of the algorithm on graphs with n states and priority functions $p_1, p_2, \ldots, p_k$ with $p_i : S \to [2d_i]$, for $i = 1, 2, \ldots, k$. Let $d = \sum_{i=1}^{k} d_i$. By Lemma 2, Step 2 takes $n^{O(\sqrt{n})} \cdot O(d)$ time. For simplicity we will drop the $O(\cdot)$ from $O(d)$; the whole analysis can be easily carried out with $O(d)$. We now analyze the following cases.

1. If Step 2 succeeds, then at least one state is removed and we need to solve a subgame with one state less (which takes time $T(n-1, d_1, d_2, \ldots, d_k)$).

2. If Step 2 fails, then any dominion for player 1 in G must have size at least $\sqrt{n}$; hence the dominion G_j discovered at Step 3.5.1 must be of size at least $\sqrt{n}$ (as otherwise it would have been discovered in Step 2). Hence the DisjParityWin call at Step 3.5.1 requires to solve a subgame of size at most $n - \sqrt{n}$, and this requires time $T(n - \sqrt{n}, d_1, d_2, \ldots, d_k)$. We now analyze the loop in Step 3.3: we analyze the work for one priority function, and then sum it up for all k priority functions. For a fixed priority function p_i, Step 3.3.2 is executed at most n times, and by Lemma 2, each time it requires at most $n^{O(\sqrt{n})} \cdot d$ time. Hence the total work of Step 3.3.2 requires at most $n \cdot n^{O(\sqrt{n})} \cdot d = n^{O(\sqrt{n})} \cdot d$ time. We now analyze Step 3.3.3: since 3.3.3 is invoked upon failure of Step 3.3.2, the discovered set $H_j \setminus W_j$ (which is a dominion) has at least size $\sqrt{n}$. Hence this step is executed $\sqrt{n}$ times; the first time on a game graph with $n-1$ states and the range of the priority function p_i being $[2d_i - 2]$, and each subsequent time, with at most $n - \sqrt{n}$ states and the range of p_i being $[2d_i - 2]$. Hence the total work of the loop for the priority function p_i is
$$n^{O(\sqrt{n})} \cdot d + T(n-1, d_1, \ldots, d_i - 1, \ldots, d_k) + \sqrt{n} \cdot T(n - \sqrt{n}, d_1, \ldots, d_i - 1, \ldots, d_k).$$
Thus the total work when Step 2 fails is obtained by summing over $i = 1$ to k, and then adding $T(n - \sqrt{n}, d_1, d_2, \ldots, d_k)$ (the work after Step 3.5.1 on the reduced game graph). Therefore we conclude that the total work when Step 2 fails is
$$\sum_{i=1}^{k} \Big(n^{O(\sqrt{n})} \cdot d + T(n-1, d_1, d_2, \ldots, d_i - 1, \ldots, d_k) \qquad (1)$$
$$+ \sqrt{n} \cdot T(n - \sqrt{n}, d_1, d_2, \ldots, d_i - 1, \ldots, d_k) \Big) + T(n - \sqrt{n}, d_1, d_2, \ldots, d_k).$$

Thus we obtain that $T(n, d_1, d_2, \ldots, d_k) = n^{O(\sqrt{n})} \cdot d + \max\{\mathsf{Term}_1, \mathsf{Term}_2\}$, where $\mathsf{Term}_1 = T(n-1, d_1, d_2, \ldots, d_k)$ (when Step 2 succeeds) and $\mathsf{Term}_2 =$ Expression (1) (when Step 2 fails). If $T(n, d_1, d_2, \ldots, d_k) = n^{O(\sqrt{n})} \cdot d + T(n-1, d_1, d_2, \ldots, d_k)$, then easily we obtain that $T(n, d_1, d_2, \ldots, d_k) = n^{O(\sqrt{n})} \cdot d \cdot n = n^{O(\sqrt{n})} \cdot d$. We now analyze the recurrence $T(n, d_1, d_2, \ldots, d_k) = n^{O(\sqrt{n})} \cdot d + \mathsf{Term}_2$, where Term_2 is the Expression (1). The following lemmas analyze the recurrence. Lemma 3 follows by induction.

Lemma 3. *Consider the following recurrence: $T(n, d_1, d_2, \ldots, d_k)$ is $n^{O(\sqrt{n})} \cdot d +$ (1) if $n \geq 2$, and $\binom{d}{d_1, d_2, \ldots, d_k}$ otherwise. Then $T(n, d_1, d_2, \ldots, d_k) \leq n^{O(\sqrt{n})} \cdot k \cdot d \cdot \binom{d}{d_1, d_2, \ldots, d_k} \cdot t(n)$, where $t(n)$ is $1 + t(n-1) + (\sqrt{n} + 1) \cdot t(n - \sqrt{n})$ if $n \geq 2$, and 1 otherwise.*

We now show that the recurrence $t(n) = 1 + t(n-1) + (\sqrt{n} + 1) \cdot t(n - \sqrt{n})$ satisfies the bound that $t(n) = n^{O(\sqrt{n})}$. In [12] a similar recurrence was analyzed. In [12] the recurrence $t(n) = 1 + t(n-1) + t(n - \sqrt{n})$ was proved to satisfy the bound $n^{O(\sqrt{n})}$. In the next lemma we show that the bound of [12] can be proved also for our recurrence.

Lemma 4. *Consider the following recurrence:* $t(n)$ *is* $1 + t(n-1) + (\sqrt{n}+1) \cdot t(n-\sqrt{n})$ *if* $n \geq 2$, *and* 1 *otherwise. Then* $t(n) = n^{O(\sqrt{n})}$.

Proof. To bound $t(n)$ we will analyze the following tree:

1. there is a root labeled n (this correspond to the term 1 of the recurrence);
2. if $n > 1$, then it has a *left* child labeled $n-1$ and the sub-tree of $t(n-1)$ is attached to this child (this correspond to the term $t(n-1)$ of the recurrence);
3. if $n > 1$, then it has $(\lceil\sqrt{n}\rceil + 1)$ *right* children labeled $n - \lfloor\sqrt{n}\rfloor$ and the sub-tree of $t\left(n - \lfloor\sqrt{n}\rfloor\right)$ is attached to each of the right children (this correspond to the term $(\sqrt{n}+1) \cdot t(n-\sqrt{n})$ of the recurrence). For simplicity we will drop the ceilings $\lceil\cdot\rceil$ and floors $\lfloor\cdot\rfloor$ below.

The number of nodes in the tree is a bound for our recurrence. We now bound the number of the nodes in the tree. A node in the tree with no sub-tree is referred as a *leaf.*

Length of a path. Any path in the tree from root down to a leaf has length at most n (as the label decrease by at least 1 at every step).

Right children in a path. We now bound the number right children on a path from the root down to a leaf. Consider a path from the root to a leaf and we consider the number of right children possible in a segment of the path between label k and $\frac{k}{2}$. For every choice of a right children appear in this segment the label goes down by at least $\sqrt{\frac{k}{2}}$; and hence the number of possible right children in this segment is at most $\frac{\frac{k}{2}}{\sqrt{\frac{k}{2}}} = \sqrt{\frac{k}{2}}$. Hence the number of right children in a path from root to the leaf can be bounded by considering the bound on segments: n to $\frac{n}{2}$; then $\frac{n}{2}$ to $\frac{n}{4}$; then $\frac{n}{4}$ to $\frac{n}{8}$; and so on. This yields the bound $\sqrt{n} \cdot \left(\sum_{i=1}^{\infty} \frac{1}{\sqrt{2^i}} \right) = O(\sqrt{n})$.

The number of paths. We now bound the number of paths in the tree. The length of a path is at most n; there are at most $O(\sqrt{n})$ right children; every choice of a left child in the path is unique and for every choice of a right children there are at most $(\sqrt{n}+1)$ choices (since any node can have at most $(\sqrt{n}+1)$ right children). Hence we obtain the following bound for the number distinct paths $\binom{n}{O(\sqrt{n})} \cdot (\sqrt{n}+1)^{O(\sqrt{n})} = n^{O(\sqrt{n})}$. Hence the desired result follows. ∎

Combining the analysis of the recurrence and the correctness of Algorithm 2, we obtain the following result.

Theorem 4 (Correctness and run time). *Given a game graph* $G = ((S,E),(S_1,S_2))$ *and priority functions* $p_1 : S \to [2d_1], p_2 : S \to [2d_2], \ldots, p_k : S \to [2d_k]$, *the following assertions hold.*

1. *If* W *is the output of Algorithm 2, then* $W = W_2(\text{DisjParity}(\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_k))$, *and* $S \setminus W = W_1(\text{ConjParity}(p_1, p_2, \ldots, p_k))$.
2. *The run time of Algorithm 2 is* $n^{O(\sqrt{n})} \cdot O(k \cdot d) \cdot \binom{d}{d_1, d_2, \ldots, d_k}$, *where* $n = |S|$ *and* $d = \sum_{i=1}^{k} d_i$.

Remark. In the special case of Rabin and Streett objectives with k pairs, the run time of Algorithm 2 is $n^{O(\sqrt{n})} \cdot O(k^2) \cdot k!$. For comparison, the algorithm in [16] works in time $O(m \cdot n^{k+1} \cdot k \cdot k!)$. We conclude that the algorithm presented above is of better complexity when the number of pairs is larger than $\sqrt{n}$.

References

1. D. Bustan and O. Grumberg. Simulation based minimization. In *Conference on Automated Deduction*, *LNCS* 1831, pages 255–270, Springer-Verlag, 2000.
2. K. Chatterjee, T.A. Henzinger, and M. Jurdziński. Games with secure equilibria. In *LICS*, pages 160–169, IEEE, 2004.
3. K. Chatterjee, T.A. Henzinger, and N. Piterman. Generalized parity games. 2006. Technical Report: UC Berkeley EECS-2006-144.
4. D.L. Dill, A.J. Hu, and H. Wong-Toi. Checking for language inclusion using simulation relations. In *CAV*, *LNCS* 575, pages 255–265, Springer-Verlag, 1991.
5. S. Dziembowski, M. Jurdziński, and I. Walukiewicz. How much memory is needed to win infinite games. In *LICS*, pages 99–110, IEEE, 1997.
6. E.A. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *FOCS*, pages 328–337, IEEE, 1988.
7. E.A. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In *FOCS*, pages 368–377, IEEE, 1991.
8. K. Etessami and G. Holzmann. Optimizing Büchi automata. In *CONCUR*, *LNCS* 1877, pages 153–167, Springer-Verlag, 2000.
9. T.A. Henzinger, O. Kupferman, and S. Rajamani. Fair simulation. In *CONCUR*, *LNCS* 1243, pages 273–287, Springer-Verlag, 1997.
10. F. Horn. Streett games on finite graphs. In *Games in Design and Verification*, 2005.
11. M. Jurdziński. Small progress measures for solving parity games. In *STACS* , *LNCS* 1770, pages 290–301, Springer-Verlag, 2000.
12. M. Jurdziński, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. In *SODA*, pages 117–123, ACM/SIAM, 2006.
13. D.A. Martin. Borel determinacy. *Annals of Mathematics*, 65:363–371, 1975.
14. R. Milner. An algebraic definition of simulation between programs. In *Second International Joint Conference on Artificial Intelligence*, pages 481–489, The British Computer Society, 1971.
15. N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *LICS*, pages 255–264, IEEE, 2006.
16. N. Piterman and A. Pnueli. Faster solution of rabin and streett games. In *LICS*, pages 275–284, IEEE, 2006.
17. M.O. Rabin. Decidability of second order theories and automata on infinite trees. *Transaction of the AMS*, 141:1–35, 1969.
18. P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *IEEE Transactions on Control Theory*, 77:81–98, 1989.
19. S. Safra. On the complexity of ω-automata. In *FOCS*, pages 319–327, IEEE, 1988.
20. F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *CAV*, *LNCS* 1855, pages 248–263, Springer-Verlag, 2000.
21. R.S. Streett. Propositional dynamic logic of looping and converse. *Information and Control*, 54:121–141, 1982.
22. W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1–2):135–183, 1998.

Tree Automata with Memory, Visibility and Structural Constraints

Hubert Comon-Lundh[1], Florent Jacquemard[2], and Nicolas Perrin[3]

[1] LSV & ENS Cachan
comon@lsv.ens-cachan.fr
[2] INRIA Futurs & LSV
florent.jacquemard@inria.fr
[3] ENS Lyon
nicolas.perrin@ens-lyon.fr

Abstract. Tree automata with one memory have been introduced in 2001. They generalize both pushdown (word) automata and the tree automata with constraints of equality between brothers of Bogaert and Tison. Though it has a decidable emptiness problem, the main weakness of this model is its lack of good closure properties.

We propose a generalization of the visibly pushdown automata of Alur and Madhusudan to a family of tree recognizers which carry along their (bottom-up) computation an auxiliary unbounded memory with a tree structure (instead of a symbol stack). In other words, these recognizers, called visibly Tree Automata with Memory (VTAM) define a subclass of tree automata with one memory enjoying Boolean closure properties. We show in particular that they can be determinized and the problems like emptiness, inclusion and universality are decidable for VTAM. Moreover, we propose an extension of VTAM whose transitions may be constrained by structural equality and disequality tests between memories, and show that this extension preserves the good closure and decidability properties.

1 Introduction

The control flow of programs with calls to functions can be abstracted as pushdown systems. This allows to reduce some program verification problems to problems (e.g. model-checking) on pushdown automata. When it comes to functional languages with *continuation passing style*, the stack must contain information on continuations and has the structure of a dag (for jumps). Similarly, in the context of asynchronous concurrent programming languages, for two concurrent threads the ordering of return is not determined (synchronized) and these threads can not be stacked. In these cases, the control flow is better modeled as a tree structure rather than a stack. That is why we are interested in tree automata with one memory, which generalize the pushdown (tree) automata, replacing the stack with a tree.

Tree automata with one memory are introduced in [4]. They compute bottom-up on a tree, with an auxiliary memory carrying a tree. Along a computation, at any node of the tree, the memory is updated incrementally from the memory reached at the sons of the node. This update may consist in building a new tree from the memories at the sons (this generalizes a push) or retrieving a subtree of one of the memories at the sons (this generalizes a pop). In addition, such automata may perform equality tests: a transition may be constrained to be performed, only when the memories reached at some of the sons are identical. In this way, tree automata with memory also generalize tree automata with equality tests between brothers [3].

Automata with one memory have been introduced in the context of the verification of security protocols, where the messages exchanged are represented as trees. In the context of (functional or concurrent) programs, the creation of a thread, or a callcc, corresponds to a push, the termination of a thread or a callcc corresponds to a pop. The emptiness problem for such automata is in EXPTIME. However, the class of tree languages defined by such automata is neither closed by intersection nor by complement. This is not surprising as they are strictly more general than context free languages.

On the other hand, Alur and Madhusudan have introduced the notion of visibility for pushdown automata [2], which is a relevant restriction in the context of control flow analysis. With this restriction, determinization is possible and actually the class of languages is closed under Boolean operations.

In this paper, we introduce the new formalism of Visibly Tree Automata with Memory (VTAM), extending on one hand Visibly pushdown languages to trees, including a tree structure instead of a stack (following former approaches [10,15,8]). On the other hand, VTAM restrict tree automata with one memory, imposing a visibility condition on the transitions: each symbol is assigned a given type of action. When reading a symbol, the automaton can only perform the assigned type of action: push or pop.

We first show in Section 3 that VTAM can be determinized, using a proof similar to the proof of [2], and do have the good closure properties. The main difficulty here is to understand what is a good notion of visibility for trees, with memories instead of stacks.

In a second part of the paper (Section 4), we consider VTAM with constraints. Our constraints here are recognizable relations; a transition can be fired only if the memory contents of the sons of the current node satisfy such a relation. We give then a general theorem, expressing conditions on such relations, which ensure the decidability of emptiness. Such conditions are shown to be necessary on one hand, and, on the other hand, we prove that they are satisfied by some examples, including equality tests and structural equality tests. As an intermediate result, we show that, in case of equality tests or structural equality tests, the language of memories that can be reached in a given state is always a regular language. This is a generalization of the well-known result that the set of stack contents in a pushdown automaton is always regular. To prove this, we observe that the memories contents are recognized by a two-way alternating tree automaton with constraints. Then we show, using a saturation strategy, that two-way alternating tree automata with (structural) equality constraints are not more expressive than standard tree automata.

We consider VTAM with structural equality tests, since the determinization and closure properties of Section 3 carry over this generalization, which we show in Section 4.4. Finally, we give in Section 4.5 some examples of languages that can be recognized by VTAM with structural equality and disequality tests: well-balanced binary trees, red-black trees, powerlists...

Generalisations of pushdown automata to trees (both for input and stack) are proposed in [10,15,8]. Our contributions are the generalization of the visibility condition of [2] to such tree automata – our VTAM (without constraints) strictly generalize the VP Languages of [2], and the addition of constraints on the stack contents. The visibly tree automata of [1] use a word stack which is less general than a tree structured memory but the comparison with VTAM is not easy as they are alternating and compute top-down on infinite trees.

2 Preliminaries

Term algebra. A *signature* Σ is a finite set of function symbols with arity, denoted by f, g... We write Σ_n the subset of function symbols of Σ of arity n. Given an infinite set $\mathcal{X}$ of variables, the set of terms built over Σ and $\mathcal{X}$ is denoted $\mathcal{T}(\Sigma, \mathcal{X})$, and the subset of ground terms is denoted $\mathcal{T}(\Sigma)$. The set of variables occurring in a term $t \in \mathcal{T}(\Sigma, \mathcal{X})$ is denoted $vars(t)$. A *substitution* σ is a mapping from $\mathcal{X}$ to $\mathcal{T}(\Sigma, \mathcal{X})$ such that $\{x | \sigma(x) \neq x\}$, the *support* of σ, is finite. The application of a substitution σ to a term t is written $t\sigma$. It is the homomorphic extension of σ to $\mathcal{T}(\Sigma, \mathcal{X})$. The *positions* $Pos(t)$ in a term t are sequences of positive integers (Λ, the empty sequence, is the root position). A subterm of t at position p is written $t|_p$, and the replacement in t of the subterm at position p by u denoted $t[u]_p$.

Rewriting. We assume standard definitions and notations for term rewriting [9]. A *term rewriting system* (TRS) over a signature Σ is a finite set of rewrite rules $\ell \to r$, where $\ell \in \mathcal{T}(\Sigma, \mathcal{X})$ and $r \in \mathcal{T}(\Sigma, vars(\ell))$. A term $t \in \mathcal{T}(\Sigma, \mathcal{X})$ rewrites to s by a TRS $\mathcal{R}$ (denoted $t \to_{\mathcal{R}} s$) if there is a rewrite rule $\ell \to r \in \mathcal{R}$, a position p of t and a substitution σ such that $t|_p = \ell\sigma$ and $s = t[r\sigma]_p$. The transitive and reflexive closure of $\to_{\mathcal{R}}$ is denoted $\xrightarrow[\mathcal{R}]{*}$.

Tree Automata. Following definitions and notation of [5], we consider tree automata which compute bottom-up (from leaves to root) on (finite) ground terms in $\mathcal{T}(\Sigma)$. At each stage of computation on a tree t, a tree automaton reads the function symbol f at the current position p in t and updates its current state, according to f and to the respective states reached at the positions immediately under p in t. Formally, a bottom-up *tree automaton* (TA) $\mathcal{A}$ on a signature Σ as a tuple $(Q, Q_\mathsf{f}, \Delta)$ where Σ is the computation signature, Q is a finite set of nullary state symbols, disjoint from Σ, $Q_\mathsf{f} \subseteq Q$ is the subset of final states and Δ is a set of rewrite rules of the form: $f(q_1, \ldots, q_n) \to q$, where $f \in \Sigma$ and

$q_1, \ldots, q_n \in Q$. A term t is *accepted* by $\mathcal{A}$ in state q iff $t \xrightarrow[\Delta]{*} q$, and the *language* $L(\mathcal{A}, q)$ of $\mathcal{A}$ in state q is the set of ground terms accepted in q. The language $L(\mathcal{A})$ of $\mathcal{A}$ is $\bigcup_{q \in Q_\mathsf{f}} L(\mathcal{A}, q)$ and a set of ground terms is called *regular* if it is the language of a TA.

3 Visibly Tree Automata with Memory

We propose in this section a subclass of the tree automata with one memory [4] which is stable under Boolean operations and has a decidable emptiness problem.

3.1 Definition of VTAM

Tree automata have been extended [4] to carry an unbounded information stored in a tree structure along the states in computations. This information is called *memory* in [4] and will keep this terminology here, and call our recognizers *tree automata with memory* (TAM). For consistency with the above formalisms, the memory contents will be ground terms over a *memory signature* Γ.

Like for TA we consider bottom-up computations of TAM in trees; at each stage of computation on a tree t, a TAM, like a TA, reads the function symbol at the current position p in t and updates its current state, according to the states reached immediately under p. Moreover, a configuration of TAM contains not only a state but also a memory, which is a tree. The current memory is updated according to the respective contents of memories reached in the nodes immediately under p in t.

As above, we use term rewrite systems in order to define the transitions allowed in a TAM. For this purpose, we add an argument to state symbols, which will contain the memory. Hence, a configuration of TAM in state q and whose memory contains is the ground term $m \in \mathcal{T}(\Gamma)$ is represented by the term $q(m)$. We propose below a very general definition of TAM (it differs from the one of [4]) which shall be restricted later on.

Definition 1. *A bottom-up* tree automaton with memory *(TAM) on a signature Σ is a tuple $(\Gamma, Q, Q_\mathsf{f}, \Delta)$ where Γ is a memory signature, Q is a finite set of unary state symbols, disjoint from $\Sigma \cup \Gamma$, $Q_\mathsf{f} \subseteq Q$ is the subset of final states and Δ is a set of rewrite rules of the form $f\big(q_1(m_1), \ldots, q_n(m_n)\big) \to q(m)$ where $f \in \Sigma_n$, $q_1, \ldots, q_n, q \in Q$ and $m_1, \ldots, m_n, m \in \mathcal{T}(\Gamma, \mathcal{X})$.*

The rules of Δ are also called *transition rules.* A term t is *accepted* by $\mathcal{A}$ in state $q \in Q$ and with memory $m \in \mathcal{T}(\Gamma)$ iff $t \xrightarrow[\Delta]{*} q(m)$, and the *language* $L(\mathcal{A}, q)$ and *memory language* $M(\mathcal{A}, q)$ of $\mathcal{A}$ in state q are respectively defined by:

$$L(\mathcal{A}, q) = \big\{t \;\big|\; t \xrightarrow[\Delta]{*} q(m),\ m \in \mathcal{T}(\Gamma)\big\}$$
$$M(\mathcal{A}, q) = \big\{m \;\big|\; t \xrightarrow[\Delta]{*} q(m),\ t \in \mathcal{T}(\Sigma)\big\}.$$

The language of $\mathcal{A}$ is the union of languages of $\mathcal{A}$ is its final states, denoted: $L(\mathcal{A}) = \bigcup_{q \in Q_\mathsf{f}} L(\mathcal{A}, q)$.

$$\begin{array}{llll}
\mathsf{PUSH} & f_2\big(q_1(y_1), & q_2(y_2)\big) & \to q\big(h(y_1,y_2)\big) \\
\mathsf{POP}_{11} & f_3\big(q_1(h(y_{11},y_{12})), & q_2(y_2)\big) & \to q(y_{11}) \\
 & f_3\big(q_1(\bot), & q_2(y_2)\big) & \to q(\bot) \\
\mathsf{POP}_{12} & f_4\big(q_1(h(y_{11},y_{12})), & q_2(y_2)\big) & \to q(y_{12}) \\
 & f_4\big(q_1(\bot), & q_2(y_2)\big) & \to q(\bot) \\
\mathsf{POP}_{21} & f_5\big(q_1(y_1), & q_2(h(y_{21},y_{22}))\big) & \to q(y_{21}) \\
 & f_5\big(q_1(y_1), & q_2(\bot)\big) & \to q(\bot) \\
\mathsf{POP}_{22} & f_6\big(q_1(y_1), & q_2(h(y_{21},y_{22}))\big) & \to q(y_{22}) \\
 & f_6\big(q_1(y_1), & q_2(\bot)\big) & \to q(\bot) \\
\mathsf{INT}_0 & a & & \to q(\bot) \\
\mathsf{INT}_1 & f_7\big(q_1(y_1), & q_2(y_2)\big) & \to q(y_1) \\
\mathsf{INT}_2 & f_8\big(q_1(y_1), & q_2(y_2)\big) & \to q(y_2)
\end{array}$$

where $q_1, \dots, q_n \in Q$, y_1, y_2 are distinct variables of $\mathcal{X}$, $h \in \Gamma_2$, $a \in \Sigma_{\mathsf{INT}_0}$, and every f_i is in the corresponding partition of Σ ($f_2 \in \Sigma_{\mathsf{PUSH}}$, $f_3 \in \Sigma_{\mathsf{POP}_{11}}$, etc).

Fig. 1. VTAM transition categories

Visibility Condition. The above formalism is of course far too expressive. As there are no restrictions on the operation performed on memory by the rewrite rules, one can easily encode a Turing machine as a TAM. We shall now define a decidable restriction called *visibly tree automata with memory* (VTAM).

First, we consider only three main families (later divided into subcategories) of operations on memory. We assume below a computation step at some position p of a term, where memories $m_1, \dots, m_n$ have been reached at the positions immediately below p:

– PUSH: the new current memory m is build with a symbol $h \in \Gamma_n$ *pushed* at the top of memories reached: $f\big(q_1(m_1), \dots, q_n(m_n)\big) \to q\big(h(m_1, \dots, m_n)\big)$. According to the terminology of [2], this corresponds to a *call* move in a program represented by an automaton.

– POP: the new current memory is a subterm of one of the the memories reached: $f\big(\dots, q_i(g(m'_1, \dots, m'_k)), \dots\big) \to q(m'_j)$. This corresponds to a function's *return* in a program.

– INT (internal): the new current memory is one of the memories reached: $f\big(q_1(m_1), \dots, q_n(m_n)\big) \to q(m_i)$. This corresponds to an internal operation (neither call nor return) in a function of a program.

Next, we adhere to the *visibility* condition of [2]. The idea behind this restriction, which was already in [12], is that the symbol read (in a term in our case and [1], in a word in the case of [2]) by an automaton corresponds to an instruction of a program, and hence belongs to one of the three above families (call, return and internal). Indeed, the effect of the execution of a given instruction on the current program state (a stack for [2] or a tree in our case) will always be in the same family. In other words, in this context, the family of the memory operations performed by a transition is completely determined by the function symbol read. We assume from now on for the sake of simplicity that all the symbols of Σ and Γ have either arity 0 or 2. This is not a real restriction, and the results of this paper can be extended straightforwardly to the case of function symbols with other arity. The signature $\varSigma$ is partitioned in eight subsets:

$$\varSigma = \varSigma_{\mathsf{PUSH}} \uplus \varSigma_{\mathsf{POP}_{11}} \uplus \varSigma_{\mathsf{POP}_{12}} \uplus \varSigma_{\mathsf{POP}_{21}} \uplus \varSigma_{\mathsf{POP}_{22}} \uplus \varSigma_{\mathsf{INT}_0} \uplus \varSigma_{\mathsf{INT}_1} \uplus \varSigma_{\mathsf{INT}_2}$$

The eight corresponding transition categories are defined formally in Figure 1. In this figure, $\perp$ is a special constant symbol in $\varGamma$, used to represent an empty memory. Note that the other constant symbols of $\varGamma$ are not relevant since they can not be pushed or popped. Note that each POP rule has a variant which read an empty memory.

Definition 2. *A* visibly tree automaton with memory *(VTAM) on $\varSigma$ is a TAM $(\varGamma, Q, Q_{\mathsf{f}}, \varDelta)$ such that every rule of $\varDelta$ belongs to one of the above categories* PUSH*,* POP_{11}*,* POP_{12}*,* POP_{21}*,* POP_{22}*,* INT_0*,* INT_1*,* INT_2*.*

A VTAM $\mathcal{A}$ is said *complete* if every term of $\mathcal{T}(\varSigma)$ belong to $L(\mathcal{A}, q)$ for at least one state $q \in Q$. Every VTAM can be completed (with a polynomial overhead) by the addition of a trash state. Hence, we shall consider from now on only complete VTAM.

3.2 Determinism

A VTAM $\mathcal{A} = (\varGamma, Q, Q_{\mathsf{f}}, \varDelta)$ is said *deterministic* iff:

- for all $a \in \varSigma_{\mathsf{INT}_0}$ there is at most one rule in $\varDelta$ with left-member a,
- for all $f \in \varSigma_{\mathsf{PUSH}} \cup \varSigma_{\mathsf{INT}_1} \cup \varSigma_{\mathsf{INT}_2}$, for all $q_1, q_2 \in Q$, there is at most one rule in $\varDelta$ with left-member $f\big(q_1(y_1), q_2(y_2)\big)$,
- for all $f \in \varSigma_{\mathsf{POP}_{11}} \cup \varSigma_{\mathsf{POP}_{12}}$ (resp. $\varSigma_{\mathsf{POP}_{21}} \cup \varSigma_{\mathsf{POP}_{22}}$), for all $q_1, q_2 \in Q$ and all $h \in \varGamma$, there is at most one rule in $\varDelta$ with left-member $f\big(q_1(h(y_{11}, y_{12})), q_2(y_2)\big)$ (resp. $f\big(q_1(y_1), q_2(h(y_{21}, y_{22}))\big)$.

Theorem 1. *For every VTAM $\mathcal{A} = (\varGamma, Q, Q_{\mathsf{f}}, \varDelta)$ there exists a deterministic VTAM $\mathcal{A}^{det} = (\varGamma^{det}, Q^{det}, Q_{\mathsf{f}}^{det}, \varDelta^{det})$ such that $L(\mathcal{A}) = L(\mathcal{A}^{det})$, where $|Q^{det}|$ and $|\varGamma^{det}|$ both are $O\big(2^{|Q|^2}\big)$.*

Proof. We follow the technique of [2] for the determinization of VPA: we do a subset construction and postpone the application (to the memory) of PUSH rules, until a matching POP is met. The construction of [2] is extended in order to handle the branching structure of the term read and of the memory.

With the visibility condition, for each symbol read, only one kind of memory operation is possible. This permits a more uniform construction of the rules of $\mathcal{A}^{det}$ for each symbol of $\varSigma$. As we shall see below, $\mathcal{A}^{det}$ wont need to keep track of the contents of memory (of $\mathcal{A}$) during its computation, it will only need to memorize information on the reachability of states of $\mathcal{A}$, following the path from the position of the PUSH symbol which has pushed the top symbol of the current memory (let us call it the *last-memory-push-position*) to the current position in the term. We let :

$$Q^{det} := \{0, 1\} \times \mathcal{P}(Q) \times \mathcal{P}(Q^2)$$

Q_{f}^{det} is the subset of states whose second component contains a final state of Q_{f}. The first component is a flag indicating whether the memory is currently empty (value 0) or not (value 1). The second component is the subset of states of Q that $\mathcal{A}$ can reach at current position, and the third component is a binary relation on Q which contains (q, q') iff starting from a state q and memory m at the last-memory-push-position, $\mathcal{A}$ can reach the current position in state q', and with the same memory m.

INT. For every $f \in \Sigma_{\mathsf{INT}_1}$, we have the following rules in Δ^{det}:

$$f\big(\langle b_1, R_1, S_1\rangle(y_1), \langle b_2, R_2, S_2\rangle(y_2)\big) \to \langle b_1, R, S\rangle(y_1)$$

where $R = \big\{q \bigm| \exists q_1 \in R_1, q_2 \in R_2, f\big(q_1(y_1), q_2(y_2)\big) \to q(y_1) \in \Delta\big\}$ and S is the update of S_1 according to the INT_1-transitions of Δ (when $b_1 = 1$, the case $b_1 = 0$ is similar):

$$S := \big\{(q, q') \bigm| \exists q_1 \in Q, q_2 \in R_2, (q, q_1) \in S_1 \text{ and } f\big(q_1(y_1), q_2(y_2)\big) \to q'(y_1) \in \Delta\big\}.$$

The case $f \in \Sigma_{\mathsf{INT}_2}$ is similar.

We consider memory symbols made of pairs of states and PUSH symbols:

$$\Gamma^{det} := \big(Q^{det}\big)^2 \times (\Sigma_{\mathsf{PUSH}})$$

PUSH. For every $f \in \Sigma_{\mathsf{PUSH}}$, we have the following rules in Δ^{det}:

$$f\big(\langle b_1, R_1, S_1\rangle(y_1), \langle b_2, R_2, S_2\rangle(y_2)\big) \to \langle 1, R, Id_Q\rangle(p(y_1, y_2))$$

where $R = \big\{q \bigm| \exists q_1 \in R_1, q_2 \in R_2, h \in \Gamma, f\big(q_1(y_1), q_2(y_2)\big) \to q\big(h(y_1, y_2)\big) \in \Delta\big\}$ and Id_Q is $\big\{(q, q) \bigm| q \in Q\big\}$ is used to initialize the memorization of state reachability from the position of the symbol f, and $p := \big\langle\langle b_1, R_1, S_1\rangle, \langle b_2, R_2, S_2\rangle, f\big\rangle$. Note that the two states reached just below the position of application of this rule are pushed on the top of the memory. They will be used later in order to update R and S when a matching POP symbol is read.

POP. For every $f \in \Sigma_{\mathsf{POP}_{11}}$, we have the following rules in Δ^{det}:

$$f\big(\langle b_1, R_1, S_1\rangle(h(y_{11}, y_{12})), \langle b_2, R_2, S_2\rangle(y_2)\big) \to \langle b, R, S\rangle(y_{11})$$

where $h = \langle Q_1, Q_2, g\rangle$, with $Q_1 = \langle b_1', R_1', S_1'\rangle \in Q^{det}$, $Q_2 = \langle b_2', R_2', S_2'\rangle \in Q^{det}$.

$$\begin{array}{l}
R = \left\{ q \;\middle|\; \begin{array}{l} \exists q_1' \in R_1', q_2' \in R_2', (q_0, q_1) \in S_1, q_2 \in R_2, h \in \Gamma, g\big(q_1'(y_1), q_2'(y_2)\big) \to \\ \quad q_0\big(h(y_1, y_2)\big) \in \Delta, f\big(q_1(h(y_{11}, y_{12})), q_2(y_2)\big) \to q(y_{11}) \in \Delta \end{array} \right\} \\
S = \left\{ (q, q') \;\middle|\; \begin{array}{l} \exists q_1' \in S_1'(q), q_2' \in R_2', (q_0, q_1) \in S_1, q_2 \in R_2, h \in \Gamma, g\big(q_1'(y_1), q_2'(y_2)\big) \\ \to q_0\big(h(y_1, y_2)\big) \in \Delta, f\big(q_1(h(y_{11}, y_{12})), q_2(y_2)\big) \to q'(y_{11}) \in \Delta \end{array} \right\}
\end{array}$$

When a POP symbol is read, the top symbol of the memory, which is popped, contains the states reached just before the application of the matching PUSH. We use this information in order to update $\langle b_1, R_1, S_1\rangle$ and $\langle b_2, R_2, S_2\rangle$ to $\langle b, R, S\rangle$.

The above constructions ensure the three invariants stated above, after the definition of Q^{det} and corresponding to the three components of these states. It follows that $L(\mathcal{A}) = L(\mathcal{A}^{det})$. □

3.3 Closure Properties

The tree automata with one memory of [4] are closed under union but not closed under intersection and complement (even their version without constraints). The visibility condition makes possible these closures for VTAM.

Theorem 2. *The class of tree languages of VTAM is closed under Boolean operations (union, intersection, complement).*

Proof. (sketch, see [7] for the complete constructions). For the union of two VTAM languages, we construct a VTAM whose memory signature, state set, final state set and rules set are the union of the respective memory signatures, state sets, final state sets and rules sets of the two given VTAM.

For the intersection, we construct a VTAM whose memory signature, state set and final state set are the Cartesian product of the respective memory signatures, state sets and final state sets of the two given VTAM. The rule set of the intersection VTAM is obtained by "product" of rules of the two given VTAM with same function symbols. The product of rules means Cartesian products of the respective states and memory symbols pushed or popped. Note that such an operation is possible only because the visibility condition ensures that two rules with the same function symbol in left-side will have the same form. Hence we can synchronise memory operations on the same symbols.

For the complement, we use the construction of Theorem 1 and take the complement of the final state set of the VTAM obtained. □

3.4 Decision Problems

Every VTAM is a particular case of tree automaton with one memory of [4]. Since the emptiness problem (whether the language accepted is empty or not) is decidable for this latter class, it is also decidable for VTAM. In comparison, the emptiness is decidable for nondeterministic visibly pushdown (top-down) tree automata (N-VPTA) of [1] but the class of languages of infinite trees that they define is not closed under complement. The alternating version of these automata (VPTA, [1]) is closed under Boolean operations but has an undecidable emptiness problem. We propose below a proof of decidability of emptiness which follows the same lines as [4].

Theorem 3. *The emptiness problem is decidable in EXPTIME for VTAM. The universality and inclusion problem are decidable in 2-EXPTIME for VTAM.*

Proof. Assume given a VTAM $\mathcal{A} = (\Gamma, Q, Q_{\mathsf{f}}, \Delta)$. By definition, for each state $q \in Q$, the language $L(\mathcal{A}, q)$ is empty iff the memory language $M(\mathcal{A}, q)$ is empty. We show that each $M(\mathcal{A}, q)$ is recognized by an alternating two-way automaton, hence is regular (see e.g. [5]). We can construct in exponential time a TA $\mathcal{A}_q$ of size exponential in the size of $\mathcal{A}$ and accepting $L(\mathcal{A}, q)$. A proof of a more general result will stated in Lemma 1 and can be found in [7].

As usual, a VTAM $\mathcal{A}$ is universal iff the language of its complement automaton $\overline{\mathcal{A}}$ is empty, and $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$ iff $L(\mathcal{A}_1) \cap L(\overline{\mathcal{A}_2}) = \emptyset$. Since these operations require a determinization of a given VTAM first, these problems can be decided in 2-EXPTIME for VTAM. □

4 Visibly Tree Automata with Memory and Constraints

In the late eighties, some models of tree recognizers obtained by adding equality and disequality constraints in transitions of tree automata have been proposed in order to solve problems with term rewrite systems or constraints systems with non-linear patterns (terms with multiple occurrence of the same variable). The tree automata of [3] for instance can perform equality and disequality test between subterms of the term read located at brother positions.

In the case of tree automata with memory, we shall apply constraints to the contents of the memory. Indeed, each step of a bottom-up computation starts with two states and two memories (and ends with one state and one memory), and therefore, it is possible to compare the contents of these two memories, with respect to some binary relation. We state first the general definition of visibly tree automata with constraints on memories, then give sufficient conditions for the emptiness decidability and show some relevant examples which satisfy these conditions. Finally, we study in Section 4.4 the particular case of VTAM with structural equality constraints. They enjoy not only decision properties but also good closure properties.

4.1 Definitions

Assume given a fixed equivalence relation R on $\mathcal{T}(\Gamma)$. We consider now four new categories for the symbols of Σ: INT_1^R, INT_2^R, $\mathsf{INT}_1^{\neg R}$, $\mathsf{INT}_2^{\neg R}$, in addition to the eight previous categories of page 173. The four new categories correspond to the the constrained versions of the transition rules INT_1 and INT_2 presented in Figure 2.

We will not extend the rules PUSH and POP with constraints for some reasons explained below. A ground term t rewrites to s by a constrained rule $f\big(q_1(y_1), q_2(y_2)\big) \xrightarrow{y_1\, c\, y_2} r$ (where c is either R or $\neg R$) if there exists a position p of t and a substitution σ such that $t|_p = \ell\sigma$, $y_1\sigma\, c\, y_2\sigma$ and $s = t[r\sigma]_p$.

For example, if R is term equality, the transition is performed only when the memory contents are identical.

Definition 3. *A visibly tree automaton with memory and constraints (*VTAM$^R_{\neg R}$*) on a signature Σ is a tuple $(\Gamma, R, Q, Q_{\mathsf{f}}, \Delta)$ where Γ, Q, Q_{f} are defined as for TAM, R is an equivalence relation on $\mathcal{T}(\Gamma)$ and Δ is a set of rewrite rules in one of the above categories:* PUSH, POP_{11}, POP_{12}, POP_{21}, POP_{22}, INT_0, INT_1, INT_2, INT_1^R, INT_2^R, $\mathsf{INT}_1^{\neg R}$, $\mathsf{INT}_2^{\neg R}$.

$$\begin{array}{lll}
\mathsf{INT}_1^R & f_9\big(q_1(y_1),\ q_2(y_2)\big) \xrightarrow{y_1\, R\, y_2} q(y_1) \\
\mathsf{INT}_2^R & f_{10}\big(q_1(y_1), q_2(y_2)\big) \xrightarrow{y_1\, R\, y_2} q(y_2) \\
\mathsf{INT}_1^{\neg R} & f_{11}\big(q_1(y_1), q_2(y_2)\big) \xrightarrow{y_1\, \neg R\, y_2} q(y_1) \\
\mathsf{INT}_2^{\neg R} & f_{12}\big(q_1(y_1), q_2(y_2)\big) \xrightarrow{y_1\, \neg R\, y_2} q(y_2)
\end{array}$$

Fig. 2. New transition categories for VTAM$^R_{\neg R}$

We denote VTAM^R the subclass of $\mathrm{VTAM}^R_{\neg R}$ with positive constraints only, i.e. without transition rules in $\mathsf{INT}_1^{\neg R}$ or $\mathsf{INT}_2^{\neg R}$. The acceptance of terms of $\mathcal{T}(\Sigma)$ and languages of term and memories are defined and denoted as in Section 3.1.

The definition of *deterministic* $\mathrm{VTAM}^R_{\neg R}$ is based on the same conditions as for VTAM for the function symbols in categories of PUSH_0, PUSH, POP_{11}, $\ldots$, POP_{22}, INT_1, INT_2, and for the function symbols of INT_1^R, INT_2^R, $\mathsf{INT}_1^{\neg R}$, $\mathsf{INT}_2^{\neg R}$, we use the same conditions as for INT_1, INT_2: for all $f \in \Sigma_{\mathsf{INT}_1^R} \cup \Sigma_{\mathsf{INT}_2^R} \cup \Sigma_{\mathsf{INT}_1^{\neg R}} \cup \Sigma_{\mathsf{INT}_2^{\neg R}}$, for all $q_1, q_2 \in Q$, there is at most one rule in Δ with left-member $f\big(q_1(y_1), q_2(y_2)\big)$.

4.2 Emptiness Decision

We propose here a generic theorem for emptiness decision. The idea of this theorem is that under some condition on R, the transition rules with negative constraints can be eliminated.

Theorem 4. *Let R be an equivalence relation satisfying these two properties:*

i. for all automaton $\mathcal{A}$ of VTAM^R *and for all state q of $\mathcal{A}$, the memory language $M(\mathcal{A}, q)$ is a regular tree language,*
ii. the size of every equivalence class of R is bounded, and its elements can be enumerated.

Then the emptiness problem is decidable for $\mathrm{VTAM}^R_{\neg R}$.

Proof. Let $\mathcal{A} = (\Gamma, R, Q, Q_\mathsf{f}, \Delta)$ be a $\mathrm{VTAM}^R_{\neg R}$. We show in [7] that there exists a VTAM^R $A^+ = (\Gamma, R, Q^+, Q_\mathsf{f}, \Delta^+)$ such that $Q \subseteq Q^+$, and for each $q \in Q$, $M(\mathcal{A}^+, q) = M(\mathcal{A}, q)$. The proof is by induction on the number n of rules with negative constraints (*i.e.* rules in categories $\mathsf{INT}_1^{\neg R}$ and $\mathsf{INT}_2^{\neg R}$) in Δ and uses the bound on the size of equivalence classes, condition *ii* of the theorem.

It follows from the condition *i.* of the theorem that emptiness is decidable for $\mathcal{A}$, since by definition $L(\mathcal{A}, q)$ is empty iff $M(\mathcal{A}, q)$ is empty. □

We will see soon (Section 4.4) two examples of relations satisfying *i.* and *ii.*

4.3 Regular Tree Relations

We first consider the general case where the equivalence R is based on an arbitrary regular binary relation on $\mathcal{T}(\Gamma)$. By regular binary relation, we mean a set of pairs of ground terms accepted by a tree automaton computing simultaneously in both terms of the pair. More formally, we use a coding of a pair of terms of $\mathcal{T}(\Sigma)$ into a term of $\mathcal{T}\big((\Sigma \cup \{\bot\})^2\big)$, where $\bot$ is a new constant symbol (not in Σ). This coding is defined recursively by:

$\otimes : \mathcal{T}(\Sigma) \cup \{\bot\} \times \mathcal{T}(\Sigma) \cup \{\bot\} \to \mathcal{T}\big((\Sigma \cup \{\bot\})^2\big)$
for all $a, b \in \Sigma_0 \cup \{\bot\}$, $a \otimes b := \langle a, b\rangle$,
for all $a \in \Sigma_0 \cup \bot$, $f \in \Sigma_2$, $t_1, t_2 \in \mathcal{T}(\Sigma)$, $f(t_1, t_2) \otimes a := \langle f, a\rangle(t_1 \otimes \bot, t_2 \otimes \bot)$
$a \otimes f(t_1, t_2) := \langle a, f\rangle(\bot \otimes t_1, \bot \otimes t_2)$,
f. a. $f, g \in \Sigma_2$, $s_1, s_2, t_1, t_2 \in \mathcal{T}(\Sigma)$, $f(s_1, s_2) \otimes g(t_1, t_2) := \langle f, g\rangle(s_1 \otimes t_1, s_2 \otimes t_2)$.

Then, a binary relation $R \subseteq \mathcal{T}(\Sigma) \times \mathcal{T}(\Sigma)$ is called regular iff the set $\{s \otimes t \mid (s,t) \in R\}$ is regular.

The class of $\mathrm{VTAM}^{R}_{\neg R}$ when R is a binary regular tree relation constitutes a nice and uniform framework. Note however the condition ii. of Theorem 4 is not always true in this case. Actually, it is too expressive.

Theorem 5. *The emptiness problem is undecidable for* VTAM^R *with some* R *based on a regular binary relation.*

Proof. We reduce the blank accepting problem for a deterministic Turing machine $\mathcal{M}$. We encode configurations of $\mathcal{M}$ as "right-combs" (binary trees) built with the tape and state symbols of $\mathcal{M}$, in Σ_{PUSH} (hence binary) and a constant symbol ε in Σ_{INT_0}. Let R be the regular relation which accepts all the pairs of configurations $c \otimes c'$ such that c' is a successor of c by $\mathcal{M}$. A sequence of configurations $c_0 c_1 \ldots c_n$ (with $n \geq 1$) is encoded as a tree $t = f(c_0(f(c_1, \ldots f(c_{n-1}, c_n)))$, where f is a binary symbol of $\Sigma_{\mathsf{INT}_1^R}$.

We construct a VTAM^R $\mathcal{A}$ which accepts exactly the term-representations t of computation sequences of $\mathcal{M}$ starting with the initial configuration c_0 of $\mathcal{M}$ and ending with is a final configuration c_n with blank tape. Following the type of the function symbols, the rules of $\mathcal{A}$ will push all the symbols read in subterms of t corresponding to configurations and a transition applied at the top of a subterm $f(c_i, f(c_{i+1}, \ldots))$ will compare, with R, c_i and c_{i+1} (the memory contents in respectively the left and right branches) and store c_i in the memory. This way, $\mathcal{A}$ checks that successive configurations in t correspond to transitions of $\mathcal{M}$, hence that the language of $\mathcal{A}$ is not empty iff $\mathcal{M}$ accepts the initial configuration c_0. □

4.4 Syntactic and Structural Equality and Disequality Constraints

We present now two examples of relations satisfying the conditions of Theorem 4. These results will be proved with the following crux Lemma.

Lemma 1. *Let R be a regular binary relation defined by a TA whose state set is $\{R_i \mid i = \{1..n\}\}$ and such that $\forall i, j \exists k, l \; R_i(x,y) \wedge R_j(y,z) \models\!\mid R_k(x,y) \wedge R_l(x,z)$. Let $\mathcal{A} = (\Gamma, R, Q, Q_{\mathsf{f}}, \Delta)$ be a tree automaton with memory and constraints (not necessarily visibly). Then for every $q \in Q$, $M(\mathcal{A}, q)$ is regular.*

Proof. (Sketch, the complete proof can be found in [7]). We first observe that $M(\mathcal{A}, q)$ is the interpretation of q in the least Herbrand model of a set of Horn clauses computed from the rules Δ. We saturate this set of clauses by resolution with an selection and eager splitting. This saturation terminates, and the set of clauses corresponding to alternating automata transitions in the saturated set recognizes the language $M(\mathcal{A}, q)$, which is therefore regular. □

We first apply Lemma 1 to the class $\mathrm{VTAM}^{=}_{\neq}$ where $=$ denotes the equality between ground terms made of memory symbols.

Corollary 1. *The emptiness problem is decidable for* $\mathrm{VTAM}^{=}_{\neq}$.

Lemma 1 applies also to another class $\mathrm{VTAM}^{\equiv}_{\not\equiv}$, where $\equiv$ denotes structural term equality, defined recursively as the smallest equivalence relation ground terms such that:

– $a \equiv b$ for all a, b of arity 0,
– $f(s_1, s_2) \equiv g(t_1, t_2)$ if $s_1 \equiv t_1$ and $s_2 \equiv t_2$, for all f, g of arity 2.

Note that it is a regular relation.

Corollary 2. *The emptiness problem is decidable for* $\mathrm{VTAM}^{\equiv}_{\not\equiv}$.

A nice property of $\mathrm{VTAM}^{\equiv}_{\not\equiv}$ is that the construction for determinization of Section 3.2 still works for this class.

Theorem 6. *For every* $\mathrm{VTAM}^{\equiv}_{\not\equiv}$ $\mathcal{A} = (\Gamma, \equiv, Q, Q_\mathsf{f}, \Delta)$ *there exists a deterministic* $\mathrm{VTAM}^{\equiv}_{\not\equiv}$ $\mathcal{A}^{det} = (\Gamma^{det}, \equiv, Q^{det}, Q^{det}_\mathsf{f}, \Delta^{det})$ *such that* $L(\mathcal{A}) = L(\mathcal{A}^{det})$*, where* $|Q^{det}|$ *and* $|\Gamma^{det}|$ *both are* $O\big(2^{|Q|^2}\big)$.

Proof. We use the same construction as in the proof of Theorem 1, with a direct extension of the construction for INT to $\mathsf{INT}^{\equiv}$ or $\mathsf{INT}^{\not\equiv}$. The key property for handling constraints is that the structure of memory (hence the result of the structural tests) is independent from the non-deterministic choices of the automaton. With the visibility condition it only depends on the term read. □

Theorem 7. *The class of tree languages of* $\mathrm{VTAM}^{\equiv}_{\not\equiv}$ *is closed under Boolean operations.*

Proof. We use the same constructions as in Theorem 2 (VTAM) for union and intersection. For the intersection, in the case of constrained rules we can safely keep the constraints in product rules, thanks to the visibility condition (as the structure of memory only depends on the term read, see the proof of Theorem 6). For instance, the product of the $\mathsf{INT}^{\equiv}_{1}$ rules $f_9\big(q_{11}(y_1), q_{12}(y_2)\big) \xrightarrow{y_1 \equiv y_2} q_1(y_1)$ and $f_9\big(q_{21}(y_1), q_{22}(y_2)\big) \xrightarrow{y_1 \equiv y_2} q_1(y_1)$, is $f_9\big(\langle q_{11}, q_{21}\rangle(y_1), \langle q_{12}, q_{22}\rangle(y_2)\big) \xrightarrow{y_1 \equiv y_2} \langle q_1, q_2\rangle(y_1)$. For the complementation, we use Theorem 6. □

Corollary 3. *The universality and inclusion problems are decidable for* $\mathrm{VTAM}^{\equiv}_{\not\equiv}$.

Proof. This is a consequence of Corollary 3 and Theorem 7. □

Constrained PUSH transitions. We did not consider a constrained extension of the rules PUSH. The main reason is that symbols of a new category $\mathsf{PUSH}^{\equiv}$, which test two memories for structural equality and then push a symbol on the top of them, permit construct a constrained VTAM $\mathcal{A}$ whose memory language $M(\mathcal{A}, q)$ is the set of well-balanced binary trees. This language is not regular, whereas the base of our emptiness decision procedure is the result (Theorem 4, Lemma 1) of regularity of these languages for the classes considered.

4.5 Some VTAM$^{\equiv}_{\not\equiv}$ Languages

The regular tree languages and VPL are particular cases of VTAM languages. In some cases, the tree automata with equality and disequality tests between brothers [3] can be simulated by VTAM$^{=}_{\neg=}$ which push all the symbol read up to (dis)equality tests. We present in this final section some other relevant examples of VTAM$^{\equiv}_{\not\equiv}$ languages.

Well balanced binary trees. The VTAM$^{\equiv}_{\not\equiv}$ with memory alphabet $\{f, \bot\}$, state set $\{q, q_\mathsf{f}\}$, unique final state q_f, and whose rules follow accepts the (non-regular) language of well balanced binary trees build with g (binary, in $\Sigma_{\mathsf{INT}^{\equiv}_{1}}$), f (binary, in Σ_{PUSH}) and a (constant in Σ_{INT_0}) with a g at the root position and only f's and a's below.

$$\begin{array}{lrcl} a \to q(\bot) & f\big(q(y_1), q(y_2)\big) & \to & q\big(f(y_1, y_2)\big) \\ & g\big(q(y_1), q(y_2)\big) & \xrightarrow{y_1 \equiv y_2} & q_\mathsf{f}(y_1) \end{array}$$

Powerlists. A powerlist [14] is roughly a list of length 2^n (for $n \geq 0$) whose elements are stored in the leaves of a balanced binary tree. This data structure has been used in [14] to specify data-parallel algorithms based on divide-and-conquer strategy and recursion (*e.g.* Batcher's merge sort and fast Fourier transform).

The following VTAM$^{\equiv}_{\not\equiv}$ with memory alphabet $\{f, \bot\}$, state set $\{q, q_\mathsf{f}\}$ and unique final state q_f and whose rules follow accepts the language of powerlists of natural numbers presented in unary notation with the symbol s (binary, in Σ_{INT_2}) and 0 (constant in Σ_{INT_0}). We use artificially a successor symbol s of arity 2 instead of 1 as usual, because of the assumption that $\Sigma = \Sigma_0 \uplus \Sigma_2$ in Section 3.1 (2 for instance is written $s(0, s(0, 0))$). The other symbols are f (binary, in Σ_{PUSH}), and g (binary, in $\Sigma_{\mathsf{INT}^{\equiv}_{1}}$), used for the root of powerlist only (as above). The rules of the VTAM$^{\equiv}_{\not\equiv}$ are the following:

$$\begin{array}{rrcl} 0 \to q_0(\bot) & f\big(q_0(y_1), q_0(y_2)\big) & \to & q\big(f(y_1, y_2)\big) \\ s(q_0(y_1), q_0(y_2)) \to q_0(y_2) & f\big(q(y_1), q(y_2)\big) & \to & q\big(f(y_1, y_2)\big) \\ & g\big(q(y_1), q(y_2)\big) & \xrightarrow{y_1 \equiv y_2} & q_\mathsf{f}(y_1) \end{array}$$

Note that only the f symbol is pushed on the memory. Therefore, only the upper structure of the powerlist is saved in the memory and tested at root position for structural equality. This way, we ensure that this upper part is well balanced, hence that the list has length 2^n.

Some equational properties of algebraic specifications of powerlists have been studied in the context of automatic induction theorem proving and sufficient completeness [13]. Tree automata with constraints have been acknowledged as a very powerful formalism in this context (see *e.g.* [6]). We therefore believe that a characterisation of powerlist (and the complement language) as VTAM$^{\equiv}_{\not\equiv}$ for the automated verification of algorithms on this data structure.

Red-black trees. A red-black is a binary search tree following these properties:

1. every node is either red or black,
2. the root node is black,

3. all the leaves are black,
4. if a node is red, then both its sons are black,
5. every path from the root to a leaf contains the same number of black nodes.

The four first properties are local and can be check with standard TA rules. The fifth property make the language red-black trees not regular and we need $\mathrm{VTAM}^{\equiv}_{\not\equiv}$ rules to recognize it. It can be checked by pushing all the black nodes read, we use for this purpose a symbol *black* $\in \Sigma_{\mathsf{PUSH}}$. When a red node is read, the number of black nodes in both its sons are check to be equal (by a test $\equiv$ on the corresponding memories) and only one corresponding memory is kept. This is done with a symbol *red* $\in \Sigma_{\mathsf{INT}^{\equiv}_{1}}$. When a red node is read, the equality of number of black nodes in its sons must also be tested, and a *black* must moreover be pushed on the top of the memory kept. The structure test is done with an auxiliary symbol *aux* $\in \Sigma_{\mathsf{INT}^{\equiv}_{1}}$, located just above the *black* symbol. It means that the $\mathrm{VTAM}^{\equiv}_{\not\equiv}$ recognizes not exactly the red-black tree but a representation with additional nodes. This can be considered as already satisfying in the context of verification. In [11] a special class of tree automata is introduced and used in a procedure for the verification of C programs which handle balanced tree data structures, like red-black tree. Based on the above example, we think that, following the same approach, $\mathrm{VTAM}^{\equiv}_{\not\equiv}$ can also be used for similar purposes.

5 Conclusion

Having a tree memory structure instead of a stack is sometimes more relevant (even when the input functions symbols are only of arities 1 and 0). We have shown how to extend the visibly pushdown languages to such memory structures, keeping determinization and closure properties of VPL. Our main contribution is then to extend this automaton model, constraining the transition rules with some regular conditions, while keeping decidability results. The structural equality and disequality tests appear to a be a good constraint class since we have then both decidability of emptiness and Boolean closure properties.

Acknowledgments

The authors wish to thank the reviewers for their useful remarks and suggestions.

References

1. R. Alur, S. Chaudhuri, and P. Madhusudan. Visibly pushdown tree languages. Available on: http://www.cis.upenn.edu/~swarat/pubs/vptl.ps, 2006.
2. R. Alur and P. Madhusudan. Visibly pushdown languages. In L. Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC 2004)*, pages 202–211. ACM, 2004.
3. B. Bogaert and S. Tison. Equality and Disequality Constraints on Direct Subterms in Tree Automata. In *9th Symp. on Theoretical Aspects of Computer Science, STACS*, volume 577 of *LNCS*, pages 161–171. Springer, 1992.

4. H. Comon and V. Cortier. Tree automata with one memory, set constraints and cryptographic protocols. *Theoretical Computer Science*, 331(1):143–214, Feb. 2005.
5. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications.* `http://www.grappa.univ-lille3.fr/tata`, 1997.
6. H. Comon and F. Jacquemard. Ground reducibility is exptime-complete. *Information and Computation*, 187(1):123–153, 2003.
7. H. Comon-Lundh, F. Jacquemard, and N. Perrin. Tree automata with memory, visibility and structural constraints. Technical Report LSV-07-01, Laboratoire Spécification et Vérification, Jan. 2007. 25 pages.
8. J.-L. Coquidé, M. Dauchet, R. Gilleron, and S. Vágvölgyi. Bottom-up tree pushdown automata: classification and connection with rewrite systems. *Theoretical Computer Science*, 127(1):69–98, 1994.
9. N. Dershowitz and J.-P. Jouannaud. *Rewrite systems*, chapter Handbook of Theoretical Computer Science, Volume B, pages 243–320. Elsevier, 1990.
10. I. Guessarian. Pushdown tree automata. *Theory of Computing Systems*, 16(1):237–263, 1983.
11. P. Habermehl, R. Iosif, and T. Vojnar. Automata-based verification of programs with tree updates. In *Proc. 12th Intern. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *LNCS*, April 2006.
12. T. Jensen, D. L. Métayer, and T. Thorn. Verification of control flow based security policies. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 89–103. IEEE Computer Society Press, 1999.
13. D. Kapur. *Essays in Honor of Larry Wos*, chapter Constructors can be Partial Too. MIT Press, 1997.
14. J. Misra. Powerlist: A structure for parallel recursion. *ACM Transactions on Programming Languages and Systems*, 16(6):1737–1767, November 1994.
15. K. M. Schimpf and J. Gallier. Tree pushdown automata. *Journal of Computer and System Sciences*, 30(1):25–40, 1985.

Enriched μ-Calculi Module Checking[⋆]

Alessandro Ferrante[1] and Aniello Murano[2]

[1] Università di Salerno, Via Ponte don Melillo, 84084 - Fisciano (SA), Italy
[2] Università di Napoli Federico II, Via Cintia, 80126 - Napoli, Italy

Abstract. The model checking problem for open finite-state systems (called *module checking*) has been intensively studied in the literature with respect to *CTL* and *CTL**. In this paper, we focus on module checking with respect to the *fully enriched μ-calculus* and some of its fragments. Fully enriched μ-calculus is the extension of the propositional μ-calculus with *inverse programs*, *graded modalities*, and *nominals.* The fragments we consider here are obtained by dropping at least one of the additional constructs. For the full calculus, we show that module checking is undecidable by using a reduction from the domino problem. For its fragments, instead, we show that module checking is decidable and ExpTime-complete. This result is obtained by using, for the upper bound, a classical automata-theoretic approach via *Forest Enriched Automata* and, for the lower bound, a reduction from the module checking problem for *CTL*, known to be ExpTime-hard.

1 Introduction

One of the most significant developments in the area of formal design verification has been the discovery of the *model-checking* technique, which is particularly suitable for verifying ongoing behaviors of reactive systems ([CE81, QS81, VW86]). In this verification method, (for a survey, see [CGP99]), the behavior of a system, formally described by a mathematical model, is checked against a behavioral constraint specified by a formula in a suitable temporal logic, which enforces either a linear model of time (formulas are interpreted over linear sequences corresponding to single computations of the system) or a branching model of time (formulas are interpreted over infinite trees, which describe all the possible computations of the system).

In system modeling, we distinguish between *closed* and *open* systems [HP85]. For a closed system, the behavior is completely determined by the state of the system. For an open system, the behavior is affected both by its internal state and by the ongoing interaction with its environment. Thus, while in a closed system all the nondeterministic choices are internal, and resolved by the system, in an open system there are also external nondeterministic choices, which are resolved by the environment [Hoa85]. Model checking algorithms used for the verification of closed systems are not appropriate for the verification of open systems. In the

⋆ Work partially supported by MIUR FIRB Project no. RBAU1P5SS and grant ex-60% 2005, Università di Salerno.

latter case, we should check the system with respect to arbitrary environments and should take into account uncertainty regarding the environment.

In [KVW01], Kupferman, Vardi, and Wolper extend model checking from closed finite-state systems to open finite-state systems. In such a framework, the open finite-state system is described by a labeled state-transition graph called *module* whose set of states is partitioned into a set of *system states* (where the system makes a transition) and a set of *environment states* (where the environment makes a transition). The problem of model checking a module (called *module checking*) has two inputs: a module M and a temporal formula φ. The idea is that an open system should satisfy a specification φ no matter how the environment behaves. Let us consider the unwinding of M into an infinite tree, say T_M. Checking whether T_M satisfies φ, (formally, $M \models \varphi$) is the usual *model-checking problem* [CE81, QS81]. On the other hand, for an open system, T_M describes the interaction of the system with a maximal environment, i.e., an environment that enables all the external nondeterministic choices. In order to take into account all the possible behaviors of the environment, we have to consider all the trees T obtained from T_M by pruning subtrees whose root is a successor of an environment state (pruning these subtrees correspond to disable possible environment choices). Therefore, a module M satisfies φ (formally, $M \models_r \varphi$, where r stands for "reactively") if all these trees T satisfy φ. The set of all the trees derived from T_M by a legal pruning is denoted by $exec(M)$.

In [KVW01], it has been showed that model checking for open finite-state systems is ExpTime-complete for specification in *CTL* and 2ExpTime-complete for specification in *CTL**. Moreover, the *program complexity*, i.e., the complexity of the problem assuming the formula to be fixed, is Ptime-complete. Recently, module checking has been also extended to infinite–state systems, by considering open pushdown systems as models [BMP05]. It has been showed that in this framework module checking is 2ExpTime-complete for specification in *CTL* and 3ExpTime-complete for specification in *CTL**.

The *μ-calculus* is a propositional modal logic augmented with least and greatest fixpoint operators [Koz83]. It is often used as a target formalism for embedding temporal and modal logics with the goal of transferring computational and model theoretic properties such as the ExpTime upper complexity bound (see [BS06] for a survey). *Fully enriched μ-calculus* is the extension of the propositional μ-calculus with *inverse programs*, *graded modalities*, and *nominals.* Intuitively, inverse programs allow to travel backwards along accessibility relations [Var98], nominals are propositional variables interpreted as singleton sets [SV01], and graded modalities enable statements about the number of successors and predecessors of a state [KSV02]. In [BP04], Bonatti and Peron showed that satisfiability is undecidable in the *fully enriched μ-calculus.* On the other hand, the satisfiability problem for interesting fragments of the fully enriched μ-calculus has been showed to be decidable and ExpTime-complete. In particular, it has been showed for the fragments of the fully enriched μ-calculus obtained by dropping at least one of graded modalities (*fully hybrid μ-calculus*)[SV01], nominals (*full graded μ-calculus*) [BLMV06], and inverse programs (*hybrid graded*

	Inverse progr.	Graded mod.	Nominals	Complexity
fully enriched μ-calculus	x	x	x	undecidable
full graded μ-calculus	x	x		ExpTime
full hybrid μ-calculus	x		x	ExpTime
hybrid graded μ-calculus		x	x	ExpTime

Fig. 1. Enriched μ-calculi and known results

μ-calculus)[BLMV06]. These enriched μ-calculi are shown in Fig. 1 together with the complexity of their satisfiability problem.

The above decidability results are based on an automata-theoretic approach via *fully enriched automata (FEAs)*, which run on infinite forests and use a parity acceptance condition. Intuitively, these automata generalize alternating automata on infinite trees in a similar way as the fully enriched μ-calculus extends the standard μ-calculus: FEAs can move up to a node's predecessor (by analogy with inverse programs), move down to at least n or all but n successors (by analogy with graded modalities), and jump directly to the roots of the input forest (which are the analogues of nominals). The decidability results follow from the fact that all the above fragments enjoy the *forest model property* (while some of them do not enjoy neither the tree model property nor the finite model property), and from the fact that the emptiness problem for fully enriched automata is decidable and ExpTime-complete. Observe that decidability of the emptiness problem for FEAs does not contradict the undecidability of the fully enriched μ-calculus: the latter does not enjoy a forest model property [BP04], and hence satisfiability cannot be decided using forest-based FEAs.

In this paper, we extend the module checking problem for finite-state systems to the *fully enriched μ-calculus* and we show that this problem is undecidable. To gain this result, we use a reduction from the domino problem [Ber66], known to be undecidable, by extending an idea due to Bonatti and Peron in [BP04].

Moreover, we consider the problem of module checking for the fragments of the full calculus as listed in Fig. 1. That is, we consider the module checking problem whit respect to formulas of the fully hybrid, full graded, and hybrid graded μ-calculus. We show that in all the above frameworks, the module checking problem is decidable and ExpTime-complete. For the upper bound, we use an automata-theoretic approach via FEA. In more details, given a model M and a formula φ, we first build a Büchi automaton $\mathcal{A}_M$, accepting $exec(M)$. In particular, since M requires to be unwound in a forest rather then a tree (since all the fragments we consider enjoy the forest model property, while those including nominals do not enjoy the tree model property), the set $exec(M)$ is a set of forests, and thus, $\mathcal{A}_M$ is a Büchi automaton running on forests (BFA, for short). Then, accordingly to [SV01] and [BLMV06], we build a FEA $\mathcal{A}_{\neg\varphi}$ accepting all models of $\neg\varphi$, with the intent to check that no models of $\neg\varphi$ are in $exec(M)$. Thus, we check that $M \models_r \varphi$ by checking whether $\mathcal{L}(\mathcal{A}_M) \cap \mathcal{L}(\mathcal{A}_{\neg\varphi})$ is empty. The results follow from the fact that BFAs are a particular case of FEAs, which are closed under intersection and have the emptiness problem solvable in ExpTime [BLMV06].

We also show a lower bound matching the obtained upper bound by using a reduction from the module checking for *CTL*, known to be ExpTime-hard.

2 Preliminaries

Labeled Forests. For a finite set X, we denote the set of finite words over X by X^*, the empty word by ε, and with X^+ we denote $X^* \setminus \{\varepsilon\}$. Given a word w in X^* and a symbol x of X, we use $w \cdot x$ to denote the word wx. Let $\mathbb{N}$ be the set of positive integers. For $n \in \mathbb{N}$, let $\mathbb{N}$ be denote the set $\{1, 2, \ldots, n\}$. A *forest* is a set $F \subseteq \mathbb{N}^+$ such that if $x \cdot c \in F$ where $x \in \mathbb{N}^+$ and $c \in \mathbb{N}$, then also $x \in F$. The elements of F are called *nodes*, and the strings consisting of a single natural number are the *roots* of F. For each root $r \in F$, the set $T = \{r \cdot x \mid x \in \mathbb{N}^* \text{ and } r \cdot x \in F\}$ is a *tree* of F (the tree *rooted in* r). For every $x \in F$, the nodes $x \cdot c \in F$ where $c \in \mathbb{N}$ are the *successors* of x, denoted $children(x)$, and x is their *predecessor*. The number of successors of a node x is called the *branching degree* of x, and is denoted by $bd(x)$. The degree of a forest is the maximum of the degrees of a node in the forest and the number of roots.

Let $F \subseteq \mathbb{N}^+$ be a forest and x a node in F. As a convention, we take $x \cdot \varepsilon = x$, $(x \cdot c) \cdot -1 = x$, and $n \cdot -1$ as undefined, for $n \in \mathbb{N}$. We call x a *leaf* if it has no successors. A *path* π in F is a word $\pi = a_1 a_2 \ldots$ of F such that a_1 is a root of F and for every $a_i \in \pi$, either a_i is a leaf (i.e., π ends in a_i) or a_i is a predecessor of a_{i+1}. Given two alphabets Σ_1 and Σ_2, a (Σ_1, Σ_2)-labeled forest is a triple $\langle F, V, E \rangle$, where F is a forest, $V : F \rightarrow \Sigma_1$ maps each node of F to a letter in Σ_1, and $E : F \times F \rightarrow \Sigma_2$ is a partial function that maps each pair (x, y), with $y \in children(x)$, to a letter in Σ_2. As a particular case, we consider a forest without labels on edges as a Σ_1-labeled forest $\langle F, V \rangle$, and a tree as a forest containing exactly one tree.

A *quasi-forest* is a forest where each node may also have roots as successors. Thus, for each node x of a quasi-forest $F \subseteq \mathbb{N}^+$, we denote with $successor(x)$ the successors of x and $children(x) = successor(x) \setminus \mathbb{N}$. All the other definitions regarding forests easily extend to quasi-forest. Notice that in a quasi-forest, a root can also have several predecessors, while every other node has always a unique one. Clearly, a quasi-forest can be always transformed in a forest by removing root successors.

Enriched Automata. For a given set Y, let $B^+(Y)$ be the set of positive Boolean formulas over Y (i.e., Boolean formulas built from elements in Y using $\wedge$ and $\vee$), where we also allow the formulas **true** and **false** and $\wedge$ has precedence over $\vee$. For a set $X \subseteq Y$ and a formula $\theta \in B^+(Y)$, we say that X satisfies θ iff assigning **true** to elements in X and assigning false to elements in $Y \setminus X$ makes θ true. For $b > 0$, let $\langle [b] \rangle = \{\langle 0 \rangle, \langle 1 \rangle, \ldots, \langle b \rangle\}$, $[[b]] = \{[0], [1], \ldots, [b]\}$, and $D_b = \langle [b] \rangle \cup [[b]] \cup \{-1, \varepsilon, \langle root \rangle, [root]\}$.

A fully enriched automaton is an automaton in which the transition function δ maps a state q and a letter σ to a formula in $B^+(D_b \times Q)$. Intuitively, an atom $(\langle n \rangle, q)$ (resp., $([n], q)$) means that the automaton sends copies in state q to $n+1$

(resp., all but n) different successors of the current node, (ε, q) means that the automaton sends a copy (in state q) to the current node, $(-1, q)$ means that the automaton sends a copy to the predecessor of the current node, and $(\langle root \rangle, q)$ and $([root], q)$ mean that the automaton sends a copy to some, respectively all of the roots of the forest. When, for instance, the automaton is in state q, reads a node x, and $\delta(q, V(x)) = (-1, q_1) \wedge ((\langle root \rangle, q_2) \vee ([root], q_3))$, it sends a copy in state q_1 to the predecessor and either sends a copy in state q_2 to one of the roots or a copy in state q_3 to all roots.

Formally, a *fully enriched automaton* (FEA, for short) is a tuple $\mathcal{A} = \langle \Sigma,\, b,\, Q,\, \delta,\, Q_0,\, \mathcal{F} \rangle$, where Σ is the input alphabet, $b > 0$ is a counting bound, Q is a finite set of states, $\delta : Q \times \Sigma \rightarrow B^+(D_b \times Q)$ is a transition function, $Q_0 \subseteq Q$ is a set of initial states, and $\mathcal{F}$ is the acceptance condition. A *run* of $\mathcal{A}$ on an input Σ-labeled forest $\langle F, V \rangle$ is a tree $\langle T_r, r \rangle$ in which each node is labeled by an element of $F \times Q$. Intuitively, a node in T_r labeled by (x, q) describes a copy of the automaton in state q that reads the node x of F. Runs start in the initial state and satisfy the transition relation. Thus, a run $\langle T_r, r \rangle$ with root z has to satisfy the following: (i) $r(z) = (c, q_0)$ for some root c of F and $q_0 \in Q_0$ (ii) for all $y \in T_r$ with $r(y) = (x, q)$ and $\delta(q, V(x)) = \theta$, there is a (possibly empty) set $S \subseteq D_b \times Q$, such that S satisfies θ, and for all $(d, s) \in S$, the following hold:

– If $d \in \{-1, \varepsilon\}$, then $x \cdot d$ is defined and there is $j \in \mathbb{N}$ such that $y \cdot j \in T_r$ and $r(y \cdot j) = (x \cdot d, s)$;
– If $d = \langle n \rangle$, then there are distinct $i_1, \ldots, i_{n+1} \in \mathbb{N}$ such that for all $1 \leq j \leq n+1$, there is $j' \in \mathbb{N}$ such that $y \cdot j' \in T_r$, $x \cdot i_j \in F$, and $r(y \cdot j') = (x \cdot i_j, s)$;
– If $d = [n]$, then there are distinct $i_1 \ldots, i_{bd(x)-n} \in \mathbb{N}$ such that for all $1 \leq j \leq bd(x) - n$, there is $j' \in \mathbb{N}$ such that $y \cdot j' \in T_r$, $x \cdot i_j \in F$, and $r(y \cdot j') = (x \cdot i_j, s)$;
– If $d = \langle root \rangle$, then for some root $c \in F$ and some $j \in \mathbb{N}$ such that $y \cdot j \in T_r$, it holds that $r(y \cdot j) = (c, s)$;
– If $d = [root]$, then for all roots $c \in F$ there exists $j \in \mathbb{N}$ such that $y \cdot j \in T_r$ and $r(y \cdot j) = (c, s)$.

A run $\langle T_r, r \rangle$ is *accepting* if all its infinite paths satisfy the acceptance condition. We consider here the *parity acceptance condition*, where $\mathcal{F} = \{F_1, \ldots, F_k\}$ is such that $F_1 \subseteq \ldots \subseteq F_k = Q$. The number k of sets in $\mathcal{F}$ is called the *index* of the automaton. Given a run $\langle T_r, r \rangle$ and an infinite path $\pi \subseteq T_r$, let $\mathit{Inf}(\pi) \subseteq Q$ be such that $q \in \mathit{Inf}(\pi)$ iff there are infinitely many $y \in \pi$ for which $r(y) \in F \times \{q\}$. A path π *satisfies* a parity acceptance condition $\mathcal{F} = \{F_1, \ldots, F_k\}$ iff there is an even i for which $\mathit{Inf}(\pi) \cap F_i \neq \emptyset$ and $\mathit{Inf}(\pi) \cap F_{i-1} = \emptyset$. An automaton *accepts* a forest iff there exists an accepting run of the automaton on the forest. We denote by $\mathcal{L}(\mathcal{A})$ the set of all Σ-labeled forests that $\mathcal{A}$ accepts. The *emptiness problem* for FEAs is to decide, given a FEA $\mathcal{A}$, whether $\mathcal{L}(\mathcal{A}) = \emptyset$. In the following theorem we recall the exact complexity of this decision problem.

Theorem 1. *[BLMV06] The nonemptiness problem for a fully enriched automaton $\mathcal{A} = \langle \Sigma, b, Q, \delta, Q_0, \mathcal{F} \rangle$ can be solved in time linear in the size of Σ and b, and exponential in the index of the automaton and number of states.*

The following results on FEAs will be useful in the rest of the paper.

Lemma 1. *[BLMV06] Given two FEAs $\mathcal{A}_1$ and $\mathcal{A}_2$, there exists a FEA $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ and whose size is linear in the size of $\mathcal{A}_1$ and $\mathcal{A}_2$.*

As a particular case of FEA, we consider nondeterministic Büchi Automata running on forests (BFA, for short). Formally, a BFA is a tuple $\mathcal{A} = \langle \Sigma, D, Q, \delta, Q_0, \mathcal{F}\rangle$, where Σ, Q, and Q_0 are defined as in FEA, D is a finite set of branching degrees, $\mathcal{F} \subseteq Q$ is a Büchi acceptance condition, and $\delta : Q \times \Sigma \times D^2 \to 2^{Q^* \times (Q \times Root)^*}$ is the transition relation satisfying $\delta(q, \sigma, d_1, d_2) \in 2^{Q^{d_1} \times (Q \times Root)^{d_2}}$, for every $q \in Q$, $\sigma \in \Sigma$ and $d_1, d_2 \in D$.

A *run* of $\mathcal{A}$ on an input Σ-labeled forest $\langle F, V\rangle$ of branching degree D is a tree $\langle T_r, r\rangle$ in which each node is labeled by an element of $F \times Q$. Formally, $\langle T_r, r\rangle$ is a run if $r(z) = (Q_0, z)$, for some root z of F and $q_0 \in Q_0$, and for all $y \in T_r$ labeled with (q, x), having d successors where d_2 are roots successors and d_1 are the remaining ones, we have that $r(y \cdot i) = \langle q_i, x \cdot i\rangle$ for all $1 \leq i \leq d_1$, $r(y \cdot (d_1 + i)) = \langle q_{d_1+i}, x_i\rangle$ for all $1 \leq i \leq d_2$ and $\langle\langle q_1, \ldots, q_{d_1}\rangle, \langle r(y \cdot (d_1 + 1)), \ldots, r(y \cdot d)\rangle\rangle \in \delta(q, V(x), d_1, d_2)$. A run $\langle F, V\rangle$ of a BFA is accepting if for all paths π of T_r, we have that $Inf(\pi) \cap \mathcal{F} \neq \emptyset$. Notice that $\mathcal{F}$ can be also expressed as the particular parity condition $\{\emptyset, \mathcal{F}\}$.

3 Fully Enriched μ-Calculus

Let *AP*, *Var*, *Prog*, and *Nom* be finite and pairwise disjoint sets of *atomic propositions*, *propositional variables*, *atomic programs*, and *nominals*. A *program* is an atomic program a or its converse a^-. The set of *formulas of the fully enriched μ-calculus* is the smallest set such that (i) **true** and **false** are formulas; (ii) p and $\neg p$, for $p \in AP \cup Nom$, are formulas; (iii) $x \in Var$ is a formula; (iv) if φ_1 and φ_2 are formulas, α is a program, n is a non-negative integer, and y is a propositional variable, then the following are also formulas: $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$, $\langle n, \alpha\rangle\varphi_1$, $[n, \alpha]\varphi_1$, $\mu y.\varphi_1(y)$, and $\nu y.\varphi_1(y)$.

Observe that we use positive normal form, i.e., negation is applied only to atomic propositions. We call μ and ν *fixpoint operators* and use λ to denote a fixpoint operator μ or ν. A propositional variable y occurs *free* in a formula if it is not in the scope of a fixpoint operator, and *bound* otherwise. A *sentence* is a formula that contains no free variables. We refer often to the *graded modalities* $\langle n, \alpha\rangle\varphi_1$ and $[n, \alpha]\varphi_1$ as *atleast formulas* and *allbut formulas* and assume that the integers in these operators are given in binary coding: the contribution of n to the length of the formulas $\langle n, \alpha\rangle\varphi$ and $[n, \alpha]\varphi$ is $\lceil \log n\rceil$ rather than n. We refer to fragments of the fully enriched μ-calculus using the names from Fig. 1. Hence, we say that a formula φ of the fully enriched μ-calculus is also a formula of the *hybrid graded*, *full hybrid*, or *full graded* μ-calculus if φ does not have inverse programs, graded modalities, or nominals, respectively. If at least one of the above holds, we also say that φ is an *enriched μ-calculus* formula. To avoid confusion, we observe that enriched formulas are also formulas of the full calculus, while the converse is not always true. We recall that enriched formulas enjoy the forest model property (as showed in [BLMV06] and [SV01]), while fully enriched formulas does not [BP04].

The semantics of the fully enriched μ-calculus is defined with respect to a *Kripke structure*, i.e., a tuple $K = \langle W, W_0, R, L\rangle$ where W is a non-empty set of *states*, $W_0 \subseteq W$ is the set of initial states, $R : \mathit{Prog} \rightarrow 2^{W\times W}$ is a total function (i.e., for each $v \in W$ there is a program $a \in \mathit{Prog}$ and a node w such that $(v, w) \in R(a)$) that assigns to each atomic program a transition relation over W, and $L : \mathit{AP} \cup \mathit{Nom} \rightarrow 2^W$ is a labeling function that assigns to each atomic proposition and nominal a set of states such that the sets assigned to nominals are singletons and subsets of W_0. To deal with inverse programs, we extend R as follows: for each $a \in \mathit{Prog}$, set $R(a^-) = \{(v, u) : (u, v) \in R(a)\}$. If $(w, w') \in R(\alpha)$, we say that w' is an *α-successor* of w. Informally, an *atleast* formula $\langle n, \alpha\rangle\varphi$ holds at a state w of a Kripke structure K if φ holds at least in $n + 1$ α-successors of w. Dually, the *allbut* formula $[n, \alpha]\varphi$ holds in a state w of a Kripke structure K if φ holds in all but at most n α-successors of w. Note that $\neg\langle n, \alpha\rangle\varphi$ is equivalent to $[n, \alpha]\neg\varphi$, and that the modalities $\langle\alpha\rangle\varphi$ and $[\alpha]\varphi$ of the standard μ-calculus can be expressed as $\langle 0, \alpha\rangle\varphi$ and $[0, \alpha]\varphi$, respectively.

To formalize semantics, we introduce valuations. Given a Kripke structure $K = \langle W, W_0, R, L\rangle$ and a set $\{y_1, \ldots, y_n\}$ of variables in Var, a *valuation* $\mathcal{V} : \{y_1, \ldots, y_n\} \rightarrow 2^W$ is an assignment of subsets of W to the variables $y_1, \ldots, y_n$. For a valuation $\mathcal{V}$, a variable y, and a set $W' \subseteq W$, we denote by $\mathcal{V}[y \leftarrow W']$ the valuation obtained from $\mathcal{V}$ by assigning W' to y. A formula φ with free variables among $y_1, \ldots, y_n$ is interpreted over the structure K as a mapping φ^K from valuations to 2^W, i.e., $\varphi^K(\mathcal{V})$ denotes the set of points that satisfy φ under valuation $\mathcal{V}$. The mapping φ^K is defined inductively as follows:

- $\mathbf{true}^K(\mathcal{V}) = W$ and $\mathbf{false}^K(\mathcal{V}) = \emptyset$;
- for $p \in \mathit{AP} \cup \mathit{Nom}$, we have $p^K(\mathcal{V}) = L(p)$ and $(\neg p)^K(\mathcal{V}) = W \setminus L(p)$;
- for $y \in \mathit{Var}$, we have $y^K(\mathcal{V}) = \mathcal{V}(y)$;
- $(\varphi_1 \wedge \varphi_2)^K(\mathcal{V}) = \varphi_1^K(\mathcal{V}) \cap \varphi_2^K(\mathcal{V})$ and $(\varphi_1 \vee \varphi_2)^K(\mathcal{V}) = \varphi_1^K(\mathcal{V}) \cup \varphi_2^K(\mathcal{V})$;
- $(\langle n, \alpha\rangle\varphi)^K(\mathcal{V}) = \{w : |\{w' \in W : (w, w') \in R(\alpha) \text{ and } w' \in \varphi^K(\mathcal{V})\}| \geq n + 1\}$;
- $([n, \alpha]\varphi)^K(\mathcal{V}) = \{w : |\{w' \in W : (w, w') \in R(\alpha) \text{ and } w' \notin \varphi^K(\mathcal{V})\}| \leq n\}$;
- $(\mu y.\varphi(y))^k(\mathcal{V}) = \bigcap\{W' \subseteq W : \varphi^K([y \leftarrow W']) \subseteq W'\}$;
- $(\nu y.\varphi(y))^k(\mathcal{V}) = \bigcup\{W' \subseteq W : W' \subseteq \varphi^K([y \leftarrow W'])\}$.

Notice that α used in the previous graded modalities is a program, i.e., α can be either an atomic program or its converse. Also, notice that no valuation is required for a sentence. Let $K = \langle W, W_0, R, L\rangle$ be a Kripke structure and φ a sentence. For a state $w \in W$, we say that K *satisfies* φ at w, denoted $K, w \models \varphi$, if $w \in \varphi^K$. K is a *model* of φ if there is a $w \in W_0$ such that $K, w \models \varphi$. In what follows, a formula φ *counts* up to b if the maximal integer in *atleast* and *allbut* restrictions used in φ is $b - 1$.

Given a formula φ of the enriched μ-calculus, accordingly to the forest model property, we can define a FEA accepting all forest models of φ. Before giving this result, there is a technical difficulty to be overcome: φ has quasi-forests as models, with labels on both edges and nodes, while FEAs can only accept forests with labels on nodes. This problem can be dealt in the following way. First, we move the label of each edge to the target node of the edge. For this purpose, we introduce a new propositional symbol p_α for each program α. Thus, for each quasi-forest model $\langle F, V, E\rangle$, we consider the corresponding quasi-forest $\langle F, V'\rangle$ obtained by removing labeling on the edges and using as labeling of nodes the extended labeling function $V'(w) = V(w) \cup \{p_\alpha \mid E(v, w) = \alpha\}$. Then, to solve the problem of edges to the roots in quasi-forests models, we introduce in node labels new propositional symbols $\uparrow_o^\alpha$ (not occurring in the input formula) that represent an α-labeled edge from the current node to the (unique) root node labeled by nominal o. We call the new labeling function V^*, and with $\langle F, V^*\rangle$ we denote the forest encoding of the quasi-forest model $\langle F, V, E\rangle$. A forest $\langle F, V^*\rangle$ can also be considered as a particular Kripke structure by letting the total property holding in $\langle F, V^*\rangle$ if all leaves have a propositional symbol $\uparrow_o^\alpha$ in their labels. Now we can give the following result.

Lemma 2. *[SV01, BLMV06] Given a sentence φ of the enriched μ-calculus that has ℓ* atleast *subsentences, counts up to b, and contains k nominals, we can construct a FEA $\mathcal{A}_\varphi$ such that it accepts exactly the forest encodings of the quasi-forest models of φ having degree at most* $\max\{k+1, \ell(b+1)\}$*, and such that it has $\mathcal{O}(|\varphi|^2)$ states, index $|\varphi|$, and counting bound b.*

4 Enriched μ-Calculus Module Checking

In this paper we consider open systems, i.e., systems that interact with their environment and whose behavior depends on this interaction. The (global) behavior of such a system is described by a *module* $M = \langle W_s, W_e, W_0, R, L\rangle$, which is a Kripke structure where the set of states $W = W_s \cup W_e$ is partitioned in a set of *system states* W_s and a set of *environment states* W_e.

Given a module M, we assume that its states are ordered and the number of successors of each state w, denoted by $bd(w)$, is finite, and W is considered to be finite. For each state $w \in W$, we denote by $succ(w)$ the ordered tuple (possibly empty) of w's successors. When the module M is in a system state w_s, then all the states in $succ(w_s)$ are possible next states. On the other hand, when M is in an environment state w_e, then the possible next states (that are in $succ(w_e)$) depend on the current environment. Since the behavior of the environment is not predictable, we have to consider all the possible sub-tuples of $succ(w_e)$. The only constraint, since we consider environments that cannot block the system, is that not all the transitions from w_e are disabled.

The set of all (maximal) computations of M starting from the initial states W_0 is described by a $(W, Prog)$-labeled quasi-forest $\langle F_M, V_M, E_M\rangle$, called *computation quasi-forest*, which is obtained by unwinding M in the usual way. The problem of deciding, for a given branching-time formula φ over $AP \cup Nom$, whether $\langle F_M, L \circ V_M, E_M\rangle$ satisfies φ, denoted $M \models \varphi$, is the usual *model-checking problem* [CE81, QS81]. On the other hand, for an open system, $\langle F_M, V_M, E_M\rangle$ corresponds to a very specific environment, i.e., a maximal environment that never restricts the set of its next states. Therefore, when we examine a branching-time specification φ w.r.t. a module M, φ should hold not only in $\langle F_M, V_M, E_M\rangle$, but in all the quasi forests obtained by pruning from $\langle F_M, V_M, E_M\rangle$ subtrees whose root is a child (successor) of a node corresponding to an environment state, as well as inhibiting some of its jumps to roots, if there are any. The set of these quasi forests is denoted by $exec(M)$, and is formally defined as follows. $\langle F, V, E\rangle \in exec(M)$ iff for each $w_i \in W_0$, we have $V(i) = w_i$, and the following holds:

- For $x \in F$ with $V(x) = w \in W_s$, $succ(w) = \langle w_1, \ldots, w_n, w_{n+1}, \ldots, w_{n+m}\rangle$, and $succ(w) \cap W_0 = \langle w_{n+1}, \ldots, w_{n+m}\rangle$, it holds that
  - $children(x) = \{x \cdot 1, \ldots, x \cdot n\}$ and for $1 \leq i \leq n$, $V(x \cdot i) = w_i$, and $E(x, x \cdot i) = \alpha$ if $(w, w_i) \in R(\alpha)$;
  - for $1 \leq i \leq m$, let $x_i \in \mathbb{N}$ such that $V(x_i) = w_{n+i}$, it holds that $E(x, x_i) = \alpha$ if $(w, w_{n+i}) \in R(\alpha)$;
- For $x \in F$ with $V(x) = w \in W_e$ it holds that there exists a sub-tuple $S = \langle w_{i_1}, \ldots, w_{i_p}, w_{i_{p+1}}, \ldots, w_{i_{p+q}}\rangle$ of $succ(w)$ with $p + q \geq 1$, $S \cap W_0 = \langle w_{i_{p+1}}, \ldots, w_{i_{p+q}}\rangle$ and such that
  - $children(x) = \{x \cdot 1, \ldots, x \cdot p\}$ and for $1 \leq j \leq p$, $V(x \cdot j) = w_{i_j}$, and $E(x, x \cdot j) = \alpha$ if $(w, w_{i_j}) \in R(\alpha)$;
  - for $1 \leq j \leq q$, let $x_j \in \mathbb{N}$ such that $V(x_j) = w_{i_{p+j}}$, it holds that $E(x, x_j) = \alpha$ if $(w, w_{i_{p+j}}) \in R(\alpha)$;

Intuitively, a quasi-forest in $exec(M)$ corresponds to a different behavior of the environment. In the following, we consider quasi-forests in $exec(M)$ as $(2^{AP \cup Nom}$, $Prog)$-labeled quasi-forests, i.e., taking the label of a node x to be $L(V(x))$.

For a module M and a formula φ of the enriched μ-calculus we say that M satisfies φ, denoted $M \models_r \varphi$, if all the quasi forests in $exec(M)$ satisfy φ. The problem of deciding whether M satisfies φ is called *module checking*, and extends to forests the analogously problem defined in [KVW01] regarding trees. Note that $M \models_r \varphi$ implies $M \models \varphi$, but the converse in general does not hold. Also, note that $M \not\models_r \varphi$ is *not* equivalent to $M \models_r \neg\varphi$, since $M \not\models_r \varphi$ just states that there is some quasi forest in $exec(M)$ satisfying $\neg\varphi$.

5 Deciding Enriched μ-Calculus Module Checking

In this section, we solve the module checking problem for the enriched μ-calculus. In particular, we show that this problem is decidable and ExpTime-complete. For the upper bound, we give an algorithm based on an automata-theoretic approach, by extending to forests and idea of [KVW01]. For the lower bound, we give a reduction from the module checking problem for *CTL*, known to be ExpTime-hard. We start with the upper bound.

Let M be a module and φ an enriched μ-calculus formula. We decide the module-checking problem for M against φ by building a FEA $\mathcal{A}_{M\times\neg\varphi}$ as the intersection of two automata. Essentially, the first automaton, denoted by $\mathcal{A}_M$, is a Büchi automaton that accepts forests encoding of labeled quasi-forests of $exec(M)$, and the second automaton is a FEA that accepts all the forests encoding of labeled quasi-forests that do not satisfy φ. Thus, $M \models_r \varphi$ iff $\mathcal{L}(\mathcal{A}_{M\times\neg\varphi})$ is empty.

The construction of $\mathcal{A}_M$ proposed here extends that given in [KVW01] for solving the module checking problem with respect to CTL and CTL^*. The extension concerns the handling of forest models instead of trees and formulas of the enriched μ-calculus. Before starting, there are few technical difficulties to be overcome. First, we notice that $exec(M)$ contains quasi-forests, with labels on both edges and nodes, while Büchi automata can only accept forests with labels on nodes. This problem can be dealt as we did in Section 3 by moving the label of each edge to the target node of the edge (formally using a new propositional symbol p_α, for each program α) and by substituting edges to roots with new propositional symbols $\uparrow^\alpha_o$ (which represent an α-labeled edge from the current node to the unique root node labeled by nominal o). Let $AP^* = AP \cup \{p_\alpha \mid \alpha \in Prog\} \cup \{\uparrow^\alpha_o \mid \alpha \in Prog \text{ and } o \in Nom\}$, we denote with $\langle F, V^* \rangle$ the $2^{AP^* \cup Nom}$-labeled forest encoding of a quasi-forest $\langle F, V, E \rangle \in exec(M)$, obtained using the above transformations.

Another technical difficulty to handle is relate to the fact that quasi-forests of $exec(M)$ (and thus their encoding) may not share the same structure, since they are obtained by pruning some subtrees from the computation quasi-forest $\langle F_M, V_M, E_M \rangle$ of M. Let $\langle F_M, V^*_M \rangle$ the *computation forest* of M obtained from $\langle F_M, V_M, E_M \rangle$ using the above encoding. By extending an idea of [KVW01], we solve the technical problem by considering each forest $\langle F, V^* \rangle$, encoding of a quasi-forest of $exec(M)$, as a $2^{AP^* \cup Nom} \cup \{\perp\}$-labeled forest $\langle F_M, V^{**} \rangle$ (where $\perp$ is a fresh proposition name not belonging to $AP \cup Nom$) such that for each node $x \in F_M$, if $x \in F$ then $V^{**}(x) = V^*(x)$, otherwise $V^{**}(x) = \{\perp\}$. Thus, we label each node pruned in the $\langle F_M, V^*_M \rangle$ with $\{\perp\}$ and recursively, we label with $\{\perp\}$ its subtrees. In this way, all forests encoding quasi-forests of $exec(M)$ have the same structure of $\langle F_M, V^*_M \rangle$, and they differ only in their labeling. Accordingly we can think of an environment as a strategy for placing $\{\perp\}$ in $\langle F_M, V^{**} \rangle$. Moreover, the environment can also disable jumps to roots. This is performed by removing from enabled environment nodes some of $\uparrow^\alpha_o$ labels. Notice that since we consider environments that do not block the system, each node associated with an environment state has at least one successor not labeled by $\{\perp\}$, unless it has $\uparrow^\alpha_o$ in its label.

Let us denote by $\widehat{exec}(M)$ the set of all $2^{AP^* \cup Nom} \cup \{\perp\}$-labeled $\langle F_M, V^{**} \rangle$ forests obtained from $\langle F, V, E \rangle \in exec(M)$ in the above described manner. The required BFA $\mathcal{A}_M$ must accept all and only the $2^{AP^* \cup Nom} \cup \{\perp\}$-labeled forests in $\widehat{exec}(M)$. Formally, let $M = \langle W_s, W_e, W_0, R, L \rangle$ be a module, $\mathcal{A}_M = \langle \Sigma, D, Q, \delta, Q_0, \mathcal{F} \rangle$ is defined as follows:

- $\Sigma = 2^{AP^* \cup Nom} \cup \{\perp\}$;
- $D = \bigcup_{w \in W} bd(w)$. That is D contains all the branching degrees in M.
- $Q = W \times \{\perp, \top, \vdash\}$. Thus every node w of M induces three states $(w, \perp)$, $(w, \top)$, and $(w, \vdash)$ in $\mathcal{A}_M$. Intuitively, when $\mathcal{A}_M$ is in state $(w, \perp)$, it can read only $\perp$, in state $(w, \top)$, it can read only letters in $2^{AP^* \cup Nom}$, and in state $(w, \vdash)$, then it can read both letters in $2^{AP^* \cup Nom}$ and $\perp$. In this last case, it is left to the environment to decide whether the transition to a state of the form $(w, \vdash)$ is enabled. The three types of states are used to ensure that

the environment enables all transitions from enabled system nodes, enables at least one transition from each enabled environment node, and disables transitions from disabled nodes.

- $Q_0 = \{\langle w_i, \top\rangle \mid w_i \in W_0\}$.
- The transition function $\delta : Q \times \Sigma \times D^2 \to 2^{Q^* \times (Q \times Root)^*}$ is defined as follows. Let $x \in F$ such that $V(x) = w$, $succ(w) = \langle w_1, \ldots, w_n, w'_1, \ldots, w'_m\rangle$, $succ(w) \cap W_0 = \langle w'_1, \ldots, w'_m\rangle$, and there exist $j_1, \ldots, j_m$ such that $V(j_h) = w'_h$, for $1 \leq h \leq m$, then:
  - For $w \in W_e \cup W_s$ and $g \in \{\vdash, \bot\}$ we have

$$\delta((w, g), \bot, n, 0) = \{\langle\langle(w_1, \bot), \ldots, (w_n, \bot)\rangle, \langle\emptyset\rangle\rangle\}$$

    That is, $\delta((w, g), \bot)$ contains exactly one n-tuple of all successors of w without jumps to roots. In this case, all transitions to successors of w are recursively disabled.
  - For $w \in W_s$ and $g \in \{\top, \vdash\}$ we have

$$\begin{aligned}\delta((w, g), L(w), n, m) = \langle\ &\langle(w_1,\top), \ldots, (w_n,\top)\rangle,\\ &\langle(\langle w'_1, j_1\rangle,\top), \ldots, (\langle w'_m, j_m\rangle,\top)\rangle\ \rangle\}.\end{aligned}$$

    That is, $\delta((w, g), m)$ contains exactly one $(n+m)$-tuple of all successors of w, containing m jumps to roots. In this case all transitions to successors of w are enabled.
  - For $w \in W_e$ and $g \in \{\top, \vdash\}$, let $J = \{\uparrow^{\alpha}_{o} \mid \alpha \in Prog$ and $o \in Nom\}$ and $X \subseteq L(w)$ such that $(X \setminus J) = (L(w) \setminus J)$, (i.e., X may have less jumps to roots of $L(w)$), we have
    - For $X \cap J = \emptyset$ we have

$$\begin{aligned}\delta((w, g), X, n, 0) = \{\ &\langle(w_1, \top),\ (w_2,\vdash), \ldots, (w_n,\vdash)\rangle,\\ &\langle(w_1,\vdash),\ (w_2, \top), \ldots, (w_n,\vdash)\rangle,\\ &\qquad\vdots\\ &\langle(w_1,\vdash),\ (w_2,\vdash), \ldots, (w_n, \top)\rangle\}.\end{aligned}$$

      That is, $\delta((w, g), X, n, 0)$ contains n different n-tuples of all successors of w, without jumps to roots. When $\mathcal{A}_M$ proceeds according to the ith tuple, the environment can disable all transitions to successors of w, except that to w_i.
    - For $X \cap J = \{\uparrow^{\alpha_1}_{o_1}, \ldots \uparrow^{\alpha_s}_{o_s}\}$ with $s \geq 1$, let $\langle w'_{j_1} \ldots w'_{j_s}\rangle$ a subtuple of $\langle w'_1 \ldots w'_m\rangle$ such that $o_i \in L(w'_{j_i})$, we have

$$\begin{aligned}\delta((w, g), X, n, s) = \{\ \langle\ &\langle(w_1,\vdash), \ldots, (w_n,\vdash)\rangle,\\ &\langle(\langle w'_{j_1}, j_1\rangle, \top), \ldots, (\langle w'_{j_s}, j_s\rangle, \top)\rangle\ \rangle\end{aligned}$$

      That is, $\delta((w, g), X, n, s)$ contains one $(n + s)$-tuple of successors of w, where the first n are all not root successors of w and they can be successively disabled.

Notice that δ is not defined when n is different from the number of successors of w that are not jumps to roots, and when the input does not meet the restriction imposed by the $\top$, $\vdash$, and $\perp$ annotations or by the labeling of w.

The automaton $\mathcal{A}_M$ has $3 \cdot |W|$ states, $2^{|AP| \cdot |R|}$ symbols, and the size of the transition relation $|\delta|$ is bounded by $|R|(|W| \cdot 2^{|R|})$.

We recall that a node labeled by $\{\perp\}$ stands for a node that actually does not exist. Thus, we have to take this into account when we interpret formulas of the enriched μ-calculus over forests $\langle F_M, V^* \rangle \in \widehat{exec}(M)$. In order to achieve this, as in [KVW01] we define a function f that transforms the input formula in a formula of the enriched μ-calculus that restricts path quantification to only paths that never visit a state labeled with $\{\perp\}$. The function f we consider extends that given in [KVW01] and is inductively defined as follows:

– $f(\mathbf{true}) = \mathbf{true}$ and $f(\mathbf{false}) = \mathbf{false}$;
– $f(p) = p$ and $f(\neg p) = \neg p$ for all $p \in AP \cup Nom$;
– $f(x) = x$ for all $x \in Var$;
– $f(\varphi_1 \vee \varphi_2) = f(\varphi_1) \vee f(\varphi_2)$ and $f(\varphi_1 \wedge \varphi_2) = f(\varphi_1) \wedge f(\varphi_2)$ for all enriched μ-calculus formulas φ_1 and φ_2;
– $f(\mu x.\varphi(x)) = \mu x.f(\varphi(x))$ and $f(\nu x.\varphi(x)) = \nu x.f(\varphi(x))$ for all $x \in Var$ and enriched μ-calculus formulas φ;
– $f(\langle n, \alpha \rangle \varphi) = \langle n, \alpha \rangle (\neg\perp \wedge f(\varphi))$ for $n \in \mathbb{N}$ and for all programs α and enriched μ-calculus formulas φ;
– $f([n, \alpha]\varphi) = [n, \alpha](\neg\perp \wedge f(\varphi))$ for $n \in \mathbb{N}$ and for all programs α and enriched μ-calculus formulas φ.

Note that the programs α in the previous definition of f can be either an atomic program $a \in Prog$ or its converse a^-. By definition of f, it follows that for each formula φ and $\langle F, V \rangle \in \widehat{exec}(M)$, $\langle F, V \rangle$ satisfies $f(\varphi)$ *iff* the $2^{AP^* \cup Nom}$-labeled tree obtained from $\langle F, V \rangle$ removing all the nodes labeled by $\{\perp\}$ satisfies φ. Therefore, the module checking problem of M against an enriched μ-calculus formula φ is reduced to check the existence of a forest $\langle F, V \rangle \in \widehat{exec}(M) = \mathcal{L}(\mathcal{A}_M)$ satisfying $f(\neg\varphi)$ (note that $|f(\neg\varphi)| = O(|\neg\varphi|)$). We reduce the latter to check the emptiness of a FEA $\mathcal{A}_{M \times \neg\varphi}$ that is defined as the intersection of the BFA $\mathcal{A}_M$ with a FEA $\mathcal{A}_{\neg\varphi}$ accepting exactly the $2^{AP^* \cup Nom} \cup \{\perp\}$ forests encodings of quasi-forest models of $f(\neg\varphi)$. By Lemma 2, if φ is an enriched μ-calculus formula, then $\mathcal{A}_{\neg\varphi}$ has $\mathcal{O}(|\varphi|^2)$ states, index $|\varphi|$, and counting bound b. Therefore, by Lemma 1, $\mathcal{A}_{M \times \neg\varphi}$ has $\mathcal{O}(|W| + |\varphi|^2)$ states, index $|\varphi|$, and counting bound b. We recall that, by Theorem 1, given a FEA, the emptiness is exponential only in its number of states and index, thus we have algorithm to decide the module checking problem for enriched μ-calculus formulas that is exponential both in the size of the module and the size of the formula.

To show a tight lower bound we recall that *CTL* module checking is ExpTime-hard [KVW01] and every *CTL* formula can be linearly transformed in a modal μ-calculus formula [Zap02]. This leads to the module checking problem w.r.t. modal μ-calculus formulas to be ExpTime-hard and thus to the following result

Theorem 2. *The module checking problem with respect to enriched μ-calculus formulas is* ExpTime*-complete.*

6 Fully Enriched μ-Calculus Module Checking

In this section, we deal with the module checking problem for the fully enriched μ-calculus and we show that it is undecidable.

Let us note that, since the fully enriched μ-calculus does not enjoy the forest model property [BP04], we cannot unwind a Kripke structure in a forest. However, it is always possible to unwind it in an equivalent acyclic graph that we call *computation graph*. In order to take into account all the possible behaviors of the environment, we consider all the possible subgraphs of the computation graph obtained disabling some transitions from environment nodes but one. We denote with $graphs(M)$ the set of this graphs. Given a fully enriched μ-calculus formula φ, we have that $M \models_r \varphi$ iff $K \models \varphi$ for all $K \in graphs(M)$.

To show the undecidability of the addressed problem, we need some further definitions. An (infinite) *grid* is a tuple $G = \langle \mathbb{N}^2, h, v \rangle$ such that h and v are defined as $h(\langle x, y \rangle) = \langle x + 1, y \rangle$ and $v(\langle x, y \rangle) = \langle x, y + 1 \rangle$. Given a finite set of *types* T, we will call *tiling* on T a function $\hat{\rho} : \mathbb{N}^2 \rightarrow T$ that associates a type from T to each vertex of an infinite grid G, and we call *tiled infinite grid* the tuple $\langle G, T, \hat{\rho} \rangle$. A *grid model* is an infinite Kripke structure $K = \langle W, \{w_0\}, R, L \rangle$, on the set of atomic programs $Prog = \{l^-, v\}$, such that K can be mapped on a grid in such a way that w_0 corresponds to the vertex $\langle 0, 0 \rangle$, $R(v)$ corresponds to v and $R(l^-)$ corresponds to h. We say that a grid model K "corresponds" to a tiled infinite grid $\langle G, T, \hat{\rho} \rangle$ if every state of K is labeled with only one atomic proposition (and zero or more nominals) and there exists a bijective function $\rho : T \rightarrow AP$ such that, if $w_{x,y}$ is the state of K corresponding with the node $\langle x, y \rangle$ of G, then $\rho(\hat{\rho}(\langle x, y \rangle)) \in L(w_{x,y})$.

Theorem 3. *The module checking problem for fully enriched μ-calculus is undecidable.*

Proof sketch. To show the result, we use a reduction from the domino problem, known to be undecidable [Ber66]. The domino problem is defined as follows.

Let T be a finite set of types, and $H, V \subseteq T^2$ be two relations describing the types that cannot be vertically and horizontally adjacent in an infinite grid. The domino problem is to decide whether there exists a tiled infinite grid $\langle G, T, \hat{\rho} \rangle$ such that $\hat{\rho}$ preserves the relations H and V. We call such a tiling function a *legal tiling* for G on T.

In [BP04], Bonatti and Peron showed undecidability for the satisfiability problem for fully enriched μ-calculus by also using a reduction from the domino problem. Hence, given a set of types T and relations H and V, they build a (alternation free) fully enriched μ-calculus formula φ such that φ is satisfiable iff the domino problem has a solution in a tiled infinite grid, with a legal tiling ρ on T (with respect to H and V). In particular, the formula they build can be only satisfiable on a grid model K corresponding to a tiled infinite grid with a legal tiling ρ on T. In the reduction we propose here, we use the formula φ used in [BP04]. It remains to define the module.

Let $\{G_1, G_2, \ldots\}$ be the set of all the infinite tiled grids on T (i.e., $G_i = \langle G, T, \hat{\rho}_i \rangle$), we build a module M such that $graphs(M)$ contains, for each $i \geq 1$, a grid models corresponding to G_i. Therefore, we can decide the domino problem by checking whether $M \models_r \neg\varphi$. Indeed, if $M \models_r \neg\varphi$, then all grid models corresponding to G_i do not satisfy φ and, therefore, there is no solution for the domino problem. On the other side, if $M \not\models_r \neg\varphi$, then there exists a model for φ; since φ can be satisfied only on a grid model corresponding to a tiled infinite grid with a legal tiling on T with respect to H and V, we have that the domino problem has a solution.

Formally, let $T = \{t_1, \ldots, t_m\}$ be the set of types, the module $M = \langle W_s, W_e, W_0, R, L\rangle$ with respect to atomic programs $Prog = \{l^-, v\}$, atomic propositions $AP = T$, and nominals $Nom = \{o_1, \ldots, o_m\}$, is defined as follows:

- $W_s = \emptyset$, $W_e = \{x_1, \ldots, x_m, y_1, \ldots, y_m\}$ and $W_0 = \{x_1, \ldots, x_m\}$;
- for all $i \in \{1, \ldots m\}$, $L(t_i) = \{x_i, y_i\}$ and $L(o_i) = \{x_i\}$;
- $R(v) = \{\langle x_i, x_j\rangle | i, j \in \{1, \ldots, m\}\} \cup \{\langle y_i, y_j\rangle | i, j \in \{1, \ldots, m\}\}$ and $R(l^-) = \{\langle x_i, y_j\rangle | i, j \in \{1, \ldots, m\}\} \cup \{\langle y_i, x_j\rangle | i, j \in \{1, \ldots, m\}\}$

Notice that we duplicate the set of nodes labeled with tiles since we cannot have pairs of nodes in M labeled with more than one atomic program (in our case, with both v and l^-). Moreover the choice of labeling nodes x_i with nominals is arbitrary. Finally, from the fact that the module contains only environment nodes, it immediately follows that, for each i, the grid model corresponding to the infinite tiled grid G_i is contained in $graphs(M)$. $\square$

7 Conclusions

In [KVW01], module checking has been introduced as a useful framework for the verification of open finite-state systems. There, it has been shown that while for *LTL* the complexity of the model checking problem coincides with that of module checking (i.e., it is PSPACE-complete), for the branching time paradigm the problem of module checking is much harder. In fact, *CTL* (resp., *CTL**) module checking is EXPTIME-complete (resp., 2EXPTIME-complete), while model checking is solvable in linear time (resp., exponential time).

In this paper, we have extended the framework of module checking problem for finite-state systems to formulas of the fully enriched μ-calculus and showed that this problem becomes undecidable in this setting. Also, we have investigated this problem with respect to formulas of interesting fragments of the full calculus and, specifically, those belonging to the full hybrid, full graded, and hybrid graded μ-calculus, and showed, in all cases, that module checking is decidable and EXPTIME-complete. In particular, for the upper bound we have proposed an algorithm that is exponential in both the size of the model and the formula. Since module checking for μ-calculus subsumes that for *CTL* and for the latter the program complexity (i.e., the complexity of the problem w.r.t. a fixed formula) is polynomial, it remains as an open problem to decide the exact program complexity of module checking for the considered fragments of the full calculus.

Finally, we recall that model checking for modal μ-calculus is in UP$\cap$co-UP (see [Zap02] for a survey). Since we have proved that module checking for modal μ-calculus is ExpTime-hard, we conclude that also for this logic module checking is harder than model checking. Moreover, the model checking algorithm considered in [Zap02] for modal μ-calculus can be easily extended to deal with formulas of the fully enriched μ-calculus, showing that also for this logic the model checking problem is in UP$\cap$co-UP. Using this conjecture, we can extend to the full calculus and its fragments all the previous observations regarding the modal μ-calculus.

References

[Ber66] R. Berger, *The undecidability of the domino problem*, Mem. AMS **66** (1966), 1–72.

[BLMV06] P.A. Bonatti, C. Lutz, A. Murano, and M.Y. Vardi, *The complexity of enriched mu-calculi*, ICALP'06, LNCS 4052, 2006, pp. 540–551.

[BMP05] Laura Bozzelli, Aniello Murano, and Adriano Peron, *Pushdown module checking.*, LPAR, 2005, pp. 504–518.

[BP04] P.A. Bonatti and A. Peron, *On the undecidability of logics with converse, nominals, recursion and counting*, Artificial Intelligence **158** (2004), no. 1, 75–96.

[BS06] J. Bradfield and C. Stirling, *Modal μ-calculi*, Handbook of Modal Logic (Blackburn, Wolter, and van Benthem, eds.), Elsevier, 2006, pp. 722–756.

[CE81] E.M. Clarke and E.A. Emerson, *Design and synthesis of synchronization skeletons using branching time temporal logic*, Proc. of Work. on Logic of Programs, LNCS 131, 1981, pp. 52–71.

[CGP99] E.M. Clarke, O. Grumberg, and D.A. Peled, *Model checking*, MIT Press, Cambridge, MA, USA, 1999.

[Hoa85] C.A.R. Hoare, *Communicating sequential processes*, Prentice-Hall International. Upper Saddle River, NJ, USA, 1985.

[HP85] D. Harel and A. Pnueli, *On the development of reactive systems*, Logics and Models of Concurrent Systems, NATO Advanced Summer Institutes, vol. F-13, Springer-Verlag, 1985, pp. 477–498.

[Koz83] D. Kozen, *Results on the propositional mu-calculus.*, Theoretical Computer Science **27** (1983), 333–354.

[KSV02] O. Kupferman, U. Sattler, and M.Y. Vardi, *The complexity of the graded μ-calculus*, CADE'02, LNAI 2392, 2002, pp. 423–437.

[KVW01] O. Kupferman, M.Y. Vardi, and P. Wolper, *Module checking*, Information and Computation **164** (2001), 322–344.

[QS81] J.P. Queille and J. Sifakis, *Specification and verification of concurrent systems in cesar*, Proc. of 5th Int. Symposium on Programming, LNCS 137, 1981, pp. 337–351.

[SV01] U. Sattler and M.Y. Vardi, *The hybrid mu-calculus*, IJCAR'01, LNAI 2083, 2001, pp. 76–91.

[Var98] M.Y. Vardi, *Reasoning about the past with two-way automata*, ICALP'98, LNCS 1443, 1998, pp. 628–641.

[VW86] M.Y. Vardi and P. Wolper, *An automata-theoretic approach to automatic program verification (preliminary report)*, LICS '86, 1986, pp. 332–344.

[Zap02] J. Zappe, *Modal μ-calculus and alternating tree automata*, Automata, Logics, and Infinite Games (E. Grädel, W. Thomas, and T. Wilke, eds.), LNCS, vol. 2500, Springer, 2002, pp. 171–184.

PDL with Intersection and Converse Is 2EXP-Complete

Stefan Göller[1,⋆], Markus Lohrey[1], and Carsten Lutz[2]

[1] Universität Stuttgart, FMI, Germany
[2] Institute for Theoretical Computer Science, TU Dresden, Germany
goeller,lohrey@informatik.uni-stuttgart.de,
lutz@tcs.inf.tu-dresden.de

Abstract. We study the complexity of satisfiability in the expressive extension ICPDL of PDL (Propositional Dynamic Logic), which admits intersection and converse as program operations. Our main result is containment in 2EXP, which improves the previously known non-elementary upper bound and implies 2EXP-completeness due to an existing lower bound for PDL with intersection. The proof proceeds by showing that every satisfiable ICPDL formula has a model of tree-width at most two and then giving a reduction to the (non)-emptiness problem for alternating two-way automata on infinite trees. In this way, we also reprove in an elegant way Danecki's difficult result that satisfiability for PDL with intersection is in 2EXP.

1 Introduction

Propositional Dynamic Logic (PDL) was introduced by Fischer and Ladner in 1979 as a modal logic for reasoning about the input/output behaviour of programs [6]. In PDL, there are two syntactic entities: formulas, built from Boolean and modal operators and interpreted as sets of nodes of a Kripke structure; and programs, built from the operators test, union, composition, and Kleene star (reflexive transitive closure) and interpreted as binary relations in a Kripke structure. Since its invention, many different extensions of PDL have been proposed, mainly by allowing additional operators on programs. Three prominent such extensions are PDL with the converse operator (CPDL), PDL with the intersection operator (IPDL), and PDL with the negation operator on programs (NPDL), see the monograph [9] and references therein. While some of these extensions such as CPDL are well-suited for reasoning about programs, many of them aim at the numerous other applications that PDL has found since its invention. Notable examples of such applications include agent-based systems [14], regular path constraints [2], and XML-querying [1,17]. In AI, PDL received attention due to its close relationship to description logics [7] and epistemic logic [18,10].

The most important decision problem for PDL is satisfiability: is there a Kripke structure which satisfies a given formula at some node? A classical result of Fischer and Ladner states that satisfiability for PDL is EXP-complete [6,16]. The EXP upper bound extends without difficulty to CPDL and can even be established for several extensions

⋆ The first and second author are supported by the DFG project GELO; the third author is supported by the EU funded IST-2005-7603 FET Project Thinking Ontologies (TONES).

thereof [19]. In contrast, the precise complexity of satisfiability for IPDL was a long standing open problem. In [4], Danecki proved a 2EXP upper bound. Alas, Danecki's proof is rather difficult and many details are omitted in the published version. One of the reasons for the difficulty of IPDL is that, unlike PDL, it lacks the tree model property, i.e., a satisfiable IPDL formula does not necessarily have a tree model. Danecki proved that every satisfiable IPDL formula has a special model which can be encoded by a tree. This observation paves the way to using automata theoretic techniques in decision procedures for IPDL. Only recently, a matching 2EXP lower bound for IPDL was shown by Lange and the third author [11]. Regarding NPDL, it is long known that satisfiability is undecidable [9]. As recently shown in [9], the fragment of NPDL in which program negation is restricted to atomic programs is decidable and EXP-complete.

In this paper, we consider extensions of PDL with (at least two of) converse, intersection, and negation. Our main result concerns the complexity of satisfiability in ICPDL, the extension of PDL with both converse and intersection. Decidability was shown by the third author in [12] using a reduction to monadic second order logic over the infinite binary tree. However, this only yields a nonelementary algorithm which does not match the 2EXP lower bound that ICPDL inherits from IPDL. We prove that satisfiability in ICPDL can be decided in 2EXP, and thus settle the complexity of ICPDL as 2EXP-complete. There are some additional virtues of our result. First, we provide a shorter and (hopefully) more comprehensible proof of the 2EXP upper bound for IPDL. Second, the information logic DAL (data analysis logic) [5] is a fragment of ICPDL (but not of IPDL) and thus inherits the 2EXP upper bound. And third, our result has applications in description logic and epistemic logic, see [12] for more details.

Our main result is proved in three clearly separated parts. In part one, we establish a model property for ICPDL based on the notion of tree width. Tree width measures how close a graph is to a tree, and is one of the most important concepts in modern graph theory with many applications in computer science. As mentioned earlier, IPDL (and hence also ICPDL) does not have the tree model property. We prove that ICPDL enjoys an "almost tree model property": every satisfiable ICPDL formula has a model of tree width at most two This part of our proof is comparable to Danecki's observation that every satisfiable IPDL formula has a special model which can be encoded by a tree.

In part two of our proof, we use the established model property to give a polytime reduction of satisfiability in ICPDL to what we call *ω-regular tree satisfiability* in ICPDL. The latter problem is defined in terms of two-way alternating parity tree automata (TWAPTAs). A TWAPTA is an alternating automaton with a parity acceptance condition that runs on infinite node-labeled trees and can move upwards and downwards in the tree. Infinite node-labeled trees can be viewed in a natural way as Kripke structures and thus we can interpret ICPDL formulas in such trees. Now, ω-regular tree satisfiability in ICPDL is the following problem: given an ICPDL formula φ and a TWAPTA $\mathcal{T}$, is there a tree accepted by $\mathcal{T}$ which is a model of φ? Our reduction of satisfiability in ICPDL to this problem is based on a suitable encoding of width two tree decompositions of Kripke structures. The TWAPTA constructed in the reduction accepts precisely such encodings.

Finally, in part three we reduce ω-regular tree satisfiability in ICPDL to the nonemptiness problem for TWAPTAs. The latter problem was shown to be EXP-complete in [20]. Since our reduction of ω-regular tree satisfiability in ICPDL to TWAPTA-non-emptiness involves an exponential blow-up in automata size, we obtain an 2EXP upper bound for ω-regular tree satisfiability in ICPDL and also for standard satisfiability in ICPDL. The reduction employs a technique from [8], where the first and second author proved that the model-checking problem for IPDL over transition graphs of pushdown automata is 2EXP-complete. In fact, this model-checking problem can be easily reduced to ω-regular tree satisfiability in ICPDL. This illustrates that ω-regular tree satisfiability in ICPDL is of interest beyond its application in the current paper.

To obtain a more complete picture, we finally investigate the option of extending ICPDL with program negation. It turns out that in the presence of intersection, program negation is problematic from a computational perspective. In particular, we prove that already IPDL extended with negation restricted to atomic programs is undecidable. This should be contrasted with the decidability result for PDL extended with atomic negation mentioned above [13].

2 ICPDL

Let $\mathbb{P}$ be a set of *atomic propositions* and $\mathbb{A}$ a set of *atomic programs*. *Formulas* φ and *programs* π of the logic ICPDL are defined by the following grammar, where p ranges over $\mathbb{P}$ and a over $\mathbb{A}$:

$$\begin{aligned}\varphi &::= p \mid \neg\varphi \mid \langle\pi\rangle\,\varphi\\ \pi &::= a \mid \pi_1 \cup \pi_2 \mid \pi_1 \cap \pi_2 \mid \pi_1 \circ \pi_2 \mid \pi^* \mid \overline{\pi} \mid \varphi?\end{aligned}$$

We introduce the usual abbreviations $\varphi_1 \wedge \varphi_2 = \langle\varphi_1?\rangle\varphi_2$, $\varphi_1 \vee \varphi_2 = \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, and $[\pi]\varphi = \neg\langle\pi\rangle\neg\varphi$. The fragment IPDL of ICPDL is obtained by dropping the $\overline{\pi}$ clause from the above grammar.

The *semantics* of ICPDL is defined in terms of Kripke structures. A *Kripke structure* is a tuple $K = (X, \{\to_a \mid a \in \mathbb{A}\}, \rho)$, where (i) X is a set of *states*, (ii) $\to_a \subseteq X \times X$ is a *transition relation* for each $a \in \mathbb{A}$, and (iii) $\rho : X \to 2^{\mathbb{P}}$ assigns to each state a set of atomic propositions. Given a Kripke structure $K = (X, \{\to_a \mid a \in \mathbb{A}\}, \rho)$, we define by mutual induction for each ICPDL program π a binary relation $[\![\pi]\!]_K \subseteq X \times X$ and for each ICPDL formula φ a subset $[\![\varphi]\!]_K \subseteq X$ as follows: [1]

$$\begin{aligned}[\![p]\!]_K &= \{x \mid p \in \rho(x)\} \text{ for } p \in \mathbb{P}\\ [\![\neg\varphi]\!]_K &= X \setminus [\![\varphi]\!]_K\\ [\![\langle\pi\rangle\varphi]\!]_K &= \{x \mid \exists y : (x,y) \in [\![\pi]\!]_K \wedge y \in [\![\varphi]\!]_K\}\\ [\![a]\!]_K &= \to_a \text{ for } a \in \mathbb{A}\\ [\![\varphi?]\!]_K &= \{(x,x) \mid x \in [\![\varphi]\!]_K\}\\ [\![\pi^*]\!]_K &= [\![\pi]\!]_K^*\\ [\![\overline{\pi}]\!]_K &= \{(y,x) \mid (x,y) \in [\![\pi]\!]_K\}\\ [\![\pi_1 \text{ op } \pi_2]\!]_K &= [\![\pi_1]\!]_K \text{ op } [\![\pi_2]\!]_K \text{ for op} \in \{\cup, \cap, \circ\}\end{aligned}$$

[1] Overloading notation, we use $\circ$ both as a program operator of ICPDL and to denote the composition operator for binary relations, i.e., $R \circ S = \{(a,b) \mid \exists c : (a,c) \in R, (c,b) \in S\}$.

For $x \in X$ we write $(K, x) \models \varphi$ if $x \in [\![\varphi]\!]_K$. If $(K, x) \models \varphi$ for some $x \in X$, then K is a *model* of φ. The formula φ is *satisfiable* if there exists a model of φ.

Since the converse operator can be pushed down to atomic programs, we assume for the rest of this paper that converse is only applied to atomic programs. Let us set $\overline{\mathbb{A}} = \{\overline{a} \mid a \in \mathbb{A}\}$. The size $|\varphi|$ of an ICPDL formula φ and the size $|\pi|$ of an ICPDL program π is defined as follows: $|p| = |a| = 1$ for all $p \in \mathbb{P}$ and $a \in \mathbb{A} \cup \overline{\mathbb{A}}$, $|\neg\varphi| = |\varphi?| = |\varphi| + 1$, $|\langle\pi\rangle\varphi| = |\pi| + |\varphi| + 1$, $|\pi_1 \text{ op } \pi_2| = |\pi_1| + |\pi_2| + 1$ for op $\in \{\cup, \cap, \circ\}$, and $|\pi^*| = |\pi| + 1$.

The main result of this paper is the following.

Theorem 1. *Satisfiability in ICPDL is* 2EXP*-complete.*

As discussed in the introduction, it suffices to give a 2EXP algorithm for satisfiability in ICPDL because of the known 2EXP lower bound for IPDL [11]. The rest of the paper is organized as follows. In Section 3, we show that every satisfiable ICPDL formula has a model of tree width at most two. In Section 4, satisfiability of ICPDL formulas in a model of tree width at most two is reduced to ω-regular tree satisfiability in ICPDL. In Section 5, the latter problem is shown to be in 2EXP. Finally, Section 6 contains the undecidability proof for IPDL extended with negation of atomic programs.

3 Models of Tree-Width Two Suffice

We start with defining tree decompositions and the tree-width of Kripke structures. Although we do not assume countability of Kripke structures in general, it suffices to consider tree decompositions and the tree width only of countable Kripke structures. Let $K = (X, \{\rightarrow_a \mid a \in \mathbb{A}\}, \rho)$ be a countable Kripke structure. A *tree decomposition* of K is a tuple $(T, (X_v)_{v \in V})$, where $T = (V, E)$ is a countable undirected tree, X_v is a subset of X (also called a *bag*) for all $v \in V$, and the following conditions are satisfied:

- $\bigcup_{v \in V} X_v = X$
- For every transition $x \rightarrow_a y$ of K there exists $v \in V$ with $x, y \in X_v$.
- For every $x \in X$, the set $\{v \in V \mid x \in X_v\}$ is a connected subset of the tree T.

The width of this tree decomposition is the supremum of $\{|X_v| - 1 \mid v \in V\}$. The *tree width* of a Kripke structure K is the minimal k such that K has a tree decomposition of width k. The purpose of this section is to prove the following theorem.

Theorem 2. *Every satisfiable ICPDL formula has a countable model of tree width at most two.*

As a preliminary to proving Theorem 2, we mutually define the set of *subprograms* $\mathsf{subp}(\alpha)$ and the set of *subformulas* $\mathsf{subf}(\alpha)$, where α is either an ICPDL formula or an ICPDL program:

- $\mathsf{subp}(a) = \{a\}$, $\mathsf{subp}(\overline{a}) = \{a, \overline{a}\}$, $\mathsf{subf}(a) = \mathsf{subf}(\overline{a}) = \emptyset$ for $a \in \mathbb{A}$;
- $\mathsf{subp}(\pi) = \{\pi\} \cup \mathsf{subp}(\pi_1) \cup \mathsf{subp}(\pi_2)$ and $\mathsf{subf}(\pi) = \mathsf{subf}(\pi_1) \cup \mathsf{subf}(\pi_2)$ if $\pi = \pi_1 \text{ op } \pi_2$ for op $\in \{\cup, \cap, \circ\}$;
- $\mathsf{subp}(\pi^*) = \{\pi^*\} \cup \mathsf{subp}(\pi)$ and $\mathsf{subf}(\pi^*) = \mathsf{subf}(\pi)$;

– $\mathsf{subp}(\varphi?) = \{\varphi?\} \cup \mathsf{subp}(\varphi)$ and $\mathsf{subf}(\varphi?) = \mathsf{subf}(\varphi)$
– $\mathsf{subp}(p) = \emptyset$ and $\mathsf{subf}(p) = \{p\}$ for $p \in \mathbb{P}$;
– $\mathsf{subp}(\neg\varphi) = \mathsf{subp}(\varphi)$ and $\mathsf{subf}(\neg\varphi) = \{\neg\varphi\} \cup \mathsf{subf}(\varphi)$;
– $\mathsf{subp}(\langle\pi\rangle\varphi) = \mathsf{subp}(\pi) \cup \mathsf{subp}(\varphi)$ and $\mathsf{subf}(\langle\pi\rangle\varphi) = \{\langle\pi\rangle\varphi\} \cup \mathsf{subf}(\pi) \cup \mathsf{subf}(\varphi)$.

To prove Theorem 2, fix a satisfiable formula φ_0, a (not necessarily countable) model $K = (X, \{\rightarrow_a \mid a \in \mathbb{A}\}, \rho)$ of φ_0, and a state $x_0 \in [\![\varphi_0]\!]_K$. Also fix choice functions W (for witness), U (for union), C (for composition), and S (for star) such that

– if $\varphi = \langle\pi\rangle\psi \in \mathsf{subf}(\varphi_0)$ and $x \in [\![\varphi]\!]_K$, then $W(x, \varphi) = y \in X$ such that $y \in [\![\psi]\!]_K$ and $(x, y) \in [\![\pi]\!]_K$;
– if $\pi = \chi \cup \sigma \in \mathsf{subp}(\varphi_0)$ and $(x, y) \in [\![\pi]\!]_K$, then $U(x, \pi, y) = \tau \in \{\chi, \sigma\}$ such that $(x, y) \in [\![\tau]\!]_K$.
– if $\pi = \chi \circ \sigma \in \mathsf{subp}(\varphi_0)$ and $(x, y) \in [\![\pi]\!]_K$, then $C(x, \pi, y) = z \in X$ such that $(x, z) \in [\![\chi]\!]_K$ and $(z, y) \in [\![\sigma]\!]_K$;
– if $\pi = \chi^* \in \mathsf{subp}(\varphi_0)$ and $(x, y) \in [\![\pi]\!]_K$ with $x \neq y$, then $S(x, \pi, y) = z \in X$ such that there exists a sequence $x_0, \ldots, x_n \in X$ with
  1. $x_0 = x$ and $x_n = y$;
  2. $(x_i, x_{i+1}) \in [\![\chi]\!]_K$ for all $i < n$;
  3. $x_0, \ldots, x_n$ is a shortest sequence with Properties 1 and 2;
  4. $x_1 = z$.

Now we inductively define a node-labeled tree $(T, (t_v)_{v \in V})$ with $T = (V, E)$ and $t_v \in X \cup X^2 \cup X^3$ for all $v \in V$. During the construction, each node in the tree is assigned a type, which may either be "singleton" or π for $\pi \in \mathsf{subp}(\varphi_0)$. Figure 1 illustrates the different cases, which are as follows:

1. Start the construction with a root node v of type singleton and set $t_v = x_0$;
2. if $v \in V$ is of type singleton and $t_v = x$, then for every $\varphi = \langle\pi\rangle\psi \in \mathsf{subf}(\varphi_0)$ such that $x \in [\![\varphi]\!]_K$, add a successor w of type π and set $t_w = (x, W(x, \varphi))$;
3. if $v \in V$ is of type a or $\overline{a}$, where $a \in \mathbb{A}$ and $t_v = (x, y)$, then add a successor w of type singleton and set $t_w = y$;
4. if $v \in V$ is of type $\pi = \chi \cup \sigma$ and $t_v = (x, y)$, then
   – add a successor w of type singleton and set $t_w = y$;
   – add a successor w' of type $U(x, \pi, y)$ and set $t_{w'} = (x, y)$;
5. if $v \in V$ is of type $\pi = \chi \cap \sigma$ and $t_v = (x, y)$, then
   – add a successor w of type singleton and set $t_w = y$;
   – add successors u, u' of type χ and σ, respectively, and set $t_u = t_{u'} = (x, y)$;
6. if $v \in V$ is of type $\pi = \chi \circ \sigma$ and $t_v = (x, y)$, then
   – add a successor w of type singleton and set $t_w = y$;
   – add a successor w' of type π and set $t_w = (x, C(x, \pi, y), y)$;
7. if $v \in V$ is of type $\pi = \chi \circ \sigma$ and $t_v = (x, z, y)$, then add successors u, u' of type χ and σ and set $t_u = (x, z)$ and $t_{u'} = (z, y)$;
8. if $v \in V$ is of type $\pi = \chi^*$ and $t_v = (x, y)$ with $x \neq y$, then
   – add a successor w of type singleton and set $t_w = y$;
   – add a successor w' of type π and set $t_w = (x, S(x, \pi, y), y)$;
9. if $v \in V$ is of type $\pi = \chi^*$ and $t_v = (x, z, y)$, then add successors u, u' of type χ and π, respectively, and set $t_u = (x, z)$ and $t_{u'} = (z, y)$.

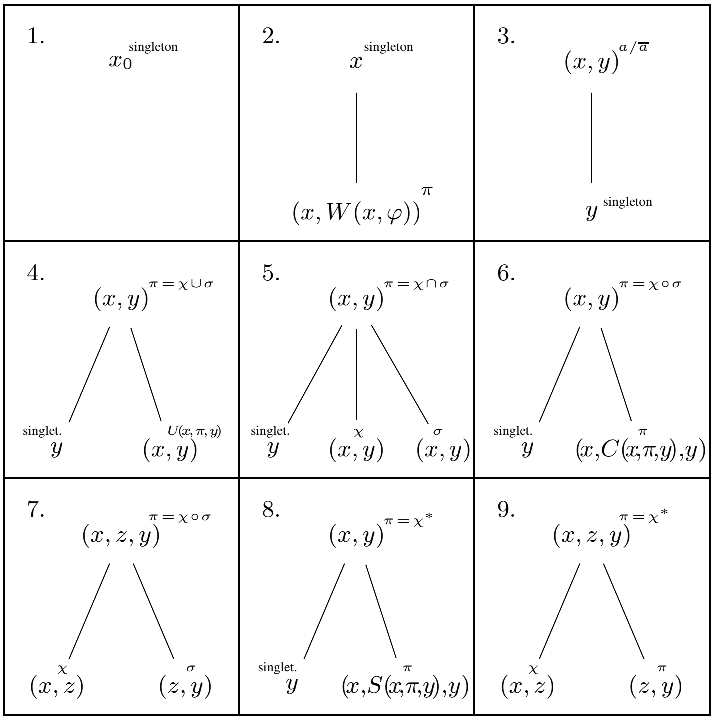


Fig. 1. Inductive definition of $(T, (t_v)_{v \in V})$

We assume that successors are added at most once to each node in the induction step and that the construction proceeds in a breadth first manner. Note that nodes of type $\psi?$ are always leafs, and so are nodes v of type χ^* with $t_v = (x,x)$ for some $x \in X$. Another important property, which illustrates the connection between K and the constructed tree, is the following:

$$\forall v \in V : \text{if } v \text{ is of type } \pi \text{ and } t_v = (x,y), \text{ then } (x,y) \in [\![\pi]\!]_K.$$

A *place* is a pair (v,x) such that x is a member of t_v. We denote the set of all places with P and let $\sim$ be the smallest equivalence relation on P which contains all pairs of the form $((u,x),(v,x))$, where $(u,v) \in E$ is an edge of the tree T. We use $[v,x]$ to denote the equivalence class of $(v,x) \in P$ w.r.t. the relation $\sim$. Define a Kripke structure $K' = (X', \{\to'_a \mid a \in \mathbb{A}\}, \rho')$ as follows:

- $X' = \{[v,x] \mid (v,x) \in P\}$;
- $[v,x] \to'_a [v',y]$ if and only if at least one of the following holds:
  - there is $u \in V$ of type a s.t. $t_u = (x,y)$, $(u,x) \sim (v,x)$, and $(u,y) \sim (v',y)$;
  - there is $u \in V$ of type $\overline{a}$ s.t. $t_u = (y,x)$, $(u,x) \sim (v,x)$, and $(u,y) \sim (v',y)$.
- $\rho'([v,x]) = \rho(x)$.

Since K' is clearly countable, to finish the proof it suffices to show the following:

1. setting $X_v = \{[v, x] \mid x$ occurs in $t_v\}$ for all $v \in V$, we obtain a tree decomposition $(T, (X_v)_{v \in V})$ of K' of width at most two;
2. K' satisfies φ_0.

Using the definitions of K' and $\sim$, it is readily checked that $(T, (X_v)_{v \in V})$ is a tree decomposition of K'. Tree width two is then immediate by construction of $(T, (t_v)_{v \in V})$. Finally, we can prove the following, whose Point 3 yields that K' is a model of φ_0.

Lemma 1. *For all $v, u \in V$, $x, y \in X$, $\pi \in \mathsf{subp}(\varphi_0)$, and $\varphi \in \mathsf{subf}(\varphi_0)$,*

1. *if $t_v = (x, y)$ and v is of type π, then $([v, x], [v, y]) \in [\![\pi]\!]_{K'}$;*
2. *if $(v, x), (u, y) \in P$ and $([v, x], [u, y]) \in [\![\pi]\!]_{K'}$, then $(x, y) \in [\![\pi]\!]_K$;*
3. *if $(v, x) \in P$, then $(K, x) \models \varphi$ if and only if $(K', [v, x]) \models \varphi$.*

4 Reduction to ω-Regular Tree Satisfiability

We exploit the model property established in the previous section to reduce satisfiability in ICPDL to ω-regular tree satisfiability in ICPDL. Since the latter is defined in terms of alternating automata on infinite trees, we start with introducing these automata and the trees on which they work.

Let Γ and Υ be finite sets. A Γ-labeled (directed) Υ-tree is a partial function $T : \Upsilon^* \to \Gamma$ such that $\mathrm{dom}(T)$ (the set of nodes) is prefix-closed. If $\mathrm{dom}(T) = \Upsilon^*$, then T is called *complete*. If Υ is understood or not important, we simply talk of Γ-labeled trees. We deliberately work with two kinds of trees here: undirected trees as a basis for tree decompositions in Section 3, and directed trees introduced here as the objects on which alternating tree automata work.

Let P be a finite set of atomic propositions and A a finite set of atomic programs, not necessarily identical to the sets $\mathbb{P}$ and $\mathbb{A}$ fixed in Section 2. A complete 2^{P}-labeled A-tree T can be viewed as a Kripke structure $K_T = (\mathsf{A}^*, \{\to_a \mid a \in \mathsf{A}\}, T)$ over the set of atomic propositions P and atomic programs A, where $\to_a = \{(u, ua) \mid u \in \mathsf{A}^*\}$ for all $a \in \mathsf{A}$. In the following, we identify T and the associated Kripke structure K_T.

We now define alternating automata on complete Γ-labeled Υ-trees. For a finite set X we denote by $\mathcal{B}^+(X)$ the set of all *positive boolean formulas* with elements of X used as variables. The constants `true` and `false` are admitted. A subset $Y \subseteq X$ can be seen as a valuation in the obvious way: it *satisfies* a formula $\theta \in \mathcal{B}^+(X)$ if and only if by assigning `true` to all elements in Y the formula θ is evaluated to `true`. Define the set of Υ*-moves* as $\mathrm{mov}(\Upsilon) = \Upsilon \uplus \overline{\Upsilon} \uplus \{\varepsilon\}$, where $\overline{\Upsilon} = \{\overline{a} \mid a \in \Upsilon\}$. For $u \in \Upsilon^*$ and $a \in \Upsilon$, define $u\overline{a} = v$ if $u = va$ for some $v \in \Upsilon^*$ and $u\overline{a} =$ undefined if $u \notin \Upsilon^* a$. A *two-way alternating parity tree automaton* (TWAPTA for short) over Γ-labeled Υ-trees is a tuple $\mathcal{T} = (S, \delta, s_0, \mathrm{Acc})$, where (i) S is a finite non-empty set of states, (ii) $\delta : S \times \Gamma \to \mathcal{B}^+(S \times \mathrm{mov}(\Upsilon))$ is the *transition function*, (iii) $s_0 \in S$ is the *initial state*, and (iv) $\mathrm{Acc} : S \to \mathbb{N}$ is the *priority function* which assigns to each state a nonnegative integer. Define $|\mathrm{Acc}| = \max\{\mathrm{Acc}(s) \mid s \in S\}$. Let T be a complete Γ-labeled Υ-tree, $u \in \Upsilon^*$ a node, and $s \in S$ a state. An $(s, u)_T$*-run* of $\mathcal{T}$ is a (not necessarily complete) $(S \times \Upsilon^*)$-labeled Ω-tree T_R for some finite set Ω such that the following two conditions are satisfied: (i) $T_R(\varepsilon) = (s, u)$, and (ii) if $\alpha \in \mathrm{dom}(T_R)$ with

$T_R(\alpha) = (q, v)$ and $\delta(q, T(v)) = \theta$, then there exists a subset $Y \subseteq S \times \mathrm{mov}(\Upsilon)$ that satisfies the formula θ and for all $(s', e) \in Y$, ve is defined and there exists a $\sigma \in \Omega$ with $\alpha\sigma \in \mathrm{dom}(T_R)$ and $T_R(\alpha\sigma) = (s', ve)$. We say that an $(s, u)_T$-run is *successful*, if for every infinite path $\alpha_1\alpha_2 \cdots \in \mathrm{dom}(T_R)^\omega$ of T_R ($\alpha_1 = \varepsilon$, $\alpha_{i+1} = \alpha_i\sigma$ for some $\sigma \in \Omega$), the number $\min\{\mathrm{Acc}(q) \mid q \in S, T_R(\alpha_i) \in \{q\} \times \Upsilon^* \text{ for infinitely many } i\}$ is even. Define

$$[\![\mathcal{T}, s]\!]_T = \{u \in \Upsilon^* \mid \text{ there exists a successful } (s, u)_T\text{-run of } \mathcal{T}\} \text{ and}$$
$$L(\mathcal{T}) = \{T \mid \varepsilon \in [\![\mathcal{T}, s_0]\!]_T\}.$$

The subscript T is omitted if clear from the context. An *ω-regular tree language* L is a set of complete Γ-labeled Υ-trees such that $L(\mathcal{T}) = L$ for some TWAPTA $\mathcal{T}$.

Our TWAPTA model differs slightly from other definitions in the literature: First, we run TWAPTA only on complete trees; this will be convenient in Section 5. Second, usually a TWAPTA has an operation $\uparrow$ for moving to the parent node of the current node. In our model, $\uparrow$ is replaced by the operations $\overline{a} \in \overline{\Upsilon}$ for all $a \in \Upsilon$. The operation $\overline{a}$ can only be executed if the current node is an a-successor of its parent node. It is easy to see that these two models are equivalent.

In Section 5, we will make use of the following result of Vardi:

Theorem 3 ([20]). *For a given TWAPTA $\mathcal{T} = (Q, \delta, s_0\mathrm{Acc})$ it can be checked in time exponential in $|Q| \cdot |\mathrm{Acc}|$ whether $L(\mathcal{T}) = \emptyset$.*

We are now in the position to formally define *ω-regular tree satisfiability in ICPDL*: given a TWAPTA $\mathcal{T}$ over 2^{P}-labeled A-trees and an ICPDL formula φ using only atomic propositions from P and atomic programs from A (in the following we simply say that φ is *over* P *and* A), decide whether there is a $T \in L(\mathcal{T})$ such that $(T, \varepsilon) \models \varphi$.

To reduce satisfiability in ICPDL to ω-regular tree satisfiability in ICPDL, we translate an ICPDL formula φ over $\mathbb{P}$ and $\mathbb{A}$ into a TWAPTA $\mathcal{T}$ and an ICPDL formula $\widehat{\varphi}$ over

$$\mathsf{A} = \{a, b, 0, 1, 2\} \quad \text{and} \quad \mathsf{P} = \{t\} \cup \mathsf{prop}(\varphi) \cup (\{0, 1, 2\} \times \mathsf{prog}(\varphi) \times \{0, 1, 2\}),$$

where $\mathsf{prop}(\varphi) = \mathsf{subf}(\varphi) \cap \mathbb{P}$ and $\mathsf{prog}(\varphi) = \mathsf{subp}(\varphi) \cap \mathbb{A}$. Intuitively, each 2^{P}-labeled A-tree T accepted by $\mathcal{T}$ encodes a tree decomposition of a Kripke structure K over $\mathbb{P}$ and $\mathbb{A}$ of tree width at most two (in a sense yet to be made precise), and T is a model of $\widehat{\varphi}$ if and only if K is a model of φ. To achieve an elegant encoding of tree decompositions, we work with *good* tree decompositions. A tree decomposition $(T, (X_v)_{v \in V})$ with $T = (V, E)$ is called good if

- $V = \{a, b\}^*$, i.e., T is a complete binary tree, and
- $X_v \subseteq X_{vc}$ or $X_{vc} \subseteq X_v$ for all $v \in V$ and $c \in \{a, b\}$.

It is easily seen how to convert a tree decomposition of a Kripke structure K of width k into a good tree decomposition of K of width k by introducing additional nodes.

Lemma 2. *Every countable Kripke structure of tree width k has a good tree decomposition of width k.*

In the following, we only need the case where $k = 2$. To encode a good tree decomposition $(T, (X_v)_{v \in \{a,b\}^*})$ of width two of a Kripke structure as a 2^{P}-labeled A-tree, we think of every tree node $v \in \{a,b\}^*$ as being divided into three slots which can be empty or filled with a state of the Kripke structure. When moving to a child, by the second condition of good tree decompositions we either add nodes to empty slots or remove nodes from slots, but not both. The three slots of the node v are described by new leafs $v0, v1, v2$. This explains our choice of A above. When slot vi is occupied by a state of the Kripke structure, then vi receives the special label $t \in \mathsf{P}$ (and probably propositional letters as additional labels). Information about the edges of the Kripke structure are stored in tree nodes from $\{a,b\}^*$. We now formally define these encodings. We work with complete trees because TWAPTAs work on such trees. Nodes that are present only to ensure completeness of the tree are labelled with the empty set. A complete 2^{P}-labeled A-tree T is called *valid* if the following holds for all $v \in \mathsf{A}^*$:

– if $v \in \{a,b\}^*$ and $i \in \{0,1,2\}$, then either $T(vi) = \emptyset$ or $\{t\} \subseteq T(vi) \subseteq \{t\} \cup \mathbb{P}$; set $X_v := \{i \mid t \in T(vi)\}$;
– if $v \in \{a,b\}^*$, then $T(v) \subseteq X_v \times \mathbb{A} \times X_v$;
– if $v \in \{a,b\}^*$ and $c \in \{a,b\}$, then $X_v \subseteq X_{vc}$ or $X_{vc} \subseteq X_v$;
– if $v \notin \{a,b\}^* \cup \{a,b\}^*\{0,1,2\}$, then $T(v) = \emptyset$.

Let T be a valid 2^{P}-labeled A-tree. We now make precise the Kripke structure $K(T)$ over $\mathbb{P}$ and $\mathbb{A}$ whose good tree decomposition is described by T. Define a set of *places* $P = \{u \in \mathsf{A}^* \mid t \in T(u)\}$ and let $\sim$ be the smallest equivalence relation on P which contains all pairs $(vi, vci) \in P \times P$, where $v \in \{a,b\}^*$, $c \in \{a,b\}$, and $0 \le i \le 2$. For $u \in P$, we use $[u]$ to denote the equivalence class of u w.r.t. $\sim$. Now set $K(T) = (X, \{\rightarrow_a \mid a \in \mathbb{A}\}, \rho)$, where:

$$\begin{aligned} X &= \{[u] \mid u \in P\} \\ \rightarrow_a &= \{([vi],[vj]) \mid v \in \{a,b\}^*, (i,a,j) \in T(v)\} \\ \rho([u]) &= \bigcup_{v \in [u]} T(v) \cap \mathbb{P} \end{aligned}$$

The structure $K(T)$ should not be confused with T *viewed* as a Kripke structure over P and A as discussed at the beginning of this section: the original formula φ whose satisfiability is to be decided is interpreted in $K(T)$ whereas the reduction formula $\widehat{\varphi}$, to be defined below, is interpreted in T viewed as a Kripke structure. The following two lemmas are easily proved.

Lemma 3. *If T is a valid 2^{P}-labeled A-tree, then the Kripke structure $K(T)$ has tree width at most two. Conversely, if K is of tree width at most two, then there exists a valid 2^{P}-labeled A-tree T such that K is isomorphic to $K(T)$.*

Lemma 4. *The set of all valid 2^{P}-labeled A-trees is an ω-regular tree language.*

Now we show how to convert formulas ψ and programs π over $\mathsf{prop}(\varphi)$ and $\mathsf{prog}(\varphi)$ into formulas $\widehat{\psi}$ and programs $\widehat{\pi}$ over P and A such that for every valid 2^{P}-labeled A-tree T, we have (i) $[\![\widehat{\pi}]\!]_T \subseteq P \times P$ and (ii) for all $u, v \in P$,

$$\begin{aligned} u \in [\![\widehat{\psi}]\!]_T &\Leftrightarrow [u] \in [\![\psi]\!]_{K(T)} \\ (u,v) \in [\![\widehat{\pi}]\!]_T &\Leftrightarrow ([u],[v]) \in [\![\pi]\!]_{K(T)} \end{aligned}$$

First, we define the auxiliary program

$$\pi_\sim^1 = \bigcup_{i\in\{0,1,2\}} t? \circ \overline{i} \circ (a \cup b \cup \overline{a} \cup \overline{b}) \circ i \circ t?$$

and let $\pi_\sim = (\pi_\sim^1)^*$. Note that $[\![\pi_\sim]\!]_T$ equals $\sim$. Now, for all $a \in \mathsf{prog}(\varphi)$ and $p \in \mathsf{prop}(\varphi)$ we define

$$\widehat{a} = \bigcup_{i,j\in\{0,1,2\}} \pi_\sim \circ \overline{i} \circ (i,a,j)? \circ j \circ \pi_\sim \quad \text{and} \quad \widehat{p} = \langle \pi_\sim \rangle p.$$

To extend this translation to complex ICPDL formulas and programs, we can simply replace all atomic programs a and formulas p with $\widehat{a}$ and $\widehat{p}$, respectively. From the construction of $\widehat{\varphi}$ and Lemmas 2 and 3, we obtain the following.

Proposition 1. *The formula φ has a model of tree width at most two if and only if there is a valid 2^{P}-labeled A-tree T such that $(T, \varepsilon) \models \langle (0 \cup 1 \cup 2) \circ t? \rangle \widehat{\varphi}$.*

From Theorem 2, Lemma 4, and Proposition 1, we obtain:

Theorem 4. *There is a polynomial time reduction from satisfiability in ICPDL to ω-regular tree satisfiability in ICPDL.*

5 ω-Regular Tree Satisfiability in ICPDL Is in 2EXP

Our remaining goal is to show that ω-regular tree satisfiability in ICPDL can be solved in doubly exponential time. This is achieved by a reduction to the EXP-complete (non)-emptiness problem for TWAPTAs. The main ingredient of the reduction is an inductive translation of ICPDL formulas into TWAPTAs and ICPDL programs into a certain kind of non-deterministic automata which we call NFAs. NFAs resemble word automata, but navigate in a complete A-tree reading symbols from $\mathsf{A} \cup \overline{\mathsf{A}}$. They can make conditional ε-transitions, which are executable only if the current tree node is accepted by some fixed TWAPTA. We start with presenting NFAs and the inductive translation.

Fix a finite set of atomic propositions P and a finite set of atomic programs A. For the rest of this section, it is more convenient to assume that a TWAPTA does not have an initial state. Hence, it is just a tuple of the form $(S, \delta, \mathrm{Acc})$. A *non-deterministic finite automaton (NFA)* A over a TWAPTA $\mathcal{T} = (S, \delta, \mathrm{Acc})$ is a pair $(Q, \rightarrow_A)$, where Q is a finite set of *states* and $\rightarrow_A$ is a set of transitions of the following form, where $q, q' \in Q$:

$$q \xrightarrow{a}_A q' \text{ with } a \in \mathsf{A} \cup \overline{\mathsf{A}} \quad \text{or} \quad q \xrightarrow{\mathcal{T},s}_A q' \text{ with } s \in S.$$

Transitions of the latter kind are called *test transitions*. Let T be a complete 2^{P}-labeled A-tree. Define the relation $\Rightarrow_{A,T} \subseteq (\mathsf{A}^* \times Q) \times (\mathsf{A}^* \times Q)$ as the smallest relation such that

- $(u,p) \Rightarrow_{A,T} (ua,q)$ if $p \xrightarrow{a}_A q$ ($a \in \mathsf{A}, u \in \mathsf{A}^*$);
- $(ua,p) \Rightarrow_{A,T} (u,q)$ if $p \xrightarrow{\overline{a}}_{A,T} q$ ($\overline{a} \in \overline{\mathsf{A}}, u \in \mathsf{A}^*$);
- $(u,p) \Rightarrow_{A,T} (u,q)$ if $p \xrightarrow{\mathcal{T},s}_A q$ and $u \in [\![\mathcal{T}, s]\!]_T$ ($u \in \mathsf{A}^*$).

For a pair $(p,q) \in Q \times Q$, define $[\![A,p,q]\!]_T = \{(u,v) \in \mathsf{A}^* \times \mathsf{A}^* \mid (u,p) \Rightarrow_{A,T}^* (v,q)\}$.

5.1 From ICPDL to Automata

For each ICPDL formula φ, we construct a TWAPTA $\mathcal{T}(\varphi)$ such that for all 2^{P}-labeled A-trees T, $[\![\mathcal{T}(\varphi), s]\!]_T = [\![\varphi]\!]_T$, where s is some selected state of $\mathcal{T}(\varphi)$. For each ICPDL program π, we construct a TWAPTA $T(\pi)$ and an NFA $A(\pi)$ over $\mathcal{T}(\pi)$ such that for all 2^{P}-labeled A-trees T, $[\![A(\pi), p, q]\!]_T = [\![\pi]\!]_T$, where p, q are two selected states of $A(\pi)$. In the following, the index T will be omitted for brevity. The construction is by induction on the structure of φ and π. We start with the construction of the TWAPTAs $T(\varphi)$ for ICPDL formulas φ.

If $\psi = p \in \mathsf{P}$, we put $\mathcal{T}(\psi) = (\{s\}, \delta, s \mapsto 1)$, where for all $Y \subseteq \mathsf{P}$ we have $\delta(s, Y) = \texttt{true}$ if $p \in Y$ and $\delta(s, Y) = \texttt{false}$ otherwise.

If $\psi = \neg\theta$, then $\mathcal{T}(\psi)$ is obtained from $\mathcal{T}(\theta)$ by applying the standard complementation procedure where all positive Boolean formulas on the right-hand side of the transition function are dualized and the acceptance condition is complemented by increasing the priority of every state by one, see e.g. [15].

If $\psi = \langle\pi\rangle\theta$, then we have inductively constructed $A = A(\pi)$ with state set Q over a TWAPTA $\mathcal{T}(\pi) = (S_1, \delta_1, \mathrm{Acc}_1)$ such that $[\![\pi]\!] = [\![A, p_0, q_0]\!]$ for some $p_0, q_0 \in Q$. We have also constructed a TWAPTA $\mathcal{T}(\theta) = (S_2, \delta_2, \mathrm{Acc}_2)$ such that $[\![\theta]\!] = [\![\mathcal{T}(\theta), s_0]\!]$ for some $s_0 \in S_2$. We construct the TWAPTA $\mathcal{T}(\psi) = (S, \delta, \mathrm{Acc})$ with $S = Q \uplus S_1 \uplus S_2$. For states in S_1 or in S_2, the transitions of $\mathcal{T}(\psi)$ are as in $\mathcal{T}(\pi)$ and $\mathcal{T}(\theta)$, respectively. It remains to simulate A. Handling transitions of A of the form $q \xrightarrow{a}_A q'$ is easy: $\mathcal{T}(\psi)$ simply navigates up or down in the tree as required. When $\mathcal{T}(\psi)$ is in state $q \in Q$ and there is a transition $q \xrightarrow{\mathcal{T}(\pi), s}_A r$, we branch universally to simulate both $\mathcal{T}(\pi)$ in state s *and* the state change of A to state r. Finally, we admit an ε-transition from state q_0 to s_0, thus simulating $\mathcal{T}(\theta)$ after finishing the simulation of A. Formally, for $q \in Q$ and $Y \subseteq \mathsf{P}$, we define

$$\begin{aligned}\delta(q, Y) = {} & \bigvee\{\langle r, a\rangle \mid r \in Q, a \in \mathsf{A} \cup \overline{\mathsf{A}}, q \xrightarrow{a}_A r\} \ \vee \\ & \bigvee\{\langle s, \varepsilon\rangle \wedge \langle r, \varepsilon\rangle \mid r \in Q, s \in S_1, q \xrightarrow{\mathcal{T}(\pi), s}_A r\} \ \vee \\ & ((q = q_0) \wedge \langle s_0, \varepsilon\rangle)\end{aligned}$$

The priority function Acc is defined by setting $\mathrm{Acc}(s) = 1$ if $s \in Q$ and $\mathrm{Acc}(s) = (\mathrm{Acc}_1 \uplus \mathrm{Acc}_2)(s)$ for $s \in S_1 \uplus S_2$. We set $\mathrm{Acc}(s) = 1$ for all $s \in Q$ since we want to assure that the NFA A is simulated for finitely many steps only, as $\psi = \langle\pi\rangle\theta$ is a diamond formula. We obtain $[\![\psi]\!] = [\![\mathcal{T}(\psi), p_0]\!]$.

We now describe the inductive construction of $A(\pi)$ and $\mathcal{T}(\pi)$ for an ICPDL program π. If $\mathcal{T}_i = (S_i, \delta_i, \mathrm{Acc}_i)$, $i \in \{1, 2\}$, are two TWAPTAs with disjoint sets of states, in what follows we use $\mathcal{T}_1 \uplus \mathcal{T}_2 = (S_1 \uplus S_2, \delta_1 \uplus \delta_2, \mathrm{Acc}_1 \uplus \mathrm{Acc}_2)$ denote their disjoint union; it is defined in the obvious way.

If $\pi = a \in \mathsf{A} \cup \overline{\mathsf{A}}$, the NFA $A(\pi)$ has two states p and q with the only transition $p \xrightarrow{a} q$. Hence, $[\![\pi]\!] = [\![A(\pi), p, q]\!]$.

If $\pi = \psi?$, we can assume that there exists a TWAPTA $\mathcal{T}(\psi)$ with a state s such that $[\![\psi]\!] = [\![\mathcal{T}(\psi), s]\!]$. The TWAPTA $\mathcal{T}(\pi)$ is $\mathcal{T}(\psi)$. The NFA $A(\pi)$ has two states p and q with the only transition $p \xrightarrow{\mathcal{T}(\pi), s} q$. Hence, we have $[\![\pi]\!] = [\![A(\pi), p, q]\!] = \{(u, u) \mid u \in [\![\mathcal{T}(\psi), s]\!]\}$.

If $\pi = \pi_1 \cup \pi_2, \pi = \pi_1 \circ \pi_2$, or $\pi = \chi^*$, we construct $A(\pi)$ by using the standard automata constructions for union, concatenation, and Kleene-star. In case $\pi = \pi_1 \cup \pi_2$ or $\pi = \pi_1 \circ \pi_2$, we set $\mathcal{T}(\pi) = \mathcal{T}(\pi_1) \uplus \mathcal{T}(\pi_2)$, whereas for $\pi = \chi^*$, we set $\mathcal{T}(\pi) = \mathcal{T}(\chi)$.

It remains to construct $A(\pi_1 \cap \pi_2)$ and $\mathcal{T}(\pi_1 \cap \pi_2)$, which is the most difficult step of the construction. Assume that the NFAs $A(\pi_i) = (Q_i, \rightarrow_{A(\pi_i)})$ over the TWAPTAs $\mathcal{T}(\pi_i)$ have already been constructed, for $i \in \{1, 2\}$. Thus, $[\![A(\pi_i), p_i, q_i]\!] = [\![\pi_i]\!]$ for some states $p_i, q_i \in Q_i$. A natural idea for defining an NFA for $\pi_1 \cap \pi_2$ is to apply a product construction to A_1 and A_2. A naive attempt to do this is bound to fail because a run of A_1 in T and a run of A_2 in T, both starting in a tree node u and ending in a tree node v, may proceed along different paths. More precisely, the two runs both travel along the unique shortest from u to v, but they may make different "detours" from this shortest path. In order to eliminate this problem and make the product construction available, we modify $A(\pi_1)$ and $A(\pi_2)$ by admitting additional test transitions that allow to short-cut the mentioned detours. These modified NFAs can always travel along the shortest path without any detours, and thus the product construction can be used.

Before we can construct $A(\pi_1 \cap \pi_2)$, we make a digression to introduce the mentioned modification of NFAs. Let $\mathcal{T} = (S, \delta, \mathrm{Acc})$ be a TWAPTA and $A = (Q, \rightarrow_A)$ an NFA over $\mathcal{T}$. Define the relation $\mathrm{loop}_A \subseteq \mathsf{A}^* \times Q \times Q$ as the smallest set such that:

(i) for all $u \in \mathsf{A}^*$ and $q \in Q$ we have $(u, q, q) \in \mathrm{loop}_A$,
(ii) if $(ua, p', q') \in \mathrm{loop}_A$, $p \xrightarrow{a}_A p'$ and $q' \xrightarrow{\overline{a}}_A q$, then $(u, p, q) \in \mathrm{loop}_A$,
(iii) if $(u, p', q') \in \mathrm{loop}_A$, $p \xrightarrow{\overline{a}}_A p'$, and $q' \xrightarrow{a}_A q$, then $(ua, p, q) \in \mathrm{loop}_A$,
(iv) if $(u, p, r) \in \mathrm{loop}_A$ and $(u, r, q) \in \mathrm{loop}_A$, then $(u, p, q) \in \mathrm{loop}_A$, and
(v) if $u \in [\![\mathcal{T}, s]\!]$ and $p \xrightarrow{\mathcal{T},s}_A q$ for $s \in S$, then $(u, p, q) \in \mathrm{loop}_A$.

Intuitively, loop_A describes detours, i.e., (parts of) a run of A that start at some node in the tree and eventually return to the very same node. It is not too difficult to prove the following.

Lemma 5. *We have* $(u, p, q) \in \mathrm{loop}_A$ *if and only if* $(u, p) \Rightarrow_A^* (u, q)$.

Since Conditions (i)–(v) can be easily translated into a TWAPTA, we obtain:

Lemma 6. *There is a TWAPTA* $\mathcal{U} = (S', \delta', \mathrm{Acc}')$ *with* $S' = S \uplus (Q \times Q)$ *s.t.*

(i) $[\![\mathcal{U}, s]\!] = [\![\mathcal{T}, s]\!]$ *for all* $s \in S$,
(ii) $[\![\mathcal{U}, (p, q)]\!] = \{u \in \mathsf{A}^* \mid (u, p, q) \in \mathrm{loop}_A\}$ *for all* $(p, q) \in Q \times Q$, *and*
(iii) $|\mathrm{Acc}'| = |\mathrm{Acc}|$.

Now define a new NFA $B = (Q, \rightarrow_B)$ over the TWAPTA $\mathcal{U}$, that results from A by adding for every pair $(p, q) \in Q \times Q$, the test transition $p \xrightarrow{\mathcal{U},(p,q)}_B q$. The following lemma shows that our modification did not damage the NFA.

Lemma 7. *Let* $u, v \in \mathsf{A}^*$ *and let* $p, q \in Q$. *Then* $(u, v) \in [\![A, p, q]\!]$ *iff* $(u, v) \in [\![B, p, q]\!]$.

We now return to the construction of $A(\pi_1 \cap \pi_2)$ and $\mathcal{T}(\pi_1 \cap \pi_2)$ from $A(\pi_1)$, $A(\pi_2)$, $\mathcal{T}(\pi_1)$, and $\mathcal{T}(\pi_2)$. For $i \in \{1, 2\}$, we first construct the NFA $B(\pi_i)$ over the TWAPTA $\mathcal{U}(\pi_i) = (S_i', \delta_i', \mathrm{Acc}_i')$ as described above. Note that $|S_i'| = |S_i| + |Q_i|^2$. We take $\mathcal{T}(\pi_1 \cap \pi_2) = \mathcal{U}(\pi_1) \uplus \mathcal{U}(\pi_2)$. The NFA $A(\pi_1 \cap \pi_2)$ is the product automaton of $B(\pi_1) = (Q_1, \rightarrow_{B(\pi_1)})$ and $B(\pi_2) = (Q_2, \rightarrow_{B(\pi_2)})$, where test transitions can be carried out asynchronously:

- The state set of $A(\pi_1 \cap \pi_2)$ is $Q_1 \times Q_2$.
- For $a \in \mathsf{A} \cup \overline{\mathsf{A}}$ we have $(r_1, r_2) \xrightarrow{a}_{A(\pi_1 \cap \pi_2)} (r_1', r_2')$ if and only if $r_1 \xrightarrow{a}_{B(\pi_1)} r_1'$ and $r_2 \xrightarrow{a}_{B(\pi_2)} r_2'$.
- For $s \in S_1' \uplus S_2'$ we have the test transition $(r_1, r_2) \xrightarrow{\mathcal{T}(\pi_1 \cap \pi_2), s}_{A(\pi_1 \cap \pi_2)} (r_1', r_2')$ if and only if (i) $s \in S_1'$, $r_2 = r_2'$, and $r_1 \xrightarrow{\mathcal{U}(\pi_1), s}_{B(\pi_1)} r_1'$ or (ii) $s \in S_2'$, $r_1 = r_1'$, and $r_2 \xrightarrow{\mathcal{U}(\pi_2), s}_{B(\pi_1)} r_2'$.

It is possible to show that $[\![A(\pi_1 \cap \pi_2), (p_1, p_2), (q_1, q_2)]\!] = [\![\pi_1 \cap \pi_2]\!]$.This finishes the inductive translation of ICPDL formulas and programs into automata. A careful analysis of the constructions outlined above, allows us to inductively establish the following bounds.

Lemma 8. *For every ICPDL formula ψ and every ICPDL program π we have:*

1. *If $\mathcal{T}(\psi) = (S, \delta, \mathrm{Acc})$, then $|S| \leq 2^{|\psi|^2}$ and $|\mathrm{Acc}| \leq |\psi|$.*
2. *If $A(\pi) = (Q, \rightarrow_{A(\pi)})$ and $\mathcal{T}(\pi) = (S, \delta, \mathrm{Acc})$ then $|Q| \leq 2^{|\pi|}$, $|S| \leq 2^{|\pi|^2}$, and $|\mathrm{Acc}| \leq |\pi|$.*

The *double* exponential bound in Point 1 of Lemma 8 is due to the fact that the construction for dealing with program intersection blows up the size of NFAs quadratically. In contrast, all other constructions involve only a linear blowup.

5.2 Wrapping Up

It is now easy to decide ω-regular tree satisfiability in ICPDL. Let $\mathcal{T}_0$ be a TWAPTA over 2^{P}-labeled A-trees and let φ be an ICPDL formula with $\mathsf{prop}(\varphi) \subseteq \mathsf{P}$ and $\mathsf{prog}(\varphi) \subseteq \mathsf{A}$. There is a state s of $\mathcal{T}(\varphi)$ such that $[\![\mathcal{T}(\varphi), s]\!]_T = [\![\varphi]\!]_T$ for all 2^{P}-labeled A-trees T. Let the TWAPTA $\mathcal{T}$ be the intersection of $\mathcal{T}_0$ and $\mathcal{T}(\varphi)$ (taking the intersection of TWAPTAs is trivial an can be done in linear time), where s becomes the initial state of $\mathcal{T}(\varphi)$. Clearly, $L(\mathcal{T}) \neq \emptyset$ if and only if there exists some tree $T \in L(\mathcal{T}_0)$ with $(T, \varepsilon) \models \varphi$. By Lemma 8 and Theorem 3, we thus obtain a 2EXP upper bound for ω-regular tree satisfiability in ICPDL. A matching lower bound is obtained by a straightforward reduction of satisfiability in ICPDL in tree-shaped Kripke structures. It was shown in [11] that this problem is 2EXP-hard.

Theorem 5. *ω-regular tree satisfiability in ICPDL is* $\mathsf{2EXP}$*-complete.*

Together with Theorem 4, this finally proves our main result Theorem 1. It is interesting to note that the bound given in Point 1 of Lemma 8 improves to single exponential if the intersection height (which can be defined in the obvious way) of ICPDL programs is bounded by a constant. Thus, we actually obtain EXP-completeness for this case.

6 Negation of Atomic Programs

We consider extensions of IPDL and ICPDL with negation of programs. It is well known that adding full program negation renders PDL undecidable [9], whereas PDL with program negation restricted to atomic programs remains decidable and EXP-complete [13].

In this section, we show that IPDL and hence also ICPDL become undecidable already when extended with atomic program negation. Since intersection of programs can be defined in terms of program union and (full) program negation, this also yields an alternative proof of the undecidability of PDL with full program negation.

Our proof proceeds by reduction from the undecidable tiling problem of the first quadrant of the plane [3]. A *tiling system* $\mathcal{T} = (T, H, V)$ consists of a finite set of *tile types* T and horizontal and vertical matching relations $H, V \subseteq T \times T$. A *solution* to $\mathcal{T}$ is a mapping $\tau : \mathbb{N} \times \mathbb{N} \to T$ such that for all $(x, y) \in \mathbb{N} \times \mathbb{N}$, we have

– if $\tau(x, y) = t$ and $\tau(x + 1, y) = t'$, then $(t, t') \in H$, and
– if $\tau(x, y) = t$ and $\tau(x, y + 1) = t'$, then $(t, t') \in V$.

The *tiling problem* is to decide, given a tiling system $\mathcal{T}$, whether $\mathcal{T}$ has a solution.

We use IPDL$^{(\neg)}$ to denote the extension of IPDL with negation of atomic programs, which we write as $\neg a$ ($a \in \mathbb{A}$). The semantics of the new constructor is defined in the obvious way, i.e., $[\![\neg a]\!]_K = (X \times X) \setminus [\![a]\!]_K$. To reduce the tiling problem to satisfiability in IPDL$^{(\neg)}$, we give a translation of tiling systems $\mathcal{T} = (T, H, V)$ into formulas $\varphi_{\mathcal{T}}$ of IPDL$^{(\neg)}$ such that $\mathcal{T}$ has a solution if and only if $\varphi_{\mathcal{T}}$ is satisfiable. In the formula $\varphi_{\mathcal{T}}$, we use two atomic programs a_x and a_y for representing the grid $\mathbb{N} \times \mathbb{N}$ and we use the elements of T as atomic propositions for representing tile types. More precisely, $\varphi_{\mathcal{T}}$ is a conjunction consisting of the following conjuncts:

(a) every element of a (connected) model of $\varphi_{\mathcal{T}}$ represents an element of $\mathbb{N} \times \mathbb{N}$ and is labelled with a unique tile type:

$$[(a_x \cup a_y)^*]\big(\bigvee_{t \in T} t \;\wedge \bigwedge_{t,t' \in T, t \neq t'} \neg(t \wedge t')\big)$$

(b) every element has an a_x-successor and an a_y-successor:

$$[(a_x \cup a_y)^*]\big(\langle a_x \rangle \texttt{true} \wedge \langle a_y \rangle \texttt{true}\big)$$

(c) the programs a_x and a_y are confluent:

$$[(a_x \cup a_y)^*]\; [(a_x; a_y) \cap (a_y; \neg a_x)]\texttt{false}$$

(d) the horizontal and vertical matching conditions are respected:

$$[(a_x \cup a_y)^*]\big(\bigwedge_{t \in T} t \;\Rightarrow\; \big([a_x] \bigvee_{(t,t') \in H} t' \;\wedge\; [a_y] \bigvee_{(t,t') \in V} t'\big)\big).$$

Lemma 9. *$\mathcal{T}$ has a solution if and only if $\varphi_{\mathcal{T}}$ is satisfiable.*

We have thus established the following result.

Theorem 6. *Satisfiability in IPDL$^{(\neg)}$ is undecidable.*

References

1. L. Afanasiev, P. Blackburn, I. Dimitriou, B. Gaiffe, E. Goris, M. J. Marx, and M. de Rijke. PDL for ordered trees. *Journal of Applied Non-Classical Logics*, 15(2):115-135, 2005.
2. N. Alechina, S. Demri, and M. de Rijke. A modal perspective on path constraints. *Journal of Logic and Computation*, 13(6):939–956, 2003.
3. R. Berger. The undecidability of the dominoe problem. *Memoirs of the American Mathematical Society*, 66, 1966.
4. R. Danecki. Nondeterministic Propositional Dynamic Logic with intersection is decidable. In *Proc. 5th Symp. Computation Theory*, LNCS 208, pages 34–53, 1984.
5. L. Farinas Del Cerro and E. Orlowska. DAL-a logic for data analysis. *Theoretical Computer Science*, 36(2-3):251–264, 1985.
6. M. J. Fischer and R. E. Ladner. Propositional Dynamic Logic of Regular Programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.
7. G. D. Giacomo and M. Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In *Proc. AAAI94*, pages 205–212, 1994.
8. S. Göller and M. Lohrey. Infinite state model-checking of propositional dynamic logics. In *Proc. CSL 2006*, LNCS 4207, pages 349–364. Springer, 2006.
9. D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. Foundations of computing. The MIT Press, 2000.
10. W. van der Hoek and J.J. Meyer. A complete epistemic logic for multiple agents – Combining distributed and common knowledge. *Epistemic Logic and the Theory of Games and Decisions*, pages 35–68. Kluwer, 1997.
11. M. Lange and C. Lutz. 2-ExpTime Lower Bounds for Propositional Dynamic Logics with Intersection. *Journal of Symbolic Logic*, 70(4):1072–1086, 2005.
12. C. Lutz. PDL with intersection and converse is decidable. In *Proc. CSL 2005*, LNCS 3634, pages 413–427. Springer, 2005.
13. C. Lutz and D. Walther. PDL with negation of atomic programs. *Journal of Applied Non-Classical Logics*, 15(2):189–213, 2005.
14. J. Meyer. Dynamic logic for reasoning about actions and agents. In J. Minker, editor, *Logic-Based Artificial Intelligence*, pages 281–311. Kluwer Academic Publishers, 2000.
15. D. Muller and P. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54(2-3):267–276, 1987.
16. V. Pratt. A near-optimal method for reasoning about action. *Journal of Computer and System Sciences*, 20:231–254, 1980.
17. B. ten Cate. The expressivity of XPath with transitive closure. In *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2006)*, pages 328–337. ACM Press, 2006.
18. H. P. van Ditmarsch, W. van der Hoek, and B. P. Kooi. Concurrent dynamic epistemic logic for MAS. In *Proc. AAMAS 2003*, pages 201–208. ACM Press, 2003.
19. M. Y. Vardi. The taming of converse: Reasoning about two-way computations. In *Proc. Logics of Programs*, LNCS 193, pages 413–423. Springer, 1985.
20. M. Y. Vardi. Reasoning about the past with two-way automata. In *Proc. ICALP '98*, LNCS 1443, pages 628–641. Springer, 1998.

Symbolic Backwards-Reachability Analysis for Higher-Order Pushdown Systems*

Matthew Hague and C.-H. Luke Ong

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, UK, OX1 3QD

Abstract. Higher-order pushdown systems (PDSs) generalise pushdown systems through the use of higher-order stacks, that is, a nested "stack of stacks" structure. We further generalise higher-order PDSs to higher-order Alternating PDSs (APDSs) and consider the backwards reachability problem over these systems. We prove that given an order-n APDS, the set of configurations from which a given regular set of configurations is reachable is itself regular and computable in n-EXPTIME. We show that the result has several useful applications in the verification of higher-order PDSs such as LTL model checking, alternation-free μ-calculus model checking, and the computation of winning regions of reachability games.

1 Introduction

Pushdown automata are an extension of finite state automata. In addition to a finite set of control states, a pushdown automaton has a stack that can be manipulated with the usual push and pop operations. *Higher-order pushdown automata* (PDA) generalise pushdown automata through the use of higher-order stacks. Whereas a stack in the sense of a pushdown automaton is an order-one stack — that is, a stack of characters — an order-two stack is a stack of order-one stacks. Similarly, an order-three stack is a stack of order-two stacks, and so on.

Higher-order PDA were originally introduced by Maslov [18] in the 1970s as generators of (a hierarchy of) finite word languages. *Higher-order pushdown systems* (PDSs) are higher-order PDA viewed as generators of infinite trees or graphs. These systems provide a natural infinite-state model for higher-order programs with recursive function calls and are therefore useful in software verification. Several notable advances in recent years have sparked off a resurgence of interest in higher-order PDA/PDSs in the Verification community. E.g. Knapik *et al.* [24] have shown that the ranked trees generated by deterministic order-n PDSs are exactly those that are generated by order-n recursion schemes satisfying the *safety* constraint; Carayol and Wöhrle [5] have shown that the ϵ-closure of the configuration graphs of higher-order PDSs exactly constitute Caucal's graph hierarchy [10]. Remarkably these infinite trees and graphs have decidable monadic second-order (MSO) theories [11,5,24].

* The full version [13] of this work is downloadable from the first author's web page.

These MSO decidability results, though powerful, only allow us to check that a property holds from a given configuration. We may wish to compute the set of configurations that satisfy a given property, especially since there may be an infinite number of such configurations. In this paper, we consider a closely-related problem:

> *Backwards Reachability*: Given a set of configurations C_{Init}, compute the set $Pre^*(C_{Init})$ of configurations that can, via any number of transitions, reach a configuration in C_{Init}.

This is an important verification problem in its own right, since safety properties (i.e. undesirable program states – such as deadlock – are never reached) feature largely in practice.

The backwards reachability problem was solved for order-one PDSs by Bouajjani *et al.* [2]. In particular, they gave a method for computing the (regular) set of configurations $Pre^*(C_{Init})$ that could reach a given regular set of configurations C_{Init}. Regular sets of configurations are represented symbolically in the form of a finite multi-automaton. That is, a finite automaton that accepts finite words (representing stacks) with an initial state for each control state of the PDS. A configuration is accepted if the stack (viewed as a word) is accepted from the appropriate initial state. The set $Pre^*(C_{Init})$ is computed by the repeated addition of a number of transitions – determined by the transition relation of the PDS – to the automaton accepting C_{Init}, until a fixed point is reached. A fixed point is guaranteed since no states are added and the alphabet is finite: eventually the automaton will become *saturated*.

The approach was extended by Bouajjani and Meyer [1] to the case of *higher-order context-free processes*, which are higher-order PDSs with a single control state. A key innovation in their work was the introduction of a new class of (finite-state) automata called *nested store automata*, which captures an intuitive notion of regular sets of n-stores. An order-one nested store automaton is simply a finite automaton over words. An order-n nested store automaton is a finite automaton whose transitions are labelled by order-$(n-1)$ nested store automata.

Our paper is concerned with the non-trivial problem[1] of extending the backwards reachability result of Bouajjani and Meyer to the general case of higher-order PDSs (by taking into account a set of control states). In fact, we consider (and solve) the backwards reachability problem for the more general case of higher-order *alternating* pushdown systems (APDSs). Though slightly unwieldy, an advantage of the alternating framework is that it conveniently lends itself to a number of higher-order PDS verification problems. Following the work of Cachat [22], we show that the winning region of a reachability game played over a higher-order PDS can be computed by a reduction to the backwards reachability problem of an appropriate APDS. We also generalise results due to Bouajjani *et al.* [2] to give a method for computing the set of configurations of a higher-order PDS that satisfy a given formula of the alternation-free μ-calculus or a linear-time temporal logic.

[1] “This does not seem to be technically trivial, and naïve extensions of our construction lead to procedures which are not guaranteed to terminate.” [1, p. 145]

Related Work. Prompted by the fact that the set of configurations reachable from a given configuration of a higher-order PDS is not regular in the sense of Bouajjani and Meyer (the stack contents cannot be represented by a finite automaton over words), Carayol [4] has proposed an alternative definition of regularity for higher-order stacks, which we shall call *C-regularity.* Our notion of regularity coincides with that of Bouajjani and Meyer, which we shall call *BM-regularity.*

A set of order-n stacks is said to be *C-regular* if it is constructible from the empty n-stack by a regular sequence of order-n stack operations. Carayol shows that C-regularity coincides with MSO definability over the canonical structure Δ_2^n associated with order-n stacks. This implies, for instance, that the winning region of a parity game over an order-n pushdown graph is also C-regular, as it can be defined as an MSO formula [22].

In this paper we solve the backwards reachability problem for higher-order PDSs and apply the solution to reachability games and model-checking. In this sense we give a weaker kind of result that uses a different notion of regularity. Because C-regularity does not imply BM-regularity[2], our result is not subsumed by the work of Carayol. However, a detailed comparison of the two approaches may provide a fruitful direction for further research.

The definition of higher-order PDSs may be extended to higher-order pushdown games. In the order-one case, the problem of determining whether a configuration is winning for Eloise with a parity winning condition was solved by Walukiewicz in 1996 [14]. The order-one backwards reachability algorithm of Bouajjani *et al.* was adapted by Cachat to compute the winning regions of order-one reachability and Büchi games [22]. Results for pushdown games have been extended to a number of winning conditions [23,3,12,20,9] including parity conditions [22,19]. In the higher-order case with a parity winning condition, a method for deciding whether a configuration is winning has been provided by Cachat [22].

Higher-order recursion schemes (HORSs) represent a further area of related work. MSO decidability for trees generated by arbitrary (i.e. not necessarily safe) HORSs has been shown by one of us [21]. A variant kind of higher-order PDSs called *collapsible pushdown automata* (extending *panic automata* [25] or *pushdown automata with links* [16] to all finite orders) has recently been shown to be equi-expressive with HORSs for generating ranked trees [8]. These new automata are conjectured to enrich the class of higher-order systems and provide many new avenues of research.

2 Preliminaries

In the sequel we will introduce several kinds of alternating automata. For convenience, we will use a non-standard definition of alternating automata that is equivalent to the standard definitions of Brzozowski and Leiss [15] and Chandra, Kozen and Stockmeyer [6]. Similar definitions have been used for the analysis

[2] For example $(push_a)^*; push_2$ defines all stacks of the form $[[a^n][a^n]]$.

of pushdown systems by Bouajjani *et al.* [2] and Cachat [22]. The alternating transition relation $\Delta \subseteq \mathcal{Q} \times \Gamma \times 2^{\mathcal{Q}}$ — where Γ is a kind of alphabet and $\mathcal{Q}$ is a state-set — is given in disjunctive normal form. That is, the image $\Delta(q, \gamma)$ of $q \in \mathcal{Q}$ and $\gamma \in \Gamma$ is a set $\{Q_1, \ldots, Q_m\}$ with $Q_i \in 2^{\mathcal{Q}}$ for $i \in \{1, \ldots, m\}$. When the automaton is viewed as a game, Eloise — the existential player — chooses a set $Q \in \Delta(q, \gamma)$; Abelard — the universal player — then chooses a state $q \in Q$.

2.1 (Alternating) Higher-Order Pushdown Systems

We begin by defining higher-order stores and their operations. We will then define higher-order PDSs and APDSs in full.

The set C_1^{Σ} of 1-stores over an alphabet Σ is the set of words of the form $[a_1, \ldots, a_m]$ with $m \geq 0$ and $a_i \in \Sigma$ for all $i \in \{1, \ldots, m\}$, $[\notin \Sigma$ and $] \notin \Sigma$. For $n > 1$, $C_n^{\Sigma} = [w_1, \ldots, w_m]$ with $m \geq 1$ and $w_i \in C_{n-1}^{\Sigma}$ for all $i \in \{1, \ldots, m\}$. There are three types of operations applicable to n-stores: *push*, *pop* and *top*. These are defined inductively. Over a 1-store, we have (for all $w \in \Sigma^*$),

$$\begin{aligned} push_w[a_1 \ldots a_m] &= [w a_2 \ldots a_m] \\ top_1[a_1 \ldots a_m] &= a_1 \end{aligned}$$

We may define the abbreviation $pop_1 = push_\varepsilon$. When $n > 1$, we have

$$\begin{aligned} push_w[\gamma_1 \ldots \gamma_m] &= [push_w(\gamma_1)\gamma_2 \ldots \gamma_m] && \\ push_l[\gamma_1 \ldots \gamma_m] &= [push_l(\gamma_1)\gamma_2 \ldots \gamma_m] && \text{if } 2 \leq l < n \\ push_n[\gamma_1 \ldots \gamma_m] &= [\gamma_1 \gamma_1 \gamma_2 \ldots \gamma_m] && \\ pop_l[\gamma_1 \ldots \gamma_m] &= [pop_l(\gamma_1)\gamma_2 \ldots \gamma_m] && \text{if } 1 \leq l < n \\ pop_n[\gamma_1 \ldots \gamma_m] &= [\gamma_2 \ldots \gamma_m] && \text{if } m > 1 \\ top_l[\gamma_1 \ldots \gamma_m] &= top_l(\gamma_1) && \text{if } 1 \leq l < n \\ top_n[\gamma_1 \ldots \gamma_m] &= \gamma_1 && \end{aligned}$$

Note that we assume wlog $\Sigma \cap \mathcal{N} = \emptyset$, where $\mathcal{N}$ is the set of natural numbers. Further, observe that when $m = 1$, pop_n is undefined. We define $\mathcal{O}_n = \{ push_w \mid w \in \Sigma^* \} \cup \{ push_l, pop_l \mid 1 < l \leq n \}$.

Definition 1. An *order-n pushdown system* (PDS) is a tuple $(\mathcal{P}, \mathcal{D}, \Sigma)$ where $\mathcal{P}$ is a finite set of control states p^j, $\mathcal{D} \subseteq \mathcal{P} \times \Sigma \times \mathcal{O}_n \times \mathcal{P}$ is a finite set of commands d, and Σ is a finite alphabet.

A configuration of an order-n PDS is a pair $\langle p, \gamma \rangle$ where $p \in \mathcal{P}$ and γ is an n-store. We have a transition $\langle p, \gamma \rangle \hookrightarrow \langle p', \gamma' \rangle$ iff we have $(p, a, o, p') \in \mathcal{D}$, $top_1(\gamma) = a$ and $\gamma' = o(\gamma)$.

Definition 2. An *order-n alternating pushdown system* (APDS) is a tuple $(\mathcal{P}, \mathcal{D}, \Sigma)$ where $\mathcal{P}$ is a finite set of control states p^j, $\mathcal{D} \subseteq \mathcal{P} \times \Sigma \times 2^{\mathcal{O}_n \times \mathcal{P}}$ is a finite set of commands d, and Σ is a finite alphabet.

A configuration of an order-n APDS is a pair $\langle p, \gamma \rangle$ where $p \in \mathcal{P}$ and γ is an n-store. We have a transition $\langle p, \gamma \rangle \hookrightarrow C$ iff we have $(p, a, OP) \in \mathcal{D}$, $top_1(\gamma) = a$, and

$$\begin{aligned} C = \{ \langle p', \gamma' \rangle \mid (o, p') \in OP \ \wedge \ \gamma' = o(\gamma) \} \\ \cup \{ \langle p, \triangledown \rangle \mid \text{if } (o, p') \in OP \text{ and } o(\gamma) \text{ is not defined} \} \end{aligned}$$

The transition relation generalises to sets of configurations via the following rule:

$$\frac{\langle p, \gamma \rangle \ \hookrightarrow \ C}{C' \cup \langle p, \gamma \rangle \ \hookrightarrow \ C' \cup C} \quad \langle p, \gamma \rangle \notin C'$$

In both the alternating and the non-alternating cases, we define $\overset{*}{\hookrightarrow}$ to be the transitive closure of $\hookrightarrow$. For a set of configurations C_{Init} we define $Pre^*(C_{Init})$ as the set of configurations $\langle p, \gamma \rangle$ such that $\langle p, \gamma \rangle \overset{*}{\hookrightarrow} c$ and $c \in C_{Init}$ or $\langle p, \gamma \rangle \overset{*}{\hookrightarrow} C$ and $C \subseteq C_{Init}$ respectively.

Observe that since no transitions are possible from an "undefined" configuration $\langle p, \triangledown \rangle$ we can reduce the reachability problem for higher-order PDSs to the reachability problem over higher-order APDSs in a straightforward manner.

In the sequel, to ease the presentation, we assume $n > 1$. The case $n = 1$ was investigated by Bouajjani *et al.* [2].

2.2 *n*-Store Multi-automata

To represent sets of configurations we will use *n-store multi-automata.* These are alternating automata whose transitions are labelled by $(n-1)$-*store automata*, which are also alternating. A set of configurations is said to be *regular* if it is accepted by an n-store multi-automaton.

Definition 3

1. A *1-store automaton* is a tuple $(\mathcal{Q}, \Sigma, \Delta, q_0, \mathcal{Q}_f)$ where $\mathcal{Q}$ is a finite set of states, Σ is a finite alphabet, q_0 is the initial state and $\mathcal{Q}_f \subseteq \mathcal{Q}$ is a set of final states. $\Delta \subseteq \mathcal{Q} \times \Sigma \times 2^{\mathcal{Q}}$ is a finite transition relation.
2. Let $\mathfrak{B}^{\Sigma}_{n-1}$ be the (infinite) set of all $(n-1)$-store automata over the alphabet Σ. An *n-store automaton* over the alphabet Σ is a tuple $(\mathcal{Q}, \Sigma, \Delta, q_0, \mathcal{Q}_f)$ where $\mathcal{Q}$ is a finite set of states, $q_0 \notin \mathcal{Q}_f$ is the initial state, $\mathcal{Q}_f \subseteq \mathcal{Q}$ is a set of final states, and $\Delta \subseteq \mathcal{Q} \times \mathfrak{B}^{\Sigma}_{n-1} \times 2^{\mathcal{Q}}$ is a *finite* transition relation.
3. An *n-store multi-automaton* over the alphabet Σ is a tuple

$$(\mathcal{Q}, \Sigma, \Delta, \{q^1, \ldots, q^z\}, \mathcal{Q}_f)$$

where $\mathcal{Q}$ is a finite set of states, Σ is a finite alphabet, $q^i \notin \mathcal{Q}_f$ for $i \in \{1, \ldots, z\}$ are separate initial states and $\mathcal{Q}_f \subseteq \mathcal{Q}$ is a set of final states, and

$$\Delta \subseteq (\mathcal{Q} \times \mathfrak{B}^{\Sigma}_{n-1} \times 2^{\mathcal{Q}}) \cup (\{q^1, \ldots, q^z\} \times \{\triangledown\} \times \{q^{\varepsilon}_f\})$$

is a *finite* transition relation where $q^{\varepsilon}_f \in \mathcal{Q}_f$ has no outgoing transitions.

To indicate a transition $(q, B, \{q_1, \ldots, q_m\}) \in \Delta$, we write,

$$q \xrightarrow{B} \{q_1, \ldots, q_m\}$$

Paths of the automata from a state q take the form,

$$q \xrightarrow{\widetilde{B}_0} \{q^1_1, \ldots, q^1_{m_1}\} \xrightarrow{\widetilde{B}_1} \ldots \xrightarrow{\widetilde{B}_m} \{q^m_1, \ldots, q^m_{m_l}\}$$

where transitions between configurations $\{q_1^x, \ldots, q_{m_x}^x\} \xrightarrow{\widetilde{B}_x} \{q_1^{x+1}, \ldots, q_{m_{x+1}}^{x+1}\}$ are such that we have $q_y^x \xrightarrow{B_y} Q_y$ for all $y \in \{1, \ldots, m_x\}$ and $\bigcup_{y\in\{1,\ldots,m_x\}} Q_y = \{q_1^{x+1}, \ldots, q_{m_{x+1}}^{x+1}\}$ and $\bigcup_{y\in\{1,\ldots,m_x\}}\{B_y\} = \widetilde{B}_x$. Observe that $\widetilde{B}_0$ is necessarily a singleton set.

We will, by abuse of notation, abbreviate a run over the word w to

$$q \xrightarrow{w} \{q_1, \ldots, q_m\}$$

Further, when a run occurs in an automaton forming part of a sequence indexed by i (for example, $A_0, A_1, \ldots$), we may write $\longrightarrow_i$ to indicate which automaton the run belongs to.

A 1-store $[a_1 \ldots a_m]$ is accepted by a 1-store automaton A (that is $[a_1 \ldots a_m] \in \mathcal{L}(A)$) iff we have a run $q_0 \xrightarrow{a_1\ldots a_m} Q$ in A with $Q \subseteq \mathcal{Q}_f$. For a given n-store automaton $A = (\mathcal{Q}, \Sigma, \Delta, q_0, \mathcal{Q}_f)$ we define

$$\mathcal{L}(A) = \{\ [\gamma_1 \ldots \gamma_m] \mid q_0 \xrightarrow{\widetilde{B}_0} \ldots \xrightarrow{\widetilde{B}_m} Q\ \wedge\ Q \subseteq \mathcal{Q}_f\ \wedge\ \forall 0 \leq i \leq m.\gamma_i \in \mathcal{L}(\widetilde{B}_i)\ \}$$

where $\gamma \in \mathcal{L}(\widetilde{B})$ iff $\gamma \in \mathcal{L}(B)$ for all $B \in \widetilde{B}$.

For an n-store multi-automaton $A = (Q, \Sigma, \Delta, \{q^1, \ldots, q^z\}, \mathcal{Q}_f)$ we define

$$\begin{aligned}\mathcal{L}(A^{q^j}) = \{\ [\gamma_1 \ldots \gamma_m] \mid q^j \xrightarrow{\widetilde{B}_0} \ldots \xrightarrow{\widetilde{B}_m} Q \\ \wedge\ Q \subseteq \mathcal{Q}_f\ \wedge\ \forall 0 \leq i \leq m.\gamma_i \in \mathcal{L}(\widetilde{B}_i)\ \} \\ \cup\ \{\ \nabla \mid q^j \xrightarrow{\nabla} q_f^{\varepsilon}\ \} \\ \mathcal{L}(A) = \{\ \langle p^j, \gamma\rangle \mid j \in \{1, \ldots, z\} \wedge \gamma \in \mathcal{L}(A^{q^j})\ \}\end{aligned}$$

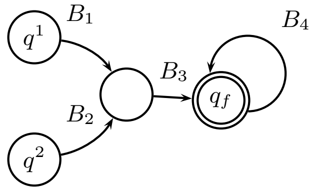
Finally, we define the automata B_l^a and X_l^a for all $1 \leq l \leq n$ and $a \in \Sigma$ and the notation q^θ. B_l^a is the l-store automaton that accepts any l-store γ such that $top_1(\gamma) = a$. X_l^a is the $(n-1)$-store automaton accepting all $(n-1)$-stores such that $top_1(\gamma) = a$ and $top_{l+1}(\gamma) = [[w']]$ for some w'. That is, $pop_l(\gamma)$ is undefined. If θ represents a store automaton, the state q^θ refers to the initial state of the automaton represented by θ.

3 Backwards Reachability

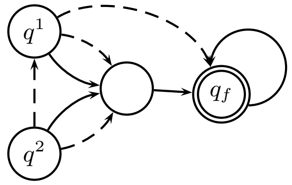
Theorem 1. *Given an n-store multi-automaton A_0 accepting the set of configurations C_{Init} of an order-n APDS, we can construct in n-EXPTIME (in the size of A_0) an n-store multi-automaton A_* accepting the set $Pre^*(C_{Init})$. Thus, $Pre^*(C_{Init})$ is regular.*

Due to space constraints, we restrict our attention in the sequel to the order-2 case. We give a brief description of the order-n construction in Section 3.5. For a formal treatment of the general case, we refer the reader to the full version of this paper [13].

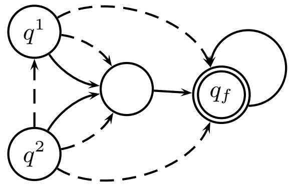
Fix an order-2 APDS. We begin by showing how to generate an infinite sequence of automata $A_0, A_1, \ldots$, where A_0 is such that $\mathcal{L}(A_0) = C_{Init}$. This



$$\begin{aligned} d_1 &= (p^1, a, push_2, p^1) \\ d_2 &= (p^1, a, push_\varepsilon, p^1), \\ d_3 &= (p^2, a, push_w, p^1) \\ d_4 &= (p^2, a, pop_2, p^1) \end{aligned}$$



$$\begin{aligned} \widetilde{G}^1_{(q^1,\circ)} &= \{\{(a, push_\varepsilon, B_1)\}\} \\ \widetilde{G}^1_{(q^1,q_f)} &= \{\{B_1^a, B_1, B_3\}\} \\ \widetilde{G}^1_{(q^2,\circ)} &= \{\{(a, push_w, B_1)\}\} \\ \widetilde{G}^1_{(q^2,q^1)} &= \{\{B_1^a\}\} \end{aligned}$$



$$\begin{aligned} \widetilde{G}^2_{(q^1,\circ)} &= \left\{\{(a, push_\varepsilon, B_1)\}, \{(a, push_\varepsilon, \widetilde{G}^1_{(q^1,\circ)})\}\right\} \\ \widetilde{G}^2_{(q^1,q^f)} &= \left\{\begin{array}{l} \{B_1^a, B_1, B_3\}, \{(a, push_\varepsilon, \widetilde{G}^1_{(q^1,q_f)})\}, \\ \{B_1^a, \widetilde{G}^1_{(q^1,q_f)}, B_4\}, \{B_1^a, \widetilde{G}^1_{(q^1,\circ)}, B_3\} \end{array}\right\} \\ \widetilde{G}^2_{(q^2,\circ)} &= \left\{\{(a, push_w, B_1)\}, \{(a, push_w, \widetilde{G}^1_{(q^1,\circ)})\}\right\} \\ \widetilde{G}^2_{(q^2,q^1)} &= \{\{B_1^a\}\} \\ \widetilde{G}^2_{(q^2,q_f)} &= \left\{\{(a, push_w, \widetilde{G}^1_{(q^1,q_f)})\}\right\} \end{aligned}$$

Fig. 1. The automata A_0, A_1 and A_2

sequence is increasing in the sense that $\mathcal{L}(A_i) \subseteq \mathcal{L}(A_{i+1})$ for all i, and sound and complete with respect to $Pre^*(C_{Init})$; that is $\bigcup_{i\geq 0} \mathcal{L}(A_i) = Pre^*(C_{Init})$. To conclude the algorithm, we construct a single automaton A_* such that $\mathcal{L}(A_*) = \bigcup_{i\geq 0} \mathcal{L}(A_i)$.

We assume wlog that all initial states in A_0 have no incoming transitions and there exist in A_0 a state q_f^* from which all valid 2-stores are accepted and a state $q_f^\varepsilon \in \mathcal{Q}_f$ that has no outgoing transitions.

3.1 Example

We give an intuitive explanation of the algorithm by means of an example. Fix the 2-state order-2 PDS and 2-store multi-automaton A_0 shown in Figure 1 with some B_1, B_2, B_3 and B_4.

We proceed via a number of iterations, generating the automata $A_0, A_1, \ldots$. We construct A_{i+1} from A_i to reflect an additional inverse application of the commands $d_1, \ldots, d_4$. Rather than manipulating order-1 store automata labelling the edges of A_0 directly, we introduce new transitions (at most one between each pair of states q_1 and q_2) and label these edges with the set $\widetilde{G}^1_{(q_1,q_2)}$. This set is a recipe for the construction of an order-1 store automaton that will ultimately label the edge. The resulting A_1 is given in Figure 1 along with the contents of the sets.

To process the command d_1 we need to add all configurations of the form $\langle p_1, [\gamma_1 \ldots \gamma_m]\rangle$ with $top_1(\gamma_1) = a$ to the set of configurations accepted by A_1 for each configuration $\langle p_1, [\gamma_1\gamma_1 \dots \gamma_m]\rangle$ accepted by A_0. This results in the transition from q^1 to q_f. The contents of $\widetilde{G}^1_{(q^1,q_f)}$ indicate that this edge must accept the product of B^a_1, B_1 and B_3.

The commands d_2 and d_3 update the *top*$_2$ stack of any configuration accepted from q^1 or q^2 respectively. In both cases this updated stack must be accepted from q^1 in A_0. Hence, the contents of $\widetilde{G}^1_{(q^1,\circ)}$ and $\widetilde{G}^1_{(q^2,\circ)}$ specify that the automaton B_1 must be manipulated to produce the automaton that will label these new transitions. Finally, d_4 requires an additional *top*$_2$ stack with a as its *top*$_1$ element to be added to any stack accepted from q^1. Thus, we introduce the transition from q^2 to q^1.

To construct A_2 from A_1 we repeat the above procedure, taking into account the additional transitions in A_1. Observe that we do not add additional transitions between pairs of states that already have a transition labelled by a set. Instead, each labelling set may contain several element sets. The resulting automaton is given in Figure 1.

If we were to repeat this procedure to construct A_3 we would notice that a kind of fixed point has been reached. In particular, the transition structure of A_3 will match that of A_2 and each $\widetilde{G}^3_{(q,q')}$ will match $\widetilde{G}^2_{(q,q')}$ in everything but the indices of the labels $\widetilde{G}^1_{(\_,\_)}$ appearing in the element sets. We may write $\widetilde{G}^3_{(q,q')} = \widetilde{G}^2_{(q,q')}[2/1]$ where the notation $[2/1]$ indicates a substitution of the element indices.

To complete the construction of A_1 (and A_2) we need to construct the automata $G^1_{(q,q')}$ (and $G^2_{(q,q')}$) represented by the labels $\widetilde{G}^1_{(q,q')}$ (and $\widetilde{G}^2_{(q,q')}$) for the appropriate q, q'. Because these new automata will be constructed from the automata labelling the egdes of A_0 (and A_1) we construct them simultaneously, constructing a single (1-store multi-)automaton $\mathcal{G}^1$ (resp. $\mathcal{G}^2$) with an initial state $g^1_{(q,q')}$ for each $G^1_{(q,q')}$. The automaton $\mathcal{G}^1$ is constructed through the addition of states and transitions to the disjoint union of $B_1, \dots, B_4, B^a_1$. Similarly, $\mathcal{G}^2$ is built through the addition of states and transitions to $\mathcal{G}^1$. This procedure is illustrated in Figure 2. For the sake of clarity, many states and transitions have been omitted. All transitions are labelled a.

In Figure 2, the innermost frame gives the disjoint union of the automata B_1, B_3 and B^a_1. The middle frame shows an excerpt of $\mathcal{G}^1$. The transition from $g^1_{(q^1,\circ)}$ is derived from the *push*$_\varepsilon$ command applied to B_1, which behaves as a pop command. We can then construct $\mathcal{G}^1_{(q^1,\circ)}$ directly from $\mathcal{G}^1$ taking $g^1_{(q^1,\circ)}$ as the initial state.

The outermost frame gives a partial representation of the automaton $\mathcal{G}^2$. The transition shown from $g^2_{(q^1,\circ)}$ derives from the *push*$_\varepsilon$ command applied to $\mathcal{G}^1_{(q^1,\circ)}$. Ommitted from the diagram is an a-transition from $g^2_{(q^1,\circ)}$ to q^{B_1} resulting from the *push*$_\varepsilon$ command applied to B_1. The branching transition from $g^2_{(q^1,q_f)}$ derives from the set $\{B^a_1, \widetilde{G}^1_{(q^1,q_f)}, B_3\}$ in $\widetilde{G}^2_{(q^1,\circ)}$. That is, we use the power of alternation to construct the product of the automata $B^a_1, \mathcal{G}^1_{(q^1,q_f)}$ and B_3.

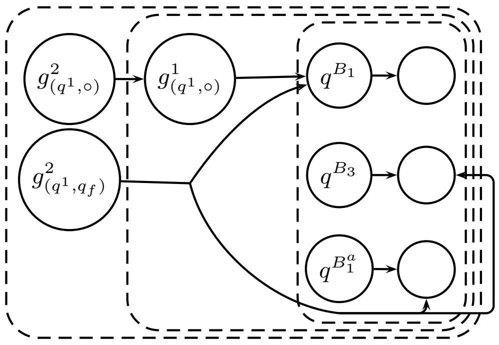


Fig. 2. The automata $\mathcal{G}^0$, $\mathcal{G}^1$ and $\mathcal{G}^2$

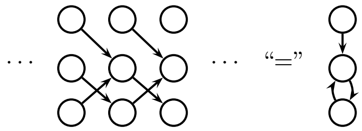


Fig. 3. Collapsing a repetitive chain of new states

We have now constructed the automata A_1 and A_2. We could then repeat this procedure to generate $A_3, A_4, \ldots$, resulting in an infinite sequence of automata that is sound and complete with respect to $Pre^*(\mathcal{L}(A_0))$.

To construct A_* we observe that since a fixed point was reached at A_2, the update to each $\mathcal{G}^i$ to create $\mathcal{G}^{i+1}$ will use similar recipes and hence become repetitive. This will lead to an infinite chain with an unvarying pattern of edges. This chain can be collapsed as shown in Figure 3.

In particular, we are no longer required to add new states to $\mathcal{G}^2$ to construct $\mathcal{G}^i$ for $i > 2$. Instead, we fix the update instructions $\widetilde{G}^2_{(q,q')}[2/1]$ for all q, q' and manipulate $\mathcal{G}^2$ as we manipulated the order-2 structure of A_0 to create A_1 and A_2. We write $\hat{\mathcal{G}}^i$ to distinguish these automata from the automata $\mathcal{G}^i$ generated without fixing the state-set.

Because Σ and the state-set are finite (and remain unchanged), this procedure will reach another fixed point $\hat{\mathcal{G}}^*$ when the transition relation is *saturated* and $\hat{\mathcal{G}}^i = \hat{\mathcal{G}}^{i+1}$. The automaton A_* has the transition structure that became fixed at A_2 labelled with automata derived from the fixed point $\hat{\mathcal{G}}^*$. This automaton will be sound and complete with respect to $Pre^*(\mathcal{L}(A_0))$.

3.2 Preliminaries

To aid in the construction of an automaton representing $Pre^*(C_{Init})$, we introduce a new kind of transition to the 2-store automata. These new transitions are introduced during the processing of the APDS commands. Furthermore, they are labelled with place-holders that will eventually be converted into 1-store automata.

Between any state q_1 and set of states Q_2 we add at most one transition. We associate this transition with an identifier $\widetilde{G}_{(q_1,Q_2)}$. To describe our algorithm we will define sequences of automata, indexed by i. The identifier $\widetilde{G}^i_{(q_1,Q_2)}$ is associated with a set that acts as a recipe for updating the 1-store automaton described by $\widetilde{G}^{i-1}_{(q_1,Q_2)}$ or creating a new automaton if $\widetilde{G}^{i-1}_{(q_1,Q_2)}$ does not exist. Ultimately, the constructed 1-store automaton will label the new transition.

The sets are in a kind of disjunctive normal form. A set $\{S_1,\ldots,S_m\}$ represents an automaton that accepts the union of the languages accepted by the automata described by $S_1,\ldots,S_m$. Each set $S \in \{S_1,\ldots,S_m\}$ corresponds to a possible effect of a command d at order-1 of the automaton. The automaton described by S accepts the intersection of languages described by its elements. An element that is an automaton B refers directly to the automaton B. Similarly, an identifier $\widetilde{G}^i_{(q_1,Q_2)}$ refers to its corresponding automaton. Finally, an element of the form $(a, push_w, \theta)$ refers to an automaton capturing the effect of applying the inverse of the $push_w$ command to the stacks accepted by the automaton represented by θ; moreover, the top_1 character of the stacks accepted by the new automaton will be a. It is a consequence of construction that for any S added during the algorithm, if $(a, push_w, \theta) \in S$ and $(a', push_{w'}, \theta') \in S$ then $a = a'$.

Formally, to each $\widetilde{G}^i_{(q_1,Q_2)}$ we attach a subset of

$$2^{\mathcal{B} \cup \widetilde{\mathcal{G}}^{i-1} \cup (\Sigma \times \mathcal{O}_1 \times (\mathcal{B} \cup \widetilde{\mathcal{G}}^{i-1}))}$$

where $\mathcal{B}$ is the union of the set of all 1-store automata occurring in A_0 and all automata of the form B^a_1 or X^a_1. Further, we denote the set of all order-1 identifiers $\widetilde{G}^i_{(q,Q)}$ in A_i as $\widetilde{\mathcal{G}}^i$. The sets $\mathcal{B}$ and $\mathcal{O}_1$ are finite by definition. If the state-set at order-1 is fixed, there is a finite bound on the size of the set $\widetilde{\mathcal{G}}^i$ for any i.

Given $\widetilde{\mathcal{G}}^i$, we build the automata for all $\widetilde{G}^i_{(q_1,Q_2)} \in \widetilde{\mathcal{G}}^i$ simultaneously. That is, we create a single automaton $\mathcal{G}^i$ associated with the set $\widetilde{\mathcal{G}}^i$. This automaton has a state $g^i_{(q_1,Q_2)}$ for each $\widetilde{G}^i_{(q_1,Q_2)} \in \widetilde{\mathcal{G}}^i$. The automaton $G^i_{(q_1,Q_2)}$ labelling the transition $q_1 \longrightarrow_i Q_2$ is the automaton $\mathcal{G}^i$ with $g^i_{(q_1,Q_2)}$ as its initial state.

The automaton $\mathcal{G}^i$ is built inductively. We set $\mathcal{G}^0$ to be the disjoint union of all automata in $\mathcal{B}$. We define $\mathcal{G}^{i+1} = T_{\widetilde{\mathcal{G}}^{i+1}}(\mathcal{G}^i)$ where $T_{\widetilde{\mathcal{G}}^j}(\mathcal{G}^i)$ is given in Definition 4. In Section 3.4 it will be seen that j is not always $(i+1)$.

Definition 4. Given an automaton $\mathcal{G}^i = (\mathcal{Q}^i, \Sigma, \Delta^i, \_, \mathcal{Q}_f)$ and a set of identifiers $\widetilde{\mathcal{G}}^j_1$, we define,

$$\mathcal{G}^{i+1} = T_{\widetilde{\mathcal{G}}^j}(\mathcal{G}^i) = (\mathcal{Q}^{i+1}, \Sigma, \Delta^{i+1}, \_, \mathcal{Q}_f)$$

where $\mathcal{Q}^{i+1} = \mathcal{Q}^i \cup \{\ g^j_{(q_1,Q_2)} \mid \widetilde{G}^j_{(q_1,Q_2)} \in \widetilde{\mathcal{G}}^j\ \}$, $\Delta^{i+1} = \Delta^{old} \cup \Delta^{new} \cup \Delta^i$, and,

$$\begin{aligned} \Delta^{old} &= \{\ g^j_{(q_1,Q_2)} \xrightarrow{a} Q \mid (g^{j-1}_{(q_1,Q_2)} \xrightarrow{a} Q) \in \Delta^i\ \} \\ \Delta^{new} &= \left\{\ g^j_{(q_1,Q_2)} \xrightarrow{b} Q \mid \widetilde{G}^j_{(q_1,Q_2)} \in \widetilde{\mathcal{G}}^j \text{ and } b \in \Sigma \text{ and } (1)\ \right\} \end{aligned}$$

where (1) requires $\{\alpha_1, \ldots, \alpha_r\} \in \widetilde{G}^j_{(q_1, Q_2)}$, $Q = Q_1 \cup \ldots \cup Q_r$ and for each $t \in \{1, \ldots, r\}$ we have,

- If $\alpha_t = \theta$, then $(q^\theta \xrightarrow{b} Q_t) \in \Delta^i$.
- If $\alpha_t = (a, push_w, \theta)$, then $b = a$ and $q^\theta \xrightarrow{w} Q_t$ is a run of $\mathcal{G}^i$.

3.3 Constructing the Sequence $A_0, A_1, \ldots$

For a given order-n APDS with commands $\mathcal{D}$ we define $A_{i+1} = T_{\mathcal{D}}(A_i)$ where the operation $T_{\mathcal{D}}$ follows.

Definition 5. Given an automaton $A_i = (\mathcal{Q}, \Sigma, \Delta^i, \{q^1, \ldots, q^z\}, \mathcal{Q}_f)$ and a set of commands $\mathcal{D}$, we define,

$$A_{i+1} = T_{\mathcal{D}}(A_i) = (\mathcal{Q}, \Sigma, \Delta^{i+1}, \{q^1, \ldots, q^z\}, \mathcal{Q}_f)$$

where Δ^{i+1} is given below.

We begin by defining the set of labels $\widetilde{G}^{i+1}$. This set contains labels on transitions present in A_i, and labels on transitions derived from $\mathcal{D}$. That is,

$$\widetilde{G}^{i+1} = \left\{ \widetilde{G}^{i+1}_{(q,Q)} \mid (q \xrightarrow{\widetilde{G}^i_{(q,Q)}} Q) \in \Delta^i \right\} \cup \left\{ \widetilde{G}^{i+1}_{(q^j,Q)} \mid (2) \right\}$$

The contents of the sets $\widetilde{G}^{i+1}_{(q,Q)} \in \widetilde{G}^{i+1}$ are defined $\widetilde{G}^{i+1}_{(q^j,Q)} = \{\ S \mid (2)\ \}$ where (2) requires $(p^j, a, \{(o_1, p^{k_1}), \ldots, (o_m, p^{k_m})\}) \in \mathcal{D}$, $Q = Q_1 \cup \ldots \cup Q_m$, $S = S_1 \cup \ldots \cup S_m$ and for each $t \in \{1, \ldots, m\}$ we have,

- If $o_t = push_2$, then $S_t = \{B^a_1\} \cup \widetilde{\theta}_1 \cup \widetilde{\theta}_2$ and there exists a path $q^{k_t} \xrightarrow{\widetilde{\theta}_1}_i Q' \xrightarrow{\widetilde{\theta}_2}_i Q_t$ in A_i.
- If $o_t = pop_2$, then $S_t = \{B^a_1\}$ and $Q_t = \{q^{k_t}\}$. Or, if $q^j \xrightarrow{\nabla}_i \{q^\varepsilon_f\}$ exists in A_i, we may have $S_t = \{B^a_1\}$ and $Q_t = \{q^\varepsilon_f\}$.
- If $o_t = push_w$ then $S_t = \{(a, push_w, \theta)\}$ and there exists a transition $q^{k_t} \xrightarrow{\theta}_i Q_t$ in A_i.

Finally, we give the transition relation Δ^{i+1}.

$$\Delta^{i+1} = \left\{ q \xrightarrow{B} Q \mid \begin{array}{c} (q \xrightarrow{B} Q) \in \Delta^i \\ \text{and } B \in \mathcal{B} \end{array} \right\} \cup \left\{ q \xrightarrow{\widetilde{G}^{i+1}_{(q,Q)}} Q \mid \widetilde{G}^{i+1}_{(q,Q)} \in \widetilde{\mathcal{G}}^{i+1} \right\}$$

We can construct an automaton whose transitions are 1-store automata by replacing each set $\widetilde{G}^{i+1}_{(q,Q)}$ with the automaton $G^{i+1}_{(q,Q)}$ which is $\mathcal{G}^{i+1}$ with initial state $g^{i+1}_{(q,Q)}$, where $\mathcal{G}^{i+1} = T_{\widetilde{\mathcal{G}}^{i+1}}(\mathcal{G}^i)$. Note that $\mathcal{G}^i$ is assumed by induction.

By repeated applications of $T_{\mathcal{D}}$ we construct the sequence $A_0, A_1, \ldots$ which is sound and complete with respect to $Pre^*(C_{Init})$.

Property 1. For any configuration $\langle p^j, \gamma\rangle$ it is the case that $\gamma \in \mathcal{L}(A^{q^j}_i)$ for some i iff $\langle p^j, \gamma\rangle \in Pre^*(C_{Init})$.

3.4 Constructing the Automaton A_*

We need to construct a finite representation of the sequence $A_0, A_1, \ldots$ in a finite amount of time. To do this we will construct an automaton A_* such that $\mathcal{L}(A_*) = \bigcup_{i \geq 0} \mathcal{L}(A_i)$. We begin by introducing some notation and a notion of subset modulo i for the sets $\widetilde{G}^i_{(q_1,Q_2)}$.

Definition 6. Given $\theta \in \mathcal{B} \cup \widetilde{\mathcal{G}}^i$ for some i, let

$$\theta[j/i] = \begin{cases} \theta & \text{if } \theta \in \mathcal{B} \\ G^j_{(q_1,Q_2)} & \text{if } \theta = G^i_{(q_1,Q_2)} \in \widetilde{\mathcal{G}}^i \end{cases}$$

For a set S we define $S[j/i]$ such that, $\theta \in S$ iff we have $\theta[j/i] \in S[j/i]$, and $(a, push_w, \theta) \in S$ iff we have $(a, push_w, \theta[j/i]) \in S[j/i]$.We extend the notation $[j/i]$ point-wise to nested sets of sets structures. Finally, we define,

1. $\widetilde{G}^i_{(q_1,Q_2)} \lesssim \widetilde{G}^j_{(q_1,Q_2)}$ iff for each $S \in \widetilde{G}^i_{(q_1,Q_2)}$ we have $S[j-1/i-1] \in \widetilde{G}^j_{(q_1,Q_2)}$.
2. $\widetilde{\mathcal{G}}^i \lesssim \widetilde{\mathcal{G}}^j$ iff for all $\widetilde{G}^i_{(q_1,Q_2)} \in \widetilde{\mathcal{G}}^i$ we have $\widetilde{G}^j_{(q_1,Q_2)} \in \widetilde{\mathcal{G}}^j$ and $\widetilde{G}^i_{(q_1,Q_2)} \lesssim \widetilde{G}^j_{(q_1,Q_2)}$.

Writing $A \simeq B$ to mean $A \lesssim B$ and $B \lesssim A$, we now show that the sets labelling the transitions of $A_0, A_1, \ldots$ reach a fixed point. Once a fixed point $\widetilde{\mathcal{G}}^i \simeq \widetilde{\mathcal{G}}^{i_1}$ has been reached, we can stop adding new states during the construction of $\mathcal{G}^{i_1}, \mathcal{G}^{i_1+1}, \ldots$.

Property 2. There exists $i_1 > 0$ such that $\widetilde{\mathcal{G}}^i \simeq \widetilde{\mathcal{G}}^{i_1}$ for all $i \geq i_1$.

Proof. (Sketch) Since the order-1 state-set in A_i remains constant and we add at most one transition between any state q_1 and set of states Q_2, there is some i_1 where no more transitions are added at order-2. That $\widetilde{\mathcal{G}}^i \simeq \widetilde{\mathcal{G}}^{i_1}$ for all $i \geq i_1$ follows since the contents of $\widetilde{G}^i_{(q_1,Q_2)}$ and $\widetilde{G}^{i_1}_{(q_1,Q_2)}$ are derived from the same transition structure.

Lemma 1. *Suppose we have a sequence of automata $\mathcal{G}^0, \mathcal{G}^1, \ldots$ and associated sets $\widetilde{\mathcal{G}}^0, \widetilde{\mathcal{G}}^1, \ldots$. Further, suppose there exists an i_1 such that for all $i \geq i_1$ we have $\widetilde{\mathcal{G}}^i \simeq \widetilde{\mathcal{G}}^{i_1}$. We can define a sequence of automata $\hat{\mathcal{G}}^{i_1}, \hat{\mathcal{G}}^{i_1+1}, \ldots$ such that the state-set in $\hat{\mathcal{G}}^i$ remains constant. The following are equivalent for all w,*

1. *The run $g^{i_1}_{(q_1,Q_2)} \xrightarrow{w}_i Q$ with $Q \subseteq \mathcal{Q}_f$ exists in $\hat{\mathcal{G}}^i$ for some i.*
2. *The run $g^{i'}_{(q_1,Q_2)} \xrightarrow{w}_{i'} Q'$ with $Q' \subseteq \mathcal{Q}_f$ exists in $\mathcal{G}^{i'}$ for some i'.*

We use $\hat{\mathcal{G}}^{i+1} = T_{\widetilde{\mathcal{G}}^{i_1}[i_1/i_1-1]}(\hat{\mathcal{G}}^i)$ to construct the sequence $\hat{\mathcal{G}}^{i_1}, \hat{\mathcal{G}}^{i_1+1}, \ldots$. Intuitively, since the transitions from the states introduced to define $\mathcal{G}^i$ for $i \geq i_1$ are derived from similar sets, we can compress the subsequent repetition into a single set of new states as shown in Figure 3. Since the state-set of this new sequence does not change and the alphabet Σ is finite, the transition structure will become saturated.

Property 3. For a sequence of automata $\mathcal{G}^0, \mathcal{G}^1, \ldots$ such that the state-set of $\mathcal{G}^i$ remains constant there exists $i_0 > 0$ such that $\mathcal{G}^i = \mathcal{G}^{i_0}$ for all $i \geq i_0$.

Thus, we have the following algorithm for constructing A_*:

1. Given A_0, iterate $A_{i+1} = T_{\mathcal{D}}(A_i)$ until the fixed point A_{i_1} is reached.
2. Iterate $\mathcal{G}^{i+1} = T_{\widetilde{\mathcal{G}}_l^{i_1}[i_1/i_1-1]}(\mathcal{G}^i)$ to generate the fixed point $\mathcal{G}^{i_0}$ from $\mathcal{G}^{i_1}$.
3. Construct A_* by labelling the transitions of A_{i_1} with automata derived from $\mathcal{G}^{i_0}$.

Property 4. There exists an automaton A_* which is sound and complete with respect to $A_0, A_1, \ldots$ and hence computes the set $Pre^*(C_{Init})$.

3.5 The General Case

We may generalise our algorithm to order-n for all n by extending Definition 4 to l-store automata using similar techniques to those used in Definition 5. Termination is reached through a cascading of fixed points. As we fixed the state-set at order-1 in the order-2 case, we may fix the state-set at order-$(n-1)$ in the order-n case. We may then generalise Property 2 and Lemma 1 to find a sequence of fixed points $i_n, \ldots, i_0$, from which A_* can be constructed. For a complete description of this procedure, we refer the reader to the long version of this paper [13].

We claim our algorithm runs in n-EXPTIME. Intuitively, when the state-set $\mathcal{Q}$ is fixed at order-1 of the store automaton, we add at most $\mathcal{O}(2^{|\mathcal{Q}|})$ transitions (since we never remove states, it is this final stage that dominates the complexity). At orders $l > 1$ we add at most $\mathcal{O}(2^{|\mathcal{Q}|})$ new transitions, which exponentially increases the state-set at order-$(l-1)$. Hence, the algorithm runs in n-EXPTIME.

4 Applications

We give a brief description of a number of applications of our result:

- *Reachability Games.* Given an order-n pushdown reachability game with a regular set of goal configurations $\mathcal{R}$, we can calculate the winning region (which is regular) for the existential player in n-EXPTIME.
- *Linear-Time Model Checking.* Given an order-n PDS $(\mathcal{P}, \mathcal{D}, \Sigma)$ and a formula ϕ of an ω-regular logic, we can calculate in $(n+2)$-EXPTIME the set of configurations C such that every run from each $c \in C$ satisfies ϕ.
- *The Alternation-Free μ-Calculus.* Given an order-n PDS $(\mathcal{P}, \mathcal{D}, \Sigma)$ and a formula ϕ of the alternation-free μ-calculus, we can compute the regular set of configurations satisfying ϕ in $((|\phi| \cdot n) + 1)$-EXPTIME.

5 Conclusion

Given an automaton representation of a regular set of higher-order APDS configurations C_{Init}, we have shown that the set $Pre^*(C_{Init})$ is regular and computable via automata-theoretic methods. This builds upon previous work on pushdown systems [2] and higher-order context-free processes [1]. The main innovation of this generalisation is the careful management of a complex automaton construction. This allows us to identify a sequence of cascading fixed points, resulting in a terminating algorithm.

Our result has many applications. We have shown that it can be used to provide a solution to the model checking problem for linear-time temporal logics and the alternation-free μ-calculus. In particular we compute the set of configurations of a higher-order PDS satisfying a given constraint. We also show that the winning regions can be computed for a reachability game played over an higher-order PDS.

There are several possible extensions to this work. Firstly, we intend to complete the complexity analysis with corresponding hardness results. Although this result is widely accepted to follow from the work of Engelfriet [7], we intend to give an alternative proof in the long version of this paper. Secondly, we plan to investigate the applications of this work to higher-order pushdown games with more general winning conditions. In his PhD thesis, Cachat adapts the reachability algorithm of Bouajjani *et al.* [2] to calculate the winning regions in Büchi games over pushdown processes [22]. It is likely that our work will permit similar extensions. Finally, we intend to generalise this work to higher-order collapsible pushdown automata, which can be used to study higher-order recursion schemes [25,8]. This may provide the first steps into the study of games over these structures.

Acknowledgments. We thank Olivier Serre and Arnaud Carayol for helpful discussions and the anonymous referees for their detailed comments.

References

1. A. Bouajjani and A. Meyer. Symbolic Reachability Analysis of Higher-Order Context-Free Processes. In *Proc. FSTTCS'04*, 2004. LNCS 3328.
2. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proc. CONCUR '97*, pp. 135–150, 1997.
3. A. Bouquet, O. Serre, and I. Walukiewicz. Pushdown games with the unboundedness and regular conditions. In *Proc. FSTTCS'03*, pages 88–99, 2003.
4. A. Carayol. Regular sets of higher-order pushdown stacks. In *Proc. MFCS*, pages 168–179, 2005.
5. A. Carayol and S. Wöhrle. The caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *Proc. FSTTCS*, pages 112–123, 2003.
6. A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.
7. J. Engelfriet. Iterated push-down automata and complexity classes. In *Proc. STOC*, pages 365–373, 1983.

8. M. Hague, A. S. Murawski, O. Serre and C.-H. L. Ong. Collapsible pushdown automata and recursion schemes, 2006. Preprint, 13 pages.
9. C. Löding, P. Madhusudan, and O. Serre. Visibly pushdown games. In *Proc. FSTTCS'04*, pages 408–420. 2004. LNCS 3328.
10. D. Caucal. On infinite terms having a decidable monadic theory. In *Proc. MFCS'02*, pages 165–176, 2002. LNCS 2420.
11. D. E. Muller and P. E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theor. Comput. Sci.*, 37:51–75, 1985.
12. H. Gimbert. Parity and exploration games on infinite graphs. In *Proc. CSL'04*, pages 56–70, 2004. LNCS 3210.
13. M. Hague and C.-H. L. Ong. Symbolic backwards-reachability analysis for higher-order pushdown systems. Preprint, 54 pages, `www.comlab.ox.ac.uk/oucl/work/matthew.hague/FoSSaCS07-long.pdf`, 2006.
14. I. Walukiewicz. Pushdown processes: Games and model checking. In *Proc. CAV '96*, pages 62–74. 1996.
15. J. A. Brzozowski and E. L. Leiss. On equations for regular languages, finite automata, and sequential networks. *Theor. Comput. Sci.*, 10:19–35, 1980.
16. K. Aehlig, J. G. de Miranda, and C.-H. L. Ong. Safety is not a restriction at level 2 for string languages. In *Proc. FoSSaCS*, pages 490–504, 2005.
17. M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Banff Higher Order Workshop*, pages 238–266, 1995.
18. A. N. Maslov. Multilevel stack automata. *Problems of Information Transmission*, 15:1170–1174, 1976.
19. O. Serre. Note on winning positions on pushdown games with ω-regular conditions. *Information Processing Letters*, 85:285–291, 2003.
20. O. Serre. Games with winning conditions of high Borel complexity. In *Proc. ICALP'04*, pages 1150–1162. Springer-Verlag, 2004. LNCS 3142.
21. C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *Proc. LICS '06*, pages 81–90. IEEE Computer Society, 2006.
22. T. Cachat. *Games on Pushdown Graphs and Extensions.* PhD thesis, RWTH Aachen, 2003.
23. T. Cachat, J. Duparc, and W. Thomas. Solving pushdown games with a Σ_3 winning condition. In *Proc. CSL'02*, pages 322–336. Springer-Verlag, 2002. LNCS 2471.
24. T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *Proc. FoSSaCS '02*, pages 205–222, London, UK, 2002. Springer-Verlag.
25. T. Knapik, D. Niwinski, P. Urzyczyn, and I. Walukiewicz. Unsafe grammars and panic automata. In *Proc. ICALP '05*, pages 1450–1461, 2005.
26. M. Y. Vardi. A temporal fixpoint calculus. In *Proc. POPL '88*, pages 250–259, New York, NY, USA, 1988. ACM Press.
27. W. Thomas. Automata on infinite objects. *Handbook of theoretical computer science (vol. B): formal models and semantics*, pages 133–191, 1990.

A Distribution Law for CCS and a New Congruence Result for the π-Calculus

Daniel Hirschkoff and Damien Pous

LIP – ENS Lyon, CNRS, INRIA, UCBL, France

Abstract. We give an axiomatisation of strong bisimilarity on a small fragment of CCS that does not feature the sum operator. This axiomatisation is then used to derive congruence of strong bisimilarity in the finite π-calculus in absence of sum. To our knowledge, this is the only nontrivial subcalculus of the π-calculus that includes the full output prefix and for which strong bisimilarity is a congruence.

Introduction

In this paper, we study strong bisimilarity on two process calculi. We first focus on *microCCS* (μCCS), the very restricted fragment of CCS that only features prefix and parallel composition. Our main result on μCCS is that adding the following *distribution law*

$$\eta.(P \,|\, \eta.P \,|\, \dots \,|\, \eta.P) \quad = \quad \eta.P \,|\, \eta.P \,|\, \dots \,|\, \eta.P$$

to the laws of an abelian monoid for parallel composition yields a complete axiomatisation of strong bisimilarity (in the law above, η is a CCS prefix, of the form a or $\overline{a}$, and P is any CCS process – the same number of copies of P appear on both sides of the equation).

The distribution law is not new: it is mentioned – among other *'mixed equations'* relating prefixed terms and parallel compositions – in a study of bisimilarity on normed PA processes [8]. In our setting, this equality can be oriented from left to right to rewrite processes into normal forms, which intuitively exhibit as much concurrency as possible. Strong bisimilarity ($\sim$) between processes is then equivalent to equality of their normal forms. This rewriting phase allows us to actually compute *unique decompositions* of processes into *prime processes*, in the sense of [10]: a process P is prime if P is not bisimilar to the inactive process $\mathbf{0}$ and if $P \sim Q \,|\, R$ implies $Q \sim \mathbf{0}$ or $R \sim \mathbf{0}$.

The distribution law is an equational schema, corresponding to an infinite family of axioms, of the form $\eta.(P \,|\, (\eta.P)^k) = (\eta.P)^{k+1}$, for $k \geq 1$ (where Q^k denotes the k-fold parallel composition of process Q). We show that although our setting is rather simple, there exists no finite axiomatisation of $\sim$ on μCCS.

We then move to the study of strong bisimilarity in the π-calculus. Because of the presence of the input prefix, and of the related phenomenon of name-passing, bisimilarity is more complex in the π-calculus than in CCS. In particular, both early and late bisimilarity, that differ in their treatment of name substitution, fail to be congruences in the full π-calculus.

There exist subcalculi of the π-calculus for which strong bisimilarity is a congruence (we discuss these in Section 5). When this is the case, this equivalence coincides with *ground bisimilarity* ($\sim_{\mathrm{g}}$), which allows one to consider only one fresh name when inspecting an input transition, instead of the usual quantification involving all free names of the process. Congruence of strong bisimilarity is hence an important property: not only is it necessary in order to reason in a compositional way, but it also helps making bisimulation proofs simpler, by reducing the size of case analyses.

In the full π-calculus, in order to get congruence, one has to work with Sangiorgi's open bisimilarity [12], which has a more involved definition than the early and late variants. Tools like the Mobility Workbench [14], for instance, have adopted this equivalence on processes.

It is known [13] that bisimilarity in the π-calculus fails to be a congruence as soon as we have prefix, parallel composition, restriction and replication. In this work, we focus on the finite, sum-free π-calculus, that we call π_0. We rely on the axiomatisation of strong bisimilarity on μCCS to prove that ground bisimilarity ($\sim_{\mathrm{g}}$) is closed under substitutions in π_0, i.e., that whenever $P \sim_{\mathrm{g}} Q$, then $P\sigma \sim_{\mathrm{g}} Q\sigma$ for any substitution σ. Closure under substitution of ground bisimilarity entails that on π_0, ground, early, late and open bisimilarities coincide, and are congruences. The problem of congruence of $\sim_{\mathrm{g}}$ on π_0 is mentioned as an open question in [13, Chapter 5], and is known since at least 1998 [2]. To our knowledge, this is the first congruence result for a subcalculus of the π-calculus that includes the full output prefix (see Section 5 for a discussion on this).

At the heart of our proof of congruence is a notion that we call *mutual desynchronisation*, and that corresponds to the existence of processes T, T_{12}, T_{21} such that $T \xrightarrow{\eta_1}\xrightarrow{\eta_2} T_{12}$ and $T \xrightarrow{\eta_2}\xrightarrow{\eta_1} T_{21}$, for two distinct actions η_1 and η_2, and with $T_{12} \sim T_{21}$. We additionally require in the two sequences of transitions from T to T_{12} and T_{21} respectively that the second prefix being fired should occur under the first prefix in T. In other words, in such a situation, the process T behaves as if the two actions η_1, η_2 were offered concurrently, but the simultaneous firing of these actions can only be emulated by triggering consecutive prefixes.

Using our analysis of strong bisimilarity on μCCS, we show that mutual desynchronisations do not exist in μCCS. This is essentially due to the fact that our axiomatisation of $\sim$ on μCCS does not allow one to relate two *distinct* prefixes when performed concurrently and sequentially. When moving to the π-calculus, it turns out that substitution closure of $\sim_{\mathrm{g}}$ amounts to observing absence of mutual desynchronisations in π_0. We exploit a transfer property, that extracts a bisimilarity proof in μCCS from a bisimilarity proof in π_0, to relate the two calculi and to show that mutual desynchronisations do not exist in π_0, yielding congruence of $\sim_{\mathrm{g}}$.

Paper outline. We introduce μCCS and the distribution law in Section 1. Section 2 is devoted to the characterisation of $\sim$ on μCCS using normal forms. In Section 3, we prove that no finite axiomatisation of $\sim$ on μCCS exists.

Section 4 presents the proof of our congruence result in the π-calculus, and we give concluding remarks in Section 5.

1 MicroCCS Processes and Normal Forms

We consider an infinite set $\mathcal{N}$ of names, ranged over with $a, b \dots$. We define on top of $\mathcal{N}$ the set of processes of μCCS, the finite, public, sum-free CCS calculus, ranged over using $P, Q, R \dots$, as follows:

$$\eta ::= a \;\big|\; \overline{a} \;, \qquad\qquad P ::= \mathbf{0} \;\big|\; \eta.P \;\big|\; P_1 \,|\, P_2 \;.$$

$\mathbf{0}$ is the nil process. η ranges over visible actions and co-actions, called *interactions*, and we let $\overline{\eta}$ stand for the co-action associated to η (we have $\overline{\overline{\eta}} = \eta$). For $k > 0$, we write P^k for the parallel composition of k copies of P, and we write $\prod_{i \in I} P_i$ for the parallel composition of all processes P_i for $i \in I$. It can be noted that our syntax does not include a construction of the form $\tau.P$ — see Remark 2.3 below.

Structural congruence, written $\equiv$, is defined as the smallest congruence satisfying the following laws:

$$(C_1) \quad P \,|\, Q \equiv Q \,|\, P \qquad (C_2) \quad P \,|\, (Q \,|\, R) \equiv (P \,|\, Q) \,|\, R \qquad (C_3) \quad P \,|\, \mathbf{0} \equiv P$$

We introduce a labelled transition system (LTS) for μCCS. Actions labelling transitions, ranged over with μ, are either interactions, or a special silent action, written τ.

Definition 1.1 (Operational semantics and behavioural equivalence). *The LTS for μCCS is given by the following rules:*

$$\eta.P \xrightarrow{\eta} P \qquad \frac{P \xrightarrow{\eta} P' \qquad Q \xrightarrow{\overline{\eta}} Q'}{P \,|\, Q \xrightarrow{\tau} P' \,|\, Q'} \qquad \frac{P \xrightarrow{\mu} P'}{P \,|\, Q \xrightarrow{\mu} P' \,|\, Q} \qquad \frac{P \xrightarrow{\mu} P'}{Q \,|\, P \xrightarrow{\mu} Q \,|\, P'}$$

A bisimulation *is a symmetrical relation $\mathcal{R}$ between processes such that whenever $P \mathrel{\mathcal{R}} Q$ and $P \xrightarrow{\mu} P'$, there exists Q' such that $Q \xrightarrow{\mu} Q'$ and $P' \mathrel{\mathcal{R}} Q'$.* Bisimilarity*, written $\sim$, is the union of all bisimulations.*

Definition 1.2 (Size). *Given P, $\#(P)$ (called the* size *of P) is defined by:*

$$\#(\mathbf{0}) \overset{\text{def}}{=} 0 \qquad \#(P_1 \,|\, P_2) \overset{\text{def}}{=} \#(P_1) + \#(P_2) \qquad \#(\eta.P) \overset{\text{def}}{=} 1 + \#(P) \;.$$

Lemma 1.3. *$P \equiv Q$ implies $P \sim Q$ which in turn implies $\#(P) = \#(Q)$.*

Proof. The first implication follows by proving that $\equiv$ is a bisimulation.

Suppose then by contradiction that there exist P, Q such that $P \sim Q$ and $\#(P) < \#(Q)$; and choose such P with minimal size. Q has at least one prefix: $Q \xrightarrow{\eta} Q'$ and we get $P \xrightarrow{\eta} P'$ with $P' \sim Q'$. Necessarily, we must have $\#(P') < \#(Q')$ and $\#(P') < \#(P)$ which contradicts the minimality hypothesis. $\square$

Definition 1.4 (Distribution law). *The* distribution law *is given by the following equation, where the same number of copies of P appears on both sides:*

$$\eta.(P \,|\, \eta.P \,|\, \ldots \,|\, \eta.P) \quad = \quad \eta.P \,|\, \eta.P \,|\, \ldots \,|\, \eta.P \ .$$

We shall use this equality, oriented from left to right, to rewrite processes. We write $P \rightsquigarrow P'$ when there exist P_1, P_2 such that $P \equiv P_1$, $P_2 \equiv P'$ and P_2 is obtained from P_1 by replacing a sub-term of the form of the left-hand side process with the right-hand side process.

Remark 1.1 (On the distribution law and PA). Among the studies about properties of $\sim$ in process algebras that include parallel composition (see [1] for a recent survey on axiomatisations), some works focus on calculi where parallel composition is treated as a primitive operator (as opposed to being expressible using sum or other constructs like the left merge operator). As mentioned above, particularly relevant to this work is [8], where Hirshfeld and Jerrum *"develop a structure theory for PA that completely classifies the situations in which a sequential composition of two processes can be bisimilar to a parallel composition"*. [8] establishes decidability of $\sim$ for normed PA processes: in that setting, the formal analogue of the distribution law (Def. 1.4) holds with η and P being two processes — the 'dot' operator is a general form of sequential composition. This equality is valid in [8] whenever η is a 'monomorphic process', meaning that η can only reduce to $\mathbf{0}$ (which corresponds to μCCS), or to η itself. [6] presents a finite axiomatisation of PA that exploits the operators of sum and left merge.

Lemma 1.5. *The relation $\rightsquigarrow$ is strongly normalising and confluent.*

Proof. If $P \rightsquigarrow P'$ then the weight of P' (defined as sum of the depths of all prefixes occuring in P') is strictly smaller than the weight of P, whence the strong normalisation. We then remark that $\rightsquigarrow$ is locally confluent, and conclude with Newman's Lemma. $\square$

Thus, for any process P, $\rightsquigarrow$ defines a normal form unique up to $\equiv$, that will be denoted by $\mathsf{n}(P)$. We let $A, B, \ldots$ range over normal forms.

The following lemma states that $\rightsquigarrow$ preserves bisimilarity:

Lemma 1.6. *If $P \rightsquigarrow P'$, then $P \sim P'$. For any P, $P \sim \mathsf{n}(P)$.*

Proof. The relation $(\rightsquigarrow \cup (\rightsquigarrow)^{-1} \cup \equiv)$ is a bisimulation. $\square$

2 Characterisation of Bisimilarity in MicroCCS

Our characterisation of $\sim$ on μCCS makes use of the notion of decomposition into *prime processes*, defined as follows:

Definition 2.1. *A process P is* prime *if $P \not\sim \mathbf{0}$ and $P \sim P_1 \,|\, P_2$ implies $P_1 \sim \mathbf{0}$ or $P_2 \sim \mathbf{0}$.*

When $P \sim P_1 \,|\, \ldots \,|\, P_n$ where the P_is are prime, we shall call $P_1 \,|\, \ldots \,|\, P_n$ a prime decomposition *of P.*

Proposition 2.2 (Unique decomposition). *Any process admits a prime decomposition which is unique up to bisimilarity: if $P_1 \mid \ldots \mid P_n$ and $Q_1 \mid \ldots \mid Q_m$ are two prime decompositions of the same process, then $n = m$ and $P_i \sim Q_i$ for all $i \in [1..n]$, up to a permutation of the indices.*

Proof. Similar to the proof of [11, Theorem 4.3.1]: the case of μCCS is not explicitly treated in that work, but the proof can be adapted rather easily. □

An immediate consequence of the above result is the following property:

Corollary 2.3 (Cancellation). *For all P, Q, R, $P \mid R \sim Q \mid R$ implies $P \sim Q$.*

Note that this is not true in presence of replication: $a \mid !a \sim \mathbf{0} \mid !a$, but $a \not\sim \mathbf{0}$.

The characterisation of $\sim$ using the distribution law follows from the observation that if a normal form is a prefixed process, then it is prime. This idea is used in the proof of Lemma 2.5. We first establish a technical result, that essentially exploits the same argument as the proof of Theorem 4.2 in [7].

Lemma 2.4. *If $\eta.P \sim Q \mid Q'$, with $Q, Q' \not\sim \mathbf{0}$, then there exist A and $k > 1$ such that $\eta.P \sim (\eta.A)^k$ and $\eta.A$ is a normal form.*

Proof. By Lemma 1.6, we have $\eta.P \sim \mathsf{n}(Q \mid Q')$. Furthermore, we have that $\mathsf{n}(Q \mid Q') \equiv \prod_{i \leq k} \eta_i.A_i$, where $k > 1$ and the processes $\eta_i.A_i$ are in normal form.

Since the η prefix must be triggered to answer any challenge from the right hand side, we have $\eta_i = \eta$ and $P \sim A_i \mid \prod_{l \neq i} \eta.A_l$ for all $i \leq k$. In particular, when $i \neq j$, we have $P \sim A_i \mid \eta.A_j \mid \prod_{l \notin \{i,j\}} \eta.A_l \sim \eta.A_i \mid A_j \mid \prod_{l \notin \{i,j\}} \eta.A_l$ and hence, by Corollary 2.3, $A_i \mid \eta.A_j \sim \eta.A_i \mid A_j$. By reasoning on the sizes of the parallel components in the prime decompositions of these two terms, we conclude that $\eta.A_i \sim \eta.A_j$ for all $i, j \leq k$.

Hence, we have $\eta.P \sim (\eta.A_1)^k$ with $k > 1$ and $\eta.A_1$ is a normal form. □

Lemma 2.5. *Let A, B be two normal forms, $A \sim B$ implies $A \equiv B$.*

Proof. We show by induction on n that for all A with $\#(A) = n$, we have

(i) if A is a prefixed process, then A is prime;
(ii) for any B, $A \sim B$ implies $A \equiv B$.

The case $n = 0$ is immediate. Suppose that the property holds for all $i < n$, with $n \geq 1$.

(i) We write $A = \eta.A'$, and suppose by contradiction $A \sim P_1 \mid P_2$ with $P_1, P_2 \not\sim \mathbf{0}$. By Lemma 2.4, we have $A \sim (\eta.B)^k$ with $k > 1$ and $\eta.B$ in normal form. By triggering the prefix on the left hand side, we have $A' \sim B \mid (\eta.B)^{k-1}$. It follows by induction that $A' \equiv B \mid (\eta.B)^{k-1}$ (using property (ii)), and hence $A \equiv \eta.(B \mid (\eta.B)^{k-1}$, which is in contradiction with the fact that A is in normal form.

(ii) Suppose now $A \sim B$.

- If A is a prefixed process, B is prime by the previous point ($\#(B) = \#(A)$ by Lemma 1.3). Necessarily, $A \equiv \eta.A'$ and $B \equiv \eta.B'$ with $A' \sim B'$. By induction, this entails $A' \equiv B'$, and $A \equiv B$.
- Otherwise, $A = \eta_1.A_1 \,|\, \dots \,|\, \eta_k.A_k$ with $k > 1$, and we know by induction (property (i)) that $\eta_i.A_i$ is prime for all $i \leq k$. Similarly, we have $B = \eta'_1.B_1 \,|\, \dots \,|\, \eta'_l.B_l$ with $\eta'_i.B_i$ prime for all $i \leq l$.
  By Proposition 2.2, $k = m$ and $\eta_i.A_i \sim \eta'_i.B_i$ (up to a permutation of the indices), which gives $\eta'_i = \eta_i$ and $A_i \sim B_i$ for all $i \leq k$. By induction, we deduce $A_i \equiv B_i$ for all i, which finally implies $A \equiv B$. □

Lemmas 1.6 and 2.5 allow us to deduce the following result.

Theorem 2.6. *Let P, Q be two μCCS processes. Then $P \sim Q$ iff $\mathsf{n}(P) \equiv \mathsf{n}(Q)$.*

Remark 2.1 (Unique decomposition of processes). Our proof relies on unique decomposition of processes (Prop. 2.2), that first appeared in [10]. Unique decomposition has been established for a variety of process algebras, and used as a way to prove decidability of behavioural equivalence and to give complexity bounds for the associated decision procedure ([9,3] cite relevant references).

In the present study, beyond the existence of a unique decomposition, we are interested in a syntactic characterisation of $\sim$ (which will in particular allow us to derive Lemma 4.6 below). In this sense, our work is close to [5], where the notion of *maximally parallel process* in CCS (with choice) is studied. [5] defines a rewriting process through which maximally parallel normal forms can be computed, and shows that in the case of μCCS, such normal forms are unique. However, no syntactical characterisation of the set of normal forms is presented, and such a characterisation cannot be directly deduced from the (rather involved) definition of the rewriting process for full CCS.

We instead restrict ourselves to μCCS from the start, and rely explicitly on the distribution law in order to 'extract' prime components of processes.

Remark 2.2 (Closure under substitutions). In (full) CCS, two strongly bisimilar processes need not remain bisimilar whenever we apply a substitution that replaces names with names. The standard counterexample is given by $a.\overline{b} + \overline{b}.a \sim a \,|\, \overline{b}$: when we replace b with a, we obtain two processes that are distinguished by $\sim$, since the latter can perform a τ transition that cannot be matched by the former. This irregularity is the basis of the standard counterexample showing that strong bisimilarity is not a congruence in the π-calculus.

In μCCS, on the other hand, $\sim$ is closed under substitutions: the intuitive reason is that two processes related by an instance of the distribution law remain equivalent when a substitution is applied (we can show in particular that for any substitution σ, $\mathsf{n}(P\sigma) \equiv \mathsf{n}(\mathsf{n}(P)\sigma)$). This is not the case for the expansion law, of which the counterexample above is an instance.

Remark 2.3 (τ prefix and weak bisimilarity). We do not address weak bisimilarity in the present work. In μCCS, strong and weak bisimilarity coincide, i.e., the internal transitions of processes are completely determined by the visible actions (interactions). When including τ prefixes in the syntax, it can be proved that adding the law $\tau.P = P$ is enough to characterise weak bisimilarity. The τ prefix is usually absent in the π-calculus, to which we shall move in Section 4. Since some results on CCS will be transferred to the π-calculus, we did not include this construct in μCCS.

3 Nonexistence of a Finite Axiomatisation

We let $\mathcal{D}$ stand for the set of equations consisting of the three axioms of structural congruence (C_1, C_2, C_3), and the infinite family of *distribution axioms*

$$(D_k): \ \eta.(P \mid (\eta.P)^k) \ = \ (\eta.P)^{k+1}, \ k \geq 1 \ .$$

We let $\mathcal{D}_k$ stand for the finite restriction of $\mathcal{D}$ where only the first k distribution axioms are included ($(D_i)_{1 \leq i \leq k}$). We shall write $\mathcal{E} \vdash P = Q$ whenever $P = Q$ can be derived in equational logic using a given set $\mathcal{E}$ of axioms, and $\mathcal{E} \nvdash P = Q$ when this is not the case.

$(D_k)_{k \geq 1}$ forms an equational schema for the distribution law, and Theorem 2.6 states that $\mathcal{D}$ is a complete axiomatisation of strong bisimilarity on μCCS. Using a rather classical approach (i.e., establishing ω-completeness and proving compactness, see [1]), this leads to the nonexistence of a finite axiomatisation of $\sim$ on μCCS. The lemma below provides the central technical property satisfied by the $(\mathcal{D}_k)_{k \geq 1}$ which is necessary to derive Theorem 3.2, that says that $\mathcal{D}$ is *intrinsically* infinite.

Lemma 3.1. *Let a be a name. For any k, there exists n s.t. $\mathcal{D}_k \nvdash a.a^n = a^{n+1}$.*

Remember that a^n stands for the n-fold parallel composition of $a.\mathbf{0}$, so that the above equality is an instance of axiom (D_n).

Proof. Let n be a number strictly greater than k such that $n+1$ is prime, and let $\theta(P, Q)$ denote the predicate: "$P \sim Q \sim a^{n+1}$, $P \equiv a.P'$, and $Q \equiv Q_1 \mid Q_2$ with $Q_1, Q_2 \not\equiv \mathbf{0}$". Suppose now that $\mathcal{D}_k \vdash a.a^n = a^{n+1}$, and consider the shortest proof of $\mathcal{D}_k \vdash P = Q$ for some processes P, Q such that either $\theta(P, Q)$ or $\theta(Q, P)$. Since $\theta(a.a^n, a^{n+1})$ holds, such a minimal proof does exist. We reason about the last rule used in the derivation of this proof in equational logic. For syntactic reasons, this cannot be reflexivity, a contextual rule, nor one of the structural congruence axioms. It can be neither symmetry nor transitivity, since otherwise this would give a shorter proof satisfying θ. The only possibility is thus the use of one of the distribution axioms, say D_i with $1 \leq i \leq k$ and $a^{n+1} \sim Q \equiv (a.Q')^{i+1}$. By Lemma 1.3, since $\#(a^{n+1}) = n+1$, $i+1$ has to divide $n+1$. This is contradictory, because we have $2 \leq i+1 \leq k+1 < n+1$, and $n+1$ is prime. □

Theorem 3.2 (No finite axiomatisation of $\sim$). *For any finite set of axioms $\mathcal{E}$, there exist processes P and Q such that $P \sim Q$ but $\mathcal{E} \nvdash P = Q$.*

Proof. Standard, by proving that $\mathcal{D}$ is ω-complete and then using the Compactness Theorem (see [1]). □

4 A New Congruence Result for the $\boldsymbol{\pi}$-Calculus

4.1 The Finite, Sum-Free $\boldsymbol{\pi}$-Calculus

π-calculus processes are built from an infinite set $\mathcal{N}_\pi$ of names, ranged over using $a, b \ldots, m, n \ldots, p, q \ldots, x, y \ldots$, according to the following grammar:

$$\phi ::= m(x) \bigm| \overline{m}n \ , \qquad\qquad P ::= \mathbf{0} \bigm| \phi.P \bigm| P_1 \,|\, P_2 \bigm| (\boldsymbol{\nu}p)P \ .$$

The input prefix $m(x)$ binds name x in the continuation process, and so does name restriction $(\boldsymbol{\nu}n)$ in the restricted process. A name that is not bound is said to be free, and we let $\mathrm{fn}(P)$ stand for the free names of P. We assume that any process that we manipulate satisfies a *Barendregt convention*: every bound name is distinct from the other bound and free names of the process. We shall use a, b, c to range over free names of processes, p, q, r (resp. x, y) to range over names bound by restriction (resp. by input), and m, n to range over any name, free or bound (note that these naming conventions are used in the above grammar). Structural congruence on π_0, written $\equiv$, is the smallest congruence that is an equivalence relation, contains α-equivalence, and satisfies the following laws:

$$P \,|\, \mathbf{0} \equiv P \qquad P \,|\, (Q \,|\, R) \equiv (P \,|\, Q) \,|\, R \qquad P \,|\, Q \equiv Q \,|\, P \qquad (\boldsymbol{\nu}p)\mathbf{0} \equiv \mathbf{0}$$

$$(\boldsymbol{\nu}p)(\boldsymbol{\nu}q)P \equiv (\boldsymbol{\nu}q)(\boldsymbol{\nu}p)P \qquad P \,|\, (\boldsymbol{\nu}p)Q \equiv (\boldsymbol{\nu}p)(P \,|\, Q) \quad \text{if } p \notin \mathrm{fn}(P)$$

We let $P[n/x]$ stand for the capture avoiding substitution of name x with name n in P. We use σ to range over substitutions in π_0 (that simultaneously replace several names).

Definition 4.1 (Late operational semantics and ground bisimilarity). *The late operational semantics of π_0 is given by a transition relation whose set of labels is defined by:*

$$\mu ::= a(x) \bigm| \overline{a}b \bigm| \overline{a}(p) \bigm| \tau \ .$$

Names x and p are said to be bound in actions $a(x)$ and $\overline{a}(p)$ respectively, and other names are free. We use $\mathrm{bn}(\mu)$ (resp. $\mathrm{fn}(\mu)$) to denote the set of bound (resp. free) names of action μ.

The late transition relation, written $\rightarrow_\pi$, is given by the following rules (symmetrical versions of the rules involving parallel composition are omitted):

$$\frac{}{\phi.P \xrightarrow{\phi}_\pi P} \qquad\qquad \frac{P \xrightarrow{a(x)}_\pi P' \qquad Q \xrightarrow{\overline{a}b}_\pi Q'}{P \,|\, Q \xrightarrow{\tau}_\pi P'[b/x] \,|\, Q'}$$

$$\frac{P \xrightarrow{\overline{a}b}_\pi P'}{(\boldsymbol{\nu}b)P \xrightarrow{\overline{a}(b)}_\pi P'} \ a \neq b \qquad\qquad \frac{P \xrightarrow{a(x)}_\pi P' \qquad Q \xrightarrow{\overline{a}(p)}_\pi Q'}{P \,|\, Q \xrightarrow{\tau}_\pi (\boldsymbol{\nu}p)(P'[p/x] \,|\, Q')}$$

$$\frac{P \xrightarrow{\mu}_\pi P'}{P \,|\, Q \xrightarrow{\mu}_\pi P' \,|\, Q} \ \mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset \qquad\qquad \frac{P \xrightarrow{\mu}_\pi P'}{(\boldsymbol{\nu}p)P \xrightarrow{\mu}_\pi (\boldsymbol{\nu}p)P'} \ p \notin \mathrm{fn}(\mu)$$

A ground bisimulation *is a symmetric relation $\mathcal{R}$ between processes such that whenever $P \mathcal{R} Q$ and $P \xrightarrow{\mu}_\pi P'$, there exists Q' s.t. $Q \xrightarrow{\mu}_\pi Q'$ and $P' \mathcal{R} Q'$.*

Ground bisimilarity, *written $\sim_\mathrm{g}$, is the union of all ground bisimulations.*

Note that we do not respect the convention on names in the rule to infer a bound output, precisely because we are transforming a free name (b) into a bound name.

Lemma 4.2. *Suppose that $P\sigma \xrightarrow{\mu}_\pi P'$.*

1. *If μ is $\overline{a}b$, $\overline{a}(p)$ or $a(x)$, then $P \xrightarrow{\mu'}_\pi P''$ with $\mu'\sigma = \mu$ and $P''\sigma = P'$.*
2. *If $\mu = \tau$ then one of the three following properties hold, where the input and output actions are offered concurrently by P in the last two cases.*
   (a) $P \xrightarrow{\tau}_\pi P''$ and $P''\sigma = P'$,
   (b) $P \xrightarrow{\overline{b}c}_\pi \xrightarrow{a(x)}_\pi P''$ where $\sigma(a) = \sigma(b)$ and $P''[c/x]\sigma \sim P'$,
   (c) $P \xrightarrow{\overline{b}(p)}_\pi \xrightarrow{a(x)}_\pi P''$ where $\sigma(a) = \sigma(b)$ and $((\boldsymbol{\nu} p)P''[p/x])\sigma \sim P'$.

Proof. Similar to the proof of Lemma 1.4.13 in [13], where the early transition semantics is treated. □

4.2 Mutual Desynchronisations

We now introduce the notion of mutual desynchronisation in μCCS, which is defined as the existence of processes obeying certain conditions in the calculus. We shall see that because of τ synchronisations, the absence of mutual desynchronisations is related to substitution closure of $\sim$.

Definition 4.3 (Mutual desynchronisation in μCCS). *We say that there exists a* mutual desynchronisation in μCCS *whenever there are two prefixes η_1, η_2, and five μCCS processes P, P', Q, Q', R such that $\eta_1 \neq \eta_2$, $P \xrightarrow{\eta_1} P'$, $Q \xrightarrow{\eta_2} Q'$ and $\eta_2.P \,|\, Q' \,|\, R \sim P' \,|\, \eta_1.Q \,|\, R$.*

The notion of mutual desynchronisation is not specific to μCCS. As explained in the introduction, it corresponds to a situation where three processes T, T_{12}, T_{21} satisfy:

– $T \xrightarrow{\eta_1}\xrightarrow{\eta_2} T_{12}$ and $T \xrightarrow{\eta_2}\xrightarrow{\eta_1} T_{21}$, where the second prefix being triggered occurs under the first one in both sequences of transitions.
– $\eta_1 \neq \eta_2$ and $T_{12} \sim T_{21}$.

The proofs of Lemmas 4.9 and 4.10 will expose analogous situations in π_0.

Definition 4.4. *We define, for any μCCS process P and prefix η, the* contribution of P at η, *written $s_\eta(P)$, by*

$$s_\eta(\mathbf{0}) \overset{\text{def}}{=} 0 \qquad\qquad s_\eta(\eta'.P) \overset{\text{def}}{=} 0 \quad \text{if } \eta \neq \eta'$$

$$s_\eta(P_1 \,|\, P_2) \overset{\text{def}}{=} s_\eta(P_1) + s_\eta(P_2) \qquad\qquad s_\eta(\eta.P) \overset{\text{def}}{=} \#(\eta.P)$$

Intuitively, $s_\eta(P)$ is the total size of the parallel components of P that start with the prefix η.

Lemma 4.5. *$P \sim Q$ implies $s_\eta(P) = s_\eta(Q)$ for all η.*

Proof. Follows from Theorem 2.6 and the observation that the distribution law preserves the contribution of a process at a given interaction prefix. □

Lemma 4.6 (No mutual desynchronisation). *There exists no mutual desynchronisation in μCCS.*

Proof. Suppose by contradiction that there are processes such that $P \xrightarrow{\eta_1} P'$, $Q \xrightarrow{\eta_2} Q'$ and $\eta_2.P \,|\, Q' \,|\, R \sim P' \,|\, \eta_1.Q \,|\, R$.

By the cancellation property (Corollary 2.3), we have $\eta_2.P \,|\, Q' \sim P' \,|\, \eta_1.Q$, hence for all η, $s_\eta(\eta_2.P \,|\, Q') = s_\eta(P' \,|\, \eta_1.Q)$ (Lemma 4.5).

Since $s_{\eta_1}(\eta_2.P \,|\, Q') = s_{\eta_1}(Q') \leq \#(Q')$ and $s_{\eta_1}(P' \,|\, \eta_1.Q)) \geq s_{\eta_1}(\eta_1.Q) = \#(Q') + 2$, by taking $\eta = \eta_1$ we finally get $\#(Q') \geq \#(Q') + 2$. □

This result will be used to show that a situation corresponding to a mutual desynchronisation cannot arise in π_0. Notice that the proof depends in an essential way on Lemma 4.5, which in turn relies on the axiomatisation of $\sim$ in μCCS (Theorem 2.6).

In what follows, we fix two distinct names a and b, that will occur free in the processes we shall consider. The definitions and results below will depend on a and b, but we avoid making this dependency explicit, in order to ease readability. Names a and b will be fixed in the proof of Lemma 4.11.

Definition 4.7 (Erasing a π_0 process). *Given a π_0 process P, we define the* erasing of P, *written $\mathcal{E}(P)$, as follows:*

$$\mathcal{E}(P_1 \,|\, P_2) \overset{\text{def}}{=} \mathcal{E}(P_1) \,|\, \mathcal{E}(P_2) \qquad \mathcal{E}((\boldsymbol{\nu} p)P) \overset{\text{def}}{=} \mathcal{E}(P) \qquad \mathcal{E}(\mathbf{0}) \overset{\text{def}}{=} \mathbf{0}$$

$$\mathcal{E}(a(x).P) \overset{\text{def}}{=} a.\mathcal{E}(P) \qquad \mathcal{E}(m(x).P) \overset{\text{def}}{=} \mathbf{0} \text{ if } m \neq a$$

$$\mathcal{E}(\overline{b}n.P) \overset{\text{def}}{=} \overline{b}.\mathcal{E}(P) \qquad \mathcal{E}(\overline{m}n.P) \overset{\text{def}}{=} \mathbf{0} \text{ if } m \neq b$$

Note that a and b play different roles in the definition of $\mathcal{E}(\cdot)$.

It is immediate from the definition that $\mathcal{E}(P)$ is a μCCS process whose only prefixes are a and $\overline{b}$. Intuitively, $\mathcal{E}(P)$ only exhibits the interactions of P at a (in input) and b (in output) that are not guarded by interactions on other names.

Lemma 4.8 (Transitions of $\mathcal{E}(P)$). *Consider a π_0 process P. We have:*

- *If $P \xrightarrow{a(x)}_\pi P'$, then $\mathcal{E}(P) \xrightarrow{a} \mathcal{E}(P')$.*
- *If $P \xrightarrow{\overline{b}c}_\pi P'$ or $P \xrightarrow{\overline{b}(p)}_\pi P'$, then $\mathcal{E}(P) \xrightarrow{\overline{b}} \mathcal{E}(P')$.*
- *Conversely, if $\mathcal{E}(P) \xrightarrow{a} P_0$, then there exist x and P' such that $P_0 = \mathcal{E}(P')$ and $P \xrightarrow{a(x)}_\pi P'$. Similarly, if $\mathcal{E}(P) \xrightarrow{\overline{b}} P_0$, there exist c, p, P' such that $P_0 = \mathcal{E}(P')$ and either $P \xrightarrow{\overline{b}c}_\pi P'$ or $P \xrightarrow{\overline{b}(p)}_\pi P'$.*

Proof. Simple reasoning on the LTSs of μCCS and π_0. □

Proposition 4.1 (Transfer). *If $P \sim_{\mathrm{g}} Q$ in π_0, then $\mathcal{E}(P) \sim \mathcal{E}(Q)$ in μCCS.*

Proof. We reason by induction on the size of P (defined as the number of prefixes in P). Consider a transition of $\mathcal{E}(P)$; as observed above, it can only be a transition along a or a transition along $\overline{b}$.

Suppose $\mathcal{E}(P) \xrightarrow{a} P_0$. By Lemma 4.8, $P \xrightarrow{a(x)}_{\pi} P'$ and $P_0 = \mathcal{E}(P')$. Since $P \sim_{\mathrm{g}} Q$, $Q \xrightarrow{a(x)}_{\pi} Q'$ for some Q' such that $P' \sim_{\mathrm{g}} Q'$. By induction, the latter relation gives $\mathcal{E}(P') \sim \mathcal{E}(Q')$, and $Q \xrightarrow{a(x)}_{\pi} Q'$ gives by Lemma 4.8 $\mathcal{E}(Q) \xrightarrow{a} \mathcal{E}(Q')$.

The case $\mathcal{E}(P) \xrightarrow{\overline{b}} P_0$ is treated similarly: by Lemma 4.8, there are two cases, according to whether P does a free output or a bound output. Reasoning like above allows us to conclude in both cases. □

We can now present our central technical result about π_0, which comes in two lemmas.

Lemma 4.9. *If $Q \sim_{\mathrm{g}} (\boldsymbol{\nu}\widetilde{p})(a(x).P_1 \,|\, \overline{b}c.P_2 \,|\, P_3)$, then there exist some Q_1, Q_2, Q_3, $\widetilde{q}$, such that $Q \equiv (\boldsymbol{\nu}\widetilde{q})(a(x).Q_1 \,|\, \overline{b}c.Q_2 \,|\, Q_3)$ and*

$$(\boldsymbol{\nu}\widetilde{p})(P_1 \,|\, P_2 \,|\, P_3) \ \sim_{\mathrm{g}} \ (\boldsymbol{\nu}\widetilde{q})(Q_1 \,|\, Q_2 \,|\, Q_3).$$

Proof. Let $P = (\boldsymbol{\nu}\widetilde{p})(a(x).P_1 \,|\, \overline{b}c.P_2 \,|\, P_3)$ and $P' = (\boldsymbol{\nu}\widetilde{p})(P_1 \,|\, P_2 \,|\, P_3)$.

Note that by our conventions on notations, $c \notin \widetilde{p}$.

Since $Q \sim_{\mathrm{g}} P$ and P can perform two transitions along $a(x)$ and $\overline{b}c$ respectively, Q can also perform these transitions, which gives

$Q \equiv (\boldsymbol{\nu}\widetilde{q})(a(x).Q_1 \,|\, \overline{b}c.Q_2 \,|\, Q_3)$ for some $\widetilde{q}, Q_1, Q_2, Q_3$,

the first (resp. second) component exhibiting the prefix that is triggered to answer the challenge on $a(x)$ (resp. $\overline{b}c$).

Consider now the challenge $P \xrightarrow{\overline{b}c}_{\pi}\xrightarrow{a(x)}_{\pi} P'$, to which Q answers by performing $Q \xrightarrow{\overline{b}c}_{\pi}\xrightarrow{a(x)}_{\pi} Q_{ba}$, with $P' \sim_{\mathrm{g}} Q_{ba}$. If $Q_{ba} = (\boldsymbol{\nu}\widetilde{q})(Q_1 \,|\, Q_2 \,|\, Q_3)$, that is, if Q triggers the prefixes on top of its first and second components, then we are done. Similarly, if Q triggers a prefix in Q_3 to answer the second challenge, say $Q_3 = a(x).Q_4 \,|\, Q_5$, we can set $Q_1' = a(x).Q_4$ and $Q_3' = Q_1 \,|\, Q_5$, and the lemma is proved.

The case that remains to be analysed is when $Q_2 \xrightarrow{a(x)}_{\pi} Q_2'$ and $Q_{ba} = (\boldsymbol{\nu}\widetilde{q})(a(x).Q_1 \,|\, Q_2' \,|\, Q_3) \sim_{\mathrm{g}} (\boldsymbol{\nu}\widetilde{p})(P_1 \,|\, P_2 \,|\, P_3)$.

We then consider the challenge where P fires its two topmost prefixes $a(x)$ and $\overline{b}c$ in the other sequence, namely $P \xrightarrow{a(x)}_{\pi}\xrightarrow{\overline{b}c}_{\pi} P'$. By hypothesis, Q triggers the prefix of its first component for the first transition. To perform the second transition, Q can fire the prefix $\overline{b}c$ either in its second or third component, in which case, as above, we are done, or, and this is the last possibility, the prefix $\overline{b}c$ occurs in Q_1. This means $Q_{ab} = (\boldsymbol{\nu}\widetilde{q})(Q_1' \,|\, \overline{b}c.Q_2 \,|\, Q_3) \sim_{\mathrm{g}} (\boldsymbol{\nu}\widetilde{p})(P_1 \,|\, P_2 \,|\, P_3)$, with $Q_1 \xrightarrow{\overline{b}c}_{\pi} Q_1'$.

To sum up, we have $Q_{ab} = (\boldsymbol{\nu}\widetilde{q})(Q'_1 \mid \overline{b}c.Q_2 \mid Q_3) \sim_{\mathrm{g}} (\boldsymbol{\nu}\widetilde{q})(a(x).Q_1 \mid Q'_2 \mid Q_3) = Q_{ba}$, with $Q_1 \xrightarrow{\overline{b}c}_\pi Q'_1$ and $Q_2 \xrightarrow{a(x)}_\pi Q'_2$: this resembles the mutual desynchronisation of Definition 4.3, translated into the π-calculus.

Indeed, we can construct a mutual desynchronisation in μCCS: $Q_{ab} \sim_{\mathrm{g}} Q_{ba}$ implies $\mathcal{E}(Q_{ab}) \sim \mathcal{E}(Q_{ba})$ by Prop. 4.1, and $Q_1 \xrightarrow{\overline{b}c}_\pi Q'_1$ (resp. $Q_2 \xrightarrow{a(x)}_\pi Q'_2$) implies by Lemma 4.8 $\mathcal{E}(Q_1) \xrightarrow{\overline{b}} \mathcal{E}(Q'_1)$ (resp. $\mathcal{E}(Q_2) \xrightarrow{a} \mathcal{E}(Q'_2)$). Finally, using Lemma 4.6, we obtain a contradiction, which concludes our proof. □

Lemma 4.10. *If $Q \sim_{\mathrm{g}} (\boldsymbol{\nu}p, \widetilde{p})(a(x).P_1 \mid \overline{b}p.P_2 \mid P_3)$, then there exist some Q_1, Q_2, Q_3, such that $Q \equiv (\boldsymbol{\nu}p, \widetilde{q})(a(x).Q_1 \mid \overline{b}p.Q_2 \mid Q_3)$ and*

$$(\boldsymbol{\nu}\widetilde{p})(P_1 \mid P_2 \mid P_3) \;\sim_{\mathrm{g}}\; (\boldsymbol{\nu}\widetilde{q})(Q_1 \mid Q_2 \mid Q_3).$$

Proof (Hint). The proof follows the same lines as for the previous lemma. The only difference is when analysing the transitions that lead to Q_{ab}: to perform the second transition, Q can either extrude the name called p in the equality $Q \equiv (\boldsymbol{\nu}p, \widetilde{q})(a(x).Q_1 \mid \overline{b}p.Q_2 \mid Q_3)$, or otherwise Q can be α-converted in order to extrude another name. In the case where Q chooses to extrude a different name, we can suppose without loss of generality that the necessary α-conversion is a swapping between name p and a name $q_1 \in \widetilde{q}$, which brings us back to the case where name p is the one being extruded.

The presence of a bound output introduces some notational complications when expressing Q_{ab}, but basically it does not affect the proof w.r.t. the proof of Lemma 4.9, because the function $\mathcal{E}(\cdot)$ is not sensitive to name permutations that do not involve a or b. □

4.3 Congruence

Theorem 4.11 (Closure of $\sim_{\mathrm{g}}$ under substitution). *If $P \sim_{\mathrm{g}} Q$ then for any substitution σ, $P\sigma \sim_{\mathrm{g}} Q\sigma$.*

Proof. We prove that the relation $\mathcal{R} \overset{\mathrm{def}}{=} \{(P\sigma, Q\sigma) \mid P \sim_{\mathrm{g}} Q\}$ is a ground bisimulation. We consider P, Q such that $P \sim_{\mathrm{g}} Q$ and suppose $P\sigma \xrightarrow{\mu}_\pi P_0$. We examine the transitions of P that make it possible for $P\sigma$ to do a μ-transition to P_0.

According to Lemma 4.2, there are two possibilities. The first possibility corresponds to the situation where μ comes from an action that P can perform, i.e., $P \xrightarrow{\mu'}_\pi P'$ for some μ', with $P'\sigma = P_0$ and $\mu'\sigma = \mu$ (cases 1 and 2a in Lemma 4.2). Since $P \sim_{\mathrm{g}} Q$, $Q \xrightarrow{\mu'}_\pi Q'$ and $P' \sim_{\mathrm{g}} Q'$ for some Q'. We can prove that $Q\sigma \xrightarrow{\mu} Q'\sigma$, and since $P' \sim_{\mathrm{g}} Q'$ we have $(P'\sigma, Q'\sigma) \in \mathcal{R}$.

The second possibility (which corresponds to the difficult case) is given by $\mu = \tau$, where the synchronisation in P' has been made possible by the application of σ. There are in turn two cases, corresponding to whether the synchronisation involves a free or a bound name. In the former case, $P \xrightarrow{a(x)}_\pi P'$ and $P \xrightarrow{\overline{b}c}_\pi P''$ for some a, x, b, c, P', P''. This entails $P \equiv (\boldsymbol{\nu}\widetilde{p})(a(x).P_1 \mid \overline{b}c.P_2 \mid P_3)$

for some $\widetilde{p}, P_1, P_2, P_3$, and, since $P \sim_{\mathrm{g}} Q$, we conclude by Lemma 4.9 that $Q \equiv (\boldsymbol{\nu}\widetilde{q})(a(x).Q_1 \,|\, \overline{b}c.Q_2 \,|\, Q_3)$ and

$$(\boldsymbol{\nu}\widetilde{p})(P_1 \,|\, P_2 \,|\, P_3) \;\sim_{\mathrm{g}}\; (\boldsymbol{\nu}\widetilde{q})(Q_1 \,|\, Q_2 \,|\, Q_3) \enspace .$$

By definition of $\mathcal{R}$, this equivalence implies that we can apply any substitution to these two processes to yield processes related by $\mathcal{R}$, and in particular $[c/x]\sigma$, which gives:

$$((\boldsymbol{\nu}\widetilde{p})(P_1 \,|\, P_2 \,|\, P_3))[c/x]\sigma \;\;\mathcal{R}\;\; ((\boldsymbol{\nu}\widetilde{q})(Q_1 \,|\, Q_2 \,|\, Q_3))[c/x]\sigma \enspace .$$

Using the Barendregt convention hypothesis, this amounts to

$$P_0 \equiv ((\boldsymbol{\nu}\widetilde{p})(P_1[c/x] \,|\, P_2 \,|\, P_3))\sigma \;\;\mathcal{R}\;\; ((\boldsymbol{\nu}\widetilde{q})(Q_1[c/x] \,|\, Q_2 \,|\, Q_3))\sigma \overset{\mathrm{def}}{=} Q_0 \enspace .$$

We can then conclude by checking that $Q\sigma \xrightarrow{\tau}_{\pi} Q_0$.

We reason similarly for the case where the synchronisation involves the transmission of a bound name, using Lemma 4.10 instead of Lemma 4.9. We remark that Lemma 4.10 gives $(\boldsymbol{\nu}\widetilde{p})(P_1 \,|\, P_2 \,|\, P_3) \sim_{\mathrm{g}} (\boldsymbol{\nu}\widetilde{q})(Q_1 \,|\, Q_2 \,|\, Q_3)$, and in this case $P\sigma \xrightarrow{\tau}_{\pi} (\boldsymbol{\nu}p,\widetilde{p})(P_1[p/x] \,|\, P_2 \,|\, P_3)\sigma$ (resp. $Q\sigma \xrightarrow{\tau}_{\pi} (\boldsymbol{\nu}p,\widetilde{q})(Q_1[p/x] \,|\, Q_2 \,|\, Q_3)\sigma$). In order to be able to add the restriction on p to the terms given by Lemma 4.10, we rely on the fact that $\sim_{\mathrm{g}}$ is preserved by restriction: $P \sim_{\mathrm{g}} Q$ implies $(\boldsymbol{\nu}p)P \sim_{\mathrm{g}} (\boldsymbol{\nu}p)Q$ for any P, Q, p. We can then reason as above to conclude. $\square$

Corollary 4.12 (Congruence of bisimilarity in π_0). *In π_0, ground, early and late bisimilarity coincide and are congruences.*

Proof. By a standard argument (see [13]): since $\sim_{\mathrm{g}}$ is closed under substitution, $\sim_{\mathrm{g}}$ is an open bisimulation. $\square$

It is known (see [13]) that adding either replication or sum to π_0 yields a calculus where strong bisimilarity fails to be a congruence.

5 Conclusion

We have presented an axiomatisation of strong bisimilarity on a small subcalculus of CCS, and a new congruence result for the π-calculus.

Technically, the notion of mutual desynchronisation is related to substitution closure of strong bisimilarity, as soon as substitutions can create new interactions by identifying two names.

We have shown in Section 4 that there exists no mutual desynchronisation in π_0, and that $\sim_{\mathrm{g}}$ is a congruence. In (full) CCS, mutual desynchronisations exist, a simple example being given by $a.\overline{b}+\overline{b}.a$. The latter process is bisimilar to $a \,|\, \overline{b}$, but the equality fails to hold when b is replaced with a. The same reasoning holds for the π-calculus with choice. It hence appears that in finite calculi, mutual desynchronisations give rise to counterexamples to substitution closure of strong bisimilarity. The situation is less clear when infinite behaviours can be expressed.

For instance, in the extension of μCCS with replication, the process $!a \,|\, !\overline{b}$ is bisimilar to $P \overset{\text{def}}{=} !a.\overline{b} \,|\, !\overline{b}.a$. Process P leads to a mutual desynchronisation: we have $P \xrightarrow{a}\xrightarrow{\overline{b}} \equiv P \xrightarrow{\overline{b}}\xrightarrow{a} \equiv P$. We do not know at present whether $\sim$ is substitution-closed in this extension of μCCS (we may remark that the two aforementioned processes remain bisimilar when b is replaced with a).

Some subcalculi of the π-calculus where strong bisimilarity is a congruence are obtained by restricting the output prefix [13]. In the *asynchronous π-calculus* ($A\pi$), mutual desynchronisations do not appear, basically because the output action is not a prefix. Strong bisimilarity is a congruence on $A\pi$. In the *private π-calculus* ($P\pi$), since only private names are emitted, no substitution generated by a synchronisation can identify two previously distinct names. Hence, although mutual desynchronisations exist in $P\pi$ (due to the presence of the sum operator), strong bisimilarity is not substitution closed, but is a congruence. Indeed, to obtain the latter property, we only need to consider the particular substitutions at work in $P\pi$, which cannot identify two names.

The question of substitution closure can also be raised in the framework of *location sensitive behavioural equivalences* such as distributed bisimilarity (see [4]). Without having a formal proof for this claim, we expect this equivalence to be substitution closed on restriction-free CCS. We believe this should be the case because in absence of restriction, distributed bisimilarity is discriminating enough to analyse the maximum degree of parallelism in processes (in particular, the expansion law is not valid for location sensitive equivalences).

Regarding future extensions of this work, we would like to study whether our approach can be adapted to analyse weak bisimilarity in π_0 (as mentioned in Remark 2.3, strong and weak bisimilarity coincide in μCCS). Another interesting direction, as hinted above, would be to study strong bisimilarity on infinite, restriction-free calculi (in CCS and the π-calculus).

Acknowledgements. We are grateful to Arnaud Carayol for interesting discussions at early stages of this work. An anonymous referee provided numerous helpful suggestions, which helped us in particular to improve the proof of Theorem 2.6. We benefited from support by the french initiative "ACI GEOCAL" and from the ANR project "MoDyFiable".

References

1. L. Aceto, W.J. Fokkink, A. Ingolfsdottir, and B. Luttik. Finite Equational Bases in Process Algebra: Results and Open Questions. In *Processes, Terms and Cycles: Steps on the Road to Infinity*, volume 3838 of *LNCS*. Springer Verlag, 2005.
2. M. Boreale and D. Sangiorgi. Some Congruence Properties for π-calculus Bisimilarities. *TCS*, 198:159–176, 1998.
3. O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification over Infinite States. In *Handbook of Process Algebra*, pages 545–623. Elsevier, 2001.
4. I. Castellani. *Handbook of Process Algebra*, chapter Process Algebras with Localities, pages 945–1045. North-Holland, 2001.

5. F. Corradini, R. Gorrieri, and D. Marchignoli. Towards parallelization of concurrent systems. *Informatique Théorique et Applications*, 32(4-6):99–125, 1998.
6. W. Fokkink and B. Luttik. An ω-complete Equational Specification of Interleaving. In *Proc. of ICALP'00*, volume 1853 of *LNCS*, pages 729–743. Springer Verlag, 2000.
7. Y. Hirshfeld and M. Jerrum. Bisimulation Equivalence is Decidable for Normed Process Algebra. Technical Report ECS-LFCS-98-386, LFCS, 1998.
8. Y. Hirshfeld and M. Jerrum. Bisimulation Equivalence is Decidable for Normed Process Algebra. In *Proc. of ICALP'99*, volume 1644 of *LNCS*, pages 412–421. Springer Verlag, 1999.
9. B. Luttik. What is Algebraic in Process Theory? *Concurrency Column, Bulletin of the EATCS*, 88, 2006.
10. R. Milner and F. Moller. Unique Decomposition of Processes. *TCS*, 107(2): 357–363, 1993.
11. F. Moller. *Axioms for Concurrency.* PhD thesis, University of Edinburgh, 1988.
12. D. Sangiorgi. A Theory of Bisimulation for the π-Calculus. *Acta Informatica*, 33(1):69–97, 1996.
13. D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes.* Cambridge University Press, 2001.
14. B. Victor, F. Moller, M. Dam, and L.-H. Eriksson. The Mobility Workbench. available from `http://www.it.uu.se/research/group/mobility/mwb`, 2006.

On the Expressiveness and Complexity of ATL[†]

François Laroussinie, Nicolas Markey, and Ghassan Oreiby[⋆]

LSV, CNRS & ENS Cachan, France
`{fl,markey,oreiby}@lsv.ens-cachan.fr`

Abstract. ATL is a temporal logic geared towards the specification and verification of properties in multi-agents systems. It allows to reason on the existence of strategies for coalitions of agents in order to enforce a given property. We prove that the standard definition of ATL (built on modalities "Next", "Always" and "Until") has to be completed in order to express the duals of its modalities: it is necessary to add the modality "Release". We then precisely characterize the complexity of ATL model-checking when the number of agents is not fixed. We prove that it is $\mathbf{\Delta}_2^{\mathsf{P}}$- and $\mathbf{\Delta}_3^{\mathsf{P}}$-complete, depending on the underlying multi-agent model (ATS and CGS resp.). We also prove that ATL^+ model-checking is $\mathbf{\Delta}_3^{\mathsf{P}}$-complete over both models, even with a fixed number of agents.

1 Introduction

Model checking. Temporal logics were proposed for the specification of reactive systems almost thirty years ago [16]. They have been widely studied and successfully used in many situations, especially for model checking —the automatic verification that a finite-state model of a system satisfies a temporal logic specification. Two flavors of temporal logics have mainly been studied: *linear-time temporal logics*, *e.g.* LTL [16], which expresses properties on the possible *executions* of the model; and *branching-time temporal logics*, such as CTL [7,17], which can express requirements on *states* (which may have several possible futures) of the model.

Alternating-time temporal logic. Over the last ten years, a new flavor of temporal logics has been defined: *alternating-time temporal logics*, *e.g.* ATL [2,3]. ATL is a fundamental logic for verifying properties in *synchronous multi-agent systems*, in which several agents can concurrently influence the behavior of the system. This is particularly interesting for modeling control problems. In that setting, it is not only interesting to know if something *can arrive* or *will arrive*, as can be expressed in CTL or LTL, but rather if some agent(s) can *control* the evolution of the system in order to enforce a given property.

The logic ATL can precisely express this kind of properties, and can for instance state that "there is a strategy for a coalition A of agents in order to

[†] Work partially supported by ACI "Sécurité & Informatique" CORTOS, a program of the French Ministry of Research.

[⋆] This author is supported by a PhD grant from Région Ile-de-France.

eventually reach an accepting state, whatever the other agents do". ATL is an extension of CTL, its formulae are built on atomic propositions and boolean combinators, and (following the seminal papers [2,3]) on modalities $\langle\langle A\rangle\rangle\,\mathbf{X}\,\varphi$ (coalition A has a strategy to immediately enter a state satisfying φ), $\langle\langle A\rangle\rangle\,\mathbf{G}\,\varphi$ (coalition A can force the system to always satisfy φ) and $\langle\langle A\rangle\rangle\,\varphi\,\mathbf{U}\,\psi$ (coalition A has a strategy to enforce $\varphi\,\mathbf{U}\,\psi$).

Multi-agent models. While linear- and branching-time temporal logics are interpreted on Kripke structure, alternating-time temporal logics are interpreted on models that incorporate the notion of *multiple agents*. Two kinds of synchronous multi-agent models have been proposed for ATL in the literature. First *Alternating Transition Systems* (ATSs) [2] have been defined: in any location of an ATS, each agent chooses one *move*, *i.e.*, a subset of locations (the list of possible moves is defined explicitly in the model) in which he would like the execution to go to. When all the agents have made their choice, the intersection of their choices is required to contain one single location, in which the execution enters. In the second family of models, called *Concurrent Game Structures* (CGSs) [3], each of the n agents has a finite number of possible moves (numbered with integers), and, in each location, an n-ary transition function indicates the state to which the execution goes.

Our contributions. While in LTL and CTL, the dual of "Until" modality can be expressed as a disjunction of "Always" and "Until", we prove that it is not the case in ATL. In other words, ATL, as defined in [2,3], is not as expressive as one could expect (while it is known that adding the dual of "Until" does not increase the complexity of the verification problems [5,9]).

We also precisely characterize the complexity of the model checking problem. The original works about ATL provide model-checking algorithms in time $O(m \cdot l)$, where m is the number of transitions in the model, and l is the size of the formula [2,3], thus in PTIME. However, contrary to Kripke structures, the number of transitions in a CGS or in an ATS is not quadratic in the number of states [3], and might even be exponential in the number of agents. PTIME-completeness thus only holds for ATS when the number of agents is bounded, and it is shown in [11,12] that the problem is strictly[1] harder otherwise, namely NP-hard on ATS and Σ_2^{P}-hard on CGSs where the transition function is encoded as a boolean function. We prove that it is in fact $\mathbf{\Delta}_2^{\mathsf{P}}$-complete and $\mathbf{\Delta}_3^{\mathsf{P}}$-complete, resp., correcting wrong algorithms in [11,12] (the problem lies in the way the algorithms handle negations). We also show that ATL$^+$ is $\mathbf{\Delta}_3^{\mathsf{P}}$-complete on both ATSs and CGSs, even when the number of agents is fixed, extending a result of [18]. Finally we study translations between ATS and CGS.

Related works. In [2,3] ATL has been proposed and defined over ATS and CGS. In [10] expressiveness issues are considered for ATL* and ATL. Complexity of

[1] We adopt the classical hypothesis that the polynomial-time hierarchy does not collapse, and that PTIME $\neq$ NP. We refer to [15] for the definitions about complexity classes, especially about oracle Turing machines and the polynomial-time hierarchy.

satisfiability is addressed in [9,19]. Complexity results about model checking (for ATL, ATL$^+$, ATL*) can be found in [3,18]. Regarding control and game theory, many papers have focused on this wide area; we refer to [20] for a survey, and to its numerous references for a complete overview.

Plan of the paper. Section 2 contains the formal definitions that are used in the sequel. Section 3 explains our expressiveness result, and Section 4 deals with the model-checking algorithms. Due to lack of space, some proofs are omitted in this article, but can be found in [13].

2 Definitions

2.1 Concurrent Game Structures and Alternating Transition Systems

Definition 1. *A* Concurrent Game Structure *(*CGS *for short)* $\mathcal{C}$ *is a* 6*-tuple* $(\mathsf{Agt}, \mathsf{Loc}, \mathsf{AP}, \mathsf{Lab}, \mathsf{Mov}, \mathsf{Edg})$ *s.t:*

– $\mathsf{Agt} = \{A_1, ..., A_k\}$ *is a finite set of* agents *(or* players*);*
– Loc *and* AP *are two finite sets of* locations *and* atomic propositions*, resp.;*
– $\mathsf{Lab}\colon \mathsf{Loc} \to 2^{\mathsf{AP}}$ *is a function labeling each location by the set of atomic propositions that hold for that location;*
– $\mathsf{Mov}\colon \mathsf{Loc} \times \mathsf{Agt} \to \mathcal{P}(\mathbb{N}) \smallsetminus \{\varnothing\}$ *defines the (finite) set of possible moves of each agent in each location.*
– $\mathsf{Edg}\colon \mathsf{Loc} \times \mathbb{N}^k \to \mathsf{Loc}$*, where* $k = |\mathsf{Agt}|$*, is a (partial) function defining the transition table. With each location and each set of moves of the agents, it associates the resulting location.*

The intended behaviour is as follows [3]: in a given location ℓ, each player A_i chooses one possible move m_{A_i} in $\mathsf{Mov}(\ell, A_i)$ and the successor location is given by $\mathsf{Edg}(\ell, m_{A_1}, ..., m_{A_k})$. We write $\mathsf{Next}(\ell)$ for the set of all possible successor locations from ℓ, and $\mathsf{Next}(\ell, A_j, m)$ for the restriction of $\mathsf{Next}(\ell)$ to locations reachable from ℓ when player A_j makes the move m.

In the original works about ATL [2], the logic was interpreted on ATSs, which are transition systems slightly different from CGSs:

Definition 2. *An* Alternating Transition System *(*ATS *for short)* $\mathcal{A}$ *is a* 5*-tuple* $(\mathsf{Agt}, \mathsf{Loc}, \mathsf{AP}, \mathsf{Lab}, \mathsf{Mov})$ *where:*

– Agt*,* Loc*,* AP *and* Lab *have the same meaning as in CGSs;*
– $\mathsf{Mov}\colon \mathsf{Loc} \times \mathsf{Agt} \to \mathcal{P}(\mathcal{P}(\mathsf{Loc}))$ *associate with each location* ℓ *and each agent* a *the set of possible moves, each move being a subset of* Loc*. For each location* ℓ*, it is required that, for any* $Q_i \in \mathsf{Mov}(\ell, A_i)$*,* $\bigcap_{i \leq k} Q_i$ *be a singleton.*

The intuition is as follows: in a location ℓ, once all the agents have chosen their moves (*i.e.*, a subset of locations), the execution goes to the (only) state that belongs to all the sets chosen by the players. Again $\mathsf{Next}(\ell)$ (resp. $\mathsf{Next}(\ell, A_j, m)$) denotes the set of all possible successor locations (resp. the set of possible successor locations when player A_j chooses the move m).

We prove in Section 4.2 that both models have the same expressiveness (w.r.t. alternating bisimilarity [4]).

2.2 Strategy, Outcomes of a Strategy

Let $\mathcal{S}$ be a CGS or an ATS. A *computation* of $\mathcal{S}$ is an infinite sequence $\rho = \ell_0\ell_1\cdots$ of locations such that for any i, $\ell_{i+1} \in \mathsf{Next}(\ell_i)$. We can use the standard notions of suffix and prefix for these computations; $\rho[i]$ denotes the i-th location ℓ_i. A *strategy* for a player $A_i \in \mathsf{Agt}$ is a function f_{A_i} that maps any finite prefix of a computation to a possible move for A_i[2]. A strategy is *state-based* (or *memoryless*) if it only depends on the current state (*i.e.*, $f_{A_i}(\ell_0\cdots\ell_m) = f_{A_i}(\ell_m)$).

A strategy induces a set of computations from ℓ —called the *outcomes* of f_{A_i} from ℓ and denoted[3] $\mathsf{Out}_{\mathcal{S}}(\ell, f_{A_i})$— that player A_i can enforce: $\ell_0\ell_1\ell_2\cdots \in \mathsf{Out}_{\mathcal{S}}(\ell, f_{A_i})$ iff $\ell = \ell_0$ and for any i we have $\ell_{i+1} \in \mathsf{Next}(\ell_i, A_i, f_{A_i}(\ell_0\cdots\ell_i))$. Let $A \subseteq \mathsf{Agt}$ be a coalition. A strategy for A is a tuple F_A containing one strategy for every player in A: $F_A = \{f_{A_i} | A_i \in A\}$. The outcomes of F_A from a location ℓ contains the computations enforced by the strategies in F_A: $\ell_0\ell_1\cdots \in \mathsf{Out}_{\mathcal{S}}(\ell, F_A)$ s.t. $\ell = \ell_0$ and for any i, $\ell_{i+1} \in \bigcap_{A_i\in A} \mathsf{Next}(\ell_i, A_i, f_{A_i}(\ell_0\cdots\ell_i))$. The set of strategies for A is denoted[3] $\mathsf{Strat}_{\mathcal{S}}(A)$. Finally $\mathsf{Out}_{\mathcal{S}}(\ell, \emptyset)$ represents the set of all computations from ℓ.

2.3 The Logic ATL and Some Extensions

Again, we follow the definitions of [3]:

Definition 3. *The syntax of* ATL *is defined by the following grammar:*

$$\mathsf{ATL} \ni \varphi_s, \psi_s ::= \top \mid p \mid \neg\varphi_s \mid \varphi_s \vee \psi_s \mid \langle\!\langle A \rangle\!\rangle\, \varphi_p$$
$$\varphi_p ::= \mathbf{X}\,\varphi_s \mid \mathbf{G}\,\varphi_s \mid \varphi_s\,\mathbf{U}\,\psi_s.$$

where p ranges over the set AP *and A over the subsets of* Agt.

In addition, we use standard abbreviations like $\bot$, $\mathbf{F}$, etc. ATL formulae are interpreted over states of a game structure $\mathcal{S}$. The semantics of the main modalities is defined as follows[3]:

$$\begin{array}{rcl}
\ell \models_{\mathcal{S}} \langle\!\langle A \rangle\!\rangle\,\varphi_p & \text{iff} & \exists F_A \in \mathsf{Strat}(A).\ \forall \rho \in \mathsf{Out}(\ell, F_A).\ \rho \models_{\mathcal{S}} \varphi_p, \\
\rho \models_{\mathcal{S}} \mathbf{X}\,\varphi_s & \text{iff} & \rho[1] \models_{\mathcal{S}} \varphi_s, \\
\rho \models_{\mathcal{S}} \mathbf{G}\,\varphi_s & \text{iff} & \forall i.\ \rho[i] \models_{\mathcal{S}} \varphi_s, \\
\rho \models_{\mathcal{S}} \varphi_s\,\mathbf{U}\,\psi_s & \text{iff} & \exists i.\ \rho[i] \models_{\mathcal{S}} \psi_s \text{ and } \forall 0 \le j < i.\ \rho[j] \models_{\mathcal{S}} \varphi_s.
\end{array}$$

It is well-known that, for the logic ATL, it is sufficient to restrict to state-based strategies (*i.e.*, $\langle\!\langle A \rangle\!\rangle\,\varphi_p$ is satisfied iff there is a state-based strategy all of whose outcomes satisfy φ_p) [3,18].

Note that $\langle\!\langle \emptyset \rangle\!\rangle\,\varphi_p$ corresponds to the CTL formula $\mathbf{A}\varphi_p$ (*i.e.*, universal quantification over all computations issued from the current state), while $\langle\!\langle \mathsf{Agt} \rangle\!\rangle\,\varphi_p$

[2] *I.e.*, $f_{A_i}(\ell_0\cdots\ell_m) \in \mathsf{Mov}(\ell_m, A_i)$.

[3] We might omit to mention $\mathcal{S}$ when it is clear from the context.

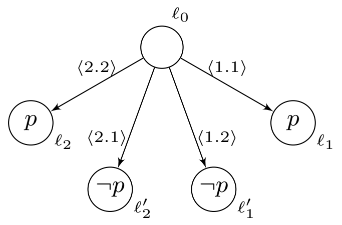


Fig. 1. A CGS that is not determined

$$\mathsf{Loc} = \{\ell_0, \ell_1, \ell_2, \ell'_1, \ell'_2\}$$

$$\mathsf{Mov}(\ell_0, A_1) = \{\{\ell_1, \ell'_1\}, \{\ell_2, \ell'_2\}\}$$
$$\mathsf{Mov}(\ell_0, A_2) = \{\{\ell_1, \ell'_2\}, \{\ell_2, \ell'_1\}\}$$

$$\text{with} \begin{cases} \mathsf{Lab}(\ell_1) = \mathsf{Lab}(\ell_2) = \{p\} \\ \mathsf{Lab}(\ell'_1) = \mathsf{Lab}(\ell'_2) = \varnothing \end{cases}$$

Fig. 2. An ATS that is not determined

corresponds to existential quantification $\mathbf{E}\varphi_p$. Note, however, that $\neg \langle\!\langle A \rangle\!\rangle \varphi_p$ is generally *not* equivalent to $\langle\!\langle \mathsf{Agt} \smallsetminus A \rangle\!\rangle \neg\varphi_p$ [3,9]. Fig. 1 displays a (graphical representation of a) 2-player CGS for which, in ℓ_0, both $\neg \langle\!\langle A_1 \rangle\!\rangle \mathbf{X}\, p$ and $\neg \langle\!\langle A_2 \rangle\!\rangle \neg\mathbf{X}\, p$ hold. In such a representation, a transition is labeled with $\langle m_1.m_2\rangle$ when it correspond to move m_1 of player A_1 and to move m_2 of player A_2. Fig. 2 represents an (alternating-bisimilar) ATS with the same properties.

Duality is a fundamental concept in modal and temporal logics: for instance, the dual of modality $\mathbf{U}$, often denoted by $\mathbf{R}$ and read *release*, is defined by $\varphi_s \,\mathbf{R}\, \psi_s \stackrel{\text{def}}{\equiv} \neg((\neg\varphi_s)\,\mathbf{U}\,(\neg\psi_s))$. Dual modalities allow, for instance, to put negations inner inside the formula, which is often an important property when manipulating formulas. In LTL, modality $\mathbf{R}$ can be expressed using only $\mathbf{U}$ and $\mathbf{G}$:

$$\varphi \,\mathbf{R}\, \psi \equiv \mathbf{G}\, \psi \vee \psi \,\mathbf{U}\, (\varphi \wedge \psi). \tag{1}$$

In the same way, it is well known that CTL can be defined using only modalities $\mathbf{EX}$, $\mathbf{EG}$ and $\mathbf{EU}$, and that we have

$$\mathbf{E}\varphi \,\mathbf{R}\, \psi \equiv \mathbf{EG}\, \psi \vee \mathbf{E}\psi \,\mathbf{U}\, (\varphi \wedge \psi) \qquad \mathbf{A}\varphi \,\mathbf{R}\, \psi \equiv \neg\, \mathbf{E}(\neg\varphi)\,\mathbf{U}\,(\neg\psi).$$

We prove in the sequel that modality $\mathbf{R}$ cannot be expressed in ATL, as defined in Definition 3. We thus define the following two extensions of ATL:

Definition 4. *We define $\mathsf{ATL}_{\mathbf{R}}$ and ATL^+ with the following syntax:*

$$\begin{aligned} \mathsf{ATL}_{\mathbf{R}} \ni \varphi_s, \psi_s &::= \top \mid p \mid \neg\varphi_s \mid \varphi_s \vee \psi_s \mid \langle\!\langle A \rangle\!\rangle \varphi_p \\ \varphi_p &::= \mathbf{X}\, \varphi_s \mid \varphi_s \,\mathbf{U}\, \psi_s \mid \varphi_s \,\mathbf{R}\, \psi_s, \end{aligned}$$

$$\begin{aligned} \mathsf{ATL}^+ \ni \varphi_s, \psi_s &::= \top \mid p \mid \neg\varphi_s \mid \varphi_s \vee \psi_s \mid \langle\!\langle A \rangle\!\rangle \varphi_p \\ \varphi_p, \psi_p &::= \neg\varphi_p \mid \varphi_p \vee \psi_p \mid \mathbf{X}\, \varphi_s \mid \varphi_s \,\mathbf{U}\, \psi_s \mid \varphi_s \,\mathbf{R}\, \psi_s. \end{aligned}$$

where p ranges over the set AP and A over the subsets of Agt.

Given a formula φ in one of the logics we have defined, the size of φ, denoted by $|\varphi|$, is the size of the tree representing that formula. The DAG-size of φ is the size of the directed acyclic graph representing that formula (*i.e.*, sharing common subformulas).

3 $\langle\!\langle A\rangle\!\rangle\,(a\,\mathrm{R}\,b)$ Cannot Be Expressed in ATL

This section is devoted to the expressiveness of ATL. We prove:

Theorem 5. *There is no ATL formula equivalent to $\Phi = \langle\!\langle A\rangle\!\rangle\,(a\,\mathbf{R}\,b)$.*

The proof of Theorem 5 is based on techniques similar to those used for proving expressiveness results for temporal logics like CTL or ECTL [8]: we build two families of models $(s_i)_{i\in\mathbb{N}}$ and $(s'_i)_{i\in\mathbb{N}}$ s.t. (1) $s_i \not\models \Phi$, (2) $s'_i \models \Phi$ for any i, and (3) s_i and s'_i satisfy the same ATL formula of size less than i. Theorem 5 is a direct consequence of the existence of such families of models. In order to simplify the presentation, the theorem is proved for formula[4] $\Phi = \langle\!\langle A\rangle\!\rangle\,(b\,\mathbf{R}\,(a \vee b))$.

The models are described by one single inductive CGS [5] $\mathcal{C}$, involving only two players. It is depicted on Fig. 3. A label $\langle\alpha.\beta\rangle$ on a transition indicates that

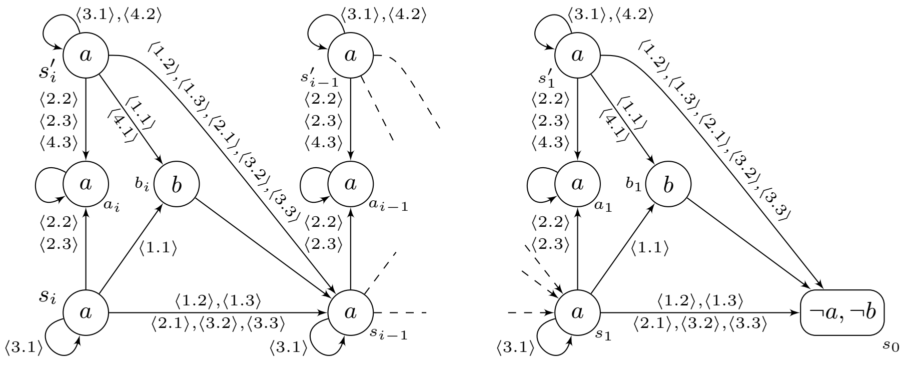


Fig. 3. The CGS $\mathcal{C}$, with states s_i and s'_i on the left

this transition corresponds to move α of player A_1 and to move β of player A_2. In that CGS, states s_i and s'_i only differ in that player A_1 has a fourth possible move in s'_i. This ensures that, from state s'_i (for any i), player A_1 has a strategy (namely, he should always play 4) for enforcing $a\,\mathbf{W}\,b$. But this is not the case from state s_i: by induction on i, one can prove $s_i \not\models \langle\!\langle A_1\rangle\!\rangle\,a\,\mathbf{W}\,b$. The base case is trivial. Now assume the property holds for i: from s_{i+1}, any strategy for A_1 starts with a move in $\{1,2,3\}$ and for any of these choices, player A_2 can choose a move (2, 1, and 2, resp.) that enforces the next state to be s_i where by i.h., A_1 has no strategy for $a\,\mathbf{W}\,b$.

We now prove that s_i and s'_i satisfy the same "small" formulae. First, we have the following equivalences:

Lemma 6. *For any $i > 0$, for any $\psi \in$ ATL with $|\psi| \le i$:*

$$b_i \models \psi \text{ iff } b_{i+1} \models \psi \qquad s_i \models \psi \text{ iff } s_{i+1} \models \psi \qquad s'_i \models \psi \text{ iff } s'_{i+1} \models \psi$$

[4] This formula can also be written $\langle\!\langle A\rangle\!\rangle\,a\,\mathbf{W}\,b$, where $\mathbf{W}$ is the "weak until" modality.
[5] Given the translation from CGS to ATS (see Sec. 4.2), the result also holds for ATSs.

Lemma 7. *$\forall i > 0$, $\forall \psi \in$ ATL with $|\psi| \leq i$: $s_i \models \psi$ iff $s'_i \models \psi$.*

Proof. The proof proceeds by induction on i, and on the structure of the formula ψ. The case $i = 1$ is trivial, since s_1 and s'_1 carry the same atomic propositions. For the induction step, dealing with CTL modalities ($\langle\!\langle\emptyset\rangle\!\rangle$ and $\langle\!\langle A_1, A_2\rangle\!\rangle$) is also straightforward, then we just consider $\langle\!\langle A_1\rangle\!\rangle$ and $\langle\!\langle A_2\rangle\!\rangle$ modalities.

First we consider $\langle\!\langle A_1\rangle\!\rangle$ modalities. It is well-known that we can restrict to state-based strategies in this setting. If player A_1 has a strategy in s_i to enforce something, then he can follow the same strategy from s'_i. Conversely, if player A_1 has a strategy in s'_i to enforce some property, two cases may arise: either the strategy consists in playing move 1, 2 or 3, and it can be mimicked from s_i. Or the strategy consists in playing move 4 and we distinguish three cases:

– $\psi = \langle\!\langle A_1\rangle\!\rangle \mathbf{X}\, \psi_1$: that move 4 is a winning strategy entails that s'_i, a_i and b_i must satisfy ψ_1. Then s_i (by i.h. on the formula) and s_{i-1} (by Lemma 6) both satisfy ψ_1. Playing move 1 (or 3) in s_i ensures that the next state will satisfy ψ_1.
– $\psi = \langle\!\langle A_1\rangle\!\rangle \mathbf{G}\, \psi_1$: by playing move 4, the game could end up in s_{i-1} (*via* b_i), and in a_i and s'_i. Thus $s_{i-1} \models \psi$, and in particular ψ_1. By i.h., $s_i \models \psi_1$, and playing move 1 (or 3) in s_i, and then mimicking the original strategy (from s'_i), enforces $\mathbf{G}\, \psi_1$.
– $\psi = \langle\!\langle A_1\rangle\!\rangle \psi_1 \mathbf{U}\, \psi_2$: a strategy starting with move 4 implies $s'_i \models \psi_2$ (the game could stay in s'_i for ever). Then $s_i \models \psi_2$ by i.h., and the result follows.

We now turn to $\langle\!\langle A_2\rangle\!\rangle$ modalities: clearly if $\langle\!\langle A_2\rangle\!\rangle \psi_1$ holds in s'_i, it also holds in s_i. Conversely, if player A_2 has a (state-based) strategy to enforce some property in s_i: If it consists in playing either 1 or 3, then the same strategy also works in s'_i. Now if the strategy starts with move 2, then playing move 3 in s'_i has the same effect, and thus enforces the same property. □

Remark 1. ATL and $\mathsf{ATL}_{\mathbf{R}}$ have the same distinguishing power as the fragment of ATL involving only the $\langle\!\langle \cdot \rangle\!\rangle \mathbf{X}$ modality (see proof of Theorem 6 in [4]). This means that we cannot exhibit two models M and M' s.t. (1) $M \models \Phi$, (2) $M' \not\models \Phi$, and (3) M and M' satisfy the same ATL formula.

Though ATL^+ would not contain the "release" modality in its syntax, it can express it, and is thus strictly more expressive than ATL. However, as for CTL and CTL^+, it is possible to translate ATL^+ into $\mathsf{ATL}_{\mathbf{R}}$ [10]. Of course, such a translation induces at least an exponential blow-up in the size of the formulae since it is already the case when translating CTL^+ into CTL [21,1]. Finally note that the standard model-checking algorithm for ATL easily extends to $\mathsf{ATL}_{\mathbf{R}}$ (and that MOCHA [5] handles $\mathsf{ATL}_{\mathbf{R}}$ formulae). In the same way, the axiomatization and satisfiability results of [9] can be extended to $\mathsf{ATL}_{\mathbf{R}}$ (as mentioned in the conclusion of [9]).

Turn-based games. In [3], a restriction of CGS —the turn-based considered. In any location of these models (named TB-CGS hereafter), only one player has several moves (the other players have only one possible choice). Such models have the property of *determinedness* (if the objectives are Borel-definable, which is the case for ATL^+): given a set of players A, either there is a strategy for A to win some objective Φ, or there is a strategy for other players $(\mathsf{Agt}\backslash A)$ to enforce $\neg\Phi$. In such systems, modality $\mathbf{R}$ can be expressed as follows: $\langle\!\langle A\rangle\!\rangle\, \varphi\, \mathbf{R}\, \psi \equiv_{\text{TB-CGS}} \neg\, \langle\!\langle \mathsf{Agt}\backslash A\rangle\!\rangle\, (\neg\varphi)\, \mathbf{U}\, (\neg\psi)$.

4 Complexity of ATL Model-Checking

In this section, we establish the precise complexity of ATL model-checking. All the complexity results below are stated for ATL but they are also true for $\mathsf{ATL}_{\mathbf{R}}$.

Model-checking issues have been addressed in the seminal papers about ATL, on both ATSs [2] and CGSs [3]. The time complexity is shown to be in $O(m \cdot l)$, where m is the size of the transition table and l is the size of the formula. The authors then claim that the model-checking problem is in PTIME (and obviously, PTIME-complete). However, it is well-known (and already explained in [2,3]) that the size m of the transition table may be exponential in the number of agents. Thus, when the transition table is not given explicitly (as is the case for ATS), the algorithm requires in fact exponential time.

Before proving that this problem is indeed not in PTIME, we define the model of *implicit* CGSs, with a succinct representation of the transition table [11]. Besides the theoretical aspect, it may be quite useful in practice since it allows to not explicitly describe the full transition table.

4.1 Explicit and Implicit CGSs

We distinguish between two classes of CGSs:

Definition 8. • *An* implicit CGS *is a CGS where, in each location ℓ, the transition function is defined by a finite sequence $((\varphi_0, \ell_0), ..., (\varphi_n, \ell_n))$, where $\ell_i \in \mathsf{Loc}$ is a location, and φ_i is a boolean combination of propositions $A_j = c$ that evaluate to true iff agent A_i chooses move c.*

The transition table is then defined as follows: $\mathsf{Edg}(\ell, m_{A_1}, ..., m_{A_k}) = \ell_j$ iff j is the lowest index s.t. φ_j evaluates to true when players A_1 to A_k choose moves m_{A_1} to m_{A_k}. We require that the last boolean formula φ_i be $\top$, so that no agent can enforce a deadlock.

• *An* explicit CGS *is a CGS where the transition table is defined explicitly.*

The size $|\mathcal{C}|$ of a CGS $\mathcal{C}$ is defined as $|\mathsf{Loc}| + |\mathsf{Edg}|$. For explicit CGSs, $|\mathsf{Edg}|$ is the size of the transition table. For implicit CGSs, $|\mathsf{Edg}|$ is the sum $\sum |\varphi|$ used for the definition of Edg.

The size of an ATS is $|\mathsf{Loc}| + |\mathsf{Mov}|$ where $|\mathsf{Mov}|$ is the sum of the number of locations in each possible move of each agent in each location.

4.2 Expressiveness of CGSs and ATSs

We prove in this section that CGSs and ATSs are closely related: they can model the same concurrent games. In order to make this statement formal, we use the following definition, which extends bisimulation to strategies of coalitions:

Definition 9 ([4]). *Let $\mathcal{A}$ and $\mathcal{B}$ be two models of concurrent games (either ATSs or CGSs) over the same set* Agt *of agents. Let $R \subseteq \mathsf{Loc}_{\mathcal{A}} \times \mathsf{Loc}_{\mathcal{B}}$ be a (non-empty) relation between states of $\mathcal{A}$ and states of $\mathcal{B}$. That relation is an* alternating bisimulation *when, for any $(\ell, \ell') \in R$, the following conditions hold:*

- $\mathsf{Lab}_{\mathcal{A}}(\ell) = \mathsf{Lab}_{\mathcal{B}}(\ell')$;
- *for any coalition $A \subseteq \mathsf{Agt}$, we have*

$$\forall m \colon A \to \mathsf{Mov}_{\mathcal{A}}(\ell, A).\ \exists m' \colon A \to \mathsf{Mov}_{\mathcal{B}}(\ell', A).\\ \forall q' \in \mathsf{Next}(\ell', A, m').\ \exists q \in \mathsf{Next}(\ell, A, m).\ (q, q') \in R.$$

- *symmetrically, for any coalition $A \subseteq \mathsf{Agt}$, we have*

$$\forall m' \colon A \to \mathsf{Mov}_{\mathcal{B}}(\ell', A).\ \exists m \colon A \to \mathsf{Mov}_{\mathcal{A}}(\ell, A).\\ \forall q \in \mathsf{Next}(\ell, A, m).\ \exists q' \in \mathsf{Next}(\ell', A, m').\ (q, q') \in R.$$

where $\mathsf{Next}(\ell, A, m)$ is the set of locations that are reachable from ℓ when each player $A_i \in A$ plays $m(A_i)$.

Two models are said to be alternating-bisimilar if there exists an alternating bisimulation involving all of their locations.

It turns out that ATSs and CGSs (both implicit and explicit ones) have the same expressive power w.r.t. this equivalence:

Theorem 10. *1. Any explicit CGS can be translated into an alternating-bisimilar implicit one in linear time; 2. Any implicit CGS can be translated into an alternating-bisimilar explicit one in exponential time; 3. Any explicit CGS can be translated into an alternating-bisimilar ATS in cubic time; 4. Any ATS can be translated into an alternating-bisimilar explicit CGS in exponential time; 5. Any implicit CGS can be translated into an alternating-bisimilar ATS in exponential time; 6. Any ATS can be translated into an alternating-bisimilar implicit CGS in quadratic time;*

Figure 4 summarizes the above results. From our complexity results (and the assumption that the polynomial-time hierarchy does not collapse), the costs of the above translations is optimal.

4.3 Model Checking ATL on Implicit CGSs

Basically, the algorithm for model-checking ATL [2,3] is similar to that for CTL: it consists in recursively computing fixpoints, *e.g.* based on the following equivalence:

$$\langle\!\langle A \rangle\!\rangle\, p \,\mathbf{U}\, q \equiv \mu Z.(q \vee (p \wedge \langle\!\langle A \rangle\!\rangle\, \mathbf{X}\, Z))$$

The difference with CTL is that we have to compute the pre-image of a set of states *for some coalition*.

It has been remarked in [11] that computing the pre-images is not in PTIME anymore when considering implicit CGSs: the algorithm has to non-deterministically guess the moves of players in A in each location, and for each pre-image, to

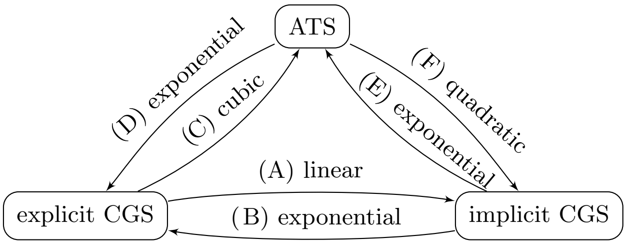


Fig. 4. Costs of translations between the three models

solve the resulting SATqueries derived from those choices and from the transition table. As a consequence, model-checking ATL on implicit CGSs is Σ_2^{P}-hard [11]. However (see below), the Σ_2^{P}-hardness proof can very easily be adapted to prove Π_2^{P}-hardness. It follows that the Σ_2^{P} algorithm proposed in [11] cannot be correct. The flaw is in the way it handles negation: games played on CGSs (and ATSs) are generally not determined, and the fact that a player has no strategy to enforce φ does not imply that the other players have a strategy to enforce $\neg\varphi$. It rather means that the other players have a *co-strategy* to enforce $\neg\varphi$ (see [9] for precise explanations about co-strategies).

Still, the Σ_2^{P}-algorithm is correct for formulas whose main operator is not a negation. As a consequence:

Proposition 11. *Model checking ATL on implicit CGSs is in $\mathbf{\Delta}_3^{\mathsf{P}}$.*

Since the algorithm consists in labeling the locations with the subformulae it satisfies, the above result holds even if we consider the DAG-size of the formula.

Before proving optimality, we briefly recall the Σ_2^{P}-hardness proof of [11]. It relies on the following Σ_2^{P}-complete problem:

EQSAT$_2$:

Input: two families of variables $X = \{x^1, ..., x^n\}$ and $Y = \{y^1, ..., y^n\}$, a boolean formula φ on the set of variables $X \cup Y$.
Output: True iff $\exists X.\ \forall Y.\ \varphi$.

This problem can be encoded in an ATL model-checking problem on an implicit CGS: the CGS has three states q_1, $q_\top$ and $q_\bot$, and $2n$ agents A^1, ..., A^n, B^1, ..., B^n, each having two possible choices in q_1 and only one choice in $q_\top$ and $q_\bot$. The transitions out of $q_\top$ and $q_\bot$ are self loops. The transitions from q_1 are given by: $\delta(q_1) = ((\varphi[x^j \leftarrow (A^j \overset{?}{=} 1), y^j \leftarrow (B^j \overset{?}{=} 1)], q_\top)(\top, q_\bot))$.

Then clearly, the coalition A^1, ..., A^n has a strategy for reaching $q_\top$, *i.e.*, $q_1 \models \langle\!\langle A^1, ..., A^n \rangle\!\rangle\, \mathbf{X}\, q_\top$, iff there exists a valuation for variables in X s.t. φ is true whatever B-agents choose for Y.

Now, this encoding can easily be adapted to the dual (thus Π_2^{P}-complete) problem AQSAT$_2$, in which, with the same input, the output is the value of $\forall X.\ \exists Y.\ \varphi$. It suffices to consider the same implicit CGS, and the formula

$\neg \langle\!\langle A^1, ..., A^n \rangle\!\rangle \, \mathbf{X} \, \neg q_\top$. It states that there is no strategy for players A^1 to A^n to avoid $q_\top$: whatever their choice, players B^1 to B^n can enforce φ.

Following the same idea, we prove the following result:

Proposition 12. *Model checking ATL on implicit CGSs is $\mathbf{\Delta}_3^{\mathsf{P}}$-hard.*

Proof. We consider the following $\mathbf{\Delta}_3^{\mathsf{P}}$-complete problem[14,18].

SNSAT$_2$:

Input: m families of variables $X_i = \{x_i^1, ..., x_i^n\}$, m families of variables $Y_i = \{y_i^1, ..., y_i^n\}$, m variables z_i, m boolean formulae φ_i, with φ_i involving variables in $X_i \cup Y_i \cup \{z_1, ..., z_{i-1}\}$.

Output: The value of z_m, defined by

$$\begin{cases} z_1 \overset{\text{def}}{=} \exists X_1.\ \forall Y_1.\ \varphi_1(X_1, Y_1) \\ z_2 \overset{\text{def}}{=} \exists X_2.\ \forall Y_2.\ \varphi_2(z_1, X_2, Y_2) \\ \qquad \ldots \\ z_m \overset{\text{def}}{=} \exists X_m.\ \forall Y_m.\ \varphi_m(z_1, ..., z_{m-1}, X_m, Y_m) \end{cases}$$

We pick an instance $\mathcal{I}$ of this problem, and reduce it to an instance of the ATL model-checking problem. Note that such an instance uniquely defines the values of variables z_i. We write $v_{\mathcal{I}} \colon \{z_1, ..., z_m\} \to \{\top, \bot\}$ for this valuation. Also, when $v_{\mathcal{I}}(z_i) = \top$, there exists a witnessing valuation for variables in X_i. We extend $v_{\mathcal{I}}$ to $\{z_1, ..., z_m\} \cup \bigcup_i X_i$, with $v_{\mathcal{I}}(x_i^j)$ being a witnessing valuation if $v_{\mathcal{I}}(z_i) = \top$.

We now define an implicit CGS $\mathcal{C}$ as follows: it contains mn agents A_i^j (one for each x_i^j), mn agents B_i^j (one for each y_i^j), m agents C_i (one for each z_i), and one extra agent D. The structure is made of m states q_i, m states $\overline{q_i}$, m states s_i, and two states $q_\top$ and $q_\bot$. There are three atomic propositions: $s_\top$ and $s_\bot$, that label the states $q_\top$ and $q_\bot$ resp., and an atomic proposition s labeling states s_i. The other states carry no label.

Except for D, the agents represent booleans, and thus always have two possible choices (0 and 1). Agent D always has m choices (0 to $m-1$). The transition relation is defined as follows: for each i,

$$\begin{aligned} \delta(\overline{q_i}) &= ((\top, s_i)); \\ \delta(s_i) &= ((\top, q_i)); \\ \delta(q_\top) &= ((\top, q_\top)); \\ \delta(q_\bot) &= ((\top, q_\bot)); \end{aligned} \qquad \delta(q_i) = \begin{pmatrix} ((D \overset{?}{=} 0) \wedge \varphi_i[x_i^j \leftarrow (A_i^j \overset{?}{=} 1), \\ \qquad y_i^j \leftarrow (B_i^j \overset{?}{=} 1), z_k \leftarrow (C_k \overset{?}{=} 1)], q_\top) \\ ((D \overset{?}{=} 0), q_\bot) \\ ((D \overset{?}{=} k) \wedge (C_k \overset{?}{=} 1), q_k) \text{ for each } k < i \\ ((D \overset{?}{=} k) \wedge (C_k \overset{?}{=} 0), \overline{q_k}) \text{ for each } k < i \\ (\top, q_\top) \end{pmatrix}$$

Intuitively, from state q_i, the boolean agents choose a valuation for the variable they represent, and agent D can either choose to check if the valuation really witnesses φ_i (by choosing move 0), or "challenge" player C_k, with move $k < i$.

The ATL formula is built recursively by $\psi_0 = \top$ and, writing AC for the coalition $\{A_1^1, ..., A_m^n, C_1, ..., C_m\}$: $\psi_{k+1} \overset{\text{def}}{=} \langle\!\langle \mathsf{AC} \rangle\!\rangle \, (\neg s) \, \mathbf{U} \, (q_\top \vee \mathbf{EX} \, (s \wedge \mathbf{EX} \, \neg\psi_k))$.

Let $f_{\mathcal{I}}(A)$ be the state-based strategy for agent $A \in \mathsf{AC}$ that consists in playing according to the valuation $v_{\mathcal{I}}$ (*i.e.* move 0 if the variable associated with A evaluates to 0 in $v_{\mathcal{I}}$, and move 1 otherwise). The following lemma completes the proof of Proposition 12:

Lemma 13. *For any $i \leq m$ and $k \geq i$, the following three statements are equivalent: (a) $\mathcal{C}, q_i \models \psi_k$; (b) the strategies $f_{\mathcal{I}}$ witness the fact that $\mathcal{C}, q_i \models \psi_k$; (c) variable z_i evaluates to $\top$ in $v_{\mathcal{I}}$.* □

Finally, Lemma 13 and Proposition 11, this implies:

Theorem 14. *Model checking ATL on implicit CGSs is $\mathbf{\Delta}_3^{\mathsf{P}}$-complete.*

4.4 Model Checking ATL on ATSs

Also for ATSs, the PTIME upper bound holds only when the number of agents is fixed. As in the previous section, the NP algorithm proposed in [11] for ATL model-checking on ATSs does not handle negation correctly. Again, the algorithm involves the fixpoint computation with pre-images, and the pre-images are now computed in NP [11]. This yields a $\mathbf{\Delta}_2^{\mathsf{P}}$algorithm for full ATL.

Proposition 15. *Model checking ATL over ATSs is in $\mathbf{\Delta}_2^{\mathsf{P}}$.*

The NP-hardness proof of [11] can be adapted in order to give a direct reduction of 3SAT, and then extended to SNSAT:

Proposition 16. *Model checking ATL on ATSs is $\mathbf{\Delta}_2^{\mathsf{P}}$-hard.*

Proof. Let us first recall the definition of the SNSAT problem [14]:

<u>SNSAT:</u>

Input: p families of variables $X_r = \{x_r^1, ..., x_r^m\}$, p variables z_r, p boolean formulae φ_r in 3-CNF, with φ_r involving variables in $X_r \cup \{z_1, ..., z_{r-1}\}$.

Output: The value of z_p, defined by

$$\begin{cases} z_1 \overset{\text{def}}{=} \exists X_1.\ \varphi_1(X_1) \\ z_2 \overset{\text{def}}{=} \exists X_2.\ \varphi_2(z_1, X_2) \\ z_3 \overset{\text{def}}{=} \exists X_3.\ \varphi_3(z_1, , z_2, X_3) \\ \qquad \dots \\ z_p \overset{\text{def}}{=} \exists X_p.\ \varphi_p(z_1, ..., z_{p-1}, X_p) \end{cases}$$

Let $\mathcal{I}$ be an instance of SNSAT, where we assume that each φ_r is made of n clauses S_r^1 to S_r^n, with $S_r^j = \alpha_r^{j,1} s_r^{j,1} \vee \alpha_r^{j,2} s_r^{j,2} \vee \alpha_r^{j,3} s_r^{j,3}$. Again, such an instance uniquely defines a valuation $v_{\mathcal{I}}$ for variables z_1 to z_r, that can be extended to the whole set of variables by choosing a witnessing valuation for x_r^1 to x_r^n when z_r evaluates to true.

We now describe the ATS $\mathcal{A}$: it contains $(8n+3)p$ states: p states $\overline{q_r}$ and p states q_r, p states s_r, and for each formula φ_r, for each clause S_r^j of φ_r, eight states $q_r^{j,0}, ..., q_r^{j,7}$, as in the previous reduction.

States s_r are labeled with the atomic proposition s, and states $q_r^{j,k}$ that do not correspond to clause S_r^j are labeled with α.

There is one player A_r^j for each variable x_r^j, one player C_r for each z_r, plus one extra player D. As regards transitions, there are self-loops on each state $q_r^{j,k}$, single transitions from each $\overline{q_r}$ to the corresponding s_r, and from each s_r to the corresponding q_r. From state q_r,

- player A_r^j will choose the value of variable x_r^j, by selecting one of the following two sets of states:

$$\{q_r^{g,k} \mid \forall l \leq 3.\ s_r^{g,l} \neq x_r^j \text{ or } \alpha_r^{g,l} = 0\} \cup \{q_t, \overline{q_t} \mid t < r\} \qquad \text{if } x_r^j = \top$$
$$\{q_r^{g,k} \mid \forall l \leq 3.\ s_r^{g,l} \neq x_r^j \text{ or } \alpha_r^{g,l} = 1\} \cup \{q_t, \overline{q_t} \mid t < r\} \qquad \text{if } x_r^j = \bot$$

  Both choices also allow to go to one of the states q_t or $\overline{q_t}$. In q_r, players A_t^j with $t \neq r$ have one single choice, which is the whole set of states.
- player C_t also chooses for the value of the variable it represents. As for players A_r^j, this choice will be expressed by choosing between two sets of states corresponding to clauses that are not made true. But as in the proof of Prop. 12, players C_t will also offer the possibility to "verify" their choice, by going either to state q_t or $\overline{q_t}$. Formally, this yields two sets of states:

$$\{q_r^{g,k} \mid \forall l \leq 3.\ s_r^{g,l} \neq z_t \text{ or } \alpha_r^{g,l} = 0\} \cup \{q_u, \overline{q_u} \mid u \neq t\} \cup \{q_t\} \qquad \text{if } z_t = \top$$
$$\{q_r^{g,k} \mid \forall l \leq 3.\ s_r^{g,l} \neq z_t \text{ or } \alpha_r^{g,l} = 1\} \cup \{q_u, \overline{q_u} \mid u \neq t\} \cup \{\overline{q_t}\} \qquad \text{if } z_t = \bot$$

- Last, player D chooses either to challenge a player C_t, with $t < r$, by choosing the set $\{q_t, \overline{q_t}\}$, or to check that a clause S_r^j is fulfilled, by choosing $\{q_r^{j,0}, ..., q_r^{j,7}\}$.

Let us first prove that any choices of all the players yields exactly one state. It is obvious except for states q_r. For a state q_r, let us first restrict to the choices of all the players A_r^j and C_r, then:

- if we only consider states $q_r^{1,0}$ to $q_r^{n,7}$, the same argument as in the previous proof ensures that precisely on state per clause is chosen,
- if we consider states q_t and $\overline{q_t}$, the choices of players B_t ensure that exactly one state has been chosen in each pair $\{q_t, \overline{q_t}\}$, for each $t < r$.

Clearly, the choice of player D will select exactly one of the remaining states.

Now, we build the ATL formula. It is a recursive formula (very similar to the one used in the proof of Prop. 12), defined by $\psi_0 = \top$ and (again writing AC for the set of players $\{A_1^1, ..., A_p^m, C_1, ..., C_p\}$):

$$\psi_{r+1} \stackrel{\text{def}}{=} \langle\!\langle \mathsf{AC} \rangle\!\rangle\, (\neg s)\, \mathbf{U}\, (\alpha \vee \mathbf{EX}\, (s \wedge \mathbf{EX}\, \neg\psi_r)).$$

Then, writing $f_{\mathcal{I}}$ for the state-based strategy associated to $v_{\mathcal{I}}$:

Lemma 17. *For any $r \leq p$ and $t \geq r$, the following statements are equivalent: (a) $q_r \models \psi_t$; (b) the strategies $f_{\mathcal{I}}$ witness the fact that $q_r \models \psi_t$; (c) variable z_r evaluates to true in $v_{\mathcal{I}}$.* □

Theorem 18. *Model checking ATL on ATSs is $\mathbf{\Delta}_2^{\mathsf{P}}$-complete.*

4.5 Model Checking ATL$^+$

The complexity of model checking ATL$^+$ over ATSs has been settled $\mathbf{\Delta}_3^{\mathsf{P}}$-complete in [18]. But $\mathbf{\Delta}_3^{\mathsf{P}}$-hardness proof of [18] is in LOGSPACE only w.r.t. the DAG-size of the formula. We prove that model checking ATL$^+$ is in fact $\mathbf{\Delta}_3^{\mathsf{P}}$-complete (with the standard definition of the size of a formula) for our three kinds of game structures.

Theorem 19. *Model checking ATL$^+$ is $\mathbf{\Delta}_3^{\mathsf{P}}$-complete on ATSs as well as on explicit CGSs and implicit CGSs.*

5 Conclusion

In this paper, we considered the basic questions of expressiveness and complexity of ATL. We showed that ATL, as originaly defined in [2,3], is not as expressive as it could be expected, and we argue that the modality "Release" should be added in its definition [12].

We also precisely characterized the complexity of ATL and ATL$^+$ model-checking, on both ATSs and CGSs, when the number of agents is not fixed. These results complete the previously known results about these formalisms and it is interesting to see that their complexity classes ($\mathbf{\Delta}_2^{\mathsf{P}}$or $\mathbf{\Delta}_3^{\mathsf{P}}$) are unusual in the model-checking area.

As future works, we plan to focus on the extensions EATL (extending ATL with a modality $\langle\!\langle \cdot \rangle\!\rangle \overset{\infty}{\mathbf{F}}$ expressing a Büchi-like winning condition, and for which state-based strategies are still sufficient) and EATL$^+$ (the obvious association of both extensions, but for which state-based strategies are not sufficient anymore).

Acknowledgement. We thank Wojtek Jamroga for pointing out that formulas in SNSAT$_2$ cannot be restricted to CNF [6].

References

1. M. Adler and N. Immerman. An $n!$ lower bound on formula size. In *Proceedings of the 16th Annual Symposium on Logic in Computer Science (LICS'01)*, p. 197–206. IEEE CSP, 2001.
2. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proc. 38th Annual Symp. Foundations of Computer Science (FOCS'97)*, p. 100–109. IEEE CSP, 1997.
3. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.
4. R. Alur, T. A. Henzinger, O. Kupferman, and M. Y. Vardi. Alternating refinement relations. In *Proc. 9th Intl Conf. Concurrency Theory (CONCUR'98)*, vol. 1466 of *LNCS*, p. 163–178. Springer, 1998.
5. R. Alur, T. A. Henzinger, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S. Tasiran. Mocha: Modularity in model checking. In *Proc. 10th Intl Conf. Computer Aided Verification (CAV'98)*, vol. 1427 of *LNCS*, p. 521–525. Springer, 1998.

6. M. Cadoli, A. Giovanardi, and M. Schaerf. An algorithm to evaluate quantified boolean formulae. *Journal of Automated Reasoning*, 28(2):101–142, 2002.
7. E. M. Clarke and E. A. Emerson. Design and synthesis of synchronous skeletons using branching-time temporal logic. In *Proc. 3rd Workshop Logics of Programs (LOP'81)*, vol. 131 of *LNCS*, p. 52–71. Springer, 1981.
8. E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science*, vol. B, chapter 16, p. 995–1072. Elsevier, 1990.
9. V. Goranko and G. van Drimmelen. Complete axiomatization and decidability of alternating-time temporal logic. *Theoretical Computer Science*, 353(1-3):93–117, Mar. 2006.
10. A. Harding, M. Ryan, and P.-Y. Schobbens. Approximating ATL* in ATL. In *Revised Papers 3rd Intl Workshop Verification, Model Checking, and Abstract Interpretation (VMCAI'02)*, vol. 2294 of *LNCS*, p. 289–301. Springer, 2002.
11. W. Jamroga and J. Dix. Do agents make model checking explode (computationally)? In *Proc. 4th Intl Centr. and East. Europ. Conf. Multi-Agent Systems (CEEMAS'05)*, vol. 3690 of *LNCS*. Springer, 2005.
12. W. Jamroga and J. Dix. Model checking abilities of agents: A closer look. Technical Report IfI-06-02, Institut für Informatik, Technische Universität Clausthal, Germany, 2006.
13. F. Laroussinie, N. Markey, and G. Oreiby. Expressiveness and complexity of ATL. Technical Report LSV-06-03, Lab. Spécification & Vérification, ENS Cachan, France, 2006.
14. F. Laroussinie, N. Markey, and Ph. Schnoebelen. Model checking CTL$^+$ and FCTL is hard. In *Proc. 4th Intl Conf. Foundations of Software Science and Computation Structure (FoSSaCS'01)*, vol. 2030 of *LNCS*, p. 318–331. Springer, 2001.
15. Ch. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
16. A. Pnueli. The temporal logic of programs. In *Proc. 18th Ann. Symp. Foundations of Computer Science (FOCS'77)*, p. 46–57. IEEE Comp. Soc. Press, 1977.
17. J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proc. 5th Intl Symp. on Programming (SOP'82)*, vol. 137 of *LNCS*, p. 337–351. Springer, 1982.
18. P.-Y. Schobbens. Alternating-time logic with imperfect recall. In *Proc. 1st Workshop Logic and Communication in Multi-Agent Systems (LCMAS'03)*, vol. 85 of *ENTCS*. Elsevier, 2004.
19. D. Walther, C. Lutz, F. Wolter, and M. Wooldridge. ATL satisfiability is indeed EXPTIME-complete. *Journal of Logic and Computation*, 2006. To appear.
20. I. Walukiewicz. A landscape with games in the background. In *Proc. 19th Ann. Symp. Logic in Computer Science (LICS'04)*, p. 356–366. IEEE CSP, 2004.
21. Th. Wilke. CTL$^+$ is exponentially more succinct than CTL. In *Proc. 19th Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS'99)*, vol. 1738 of *LNCS*, p. 110–121. Springer, 1999.

Polynomial Constraints for Sets with Cardinality Bounds

Bruno Marnette[1], Viktor Kuncak[2], and Martin Rinard[2]

[1] ENS de Cachan, France
bruno@marnette.fr
[2] MIT CSAIL, Cambridge, USA
{vkuncak,rinard}@csail.mit.edu

Abstract. Logics that can reason about sets and their cardinality bounds are useful in program analysis, program verification, databases, and knowledge bases. This paper presents a class of constraints on sets and their cardinalities for which the satisfiability and the entailment problems are computable in polynomial time. Our class of constraints, based on tree-shaped formulas, is unique in being simultaneously tractable and able to express 1) that a set is a union of other sets, 2) that sets are disjoint, and 3) that a set has cardinality within a given range. As the main result we present a polynomial-time algorithm for checking entailment of our constraints.

1 Introduction

Hierarchical representations of sets of entities are ubiquitous in computer science, arising in programming languages, program analysis, software engineering and knowledge bases. When considering a class of constraints, we are interested in two main questions:

- **satisfiability:** is a set of constraints consistent (satisfiable)?
- **entailment:** does one set of constraints imply another set of constraints?

Note that a solution to the second problem is also a solution to the first problem: checking whether a set of constraints implies (entails) a fixed contradictory constraint solves the satisfiability problem.

In object-oriented programming and software modelling, set hierarchies model classification of entities into classes and are an important component of object models represented using notations such as UML [11] and Alloy [14]. The entailment problem for set hierarchies arises when checking, for example, that one UML diagram is a refinement of another diagram. Satisfiability checking can detect contradictory constraints that indicate an error in the model or system requirements.

Set hierarchies are also essential in knowledge representation [26]. Entailment checking allows one to check that the classification in a particular knowledge-base is a consequence of the classification in a more general ontology.

Recently, researchers have considered the (typestate) generalization of static class hierarchies in object-oriented languages to dynamically changing hierarchies of sets of objects [9, 17]. Using the ideas of [20], we can statically approximate dynamically changing set hierarchy at each program point by propagating constraints between sets of objects using a data-flow analysis. A modular approach to such analysis needs to check that 1) each procedure precondition is satisfied at each procedure call site, and 2) the postcondition holds at the end of each procedure. When the propagated information encodes a set hierarchy, these two checks require deciding the entailment of such hierarchies.

Sets with cardinality constraints. One often wishes to express constraints not only on sets but also on certain distinguished elements of these sets. A simple and unified way to reason about elements is to represent them as sets of cardinality one. Similarly, it is often desirable to state that a set is non-empty or, more generally, that the number of its elements is within given bounds. This motivates the use of cardinality constraints on sets that participate in hierarchies.

We have previously considered expressive logics that can express such constraints by combining the Boolean Algebra of sets with a cardinality operator and Presburger Arithmetic [18], [16, Chapter 7]. However, the NP-hardness of these constraints potentially limits their practical use, which motivated us to find constraints that have polynomial-time algorithms. The result is the class presented in this paper, for which we construct a polynomial-time algorithm for entailment (and therefore satisfiability). This class can express a combination of constraints that, to the best of our knowledge, cannot be represented using existing polynomial-time formalisms (see Section 6).

Our result. We call our notion of set hierarchy *itree*, standing for *inclusion tree*, because the edges in the hierarchy represent set inclusion $B \subseteq A$ and because the inclusion edges in an itree form an inverted tree. Moreover, an itree can specify that a set is covered by some of its subsets ($A = B \cup C \cup D$), or/and that these subsets are pairwise disjoint ($B \cap C = C \cap D = B \cap D = \emptyset$). An itree can also specify multiple orthogonal divisions of one set into subsets, such as $A = B \cup C \wedge A = D \cup E \wedge D \cap E = \emptyset$. Finally, an itree can specify constant cardinality constraints on sets, such as $1 \leq |A| \leq 10000$. Our algorithm checks entailment of conjunctions of such constraints.

The key idea of our polynomial-time algorithm is to define a notion of normal form where each tree node satisfies certain local constraints. We show that this normal form can be enforced in polynomial time using a set of rewrite rules. We then give polynomial-time conditions for checking whether a normalized itree implies a given constraint on variables. This yields an algorithm for checking whether an itree implies a conjunction of such constraints, and we show that an itree can always be represented as a conjunction of quantifier-free constraints. We therefore obtain a polynomial entailment test for itree constraints.

Contributions. The contributions of our paper include the following:

– We introduce itree constraints for expressing hierarchies of sets, and permitting a simple form of existential quantification over sets (Section 2.2).
– We show that generalizing the definition of itrees to permit acyclic graphs yields constraints whose satisfiability is NP-hard (Section 2.3).
– We give a polynomial-time algorithm for checking the satisfiability of itrees by proving sufficient conditions for the existence of their models (Section 3).

- We give a polynomial-time algorithm for checking whether an itree entails a given cardinality, inclusion or disjointness constraint (Section 4).
- We show that the quantifiers in an itree can be eliminated, which, with the previous result, gives polynomial-time entailment for itrees (Section 5).

A preliminary version of the current polynomial-time results (including proofs) appears in the technical report [22], using the same ideas but slightly different definitions. Due to space limitations we here present only proof outlines, describing the main ideas and revealing the underlying algorithms.

2 Constraints on Sets and Their Graphical Representation

The constraints that we consider in this paper are expressible using existentially quantified conjunctions of boolean algebra formulas whose variables range over sets of uninterpreted objects. We call these formulas Conjunctive constraints on Sets with Cardinalities and denote them CSC.

Definition 1. CSC *formulas are given by the following syntax:*

$$\begin{array}{l} \phi ::= \exists \nu_1, \ldots, \nu_n.\ P_1 \wedge \ldots \wedge P_m \\ P := \ S_1 \subseteq S_2 \mid S_1 \cap S_2 = \emptyset \ \mid \ |\nu| \leq k \ \mid \ |\nu| \geq k \\ S := \ s \mid \nu \mid S_1 \cup S_2 \end{array}$$

Variables in CSC formulas denote sets and can be free set variables (denoted s, s', s_i) or bound set variables (denoted ν, ν',ν_i). Sets in CSC formulas are denoted by variables or unions of variables. Cardinality constraints apply only to bound variables.

Lemma 1. *Satisfiability of* CSC *formulas is NP-hard.*

Lemma 1 holds because CSC can express boolean algebra constraints on subsets of a fixed set variable U. Namely, union together with disjointness from U can define set complement; union and complement then allow encoding arbitrary propositional operations.

2.1 Graph Representation IGRAPH for CSC

As a first step towards identifying polynomial constraints, we introduce a representation of CSC by *igraphs* (standing for *inclusion graphs*). In the following definition of igraphs, the nodes VN are bound set variables ν and the edges $\rightsquigarrow$ represent the subset inclusion of sets. Nodes are tagged with cardinality constraints and with *mode symbols* establishing additional constraints between a node and its direct sons. If ν is tagged with the mode symbol ○, the sons of ν are pairwise disjoint. If ν is tagged with the mode symbol ■, the sons $\{\nu_1, \ldots, \nu_n\}$ of ν cover entirely ν, that is $\nu \subseteq \cup_i \nu_i$. If ν is tagged with the mode symbol ⧫, then ν is *equal* to each of its sons. When a set ν participates in several atomic formulas, we can use the ♦ mode to introduce synonyms for ν. Finally, a mapping σ establishes equalities between free set variables $s \in \mathsf{SN}$ and bound variables $\nu \in \mathsf{VN}$. It also enables the encoding of set emptiness using a special symbol $\emptyset_I$.

Definition 2 **(IGRAPH).** *An* igraph $G \in \mathsf{IGRAPH}$ *is either the* false igraph $\bot_I$ *or a tuple* $(\mathsf{SN}, \mathsf{VN}, \rightsquigarrow, \mathsf{CInf}, \mathsf{CSup}, \mathsf{M}, \sigma)$ *such that*

$$\begin{array}{ll}
\mathsf{SN} \ \textit{and}\ \mathsf{VN} \ \textit{are two disjoint sets of set variables} & \\
(\mathsf{VN}, \rightsquigarrow) \ \textit{is a directed graph} & \\
\mathsf{CInf} : \mathsf{VN} \to \mathbb{N} & (\mathbb{N} = \{0, 1, 2, \ldots\}) \\
\mathsf{CSup} : \mathsf{VN} \to \mathbb{N} \cup \{\infty\} & (\forall k \in \mathbb{N}.\ k < \infty) \\
\mathsf{M} : \mathsf{VN} \to \mathcal{P}(\{\blacklozenge, \blacksquare, \bigcirc\}) & \\
\sigma : \mathsf{SN} \to \mathsf{VN} \cup \{\emptyset_I\} &
\end{array}$$

The set SN corresponds to the free variables s of G. The elements of VN correspond to the bound variables ν and are also called *nodes* by graph analogy. $\mathcal{P}(\{\blacklozenge, \blacksquare, \bigcirc\})$ denotes the set of subsets of $\{\blacklozenge, \blacksquare, \bigcirc\}$. We write $\nu \rightsquigarrow \nu'$ when $(\nu, \nu') \in \rightsquigarrow$. We define the set of *sons* of $\nu \in \mathsf{VN}$ by $\mathsf{Sons}(\nu) = \{\nu' | \nu' \rightsquigarrow \nu\}$ and the *incoming degree* of ν by $\mathsf{d}(\nu) = |\mathsf{Sons}(\nu)|$.

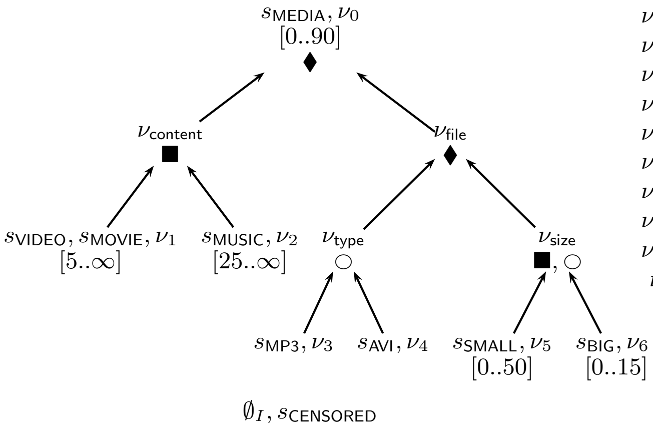
Definition 3 **(IGRAPH semantics).** *The semantics* $\mathsf{Sem}(\bot_I)$ *of the* false igraph $\bot_I$ *is by definition the formula* false. *With each igraph* $G \neq \bot_I$ *we associate a quantifier-free* CSC *formula* $\mathsf{Sem}_0(G)$ *as follows:*

$$\mathsf{Sem}_0(G) \overset{def}{=} \bigwedge \begin{cases}
\bigwedge\{\nu' \subseteq \nu \mid \nu, \nu' \in \mathsf{VN} \ \wedge \ \nu' \rightsquigarrow \nu\} \\
\bigwedge\{\nu' = \nu \mid \nu, \nu' \in \mathsf{VN} \ \wedge \ \nu' \rightsquigarrow \nu \wedge \ \blacklozenge \in \mathsf{M}(\nu)\} \\
\bigwedge\{\nu \subseteq \bigcup \mathsf{Sons}(\nu) \mid \nu \in \mathsf{VN} \ \wedge \ \blacksquare \in \mathsf{M}(\nu)\} \\
\bigwedge\{\nu' \cap \nu'' = \emptyset \mid \nu \in \mathsf{VN} \ \wedge \ \nu', \nu'' \in \mathsf{Sons}(\nu) \wedge \nu' \neq \nu'' \ \wedge \bigcirc \in \mathsf{M}(\nu)\} \\
\bigwedge\{\mathsf{CInf}(\nu) \leq |\nu| \leq \mathsf{CSup}(\nu) \mid \nu \in \mathsf{VN}\}
\end{cases}$$

The semantics $\mathsf{Sem}(G)$ *of* G *is then:*

$$\mathsf{Sem}(G) \overset{def}{=} \exists \nu_1, \ldots, \nu_n.\ \ \mathsf{Sem}_0(G) \wedge \bigwedge_{s \in \mathsf{SN}} \begin{cases} s = \emptyset, & \textit{if} \ \ \sigma(s) = \emptyset_I \\ s = \nu, & \textit{if} \ \ \sigma(s) = \nu \end{cases}$$

Figure 1 gives an example of an igraph G (represented graphically) with its semantics $\mathsf{Sem}(G)$. Given two igraphs G and G' we write $G \models G'$ iff $\mathsf{Sem}(G)$ entails $\mathsf{Sem}(G')$ and we write $G \equiv G'$ when both $G \models G'$ and $G' \models G$. We say that G is satisfiable iff $\mathsf{Sem}(G)$ is satisfiable. We also use the symbols $\models$ and $\equiv$ to compare igraphs and CSC formulas, identifying igraphs G with their semantics $\mathsf{Sem}(G)$. To avoid confusion between the syntax and the semantics of formulas we use square brackets around formulas. Thus, in the following sections, $[\nu = \nu']$ denotes an equality between two sets while $\nu = \nu'$ only states that ν and ν' are the same variable symbol (or the same node).


$s_{\mathsf{MEDIA}}, \nu_0$
$[0..90]$
ν_{content}
ν_{file}
$s_{\mathsf{VIDEO}}, s_{\mathsf{MOVIE}}, \nu_1$
$[5..\infty]$
$s_{\mathsf{MUSIC}}, \nu_2$
$[25..\infty]$
ν_{type}
ν_{size}
s_{MP3}, ν_3
s_{AVI}, ν_4
$s_{\mathsf{SMALL}}, \nu_5$
$[0..50]$
s_{BIG}, ν_6
$[0..15]$
$\emptyset_I, s_{\mathsf{CENSORED}}$

$$\begin{aligned}
&\mathsf{Sem}(G) = \\
&\exists \nu_1, \nu_2, \nu_3, \nu_4, \nu_5, \nu_6, \\
&\quad \nu_{\mathsf{content}}, \nu_{\mathsf{file}}, \nu_{\mathsf{type}}, \nu_{\mathsf{size}}. \\
&\quad \nu_{\mathsf{content}} \subseteq \nu_0 \wedge \nu_{\mathsf{file}} \subseteq \nu_0 \wedge \\
&\quad \nu_1 \subseteq \nu_{\mathsf{content}} \wedge \nu_2 \subseteq \nu_{\mathsf{content}} \wedge \\
&\quad \nu_{\mathsf{type}} \subseteq \nu_{\mathsf{file}} \wedge \nu_3 \subseteq \nu_{\mathsf{type}} \wedge \\
&\quad \nu_4 \subseteq \nu_{\mathsf{type}} \wedge \nu_{\mathsf{size}} \subseteq \nu_{\mathsf{file}} \wedge \\
&\quad \nu_5 \subseteq \nu_{\mathsf{size}} \wedge \nu_6 \subseteq \nu_{\mathsf{size}} \\
&\quad \nu_0 = \nu_{\mathsf{content}} = \nu_{\mathsf{file}} \wedge \\
&\quad \nu_{\mathsf{file}} = \nu_{\mathsf{type}} = \nu_{\mathsf{size}} \wedge \\
&\quad \nu_{\mathsf{content}} = \nu_1 \cup \nu_2 \wedge \\
&\quad \nu_{\mathsf{size}} = \nu_5 \cup \nu_6 \wedge \\
&\quad \nu_3 \cap \nu_4 = \emptyset \wedge \\
&\quad \nu_5 \cap \nu_6 = \emptyset \wedge \\
&\quad |\nu_0| \leq 90 \wedge |\nu_1| \geq 5 \wedge \\
&\quad |\nu_2| \geq 25 \wedge \\
&\quad |\nu_5| \leq 50 \wedge |\nu_6| \leq 15 \wedge \\
&\quad s_{\mathsf{MEDIA}} = \nu_0 \wedge s_{\mathsf{MOVIE}} = \nu_1 \wedge \\
&\quad s_{\mathsf{VIDEO}} = \nu_1 \wedge s_{\mathsf{MUSIC}} = \nu_2 \wedge \\
&\quad s_{\mathsf{MP3}} = \nu_3 \wedge s_{\mathsf{AVI}} = \nu_4 \wedge \\
&\quad s_{\mathsf{SMALL}} = \nu_5 \wedge s_{\mathsf{BIG}} = \nu_6 \wedge \\
&\quad s_{\mathsf{CENSORED}} = \emptyset
\end{aligned}$$

Fig. 1. An example of itree (a particular case of igraph) and its semantics

By construction, the semantics of an igraph is expressible by a CSC formula. The following lemma shows that the converse holds as well.

Lemma 2. *For each $\phi \in$ CSC we can compute in linear time an equivalent igraph.*

As a consequence, the satisfiability of igraphs is also NP-hard.

2.2 Definition of Itrees

We can now define our subclass of tree-shaped igraphs. We call this subclass *itrees*. Polynomial-time algorithms for satisfiability and entailment of itrees are the subject of this paper.

Definition 4 (ITREE). *A* generalized itree *(gitree) T is either the false igraph $\bot_I$ or an igraph $G \in$ IGRAPH such that $(\mathsf{VN}, \rightsquigarrow)$ is a tree, oriented from the leaves to the root.*

An itree *is a generalized itree such that, for each $\nu \in \mathsf{VN}$*

$$\sigma^{-1}(\nu) = \emptyset \quad \Rightarrow \quad \mathsf{CInf}(\nu) = 0 \wedge \mathsf{CSup}(\nu) = \infty \tag{QE}$$

Thanks to the tree-shape condition, itrees (and even generalized itrees) satisfy some important properties that are not true for general (or acyclic) igraphs. For example, it follows from Lemma 10 of Section 4.2 that the semantics ϕ of an itree always satisfies, for all set variables s_1, s_2, s_3, the following property:

$$\phi \models [s_1 \subseteq s_2 \wedge s_1 \subseteq s_3] \;\Rightarrow\; \big(\phi \models [s_1 = \emptyset] \;\vee\; \phi \models [s_2 \subseteq s_3] \;\vee\; \phi \models [s_3 \subseteq s_2]\big)$$

This property allows us to prove, for example, that the CSC formula $[A \subseteq B \wedge A \subseteq C]$ is not expressible as a (generalised) itree. Therefore, the class ITREE is a strict subclass of IGRAPH and is a good candidate for a more efficient fragment of IGRAPH.

The QE condition (standing for *quantifier elimination*) in the definition of ITREE ensures that the semantics of itrees can in fact be expressed using a quantifier-free CSC formula, as proved in Section 5. Note that a sufficient condition for QE is that $\sigma^{-1}(\nu) \neq \emptyset$ for each $\nu \in \mathsf{VN}$.

Because we can check whether a graph is a tree by depth-first traversal of the graph, we have the following result.

Lemma 3. *Deciding whether a given igraph $G \in \mathsf{IGRAPH}$ is an itree ($G \in \mathsf{ITREE}$) can be done in linear time.*

2.3 Hardness of Acyclic Igraphs

We have observed that satisfiability of igraphs is NP-hard. In contrast, we prove in the rest of this paper that itrees have polynomial-time satisfiability and entailment problems. A natural question to ask is whether we could obtain polynomial-time algorithms for igraphs where inclusions are acyclic but not tree-like. The following lemma (see also [22, Section 4, Lemma 4]) suggests a negative answer to this question.

Lemma 4. *Let IDAG denote the class of igraphs for which $(\mathsf{VN}, \rightsquigarrow)$ is a directed acyclic graph (DAG). For each igraph in IGRAPH we can compute in polynomial time an equivalent igraph in IDAG. Therefore, satisfiability in IDAG is NP-hard.*

The essence of the proof of Lemma 4 is that we can collapse (in polynomial time) cycles in an igraph to obtain an equivalent acyclic igraph. In addition to NP-hardness of the class of acyclic igraphs, we can prove NP-hardness for several subclasses of IDAG, using the construction in [22, Section 5, Theorem 2]. We therefore believe that considering tree-like restrictions on igraphs is a reasonable approach to identifying polynomial constraints.

3 Deciding Satisfiability of Generalized Itrees in Polynomial Time

This section gives a linear-time algorithm for satisfiability of generalized itrees. This result is a first step to an algorithm for checking entailment, which we describe in Section 5, building on the results in this section. Moreover, the satisfiability algorithm is of interest in itself.

We proceed by first showing (Lemma 5) that the bottom-up propagation of constraints (rewriting rules $\mathcal{R}_1$ and $\mathcal{R}_2$) allows transforming in linear time any gitree T into an equivalent gitree $\mathcal{R}_2^{\downarrow}(T)$ such that either **a)** $\mathcal{R}_2^{\downarrow}(T) =\perp_I$, in which case T is clearly unsatisfiable, or **b)** $\mathcal{R}_2^{\downarrow}(T)$ satisfies two properties C_1

and C_2. We then show (Lemma 6) that any gitree for which C_1 and C_2 hold is satisfiable. As a result, we can decide in linear time whether T is satisfiable by first computing $\mathcal{R}_2^{\downarrow}(T)$ and then returning *satisfiable* if and only if $\mathcal{R}_2^{\downarrow}(T)$ is different from $\perp_I$.

Lemma 5. *For each gitree T we can compute in linear time an equivalent gitree $\mathcal{R}_2^{\downarrow}(T)$ such that either $\mathcal{R}_2^{\downarrow}(T) = \perp_I$ or $\mathcal{R}_2^{\downarrow}(T)$ satisfies (for each node ν) both:*

$$\mathsf{M}(\nu) \in \{\emptyset, \{\blacklozenge\}, \{\bigcirc\}, \{\blacksquare\}, \{\bigcirc, \blacksquare\}\} \qquad (C_1(\nu))$$

and

$$\mathsf{BUInf}(\nu) \leq \mathsf{CInf}(\nu) \leq \mathsf{CSup}(\nu) \leq \mathsf{BUSup}(\nu) \qquad (C_2(\nu))$$

where, for $\mathsf{Sons}(\nu) = \{\nu_1, \ldots, \nu_n\}$,

$$\mathsf{BUInf}(\nu) \overset{def}{=} \begin{cases} \sum_i \mathsf{CInf}(\nu_i), & \textit{if } \bigcirc \in \mathsf{M}(\nu) \\ \max_i \mathsf{CInf}(\nu_i), & \textit{otherwise} \end{cases}$$

$$\mathsf{BUSup}(\nu) \overset{def}{=} \begin{cases} \min_i \mathsf{CSup}(\nu_i), & \textit{if } \blacklozenge \in \mathsf{M}(\nu) \\ \sum_i \mathsf{CSup}(\nu_i), & \textit{if } \blacksquare \in \mathsf{M}(\nu) \\ \infty, & \textit{otherwise} \end{cases}$$

Proof. Such a form $\mathcal{R}_2^{\downarrow}(T)$ can be obtained from T in two steps. The first steps consists in simplifying the mode combinations by applying the following rewriting rule R_1 to every node (in any order).

if	apply
$\mathsf{d}(\nu) = 0$	$\mathsf{M}(\nu) := (\mathsf{M}(\nu) - \{\blacklozenge\})$
$\mathsf{d}(\nu) \leq 1$	$\mathsf{M}(\nu) := (\mathsf{M}(\nu) - \{\bigcirc\})$
$\mathsf{d}(\nu) \geq 1$ $\blacklozenge \in \mathsf{M}(\nu)$	$\mathsf{M}(\nu) := (\mathsf{M}(\nu) - \{\blacksquare\})$
$\mathsf{d}(\nu) \geq 2$ $\{\blacklozenge, \bigcirc\} \subseteq \mathsf{M}(\nu)$	$\mathsf{M}(\nu) := (\mathsf{M}(\nu) - \{\bigcirc\})$ $\forall \nu' \in \mathsf{Sons}(\nu), \quad \mathsf{CSup}(\nu') := 0$

$(\mathcal{R}_1(\nu))$

The second step consists in applying the rule $\mathcal{R}_2$ below to every node, proceeding *from the leaves towards the root*, in order to 1) refine the cardinality bounds and 2) recognize the contradictory bounds such that $\mathsf{CInf}(\nu) > \mathsf{CSup}(\nu)$.

$\mathsf{CInf}(\nu)$	$:=$	$\mathsf{Max}(\mathsf{CInf}(\nu), \mathsf{BUInf}(\nu))$
$\mathsf{CSup}(\nu)$	$:=$	$\mathsf{Min}(\mathsf{CSup}(\nu), \mathsf{BUSup}(\nu))$
If $\mathsf{CInf}(\nu) > \mathsf{CSup}(\nu)$ then $T := \perp_I$		

$(\mathcal{R}_2(\nu))$

We say that C_i *holds for* T (i.e. "*T satisfies C_i*") iff $C_i(\nu)$ holds for each node ν of T.

Lemma 6. *Every gitree $T \neq \perp_I$ for which both C_1 and C_2 hold is satisfiable.*

Proof. We first note that a gitree T such that $T \neq \perp_I$ is satisfiable if and only if there exists a model for $\mathsf{Sem}_0(T)$. Indeed, a model $(\Delta, \alpha : \mathsf{VN} \rightarrow \mathcal{P}(\Delta))$ for $\mathsf{Sem}_0(T)$ can be turned into a model $(\Delta, \alpha' : \mathsf{SN} \rightarrow \mathcal{P}(\Delta))$ for $\mathsf{Sem}(T)$ by taking $\alpha'(s) = \emptyset$ when $\sigma(s) = \emptyset_I$ and $\alpha'(s) = \alpha(\sigma(s))$ when $\sigma(s) \in \mathsf{VN}$.

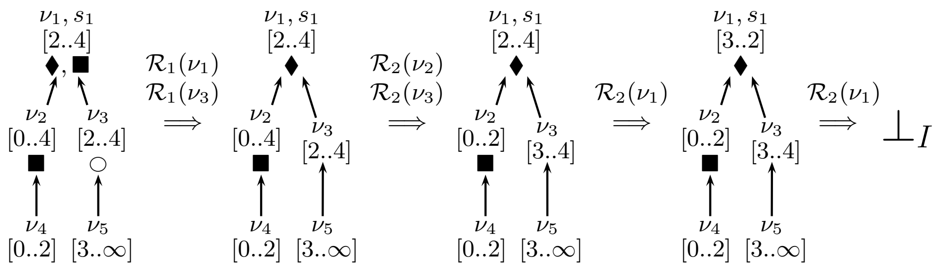


Fig. 2. Example of $\mathcal{R}_1$ and $\mathcal{R}_2$ derivation

When T satisfies both C_1 and C_2 we can build a model for $\mathsf{Sem}_0(T)$ in two steps. We first choose (in linear time) relevant cardinalities $\psi(\nu) \in \mathbb{N}$ for the nodes ν, proceeding *from the root to the leaves*. More precisely, we take $\psi(\nu) = \mathsf{CInf}(\nu)$ for the root ν of T and define recursively the values $\psi(\nu_i)$ for the sons ν_i of a node ν, for a chosen ordering of $\mathsf{Sons}(\nu) = \{\nu_1, \ldots, \nu_n\}$, and by induction on $i = 1..n$:

$$\psi(\nu_i) \overset{def}{=} \begin{cases} \mathsf{CInf}(\nu_i) & \text{if } \mathsf{M}(\nu) = \emptyset \\ \mathsf{CInf}(\nu_i) & \text{if } \mathsf{M}(\nu) = \{\bigcirc\} \\ \psi(\nu) & \text{if } \mathsf{M}(\nu) = \{\blacklozenge\} \\ \min(\mathsf{CSup}(\nu_i), \psi(\nu)) & \text{if } \mathsf{M}(\nu) = \{\blacksquare\} \\ \min(\mathsf{CSup}(\nu_i), \psi(\nu) - \sum\limits_{i'<i} \psi(\nu_{i'}) - \sum\limits_{i'>i} \mathsf{CInf}(\nu_{i'})) & \text{if } \mathsf{M}(\nu) = \{\bigcirc, \blacksquare\} \end{cases}$$

The conditions C_1 and C_2 guarantee that this cardinality choice satisfies the following property H^ψ for every node ν such that $\mathsf{Sons}(\nu) = \{\nu_1, \ldots, \nu_n\}$:

$$\left.\begin{array}{l} \mathsf{CInf}(\nu) \leq \psi(\nu) \leq \mathsf{CSup}(\nu) \\ \bigwedge_i \ \ \psi(\nu_i) \leq \psi(\nu) \\ \blacklozenge \in \mathsf{M}(\nu) \Rightarrow \ \bigwedge_i \psi(\nu_i) = \psi(\nu) \\ \bigcirc \in \mathsf{M}(\nu) \Rightarrow \ \sum_i \psi(\nu_i) \leq \psi(\nu) \\ \blacksquare \in \mathsf{M}(\nu) \Rightarrow \ \sum_i \psi(\nu_i) \geq \psi(\nu) \end{array}\right\} \ (H^\psi(\nu))$$

When a cardinality choice satisfying $H^\psi(\nu)$ for all nodes ν of T is chosen, the second step consists in building for every node ν, taken from the leaves to the root, a model for the formula $\mathsf{Sem}_0(T|_\nu)$ where $T|_\nu$ denotes the sub-itree $T|_\nu$ of T of root ν. The role played by $H^\psi(\nu)$ in this construction is the following: the property $\psi(\nu_i) \leq \psi(\nu)$ ensures that the son ν_i of ν is small enough to fit in ν; when $\mathsf{M}(\nu) = \{\blacklozenge\}$, the property $\psi(\nu_i) = \psi(\nu)$ ensures that the sons ν_i of ν have the right cardinality to be made equal to ν; when $\bigcirc \in \mathsf{M}(\nu)$ the property $\sum_i \psi(\nu_i) \leq \psi(\nu)$ ensures that the disjoint union of the sons of ν can fit in ν; when $\blacksquare \in \mathsf{M}(\nu)$, the property $\sum_i \ \psi(\nu_i) \geq \psi(\nu)$ ensures that the sons of ν contain enough elements to cover entirely ν; the property $\mathsf{CInf}(\nu) \leq \psi(\nu) \leq \mathsf{CSup}(\nu)$ ensures that the cardinality constraints are not violated.

Corollary 1. *A gitree T is satisfiable iff $\mathcal{R}_2^{\downarrow}(T) \neq \perp_I$.*

Corollary 2. *We can decide the satisfiability of a gitree in linear time.*

4 Entailment of Quantifier-Free Formulas

The goal of this section is to show that for every gitree T (and, in particular, for every itree $T \in \mathsf{ITREE}$), and every formula ϕ from a quantifier-free fragment QFCSC of CSC defined below, we can decide whether T entails ϕ in polynomial time.

Definition 5 (QFCSC).
$$\begin{aligned} \phi &::= P_1 \wedge \ldots \wedge P_m \\ P &:= S_1 \subseteq S_2 \mid S_1 \cap S_2 = \emptyset \mid |s| \leq k \mid |s| \geq k \\ S &:= s \mid S_1 \cup S_2 \end{aligned}$$

Because QFCSC formulas are conjunctions of atomic formulas, we can decide whether T entails a formula $\Phi = P_1 \wedge \ldots \wedge P_n$ by checking whether $T \models P_i$ for all $i = 1..n$. For deciding $T \models P$ we start by applying additional rewriting rules that enforce stronger properties on gitrees than in the previous section. For each kind of atomic proposition P (cardinality constraint, inclusion, or disjointness) we then define conditions on normalized gitrees that 1) characterize the property $T \models P$, and 2) are computable in polynomial time.

4.1 Checking Cardinality Constraints

Analogously to the definition of BUInf and BUSup (in Lemma 5) we next define for every node ν of a gitree T a lower bound $\mathsf{TDInf}(\nu)$ and an upper bound $\mathsf{TDSup}(\nu)$ for the cardinality of ν, this time corresponding to top-down reasoning. Given a node ν such that $\nu = \mathsf{Root}(T)$ or $\nu \rightsquigarrow \nu'$ and $\mathsf{Sons}(\nu') = \{\nu, \nu_1, \ldots, \nu_n\}$ we define

$$\mathsf{TDInf}(\nu) \overset{def}{=} \begin{cases} 0, & \text{if } \nu = \mathsf{Root}(T) \\ 0, & \text{if } \mathsf{M}(\nu') \in \{\emptyset, \{\blacksquare\}\} \\ \mathsf{CInf}(\nu'), & \text{if } \mathsf{M}(\nu') = \{\blacklozenge\} \\ \mathsf{CInf}(\nu') - \sum_i \mathsf{CSup}(\nu_i), & \text{if } \mathsf{M}(\nu') \in \{\{\bigcirc\}, \{\bigcirc, \blacksquare\}\} \end{cases}$$

$$\mathsf{TDSup}(\nu) \overset{def}{=} \begin{cases} \infty, & \text{if } \nu = \mathsf{Root}(T) \\ \mathsf{CSup}(\nu'), & \text{if } \mathsf{M}(\nu') \in \{\emptyset, \{\blacklozenge\}, \{\bigcirc\}\} \\ \mathsf{CSup}(\nu') - \sum_i \mathsf{CInf}(\nu_i), & \text{if } \mathsf{M}(\nu') \in \{\{\blacksquare\}, \{\bigcirc, \blacksquare\}\} \end{cases}$$

Lemma 7. *For each satisfiable gitree T we can compute in linear time an equivalent gitree $\mathcal{R}_3^{\downarrow}(T)$ satisfying C_1, C_2, and, for each $\nu \in \mathsf{VN}$,*

$$\mathsf{TDInf}(\nu) \leq \mathsf{CInf}(\nu) \;\wedge\; \mathsf{CSup}(\nu) \leq \mathsf{TDSup}(\nu) \qquad (C_3(\nu))$$

Proof. Such a gitree $\mathcal{R}_3^{\downarrow}(T)$ can be obtained by applying the following rule $\mathcal{R}_3$ to $\mathcal{R}_2^{\downarrow}(T)$ using a top-down strategy

$$\boxed{\begin{array}{l} \mathsf{CInf}(\nu) := \mathsf{Max}(\mathsf{CInf}(\nu), \mathsf{TDInf}(\nu)) \\ \hline \mathsf{CSup}(\nu) := \mathsf{Min}(\mathsf{CSup}(\nu), \mathsf{TDSup}(\nu)) \end{array}} \qquad (\mathcal{R}_3(\nu))$$

Lemma 8 (Checking cardinality constraints). *For each gitree T satisfying C_1, C_2 and C_3, each $s \in \mathsf{SN}$, and each $a, b \in \mathbb{N}$ we have*

$$T \models [a \leq |s| \leq b] \iff \begin{cases} \textit{either} \;\; \sigma(s) = \emptyset_I \;\wedge\; a = 0 \\ \textit{or} \;\; a \leq \mathsf{CInf}(\sigma(s)) \leq \mathsf{CSup}(\sigma(s)) \leq b \end{cases}$$

We can therefore decide whether $T \models [a \leq |s| \leq b]$ in linear time.

Proof. For each $T \neq \perp_I$ satisfying C_1, C_2, C_3, for each $\nu \in \mathsf{VN}$ and for each $k \in [\mathsf{CInf}(\nu), \mathsf{CSup}(\nu)]$, the gitree $T_{|\nu| \leftarrow k}$ obtained from T by applying $\mathsf{CInf}(\nu) := k$ and $\mathsf{CSup}(\nu) := k$ satisfies $\mathcal{R}_2^{\downarrow}\ (T_{|\nu| \leftarrow k}) \neq \perp_I$. Therefore, by Corollary [1], there exists a model (Δ, α) of $\mathsf{Sem}_0(T)$ such that $|\alpha(\nu)| = k$.

When T satisfies C_1, C_2, C_3 we can then check that the cardinality bounds CInf and CSup are optimal. That is, for every node ν of such a gitree we have $\mathsf{CInf}(\nu) = \min\{|\alpha(\nu)|, (\Delta, \alpha) \models \mathsf{Sem}_0(T)\}$ and $\mathsf{CSup}(\nu) = \max\{|\alpha(\nu)|, (\Delta, \alpha) \models \mathsf{Sem}_0(T)\}$. The result follows directly from this observation.

4.2 Checking Inclusion and Disjointness Constraints

Now that we have optimal cardinality bounds, it is natural to look at the influence of cardinality constraints on other types of constraints. This approach allows us to enforce an additional property C_4 on gitrees using a rewriting system $\mathcal{R}_4$. Finally, we show how to take advantage of C_4 to decide which inclusion constraints (Lemma [10]) or which pairwise disjointness (Lemma [11]) hold in a gitree T.

Lemma 9. *For each satisfiable gitree T we can compute in linear time an equivalent gitree $\mathcal{R}_4^{\downarrow}(T)$ satisfying C_1, C_2, C_3, and, for each $\nu \in \mathsf{VN}$,*

$$\left.\begin{array}{l} \mathsf{CSup}(\nu) > 0 \\ \mathsf{d}(v) = 1 \Rightarrow \mathsf{M}(\nu) \in \{\{\blacklozenge\}, \{\bigcirc\}\} \\ \mathsf{M}(\nu) = \{\blacksquare\} \Rightarrow \mathsf{CInf}(\nu) < \sum\limits_{\nu' \rightsquigarrow \nu} \mathsf{CSup}(\nu') \\ \mathsf{M}(\nu) = \{\bigcirc\} \Rightarrow \mathsf{CSup}(\nu) > \sum\limits_{\nu' \rightsquigarrow \nu} \mathsf{CInf}(\nu') \end{array}\right\} \quad (C_4(\nu))$$

Proof. Such a gitree $\mathcal{R}_4^{\downarrow}(T)$ can be obtained by applying with a bottom-up strategy the following rule $\mathcal{R}_4$ to $\mathcal{R}_3^{\downarrow}(T)$

if	apply
$\mathsf{d}(\nu) = 1$ $\mathsf{M}(\nu) = \emptyset$	$\mathsf{M}(\nu) := \{\bigcirc\}$
$\mathsf{M}(\nu) = \{\bigcirc\}$ $\mathsf{CSup}(\nu) \leq \sum\limits_{\nu' \rightsquigarrow \nu} \mathsf{CInf}(\nu')$	$\mathsf{M}(\nu) := \{\blacksquare, \bigcirc\}$
$\mathsf{M}(\nu) = \{\blacksquare\}$ $\mathsf{CInf}(\nu) \geq \sum\limits_{\nu' \rightsquigarrow \nu} \mathsf{CSup}(\nu')$	$\mathsf{M}(\nu) := \{\blacksquare, \bigcirc\}$
$\mathsf{d}(\nu) = 1$ $\blacksquare \in \mathsf{M}(\nu)$	$\mathsf{M}(\nu) := \{\blacklozenge\}$
$\mathsf{CSup}(\nu) = 0$ $\mathsf{d}(\nu) = 0$	$\mathsf{VN} := \mathsf{VN} - \{\nu\}$ $\forall s \in \sigma^{-1}(\nu)$ $\quad \sigma(s) := \emptyset_I$

$(\mathcal{R}_4(\nu))$

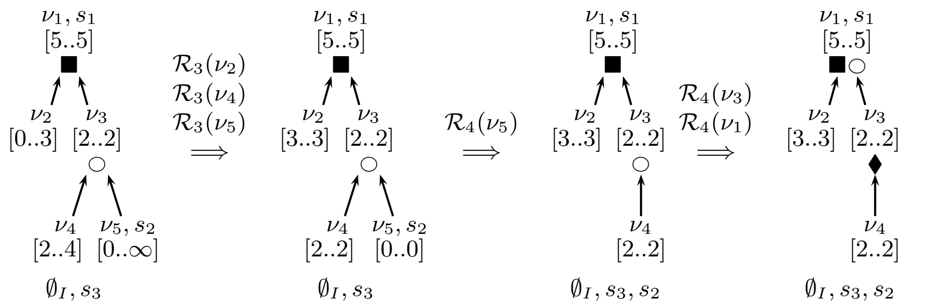


Fig. 3. Example of $\mathcal{R}_3$ and $\mathcal{R}_4$ derivations

Lemma 10 (Checking inclusion constraints). *For each gitree T and each $X \subseteq \mathsf{VN}$ we define a unary predicate $\mathcal{I}_X$ on VN as the least fixed point of*

$$\begin{aligned}
&\mathcal{I}_X(\nu) \Leftarrow \nu \in X\\
&\mathcal{I}_X(\nu) \Leftarrow \blacklozenge \in \mathsf{M}(\nu) \wedge (\exists \nu' \in \mathsf{Sons}(\nu)\ \mathcal{I}_X(\nu))\\
&\mathcal{I}_X(\nu) \Leftarrow \blacksquare \in \mathsf{M}(\nu) \wedge (\forall \nu' \in \mathsf{Sons}(\nu)\ \mathcal{I}_X(\nu))\\
&\mathcal{I}_X(\nu) \Leftarrow \nu \rightsquigarrow \nu' \wedge \mathcal{I}_X(\nu')
\end{aligned}$$

Then, if T satisfies C_1, C_2, C_3, C_4, then for all subsets $S, S' \subseteq \mathsf{SN}$, for $X' = \{\sigma(s) \mid s \in S' \wedge \sigma(s) \in \mathsf{VN}\}$ we have:

$$T \models [(\cup S) \subseteq (\cup S')] \iff \forall s \in S\ \left(\ \sigma(s) = \emptyset_I \vee \mathcal{I}_{X'}(\sigma(s))\ \right)$$

Proof. The result is a consequence of the following observation: for each T satisfying C_1, C_2, C_3, C_4, for each $\nu \in \mathsf{VN}$ and each $X \subseteq \mathsf{VN}$ we have:

$$\mathsf{Sem}_0(T) \models [\nu \subseteq (\cup X)] \iff \mathcal{I}_X(\nu)$$

The proof of this claim relies on a refinement of the algorithm of model construction used in the proof of Lemma 6.

Lemma 11 (Checking disjointness constraints). *For each gitree T we define the binary predicates $\mathcal{D}$ and $\mathcal{D}^*$ on $\mathsf{VN} \times \mathsf{VN}$ by*

$$\begin{aligned}
\mathcal{D}(\nu, \nu') &\iff \nu \neq \nu' \wedge \exists \nu'' \in \mathsf{VN}, \{\nu, \nu'\} \subseteq \mathsf{Sons}(\nu'') \wedge \bigcirc \in \mathsf{M}(\nu'')\\
\mathcal{D}^*(\nu, \nu') &\iff \exists \nu_0, \nu'_0 \in \mathsf{VN}, \nu \rightsquigarrow^* \nu_0 \wedge \nu' \rightsquigarrow^* \nu'_0 \wedge \mathcal{D}(\nu_0, \nu'_0)
\end{aligned}$$

Then, if C_1, C_2, C_3, C_4, for all subsets S, S' of SN we have

$$T \models [(\cup S) \cap (\cup S') = \emptyset] \iff \forall (s, s') \in (S \times S') \begin{cases} \textit{either } \sigma(s) = \emptyset_I \vee \sigma(s') = \emptyset_I\\ \textit{or} \quad \mathcal{D}^*(\sigma(s), \sigma(s')) \end{cases}$$

Proof. The result is a consequence of the following observation, which again relies on a refinement of the algorithm of model construction: when T satisfies C_1, C_2, C_3, C_4 then for all $\nu, \nu' \in \mathsf{VN}$ we have $\mathsf{Sem}_0(T) \models [\nu \cap \nu' = \emptyset] \iff \mathcal{D}^*(\nu, \nu')$.

We conclude this section by combining Lemmas 8,10, and 11 to prove the following theorem.

Theorem 1. *We can decide whether a gitree entails a* QFCSC *formula in polynomial time.*

5 Testing Entailment of Itrees in Polynomial-Time

The test of entailment between two arbitrary gitrees ($T \models T'$) is complicated by the existential quantifiers in the semantics of T' which prevent us from decomposing $\mathsf{Sem}(T')$ into a conjunction of independent atomic formulas. However, if we can express a gitree T' as a QFCSC formula, the previous section yields a polynomial-time algorithm for checking whether a gitree entails T'. In this section we show that the condition

$$\sigma^{-1}(\nu) = \emptyset \quad \Rightarrow \quad \mathsf{CInf}(\nu) = 0 \land \mathsf{CSup}(\nu) = \infty \tag{QE}$$

in the definition of itrees ensures that we can indeed compute a QFCSC formula associated with the itree, which motivates the definition of itrees as a subclass of gitrees.

As a first step, the following lemma gives a sufficient condition for a node of an itree (that is, a bound variable) to be expressible as a union of some free variables.

Lemma 12. *Given an itree T we define the unary predicate* Det *on* VN *as the least fixed point of*

$$\begin{array}{lcl}
\mathsf{Det}(\nu) & \Leftarrow & \sigma^{-1}(\nu) \neq \emptyset \\
\mathsf{Det}(\nu) & \Leftarrow & \blacksquare \in \mathsf{M}(\nu) \land \forall \nu' \in \mathsf{Sons}(\nu).\ \mathsf{Det}(\nu') \\
\mathsf{Det}(\nu) & \Leftarrow & \mathsf{M}(\nu) = \{\blacklozenge\} \land \exists \nu' \in \mathsf{Sons}(\nu).\ \mathsf{Det}(\nu') \\
\mathsf{Det}(\nu) & \Leftarrow & \exists \nu'.\ \nu \rightsquigarrow \nu' \land \mathsf{M}(\nu') = \{\blacklozenge\} \land \mathsf{Det}(\nu')
\end{array}$$

Then for each node ν we have: $\mathsf{Det}(\nu) \Rightarrow \big(\exists S_\nu \subseteq \mathsf{SN}.\ T \models [\nu = (\cup S_\nu)]\big)$
Moreover, the predicate Det *and a mapping $\nu \mapsto S_\nu$ are computable in polynomial time.*

When $\mathsf{Det}(\nu)$ holds for all nodes ν of a gitree T, it is clear that we can transform the formula $\phi = \mathsf{Sem}(T)$ into an equivalent formula ϕ' such that no quantified variable appears inside inclusion constraints or disjointness constraints. However, it is not sufficient to check that $\mathsf{Det}(\nu)$ holds for all nodes ν to ensure that $T \in \mathsf{QFCSC}$. Indeed, QFCSC only allows expressing cardinality constraints on free set variables and not on arbitrary union of set variables. It is for this reason that we are naturally interested in the class ITREE of gitrees for which non trivial cardinality constraints can only be enforced to nodes ν for which there exists $s \in \mathsf{SN}$ such that $\sigma(s) = \nu$.

Lemma 13. *For each itree $T \in \mathsf{ITREE}$ we can compute in polynomial time an equivalent itree $\mathcal{R}'^{\downarrow}(T)$ satisfying* $\mathsf{Det}(\nu)$ *for all nodes $\nu \in \mathsf{VN}$.*

if	apply
$\nu \rightsquigarrow \nu'$ $\mathsf{d}(\nu) = 0$ $\sigma^{-1}(\nu) = \emptyset$	$\mathsf{VN} := \mathsf{VN} - \{\nu\}$ $\mathsf{M}(\nu') := \mathsf{M}(\nu') - \{\blacksquare\}$
$\nu = \mathsf{Root}(T)$ $\mathsf{d}(\nu) = 0$ $\sigma^{-1}(\nu) = \emptyset$	$\mathsf{VN} := \mathsf{VN} - \{\nu\}$
$\mathsf{M}(\nu) \in \{\emptyset, \{\bigcirc\}\}$ $\nu \rightsquigarrow \nu'$ $\mathsf{M}(\nu') \neq \{\blacklozenge\}$ $\sigma^{-1}(\nu) = \emptyset$	$\mathsf{M}(\nu) := \mathsf{M}(\nu) \cup \{\blacksquare\}$ $\mathsf{M}(\nu') := \mathsf{M}(\nu') - \{\blacksquare\}$
$\mathsf{M}(\nu') = \{\blacklozenge\}$ $\forall \nu \in \mathsf{Sons}(\nu')$ $\quad \mathsf{M}(\nu) \in \{\emptyset, \{\bigcirc\}\}$ $\quad \sigma^{-1}(\nu) = \emptyset$	$\mathsf{M}(\nu') := \emptyset$ $\forall \nu \in \mathsf{Sons}(\nu')$ $\quad \mathsf{M}(\nu) := \mathsf{M}(\nu) \cup \{\blacksquare\}$

$(\mathcal{R}'(\nu))$

Fig. 4. Rewriting rule $\mathcal{R}'$

Proof. Given an itree T we can first compute an itree T_1 equivalent to T and satisfying the condition C_1 relative to modes (see Lemma 5). Because $\mathcal{R}_1$ does not preserve the QE condition, we compute T_1 in three steps: 1) discard the cardinality constraints of T by applying $\mathsf{CInf}(\nu) := 0$ and $\mathsf{CSup}(\nu) := \infty$ to every node; 2) apply $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4$; and 3) recover the initial cardinality constraints on the nodes that remain in the tree. Step 2) makes some subtrees of T empty and changes $\mathsf{M}, \mathsf{CSup}, \mathsf{CInf}$ for existing nodes, but never introduces new nodes or causes $\sigma^{-1}(\nu) = \emptyset$ to hold for additional nodes that remain in the tree. Thus, QE holds after step 3). We can compute the final itree $\mathcal{R}'^{\downarrow}(T)$, equivalent to T and T_1, by applying the rewriting rule $\mathcal{R}'$ of Figure 4 to T_1 using a bottom-up strategy.

Lemma 14 ($\mathsf{ITREE} \subseteq \mathsf{QFCSC}$)**.** *For each itree $T \in \mathsf{ITREE}$ we can compute in polynomial time an equivalent formula of* QFCSC*.*

Given a mapping $\nu \mapsto S_\nu$ from Lemma 12, this QFCSC formula can be computed from $\mathsf{Sem}(\mathcal{R}'^{\downarrow}(T))$ by first substituting each ν with $\cup S_\nu$ in the formula $\mathsf{Sem}(\mathcal{R}'^{\downarrow}(T))$ and then eliminating the quantifiers.

Figure 5 gives an example of an application of $\mathcal{R}'$ to an itree and indicates which substitution can finally be applied to obtain a quantifier-free formula. Finally, combining Lemma 14 and Theorem 1, we obtain the main theorem of this paper.

Theorem 2. *We can decide entailment of itrees in polynomial time.*

6 Related Work

We are not aware of any previously known constraints on sets with cardinality constraints that have polynomial-time entailment while supporting all the constraints present in the ITREE class.

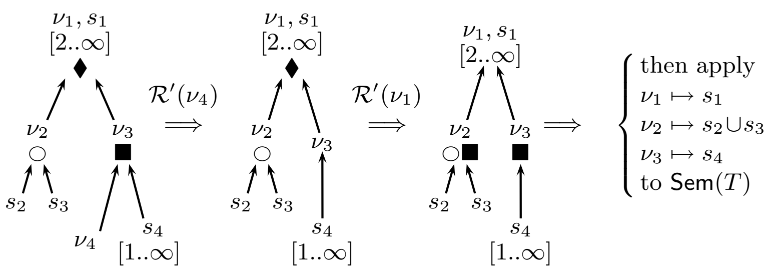


Fig. 5. Use of $\mathcal{R}'$ on an itree and quantifier elimination

Set algebras with cardinalities. Quantified formulas of boolean algebra are complete for the class of alternating exponential time with a linear number of alternations [15], and even a small number of alternations leads to exponential complexity [13]. Cardinality constraints naturally arise in quantifier elimination for boolean algebras [21]. The quantifier-free case of Boolean Algebra with Presburger Arithmetic is described in [5, Section 11], [28] with an non-deterministic exponential time decision procedure, which is also achieved as a special case of [18, 19, 24], [10, Section 8, Page 90]. Recently, [16, Section 7.9] gave a non-deterministic *polynomial-time* algorithm for quantifier-free Boolean Algebra with Presburger Arithmetic. All these constraints are NP-hard.

Description logics. Description logics [3] can reason about sets (concepts) and relations (roles). However, polynomial-time description logics such as the ones described in [8, Section 7] and [2], [3, Section 3.9.2] do not support set unions; the presence of union is generally considered to lead to intractability. Note also that the subsumption in the context of description logic typically refers to testing $A \subseteq B$ for two defined concepts A and B, as opposed to testing whether a *conjunction* of constraints on sets implies another constraint on sets, as in our case. Furthermore, cardinality constraints in description logics typically apply to a relation and are used to designate a new set, as opposed to imposing a constraint on an existing set.

Horn clause fragments. Polynomial-time fragments of first-order logic Horn clauses such as [23, 12] can in principle encode some relationships on sets by representing them as predicates, but they do not support cardinality constraints.

Constraint satisfaction problems. Constraint satisfaction problems (CSP) [7] also identify the important idea of propagating constraints along tree-like structures. For example, the Yannakakis algorithm has linear time complexity for the satisfiability of acyclic sets of constraints [27]. However, such algorithms typically work on concrete domains such as booleans or integers; we are not aware of their application to constraints that involve set variables along with their cardinalities. Indeed, an attempt to generalize itrees to acyclic graphs yields NP-hard constraints. Note that representing the values of set variables explicitly (as done in many constraint satisfaction problems over finite domains) would result in exponentially large models. Like [8, Section 7], our polynomial algorithm avoids this problem using polynomial representation of models, but, unlike [8, Section 7], can express conjunctions of constraints of the form $A = B \cup C$.

Tree-width. The notion of tree-width [25] can be used as a measure of the "treeness" of a conjunctive formula and often leads to polynomial results on classes of formulas with bounded tree-width. However, although inclusion constraints in an itree form a tree and syntactically have bounded tree-width, disjointness and union constraints introduce dependencies between siblings of a tree. Therefore, the overall tree-width of an itree formula is not bounded. Similarly, the result [6], stating that monadic second-order logic queries over graph structures of bounded tree-width are polynomial, does not seem to simplify the problem of checking entailment (or satisfiability) of itrees. Indeed, there is no natural way of representing, for example, cardinality bounds on sets in monadic second-order logic.

Constraints in program analysis. Set constraints [1,4] are incomparable to our constraints. On the one hand, set constraints are interpreted over ground terms and contain operations that apply a given free function symbol to each element of the set. On the other hand, unlike our constraints, set constraints do not support cardinality operators.

References

1. A. Aiken, D. Kozen, M. Vardi, and E. Wimmers. The complexity of set constraints. In *Proceedings of Computer Science Logic 1993*, 1993.
2. F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proc. 19th Int. Joint Conf. on Artificial Intelligence IJCAI-05*, 2005.
3. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications.* CUP, 2003.
4. L. Bachmair, H. Ganzinger, and U. Waldmann. Set constraints are the monadic class. In *Logic in Computer Science*, pages 75–83, 1993.
5. D. Cantone, E. Omodeo, and A. Policriti. *Set Theory for Computing.* Springer, 2001.
6. B. Courcelle. The monadic second-order logic of graphs III: tree-decompositions, minor and complexity issues. *ITA*, 26:257–286, 1992.
7. R. Dechter. *Constraint Processing.* Morgan-Kaufmann, 2003.
8. F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. *Information and Computation*, 134(1):1–58, 1997.
9. S. Drossopoulou, F. Damiani, M. Dezani-Ciancaglini, and P. Giannini. More Dynamic Object Re-classification: FickleII. *ACM Trans. Programming Languages and Systems*, 24(2):153–191, 2002.
10. S. Feferman and R. L. Vaught. The first order properties of products of algebraic systems. *Fundamenta Mathematicae*, 47:57–103, 1959.
11. M. Fowler. *UML Distilled (Second Edition).* Addison-Wesley, Reading, Mass., 2000.
12. R. Givan and D. Mcallester. Polynomial-time computation via local inference relations. *ACM Trans. Comput. Logic*, 3(4):521–541, 2002.

13. E. Grädel. Domino games with an application to the complexity of boolean algebras with bounded quantifier alternations. In *STACS*, pages 98–107, 1988.
14. D. Jackson. *Software Abstractions: Logic, Language, & Analysis*. MIT Press, 2006.
15. D. Kozen. Complexity of boolean algebras. *Theoretical Computer Science*, 10: 221–247, 1980.
16. V. Kuncak. *Modular Data Structure Verification*. PhD thesis, EECS Department, Massachusetts Institute of Technology, February 2007.
17. V. Kuncak, P. Lam, and M. Rinard. Role analysis. In *Annual ACM Symp. on Principles of Programming Languages (POPL)*, 2002.
18. V. Kuncak, H. H. Nguyen, and M. Rinard. Deciding Boolean Algebra with Presburger Arithmetic. *J. of Automated Reasoning*, 2006. http://dx.doi.org/10.1007/s10817-006-9042-1.
19. V. Kuncak and M. Rinard. The first-order theory of sets with cardinality constraints is decidable. Technical Report 958, MIT CSAIL, July 2004.
20. P. Lam, V. Kuncak, and M. Rinard. Generalized typestate checking for data structure consistency. In *6th Int. Conf. Verification, Model Checking and Abstract Interpretation*, 2005.
21. L. Loewenheim. Über Mögligkeiten im Relativkalkül. *Math. Annalen*, 76:228–251, 1915.
22. B. Marnette, V. Kuncak, and M. Rinard. On algorithms and complexity for sets with cardinality constraints. Technical report, MIT CSAIL, August 2005.
23. F. Nielson, H. R. Nielson, and H. Seidl. Normalizable Horn clauses, strongly recognizable relations, and Spi. In *SAS*, pages 20–35, 2002.
24. P. Revesz. Quantifier-elimination for the first-order theory of boolean algebras with linear cardinality constraints. In *Proc. Advances in Databases and Information Systems (ADBIS'04)*, 2004.
25. N. Robertson and P. D. Seymour. Graph minors. ii. algorithmic aspects of treewidth. *J. Algorithms*, 7(3):309–322, 1986.
26. J. G. Schmolze and T. A. Lipkis. Classification in the KL-ONE knowledge representation system. In *IJCAI*, pages 330–332, 1983.
27. M. Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, pages 82–94, 1981.
28. C. G. Zarba. Combining sets with cardinals. *J. of Automated Reasoning*, 34(1), 2005.

A Lower Bound on Web Services Composition

Anca Muscholl⋆ and Igor Walukiewicz⋆

LaBRI, Université Bordeaux 1 and CNRS
351, Cours de la Libération
F-33 405, Talence cedex, France

Abstract. A web service is modeled here as a finite state machine. A composition problem for web services is to decide if a given web service can be constructed from a given set of web services; where the construction is understood as a simulation of the specification by a fully asynchronous product of the given services. We show an EXPTIME-lower bound for this problem, thus matching the known upper bound. Our result also applies to richer models of web services, such as the Roman model.

Keywords: Automata simulation, complexity, web services composition.

1 Introduction

Inherently distributed applications such as web services [1] increasingly get into the focus of automated verification techniques. Often, some basic e-services are already implemented, but no such simple service can answer to a more complex query. For instance, a user interested in hiking Mt. Everest will ask a travel agency for information concerning weather forecast, group travels, guides etc. The travel agency will contact different e-services, asking for such information and making appropriate reservations, if places are available. In general, single services such as weather forecast or group reservations, are already available and it is important to be able to reuse them without any change. The task of the travel agency is to compose basic e-services in such a way that the user's requirements are met (and eventually some constraints wrt. the called services, such as avoiding unreliable ones). Thus, one main objective is to be able to check automatically that the composition of basic e-services satisfies certain desirable properties or realizes another complex e-service.

In this paper we study a problem that arises in the *composition* of e-services as considered in [2,3,4]. The setting is the following: we get as input a specification (goal) $\mathcal{B}$, together with n available services $\mathcal{A}_1, \ldots, \mathcal{A}_n$. Then we ask whether the composition of the services $\mathcal{A}_i$ can simulate the behavior of the goal $\mathcal{B}$. This problem is known as *composition synthesis*. It amounts to synthesize a so-called *delegator*, that tells at any moment which service must perform an action. In essence, a delegator corresponds to a simulation of the goal service

⋆ Work supported by the ACI Sécurité Informatique VERSYDIS.

$\mathcal{B}$ by the composition of the available services $\mathcal{A}_i$. In the most general setting, as considered for instance in [9,8], services are modeled by communicating finite state machines [5], that have access to some local data. In this paper, we reconsider the simplified setting of the so-called Roman model [2] where services are finite state processes with no access to data. This restriction is severe, but it captures some quite natural cases. First, messages exchanged by services are often synchronous (hand-shaking), which means that we do not need the full power of communication channels. Second, even when data-driven web applications are considered, some restrictions on data are needed. For instance, [7] assumes that specific user information is considered as constants in the data base scheme.

The main result of this paper is the EXPTIME lower bound for the composition synthesis problem in the very simple setting where the composition of the finite state machines $\mathcal{A}_i$ is fully asynchronous (in particular there is no communication). We also show that the same question can be solved in polynomial time if we assume that the sets of actions of the available machines are pairwise disjoint, i.e., each request can be handled by precisely one service. Note that in the latter case, the set of actions depends of course on the number of processes, whereas for the first result we show that the case where the set of actions is fixed is already EXPTIME-hard. Thanks to the simplicity of the considered model the same lower bounds hold also for much richer frameworks, as for example Colombo [3]. For the Roman model a matching EXPTIME upper bound is known [2]. The complexity of the composition problem for Colombo depends on restrictions of the model and is undecidable in the most general case.

As related work, it is worth mentioning the approach of Pistore et al. [11] who use planning techniques. The other difference is that the final goal is specified by a formula, and not as a simulation condition as we have here. Moreover, the accent there is put on satisfying one demand, i.e., constructing a sequence of actions rather than a transition system, i.e. a new service. The other possibility is to consider bisimulation instead of simulation relation. This corresponds to the so-called *orchestration problem*, where the issue is to find a communication architecture of the available services, that is equivalent to the goal, modulo bisimulation. We think that this is less natural in our simple setting, mainly due the nature of the service composition which is modeled here as a fully asynchronous product. Bisimulation requirement would mean also that the client should be prepared to admit all the interleavings possible in the composition, which usually makes the specification of the client's goal too complex. A result that is closely related to ours is the EXPTIME completeness of the simulation and bisimulation problems between non-flat systems [10]. The main difference to our setting is that both system and services are given as composition of finite state machines using (binary) synchronization on actions, i.e., an action can synchronize two services. In a sense this paper shows that the lower bound for the simulation relation holds even without any synchronization.

2 Notations

An *asynchronous* product of n deterministic automata

$$\mathcal{A}_i = \langle Q_i, \Sigma_i, q_i^0, \delta_i : Q_i \times \Sigma_i \to Q_i \rangle$$

is a nondeterministic automaton:

$$\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n = \langle Q, \Sigma, \boldsymbol{q}, \delta : Q \times \Sigma \to \mathcal{P}(Q) \rangle$$

where: $Q = Q_1 \times \cdots \times Q_n$; $\Sigma = \bigcup_{i=1,\ldots,n} \Sigma_i$; $\boldsymbol{q} = (q_1^0, \ldots, q_n^0)$; and δ is defined by:

$\boldsymbol{t} \in \delta(\boldsymbol{s}, a)$ iff for some i, $t_i = \delta_i(s_i, a)$ and for all $j \neq i$ we have $t_j = s_j$.

Observe that the product automaton can be non deterministic because the alphabets Σ_i are not necessarily disjoint.

We define a *simulation relation* on nondeterministic automata in a standard way. Take two nondeterministic automata $\mathcal{A} = \langle Q_A, \Sigma, q_A^0, \delta_A : Q_A \times \Sigma \to \mathcal{P}(Q_A) \rangle$ and $\mathcal{B} = \langle Q_B, \Sigma, q_B^0, \delta_B : Q_B \times \Sigma \to \mathcal{P}(Q_B) \rangle$ over the same alphabet. The simulation relation $\preccurlyeq \subseteq Q_A \times Q_B$ is the biggest relation such that if $q_A \preccurlyeq q_B$ then for every $a \in \Sigma$ and every $q'_A \in \delta_A(q_A, a)$ there is $q'_B \in \delta_B(q_B, a)$ such that $q'_A \preccurlyeq q'_B$. We write $\mathcal{A} \preccurlyeq \mathcal{B}$ if $q_A^0 \preccurlyeq q_B^0$.

Problem: Given n deterministic automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$ and a deterministic automaton $\mathcal{B}$ decide if $\mathcal{B} \preccurlyeq \mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$.

We will show that this problem is Exptime-complete. It is clearly in Exptime as one can construct the product $\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$ explicitly and calculate the biggest simulation relation with $\mathcal{B}$. The rest of this paper will contain the proof of Exptime-hardness. We will start with the Pspace-hardness, as this will allow us to introduce the method and some notation.

3 Pspace-Hardness

We will show Pspace-hardness of the problem by reducing it to the existence of a looping computation of a linearly space bounded deterministic Turing machine. The presented proof of the Pspace bound has the advantage to generalize to the encoding of alternating machines that we will present in the following section.

Fix a deterministic Turing machine M working in space bounded by the size of its input. We want to decide if on a given input the computation of the machine loops. Thus we do not need any accepting states in the machine and we can assume that there are no transitions from rejecting states. We denote by Q the states of M and by Γ the tape alphabet of M. A *configuration* of M is a word over $\Gamma \cup (Q \times \Gamma)$ with exactly one occurrence of a letter from $Q \times \Gamma$. A configuration is of size of n if it is a word of length n. Transitions of M will be denoted as $qa \longrightarrow q'bd$, where q, q' are the old/new state, a, b the old/new tape symbol and $d \in \{l, r\}$ the head move.

Suppose that the input is a word w of size n. We will construct automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$ and $\mathcal{B}$ such that $\mathcal{B} \preccurlyeq \mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$ iff the computation of M on w is infinite.

We start with some auxiliary alphabets. For every $i = 1, \ldots, n$ let

$$\Gamma_i = \Gamma \times \{i\} \quad \text{and} \quad \Delta_i = (Q \times \Gamma_i) \cup (Q \times \Gamma_i \times \{l, r\}) .$$

We will write a_i instead of (a, i) for elements of Γ_i. Let also $\Delta = \bigcup_{i=1,\ldots,n} \Delta_i$.

The automaton $\mathcal{A}_i = \langle Q_i, \Sigma_i, q_i^0, \longrightarrow \rangle$ is defined as follows:

– The set of states is $Q_i = \Gamma \cup (Q \times \Gamma) \cup \{\top\}$, and the alphabet of the automaton is $\Sigma_i = \Delta$.
– We have transitions:
  - $a \xrightarrow{qa_i} qa$, for all $a \in \Gamma$ and $q \in Q$,
  - $qa \xrightarrow{q'b_id} b$, for $qa \to q'bd$ the transition of M on qa (there is at most one).
  - From a, transitions on letters in $\Delta_i \setminus \{qa_i : q \in Q\}$ go to $\top$. Similarly, from qa transitions on $\Delta_i \setminus \{qb_id\}$ go to $\top$ if there is a transition of M on qa; if not, then qa has no outgoing transitions. From $\top$ there are self-loops on all letters from Δ.
– For $i = 2, \ldots, n$ the initial state of $\mathcal{A}_i$ is w_i, the i-th letter of w; for $\mathcal{A}_1$ the initial state is $q^0 w_1$, i.e., the initial state of M and the first letter of w.

Figure 1 shows a part of $\mathcal{A}_i$:

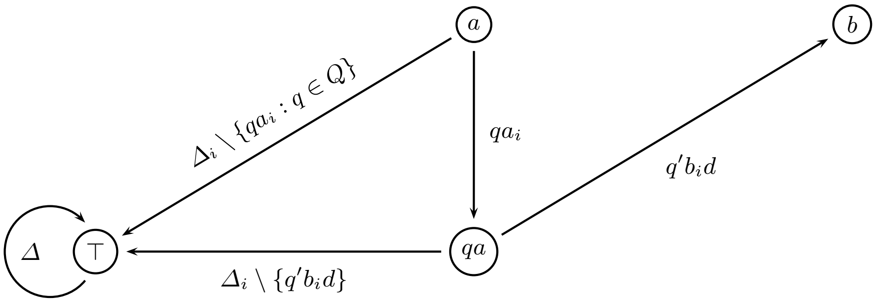

a
b
$\Delta_i \setminus \{qa_i : q \in Q\}$
qa_i
$q'b_id$
Δ
$\top$
qa
$\Delta_i \setminus \{q'b_id\}$

Fig. 1. Part of $\mathcal{A}_i$

The idea is classical: automaton $\mathcal{A}_i$ controls the i-th tape symbol, whereas automaton $\mathcal{B}$ defined below is in charge of the control part of M. The challenge is to do this without using any synchronization between adjacent automata $\mathcal{A}_i, \mathcal{A}_{i+1}$. Next, we introduce an automaton K that will be then used to define $\mathcal{B}$. The set of states of K is $Q_K = \{s, e\} \cup (Q \times \bigcup \Gamma_i \times \{l, r\})$; the initial state is s and the final one e; the alphabet is Δ; the transitions are defined by:

– $s \xrightarrow{q'b_ir} q'b_ir$ for $i = 1, \ldots, n-1$, whenever we have a transition $qa \to q'br$ in M for some state q and some letter a;

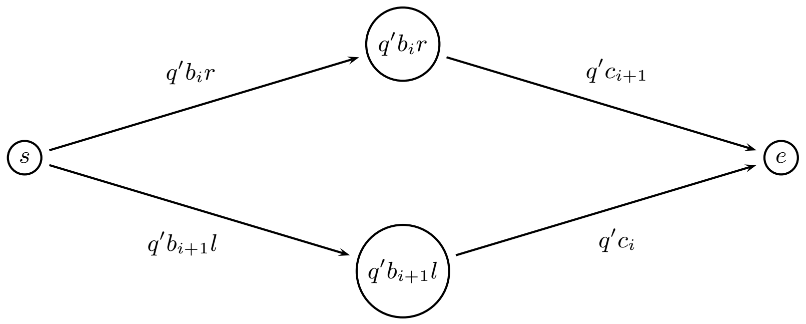


Fig. 2. Automaton K

- $s \xrightarrow{q'b_{i+1}l} q'b_{i+1}l$ for $i = 2, \dots, n$, whenever we have a transition $qa \to qbl$ in M for some state q and some letter a;
- $q'b_ir \xrightarrow{q'c_{i+1}} e$ and $q'b_{i+1}l \xrightarrow{q'c_i} e$ for all $c \in \Gamma$.

We define $\mathcal{B}$ as the minimal deterministic automaton recognizing $(L(K))^*$. In other words, $\mathcal{B}$ is obtained by gluing together states s and e. Figure 2 is a schema of the automaton K.

Remark 1. All $\mathcal{A}_i$ and $\mathcal{B}$ are deterministic automata of size polynomial in n. The input alphabets of the $\mathcal{A}_i$ are almost pairwise disjoint: the only states with common labels on outgoing transitions are the $\top$ states.

Definition 1. *We say that a configuration C of size n of M* corresponds *to a global state $\boldsymbol{s}$ of $\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$ iff $s_i = C(i)$ for $i = 1, \dots, n$; in other words, if the state of $\mathcal{A}_i$ is the same as the i-th letter of C.*

Definition 2. *We say that a global state $\boldsymbol{s}$ of $\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$ is* proper *when there is no $\top$-state in $\boldsymbol{s}$.*

Lemma 1. *If $\boldsymbol{s}$ is a proper state then for every letter $a \in \Delta$ there is at most one transition of $\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$ from $\boldsymbol{s}$ on a. Once the automaton enters a state that is not proper it stays in non proper states.*

It is easy to see that from a non proper state, $\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$ can simulate any state of $\mathcal{B}$. The reason is that from $\top$, any move on letters from Δ is possible.

Lemma 2. *Suppose that $\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$ is in a state $\boldsymbol{s}$ that corresponds to a configuration C of M.*

- *If C is a configuration with no successor, then there is a word $v \in L(K)$ that cannot be simulated by $\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$ from $\boldsymbol{s}$.*
- *Otherwise the successor configuration $C \vdash C'$ exists, and there is a unique word $v \in L(K)$ such that $\boldsymbol{s} \xrightarrow{v} \boldsymbol{t}$ and $\boldsymbol{t}$ is proper. Moreover $\boldsymbol{t}$ corresponds to C'. All other words from $L(K)$ lead to non proper configurations of $\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$.*

Proof. For the first claim, assume that $\boldsymbol{s}$ corresponds to a configuration, thus there is exactly one i such that $\mathcal{A}_i$ is in a state from $Q \times \Gamma$. The other automata are in states from Γ.

If C is terminal then $\mathcal{A}_i$ is in a state qa which has no outgoing transition. This means that this state can simulate no move on letters $q'b_ir$, for $q' \in Q$ and $b_i \in \Gamma_i$ (and such a move exists in K, as the machine M must have a move to the right if it is nontrivial). All other automata are also not capable to simulate $q'b_ir$ as they can do only moves on letters Δ_j for $j \neq i$.

Now suppose that $C \vdash C'$. To avoid special, but simple, cases suppose that the position i of the state is neither the first nor the last. Let $s_i = qa$ and suppose also that $qa \to q'br$ is the move of M on qa. The case when the move is to the left is similar.

The only possible move of K from s which will put $\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$ into a proper state is $q'b_ir$. This makes $\mathcal{A}_i$ to change the state to b and it makes K to change the state to $q'b_ir$. From this latter state the only possible move of K is on letters $q'c'_{i+1}$ for arbitrary $c' \in \Gamma$. Suppose that $\mathcal{A}_{i+1}$ is in the state $c = s_{i+1} \in \Gamma$, then all moves of K on $q'c'_{i+1}$ with $c' \neq c$ can be matched with a move to $\top$ of $\mathcal{A}_{i+1}$. On $q'c_{i+1}$ the automaton $\mathcal{A}_{i+1}$ goes to $q'c$ and automaton K goes to e. This way the state in the configuration is changed and transmitted to the right. We have that the new state of $\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$ corresponds to the configuration C'.

Lemma 3. *We have $\mathcal{B} \preccurlyeq \mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$ iff M has an infinite computation.*

Proof. Recall that $\mathcal{B}$ is the minimal deterministic automaton recognizing $(L(K))^*$, and has initial state s. The initial state of $\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$ corresponds to the initial configuration C_0 of M. We show that $s \preccurlyeq \boldsymbol{t}$ with $\boldsymbol{t}$ corresponding to a configuration C of M, iff the computation of M starting in C is infinite.

From a configuration C, machine M has only one computation: either infinite, or finite that is blocking. Suppose that the computation from C has at least one step and let C_1 be the successor configuration. By Lemma 2 from state s there is exactly one word $v_1 \in L(K)$ such that $\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$ in order to simulate it is forced to go to a proper state $\boldsymbol{t}_1$. Morover $\boldsymbol{t}_1$ corresponds to C_1. On all other words from $L(K)$, the product $\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$ can go to a non proper state and from there it can simulate any future behaviour of $\mathcal{B}$. If C_1 has no successor configuration then, again by Lemma 2, there is a word in $L(K)$ that cannot be simulated by $\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$ from $\boldsymbol{t}_1$. If C_1 has a successor then we repeat the whole argument. Thus the behaviour of $\mathcal{B}$ from s can be simulated by $\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$ from the state corresponding to C iff the machine M has an infinite computation starting from C.

One can note that the construction presented in this section uses actions that are common to several processes in a quite limited way: the only states that have common outgoing labels are the $\top$ states from which all behaviours are possible. This observation motivates the question of the complexity when the automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$ have pairwise disjoint alphabets. With this restriction, the simulation problem can be solved efficiently:

Theorem 1. *The following question can be solved in polynomial time:*
Input: *n deterministic automata $\mathcal{A}_1, \dots, \mathcal{A}_n$ over pairwise disjoint input alphabets, and a deterministic automaton $\mathcal{B}$.*
Output: *decide if $\mathcal{B} \preccurlyeq \mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$.*

Proof. Let $\mathcal{C}_i$ be a automaton with a single state $\top$, and with self-loops on every letter from the alphabet Σ_i of $\mathcal{A}_i$. We write $\mathcal{A}^{(i)}$ for the asynchronous product of all $\mathcal{C}_j$, $j \neq i$, and of $\mathcal{A}_i$. Similarly, $\boldsymbol{t}^{(i)}$ will denote $\boldsymbol{t}$ with all components but i replaced by $\top$. Suppose now that p is a state of $\mathcal{B}$, and $\boldsymbol{t}$ a state of $\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$. We write $p \preccurlyeq_i \boldsymbol{t}$ if p is simulated by $\boldsymbol{t}^{(i)}$ in $\mathcal{A}^{(i)}$. Notice that since $\mathcal{B}$ and $\mathcal{A}_i$ are both deterministic, we can decide if $p \not\preccurlyeq_i \boldsymbol{t}$ in logarithmic space (hence in polynomial time), by guessing simultaneously a path in $\mathcal{B}$ and one in $\mathcal{A}_i$.

We show now that $p \preccurlyeq \boldsymbol{t}$ in $\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$ iff $p \preccurlyeq_i \boldsymbol{t}$ for all i.

If $p \preccurlyeq \boldsymbol{t}$, then all the more $p \preccurlyeq \boldsymbol{t}^{(i)}$, since $\mathcal{C}_j$ can simulate $\mathcal{A}_j$ for all $j = 1, \dots, n$. Conversely, assume that $p \preccurlyeq_i \boldsymbol{t}$ for all i, but $p \not\preccurlyeq \boldsymbol{t}$. This means that there exist computations $p \xrightarrow{a_1 \dots a_k} p'$ in $\mathcal{B}$, $\boldsymbol{t} \xrightarrow{a_1 \dots a_k} \boldsymbol{u}$ in $\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$ and a letter $a \in \Sigma_i$ for some i, such that p' has an outgoing a-transition, but $\boldsymbol{u}_i$ does not (in $\mathcal{A}_i$). Clearly, we also have a computation $\boldsymbol{t}^{(i)} \xrightarrow{a_1 \dots a_k} \boldsymbol{u}^{(i)}$ in $\mathcal{A}^{(i)}$. Since $\boldsymbol{u}_i$ has no outgoing a-transition, so neither does $\boldsymbol{u}^{(i)}$, which contradicts $p \preccurlyeq_i \boldsymbol{t}$.

4 Exptime-Hardness

This time we take an alternating Turing machine M working in space bounded by the size of the input. We want to decide if M has an infinite computation. This means that the machine can make choices of existential transitions in such a way that no matter what are the choices of universal transitions the machine can always continue. Clearly, one can reduce the word problem to this problem, hence it is Exptime-hard (see [6]; for more details on complexity see any standard textbook on complexity).

We will assume that M has always a choice between two transitions, i.e., for each non blocking state/symbol pair qa there will be precisely two distinct tuples $q'b'd'$, $q''b''d''$ such that $qa \to q'b'd'$ and $qa \to q''b''d''$. If q is existential then it is up to the machine to choose a move; if q is universal then the choice is made from outside. To simplify the presentation we will assume that $d' = d''$, i.e., both moves go in the same direction. Every machine can be transformed to an equivalent one with this property. We will also assume that the transitions are ordered in some way so we will be able to say that $qa \to q'b'd$ is the first transition and $qa \to q''b''d$ is the second one.

Suppose that the input word is w of size n. We will construct automata $\mathcal{A}'_1, \mathcal{A}''_1, \dots, \mathcal{A}'_n, \mathcal{A}''_n$ and $\mathcal{B}$ such that $\mathcal{B}$ is simulated by $\mathcal{A}'_1 \otimes \mathcal{A}''_1 \cdots \otimes \mathcal{A}'_n \otimes \mathcal{A}''_n$ iff there is an infinite alternating computation of M on w. The main idea is that automata $\mathcal{A}'_i$ and $\mathcal{A}''_i$ control the i-th tape symbol, as in the previous section, and each one is in charge of one of the two possible transitions (if any) when the input head is at position i in an existential state (universal moves are simpler).

We will modify a little the alphabets that we use. Let

$$\begin{aligned}\Delta'_i =& (Q \times \Gamma_i) \cup (Q \times \Gamma_i \times \{l, r\} \times \{1\})\\ \Delta''_i =& (Q \times \Gamma_i) \cup (Q \times \Gamma_i \times \{l, r\} \times \{2\})\end{aligned}$$

We then put $\Delta_i = \Delta'_i \cup \Delta''_i$, $\Delta = \bigcup_i \Delta_i$, $\Delta' = \bigcup_i \Delta'_i$ and $\Delta'' = \bigcup_i \Delta''_i$.

The automaton $\mathcal{A}'_i$ is defined as follows:

- The set of states is $Q'_i = \{\top\} \cup \Gamma \cup (Q \times \Gamma) \cup (Q \times \Gamma \times \{l, r\})$, the alphabet of the automaton is $\Sigma'_i = \Delta \cup \{\zeta\}$; where ζ is a new letter common to all automata.
- We have the following transitions:
  - $a \xrightarrow{qa_i} qa$ for all $a \in \Gamma$ and $q \in Q$,
  - $qa \xrightarrow{q'b'_i d1} b'$ and $qa \xrightarrow{q''b''_i d1} b''$ if q is an universal state and $qa \to q'b'd$, $qa \to q''b''d$ are the two transitions from qa. We have also transitions to $\top$ on all the letters from $\Delta'_i \setminus \{q'b'_i d1, q''b''_i d1\}$.
  - $qa \xrightarrow{\zeta} q'b'd \xrightarrow{q'b'_i d1} b'$ and $qa \xrightarrow{q''b''_i d1} b''$ if q is an existential state and $qa \to q'b'd$, $qa \to q''b''d$ are the first and the second transitions from qa, respectively. We have also transitions to $\top$ on all the letters from $\Delta'_i \setminus \{q'b'_i d1, q''b''_i d1\}$. From $q'b'd$ all transitions on $\Delta'_i \setminus \{q'b'_i d1\}$ go to $\top$.
  - From a, transitions on letters in $\Delta'_i \setminus \{qa_i : q \in Q\}$ go to $\top$. If qa is terminal then there are no outgoing transitions from qa. From $\top$ there are self-loops on all letters from $\Delta^c := \Delta \cup \{\zeta\}$.
- The initial state of $\mathcal{A}'_i$ is w_i, the i-th letter of w except for $\mathcal{A}_1$ whose initial state is $q^0 w_1$, the initial state of M and the first letter of w.

Figure 3 below presents parts of $\mathcal{A}'_i$ corresponding to universal and existential states.

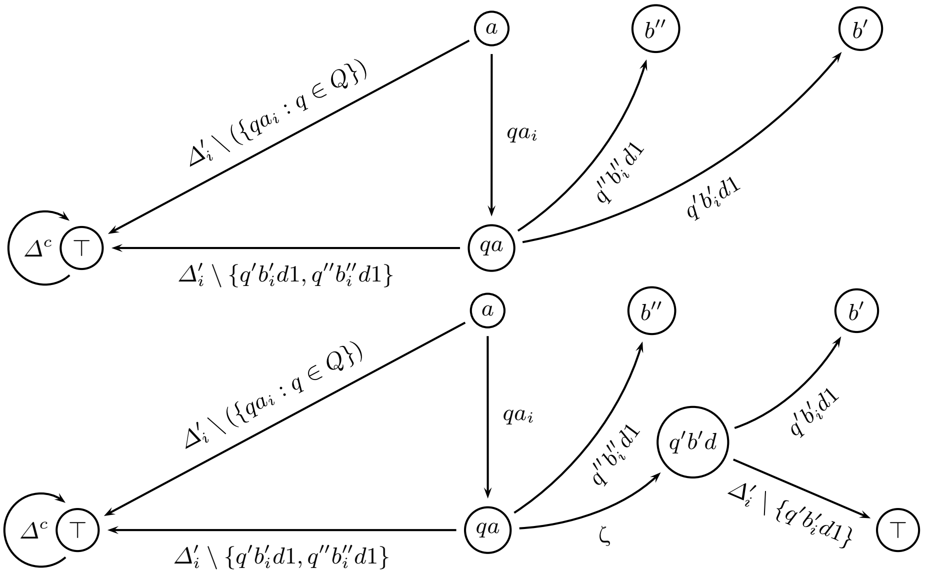
The automaton $\mathcal{A}''_i$ is the same as $\mathcal{A}'_i$ with the difference that we have $q'b'd2$ instead of $q''b''d1$, $q''b''d2$ instead of $q'b'd1$ (notice the change of primes and double primes), and Δ'' instead of Δ'.

Next, we define a new automaton K that will be used to define new automaton $\mathcal{B}$. The states of K are

$$Q_K = \{s, e, \mathit{choice}\} \cup (Q \times \bigcup_i \Gamma_i \times \{l, r\})$$

plus some auxiliary states to implement transitions on two letters at a time. We will write transitions with two letters on them for readability. The initial state is s and the final one is e. The alphabet is $\Sigma_K = \bigcup \Sigma_i$. The transitions are defined by (cf. Figure 4):

- $s \xrightarrow{\zeta} \mathit{choice}$;
- $s \xrightarrow{(q'b_i r1)(q'b_i r2)} q'b_i r$ whenever we have a transition $qa \to q'br$ in M for some universal state q and some letter a, and similarly from choice instead of s when q is existential;



a
b''
b'
$\Delta'_i \setminus (\{qa_i : q \in Q\})$
qa_i
$q''b''_i d1$
$q'b'_i d1$
Δ^c
$\top$
qa
$\Delta'_i \setminus \{q'b'_i d1, q''b''_i d1\}$
$q'b'd$
ζ
$\Delta'_i \setminus \{q'b'_i d1\}$

Fig. 3. Parts of the automaton $\mathcal{A}'_i$ corresponding to universal and existential states q, respectively. The alphabet Δ^c is $\Delta \cup \{\zeta\}$.

- $s \xrightarrow{(q'b_{i+1}l1)(q'b_{i+1}l2)} q'b_{i+1}l$ whenever we have a transition $qa \to q'bl$ in M for some universal state q and some letter a, and similarly from *choice* instead of s when q is existential;
- $q'b_ir \xrightarrow{(q'c_{i+1})^2} e$ and $q'b_{i+1}l \xrightarrow{(q'c_i)^2} e$ for all $c \in \Gamma$.

We define $\mathcal{B}$ as the minimal deterministic automaton recognizing $(L(K))^*$. It is obtained by gluing together states s and e.

Remark 2. All $\mathcal{A}'_i$, $\mathcal{A}''_i$ and $\mathcal{B}$ are deterministic and of size polynomial in n.

Definition 3. *A configuration C of size n* corresponds *to a global state $\boldsymbol{s}$ of $\mathcal{A}'_1 \otimes \mathcal{A}''_1 \cdots \otimes \mathcal{A}'_n \otimes \mathcal{A}''_n$ if $s_{2i} = s_{2i-1} = C(i)$ for $i = 1, \dots, n$; in other words, if the states of $\mathcal{A}'_i$ and $\mathcal{A}''_i$ are the same as the i-th letter of C.*

Definition 4. *We say that a global state $\boldsymbol{s}$ of $\mathcal{A}'_1 \otimes \mathcal{A}''_1 \cdots \otimes \mathcal{A}'_n \otimes \mathcal{A}''_n$ is* proper *when $\top$ does not appear in $\boldsymbol{s}$.*

It is easy to see that from a non proper state, $\mathcal{A}'_1 \otimes \mathcal{A}''_1 \cdots \otimes \mathcal{A}'_n \otimes \mathcal{A}''_n$ can simulate any state of $\mathcal{B}$. The reason is that from $\top$, any move on letters from Δ^c is possible.

Lemma 4. *Suppose that $\mathcal{A}'_1 \otimes \mathcal{A}''_1 \cdots \otimes \mathcal{A}'_n \otimes \mathcal{A}''_n$ is in a state $\boldsymbol{s}$ corresponding to a configuration C of M. If C has no successor configuration then there is a word $v \in L(K)$ that cannot be simulated by $\mathcal{A}'_1 \otimes \mathcal{A}''_1 \cdots \otimes \mathcal{A}'_n \otimes \mathcal{A}''_n$ from $\boldsymbol{s}$. Otherwise, C has two successor configurations $C \vdash C'$ and $C \vdash C''$. We have two cases:*

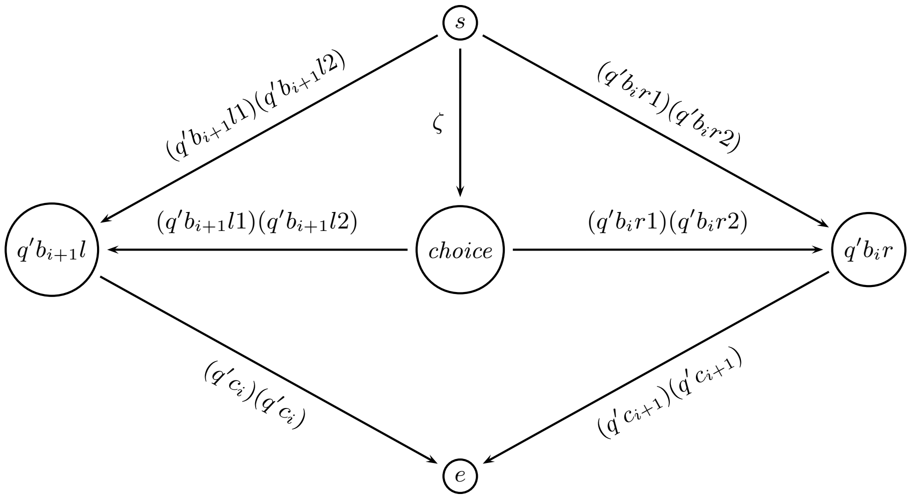


Fig. 4. Automaton K

- *If C is universal then there are two words v' and v'' in $L(K)$ that lead from $\boldsymbol{s}$ to proper states only, one state for v' and one for v''. These states correspond to C' and C'', respectively. On all other words from $L(K)$, non proper states can be reached from $\boldsymbol{s}$.*
- *If C is existential, then on the letter ζ the automaton $\mathcal{A}'_1 \otimes \mathcal{A}''_1 \cdots \otimes \mathcal{A}'_n \otimes \mathcal{A}''_n$ can reach only one of the two states $\boldsymbol{s'}$ or $\boldsymbol{s''}$. From $\boldsymbol{s'}$ there is a word v' such that $\zeta v' \in L(K)$ and on v' from $\boldsymbol{s'}$ the automaton $\mathcal{A}'_1 \otimes \mathcal{A}''_1 \cdots \otimes \mathcal{A}'_n \otimes \mathcal{A}''_n$ can reach a unique state, which moreover corresponds to C'. Similarly for $\boldsymbol{s''}$ and C''. On all words from $L(K)$ that are different from $\zeta v'$ and $\zeta v''$, non proper states can be reached from $\boldsymbol{s}$.*

Proof. As $\boldsymbol{s}$ corresponds to the configuration C, there is some i such that both automata $\mathcal{A}'_i$ and $\mathcal{A}''_i$ are in state qa, for some $q \in Q$ and $a \in \Gamma$, and all other automata are in states from Γ.

If C is a configuration without successor, then the state qa in $\mathcal{A}'_i$ and $\mathcal{A}''_i$ does not have any outgoing transition. Thus these automata cannot simulate the ζ transition of K from s. No other automaton $\mathcal{A}'_j$, or $\mathcal{A}''_j$ can simulate the ζ transition either, as they are all in states from Γ.

Suppose that C is an universal configuration with two possible transitions to the right, $qa \to q'b'r$ and $qa \to q''b''r$. The case when the moves are to the left is similar. In $\mathcal{A}'_i$ from the state qa we have a transition on $q'b'_ir1$ leading to b' and on $q''b''_ir1$ leading to b''. Similarly for $\mathcal{A}''_i$, but on $q'b'_ir2$ and $q''b''_ir2$. These transitions can simulate both transitions $(q'b'_ir1)(q'b'_ir2)$ and $(q''b''_ir1)(q''b''_ir2)$ that are possible from s in K. (All other transitions from s in K lead from $\boldsymbol{s}$ to a non proper state of $\mathcal{A}'_1 \otimes \mathcal{A}''_1 \cdots \otimes \mathcal{A}'_n \otimes \mathcal{A}''_n$.) Let us focus only on the first case, when $(q'b'_ir1)(q'b'_ir2)$ is executed in K and the state $q'b'_ir$ is reached. From this state only transitions $(q'c'_{i+1})^2$ are possible, for all $c' \in \Gamma$. Suppose that $\mathcal{A}'_{i+1}$

and $\mathcal{A}''_{i+1}$ are in state $c \in \Gamma$. Transition $(q'c_{i+1})^2$ of K is simulated by moves to $q'c$ in both $\mathcal{A}'_{i+1}$ and $\mathcal{A}''_{i+1}$. This way the new state is transferred to the right. Transitions $(q'c'_{i+1})^2$ where $c \neq c'$ are simulated in $\mathcal{A}'_1 \otimes \mathcal{A}''_1 \cdots \otimes \mathcal{A}'_n \otimes \mathcal{A}''_n$ by moves of $\mathcal{A}'_{i+1}$ and $\mathcal{A}''_{i+1}$ to $\top$.

Suppose that C is an existential configuration, with possible transitions $qa \to q'b'r$ and $qa \to q''b''r$. The case when moves are to the left is similar. Consider first the transition of K from s that corresponds to the letter ζ. Both $\mathcal{A}'_i$ and $\mathcal{A}''_i$ can simulate this transition: the first goes to state $q'b'r$, and the second goes to $q''b''r$. Assume that it is the transition of $\mathcal{A}'_i$ that is taken; the other case is symmetric. We get to the position when K is in the state *choice*, $\mathcal{A}'_i$ is in the state $q'b'r$ and $\mathcal{A}''_i$ in the state qa. From *choice*, automaton K can do $(q'b'_ir1)(q'b'_ir2)$ that can be simulated by the transitions of $\mathcal{A}'_i$ and $\mathcal{A}''_i$ (every other transition of K can be simulated by a move of $\mathcal{A}'_1 \otimes \mathcal{A}''_1 \cdots \otimes \mathcal{A}'_n \otimes \mathcal{A}''_n$ to a non proper state). Both automata reach the state b'. Automaton K is now in state $q'b_ir$ from where it can do $(q'c_{i+1})^2$ for any $c \in \Gamma$. The result of simulating these transitions while reaching a proper state is the transfer of the state to the right, in the same way as in the case of the universal move. Finally, it remains to see what happens if K makes a move from s that is different from ζ. In this case, at least one of the automata $\mathcal{A}'_i$, $\mathcal{A}''_i$ can simulate the corresponding transition on (pe_id1), (pe_id2) respectively, by going to state $\top$, since we suppose that in any configuration of M, the two outgoing transitions are distinct. Hence, a non proper state can be reached.

Theorem 2. *The following problem is* EXPTIME-*complete:*
Input: *deterministic automata* $\mathcal{A}_1, \dots, \mathcal{A}_n$ *and a deterministic automaton* $\mathcal{B}$.
Output: *decide if* $\mathcal{B} \preccurlyeq \mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$.

Proof. We use the construction presented above. By Lemma 4 we can show similarly to the previous section, that the initial state s of $\mathcal{B}$ can be simulated from a state $\boldsymbol{t}$ of $\mathcal{A}'_1 \otimes \mathcal{A}''_1 \cdots \otimes \mathcal{A}'_n \otimes \mathcal{A}''_n$ that corresponds to a configuration C of the alternating Turing machine M, iff M has an infinite alternating computation from C. The problem is clearly in EXPTIME as the state space of $\mathcal{A}'_1 \otimes \mathcal{A}''_1 \cdots \otimes \mathcal{A}'_n \otimes \mathcal{A}''_n$ can be constructed in EXPTIME.

We conclude the section by showing that Theorem 2 still holds under the assumption that the alphabet of the automata is of constant size.

Theorem 3. *Let Σ be a fixed alphabet of at least 2 letters. The following problem is* EXPTIME-*complete:*
Input: *deterministic automata* $\mathcal{A}_1, \dots, \mathcal{A}_n$ *and a deterministic automaton* $\mathcal{B}$ *over the input alphabet* Σ.
Output: *decide if* $\mathcal{B} \preccurlyeq \mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$.

Proof. We reduce directly from Theorem 2. Suppose that the input alphabet of all automata $\mathcal{A}_i, \mathcal{B}$ is $\Sigma \times \{1, \dots, m\}$, for some m. Moreover, let S be the set of states of $\mathcal{B}$ and let $Q = Q_1 \times \cdots \times Q_n$ be the set of global states of $\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$.

In each automaton $\mathcal{A}_i$, $\mathcal{B}$ we replace every transition $s \xrightarrow{a_l} t$ by a sequence of transitions with labels from $\Sigma \cup \{\#, \$\}$ as follows:

$$s \xrightarrow{a} (stl0) \xrightarrow{\#} (stl1) \xrightarrow{\#} (stl2) \cdots \xrightarrow{\#} (stll) \xrightarrow{\$} t$$

The $(l+1)$ states $(stl0), \ldots, (stll)$ are new. Let $\mathcal{A}'_i, \mathcal{B}'$ be the automata obtained from $\mathcal{A}_i$, $\mathcal{B}$, with state space Q' and S', respectively.

Take $\preccurlyeq$, the largest simulation relation from $\mathcal{B}$ to $\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_n$. We show how to extend $\preccurlyeq$ to $\preccurlyeq'$ such that $\preccurlyeq'$ is a simulation relation from $\mathcal{B}'$ to $\mathcal{A}'_1 \otimes \cdots \otimes \mathcal{A}'_n$ (not necessarily the largest one). Let $\preccurlyeq'$ be the union of $\preccurlyeq$ with the set of all pairs $((stlk), \boldsymbol{u}')$, where $s, t \in S$, $\boldsymbol{u}' = (u'_1, \ldots, u'_n) \in Q'$, and such that:

- $s \xrightarrow{a_l} t$ and $\boldsymbol{v} \xrightarrow{a_l} \boldsymbol{w}$ for some $a \in \Sigma$, $\boldsymbol{v} = (v_1, \ldots, v_n)$ and $\boldsymbol{w} = (w_1, \ldots, w_n)$ such that $s \preccurlyeq \boldsymbol{v}$, $t \preccurlyeq \boldsymbol{w}$,
- there is some i with $u'_i = (v_i w_i l k)$, and $u'_j = v_j = w_j$ for $j \neq i$.

It is immediate to check that $\preccurlyeq'$ is a simulation relation. First, (old) states from S can only be simulated by (old) states from Q. Second, a new state $(stlj)$ of $\mathcal{B}$ can be simulated only by states $\boldsymbol{u}' \in Q' \setminus Q$. It can be shown easily that the largest simulation relation from $\mathcal{B}'$ to $\mathcal{A}'_1 \otimes \cdots \otimes \mathcal{A}'_n$ coincides with $\preccurlyeq'$ (hence with $\preccurlyeq$) on the set $S \times Q$ of pairs of old states.

5 Conclusions

We have shown an Exptime lower bound for the composition of e-services that are described as a fully asynchronous product of finite state machines. Thus, we answer the question left open in [2]. Since our lower bound holds for the simplest model one can think of (no synchronization at all), it also applies to richer models, such as products with synchronization on actions as in [10] or communicating finite-state machines (CFSM) as in [9,8]. It is easy to see that the simulation of a finite-state machine by a CFSM $\mathcal{A}$ with bounded message queues is in Exptime, since the state space of $\mathcal{A}$ is exponential in this case. Hence, this problem, as well as any of its variants with some restricted form of communication, is Exptime-complete as well.

It remains open whether the bisimulation problem for a finite automaton and a fully asynchronous product of finite automata is also Exptime-hard. Another interesting question is how far one can relax the restrictions on e-services given by communicating finite-state machines, in order to preserve decidability.

Acknowledgement. We thank the anonymous referees for interesting comments and suggestions for improvement.

References

1. G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services. Concepts, Architectures and Applications.* Springer, 2004.
2. D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of e-services that export their behavior. In *Proc. of the 1st Int. Conf. on Service Oriented Computing (ICSOC 2003)*, volume 2910 of *Lecture Notes in Computer Science*, pages 43–58. Springer, 2003.
3. D. Berardi, D. Calvanese, G. D. Giacomo, R. Hull, and M. Mecella. Automatic composition of web services in colombo. In *SEBD 2005*, pages 8–15, 2005.
4. D. Berardi, D. Calvanese, G. D. Giacomo, R. Hull, and M. Mecella. Automatic composition of web services with messaging. In *VLDB 2005*, pages 613–624, 2005.
5. D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.
6. A. K. Chandra, D. Kozen and L. J. Stockmeyer. Alternation. *J. ACM* 28(1): 114–133, 1981.
7. A. Deutsch, L. Sui, and V. Vianu. Specification and verification of data-driven web services. In *Symposium on Principles of Database Systems (PODS)*, 2004.
8. X. Fu, T. Bultan, and J. Su. Conversation protocols: a formalism for specification and verification of reactive electronic services. In *Theor. Comput. Sci.* 328(1-2): 19–37, 2004.
9. R. Hull, M. Benedikt, V. Christophides, J. Su. E-services: a look behind the curtain. In *Symposium on Principles of Database Systems (PODS)*, pp. 1-14, 2003.
10. F. Laroussinie and Ph. Schnoebelen. The state explosion problem from trace to bisimulation equivalence. In *FoSSaCS 2000*, pages 192–207, 2000.
11. M. Pistore, A. Marconi, P. Bertoli, and P. Traverso. Automated composition of web services by planning at the knowledge level. In *International Joint Conference on Artificial Intelligence (IJCAI).* 2005.

Logical Characterizations of Bisimulations for Discrete Probabilistic Systems*

Augusto Parma and Roberto Segala

Dipartimento di Informatica - Università di Verona

Abstract. We give logical characterizations of bisimulation relations for the probabilistic automata of Segala in terms of three Hennessy-Milner style logics. The three logics characterize strong, strong probabilistic and weak probabilistic bisimulation, and differ only in the kind of diamond operator used. Compared to the Larsen and Skou logic for reactive systems, these logics introduce a new operator that measures the probability of the set of states that satisfy a formula. Moreover, the satisfaction relation is defined on measures rather than single states.

We rederive previous results of Desharnais et al. by defining sublogics for Reactive and Alternating Models viewed as restrictions of probabilistic automata. Finally, we identify restrictions on probabilistic automata, weaker than those imposed by the Alternating Models, that preserve the logical characterization of Desharnais et al. These restrictions require that each state either enables several ordinary transitions or enables a single probabilistic transition.

1 Introduction

Bisimulation relations are one of the simplest tools for the comparison of concurrent systems. Though they are quite detailed in their distinguishing power, they are mathematically simple and yet powerful, thus attracting a lot of interest in the literature. Axiomatizations and logical characterizations of relations are other important tools to improve our understanding of systems. In particular, axiomatizations permit to understand algebraic and compositional properties of processes, while logical characterizations permit to understand what properties are preserved under different equivalences or preorders. Axiomatizations and logical characterizations are also very useful for comparative analysis of relations.

Randomization within concurrent systems has received considerable attention as well in the literature (cf. the surveys in [1,16,13]). Our interest in this paper is in the model of Probabilistic Automata [12], which are a conservative extension of Labeled Transition Systems. They are also known as the *non-alternating* model of concurrency as opposed to the Labeled Concurrent Markov Chains of [7,11], called *alternating* models. Specifically, the alternating models impose some distinction between states that enable several ordinary transitions and states that enable a unique probabilistic transition, while in the non-alternating model each

* Supported by INRIA project ProNoBiS and MIUR project AIDA.

state may enable several probabilistic transitions. In other words, in the alternating models there is a total separation between probability and nondeterminism, while in the non-alternating models probability and nondeterminism coexist in the same states.

The goal of this paper is to study logical characterizations of bisimulation relations for probabilistic automata in the context of Hennessy-Milner style logics [8]. Logical characterizations have been studied already by Larsen and Skou [10] for reactive systems [6] and by Desharnais et al. [5] for labeled concurrent Markov chains (*alternating model*) [7,11]. These logics are derived from the Hennessy-Milner logic by replacing the diamond operator with a probabilistic diamond operator that measures bounds on the probability of performing an externally visible action and then satisfying some formula. Unfortunately, such characterizations are not adequate for probabilistic automata where, as opposed to reactive systems and alternating models, each state may enable both nondeterministic and probabilistic transitions.

Our main contribution is a Hennessy-Milner logic that keeps the original diamond operator of [8], is defined on measures over states rather than on single states and includes a new operator $[\varphi]_p$ that is true whenever the probability of the states that satisfy a formula φ is at least p. Thus, for instance, a conjunction of formulas with such operator can characterize entire probability measures. This means that our logical characterization permits to describe situations where different probability measures can be reached from a given state via transitions labeled by the same action. We study three logics for strong, strong probabilistic and weak probabilistic bisimulation, respectively, each one differing only on the definition of the diamond operator to account for the kind of transitions that are used in the definition of the bisimulation relation.

We then view the logics studied for the reactive and alternating models as restrictions of our logic, where $\Diamond_p a\varphi$ can be encoded by $\Diamond a[\varphi]_p$, and we rederive the known logical characterizations for such systems. In particular, for the alternating models we study minimal restrictions to impose on probabilistic automata so that the logical characterizations of [5] continue to hold. It turns out that it is sufficient to require that each state that enables a probabilistic transition enables only one transition, that is, each probabilistic choice should have a state that describes it. Indeed, by this restrictions, we obtain a model that is more general than the alternating models of [11,5], and the characterization of bisimulation in terms of maximal probabilities [11,5], the key technical machinery to derive the corresponding logical characterizations, continues to hold.

The paper is organized as follows. Section 2 gives some preliminary mathematical notions; Section 3 defines probabilistic automata and related concepts; Section 4, recalls the definition for Hennessy-Milner logic [8] and introduces our logics for probabilistic automata; Section 5 recalls the results for reactive and alternating systems and compares them with our results; Section 6 gives two logics for strong and weak probabilistic bisimulation for the restriction of probabilistic automata where states that enable probabilistic transitions enable only one transition, which embed all known alternating models.

2 Mathematical Preliminaries

Given a set S, a *σ-algebra* over S is a family Σ of subsets of S that is closed under complementation and countable union. A *measurable space* is the pair (S, Σ), and each element of Σ is called a *measurable set*. The σ-algebra *generated* by a family G of subsets of S is the smallest σ-algebra including G, and is denoted by $\sigma(G)$. Given a measurable space (S, Σ), a *measure* over (S, Σ) is a function $\mu\colon \Sigma \to \mathbb{R}^{\geq 0}$ such that, for every countable family $\{A_i\}_I \subseteq \Sigma$ of pairwise disjoint measurable sets, $\mu(\cup_I A_i) = \sum_I \mu(A_i)$. A *(sub-)probability measure* is a measure $\mu\colon \Sigma \to [0,1]$ for which $(\mu(S) \leq 1)$ $\mu(S) = 1$. Probability measures are ranged over by $\mu, \eta, \ldots$ and we propagate indices and primes where necessary. A set $A \subseteq S$ is called a *support* for a measure μ on Σ if $\mu(S - A) = 0$. Denote by ($\mathit{SubDisc}(S)$) $\mathit{Disc}(S)$ the set of discrete (sub-)probability measures over S and, given an element $s \in S$, denote by $\delta(s)$ the probability measure that assigns probability 1 to $\{s\}$. This is called the *Dirac measure* on s. Given a countable set of distributions $\{\mu_i\}_I$ and a set $\{p_i\}_I$ of real numbers in $[0,1]$ such that $\sum_I p_i = 1$, define the *convex combination* $\sum_I p_i\mu_i$ of $\{\mu_i\}_I$ as the probability measure μ such that, for each set X, $\mu(X) = \sum_I p_i\mu_i(X)$.

Sometimes it is necessary to lift a relation over sets to a relation over measures on sets. We give here a definition proposed in [12] using an idea of [9]. Let $\mathcal{R} \subseteq X \times Y$. The *lifting* of $\mathcal{R}$ is a new relation $\mathcal{L}(R) \subseteq \mathit{Disc}(X) \times \mathit{Disc}(Y)$, such that $\mu_1\ \mathcal{L}(\mathcal{R})\ \mu_2$ iff there exists a *weight function* (or *witness function*) $\omega\colon X \times Y \longrightarrow [0,1]$ such that the following *lifting conditions* hold:

1. $\forall (x,y) \in X \times Y$, if $\omega(x,y) > 0$ then $x\ \mathcal{R}\ y$;
2. $\forall x \in X, \sum_{y \in Y} \omega(x,y) = \mu_1(x)$;
3. $\forall y \in Y, \sum_{x \in X} \omega(x,y) = \mu_2(y)$.

If $\mathcal{R}$ is an equivalence relation, then for each pair of measures μ_1, μ_2, it can be shown that $\mu_1\ \mathcal{L}(R)\ \mu_2$ if and only if $\mu([t]) = \mu'([t])$ for each equivalence class $[t]$ of $\mathcal{R}$. In the following we will use $\mathcal{R}$ instead of $\mathcal{L}(\mathcal{R})$ if it is clear from the context that we refer to a lifted relation. A set E is *$\mathcal{R}$-closed* if $E = \{s \mid \exists r \text{ s.t. } s\ \mathcal{R}\ r\}$, that is, it is a collection of equivalence classes of $\mathcal{R}$.

A subset C of a metric space (X, d) is *compact* if every subset S of C has a limit point in C, i.e., for each $S \subseteq C$, there exists $l \in C$ such that for each $\epsilon \in \mathbb{R}^+$ there exists $x \in S$ such that $d(l, x) < \epsilon$.

3 Probabilistic Automata

In this section we recall some basic definitions for probabilistic automata, including the notions of probabilistic bisimulations. The original definitions appear in [12], and the bisimulation relations that we use were first proposed in [14] for probabilistic automata; however, the notion of strong bisimulation is an extension of the original proposal of [10] for reactive systems.

An *automaton* is a tuple $\mathcal{A} = (S, \mathit{Act}, \mathcal{D})$ where S is the set of *states*, Act is the set of *actions*, and $\mathcal{D} \subseteq S \times \mathit{Act} \times S$ is the *transition relation*. Each triple $(s, a, s') \in \mathcal{D}$ is called a *transition*, and is denoted by $s \xrightarrow{a} s'$. The set Act is partitioned into two sets E, H of *external* and *internal* actions, respectively. For the purpose of this paper we assume that $H = \{\tau\}$.

Probabilistic automata are conservative extensions of Labeled Transition Systems where transitions lead to discrete probability measures over states instead of single states. Indeed, an ordinary automaton can be seen as a probabilistic automaton where each transition leads to a Dirac measure. A *probabilistic automaton* is a tuple $\mathcal{P} = (S, Act, \mathcal{D})$, where S is the set of *states*, Act is the set of *actions*, and $\mathcal{D}$ is the *transition relation*, where $\mathcal{D} \subseteq S \times Act \times Disc(S)$. Denote a transition $(s, a, \mu) \in \mathcal{D}$ by $s \xrightarrow{a} \mu$. States and actions are ranged over by $s, r, t, \ldots$ and $a, b, c \ldots$, respectively.

An *execution* of a probabilistic automaton $\mathcal{P}$ is a finite or infinite sequence $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$ of alternating states and actions, starting with a state and, if the sequence is finite, ending in a state, where for each i, there exists a measure μ such that $(s_i, a_{i+1}, \mu) \in \mathcal{D}$ and $\mu(s_{i+1}) > 0$. State s_0 is called the first state of α and is denoted by $\mathit{fstate}(\alpha)$. If α is a finite sequence, then the last state of α is denoted by $\mathit{lstate}(\alpha)$. Denote by $\mathit{execs}(\mathcal{P})$ the set of executions of $\mathcal{P}$ and by $\mathit{execs}^*(\mathcal{P})$ the set of finite executions of $\mathcal{P}$. Executions are the result of the resolution of both probabilistic and nondeterministic choices. If we resolve nondeterministic choices only, then we obtain a structure on which we can study probability measures over executions. Nondeterminism is resolved in a randomized way by an entity called scheduler.

A *scheduler* for a probabilistic automaton $\mathcal{P}$ is a function $\sigma \colon \mathit{execs}^*(\mathcal{P}) \to \mathit{SubDisc}(\mathcal{D})$ such that $\sigma(\alpha)(s, a, \mu) > 0$ implies $s = \mathit{lstate}(\alpha)$. A scheduler σ is *deterministic* if for each finite execution α, $\sigma(\alpha) \equiv 0$ or $\sigma(\alpha) = \delta(tr)$, with $tr \in \mathcal{D}$. A scheduler induces a probability measure over executions on a σ-field whose construction is standard. Specifically, we consider the σ-field generated by *cones*, where the cone of a finite execution α, denoted by C_α, is the set of executions that have α as a prefix, i.e., $C_\alpha = \{\alpha' \in \mathit{execs}(\mathcal{P}) \mid \alpha \leq \alpha'\}$. Fixed a starting state s_0, the measure of a cone C_α, where $\alpha = s_0 a_1 s_1 \cdots s_k$, is defined as follows:

$$\mu(C_\alpha) = \prod_{i \in \{0, k-1\}} \left(\sum_{(s_i, a_{i+1}, \mu') \in \mathcal{D}} \sigma(s_0 a_1 \cdots a_i s_i)(s_i, a_{i+1}, \mu') \mu'(s_{i+1}) \right).$$

Standard measure theoretical arguments ensure that the measure defined on cones extends uniquely to a measure defined on the generated σ-field.

Let $\{s \xrightarrow{a} \mu_i\}_{i \in I}$ be a collection of transitions of $\mathcal{P}$, and let $\{p_i\}_{i \in I}$ be a collection of probabilities such that $\sum_{i \in I} p_i = 1$. Then, the triple $\{s, a, \sum_{i \in I} p_i \mu_i\}$ is called a *combined transition*. Combined transitions represent the result of choosing the transitions randomly from some state s. They are useful in the definition of probabilistic bisimulations.

We say that $s \overset{a}{\Longrightarrow} s'$ is a *weak transition* of an automaton $\mathcal{A}$ if there is a finite execution α of $\mathcal{A}$ with $\mathit{fstate}(\alpha) = s$ and $\mathit{lstate}(\alpha) = s'$, and such that $\mathit{trace}(\alpha) = \mathit{trace}(a)$, where the trace function restricts a sequence to external actions only. In other words, a weak transition is a way to abstract from internal computation. For probabilistic automata, consider a measure μ, induced by a scheduler σ from a starting state s_0, that assigns probability 1 to the set of all finite executions with trace a. Let μ' be the measure defined by $\mu'(s) = \mu(\{\alpha \mid \mathit{lstate}(\alpha) = s\})$. Then $s \overset{a}{\Longrightarrow} \mu'$ is a *weak combined transition* of $\mathcal{P}$. The term "combined" reflects the fact that σ is a randomized scheduler.

We define three types of bisimulation relations that will be studied in the rest of this paper. These relations differ for the kind of transitions used.

1. An equivalence relation $\mathcal{R} \subseteq S \times S$ is a *strong bisimulation* if for each pair s, r of states such that $s \mathrel{\mathcal{R}} r$ and for each transition $s \xrightarrow{a} \mu$, there exists μ' such that $r \xrightarrow{a} \mu'$ and $\mu \mathrel{\mathcal{R}} \mu'$. Denote by $\sim$ the largest strong bisimulation.
2. An equivalence relation $\mathcal{R} \subseteq S \times S$ is a *strong probabilistic bisimulation* if for each pair s, r of states such that $s \mathrel{\mathcal{R}} r$ and for each transition $s \xrightarrow{a} \mu$, there exists a combined transition $r \xrightarrow{a} \mu'$ such that $\mu \mathrel{\mathcal{R}} \mu'$. Denote by $\sim^p$ the largest strong probabilistic bisimulation.
3. An equivalence relation $\mathcal{R} \subseteq S \times S$ is a *weak probabilistic bisimulation* if for each pair s, r of states such that $s \mathrel{\mathcal{R}} r$ and for each transition $s \xrightarrow{a} \mu$, there exists a weak combined transition $r \overset{a}{\Longrightarrow} \mu'$ such that $\mu \mathrel{\mathcal{R}} \mu'$. Denote by $\approx^p$ the largest weak probabilistic bisimulation.

There would be a fourth natural relation that uses weak transitions induced by deterministic schedulers. However, this relation is not transitive, as shown in [3], and thus it is not interesting.

We recall an alternative way of defining bisimulation [8] as $\bigcap_{i \geq 0} \sim_n$, where $\sim_0 = S \times S$ (all states are related) and for each pair of states s, r, $s \sim_{n+1} r$ if for each action a, $s \xrightarrow{a} \mu$ implies that there exists μ' such that $r \xrightarrow{a} \mu'$ and $\mu \sim_n \mu'$. The same definition style applies to strong and weak probabilistic bisimulation, as well.

4 Hennessy-Milner Logic for Probabilistic Automata

In this section, we give logical characterizations of bisimulations for probabilistic automata. We start by recalling the logic from [8]; then we analyze in detail the logics for strong, strong probabilistic and weak probabilistic bisimulations.

4.1 Hennessy-Milner Logic

The syntax of the Hennessy-Milner Logic [8] is the following:

$$\mathcal{L}^{hm} ::= \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Diamond a\varphi.$$

The satisfaction relation $\models \subseteq S \times \mathcal{F}$ is defined by structural induction on the formulas of $\mathcal{L}^{hm}$ as follows:

- $s \models \top$ for each state s
- $s \models \neg\varphi$ iff $s \not\models \varphi$
- $s \models \varphi_1 \wedge \varphi_2$ iff $s \models \varphi_1$ and $s \models \varphi_2$
- $s \models \Diamond a\varphi$ iff there exists a transition $s \xrightarrow{a} s'$ such that $s' \models \varphi$.

The only non-trivial operator is the "diamond" $\Diamond$, which is used to describe the existence of transitions. It was shown in [8] that the logic $\mathcal{L}^{hm}$ characterizes strong bisimulation for ordinary automata. In particular, two states s and r of an automaton are bisimilar if and only if they satisfy the same formulas of $\mathcal{L}^{hm}$. If we extend the logic above with a countably infinite conjunction, then the characterization of strong bisimulation holds also for automata with countably many states. In this paper we deal with countable state spaces, and therefore we use an infinitary conjunction operator. However, our results hold for finite conjunction operators and finite-state spaces.

The logics that we study in this paper share a lot of structure of $\mathcal{L}^{hm}$, thus, we introduce here some useful notation. Let $\varphi, \psi, \ldots$ range over formulas, and we define the *depth* of a formula φ as the maximum number of nested diamond operators that occur in φ. Let $\mathcal{F}_{\mathcal{L}}$ denote the set of the formulas of $\mathcal{L}$, and $\mathcal{F}_{\mathcal{L},n}$ denote the set of the formulas of $\mathcal{L}$ of depth at most n. Moreover, denote $\mathcal{F}_{\mathcal{L}}(s)$ and $\mathcal{F}_{\mathcal{L}}(\mu)$ as the sets of the formulas of $\mathcal{L}$ which are satisfied by the state s and by the distribution μ, respectively. We define a new relation $\bowtie_{\mathcal{L}} \subseteq S \times S$ such that $s \bowtie_{\mathcal{L}} r$ if and only if $\mathcal{F}_{\mathcal{L}}(s) = \mathcal{F}_{\mathcal{L}}(r)$, and similarly we define $\bowtie_{\mathcal{L},n} \subseteq S \times S$ as the relation such that $s \bowtie_{\mathcal{L},n} r$ if and only if $\mathcal{F}_{\mathcal{L},n}(s) = \mathcal{F}_{\mathcal{L},n}(r)$. We drop the subscript $\mathcal{L}$ whenever it is clear from the context. Finally, denote by $[\![\varphi]\!]$ the set of all the states that satisfy a formula φ.

4.2 Hennessy-Milner Logic for Strong Bisimulation

The main difference between probabilistic automata and ordinary automata is that in probabilistic automata the target of each transition is a probability measure. Thus, the modal operators defined for probabilistic automata must take into account this possibility. Moreover from every state there might be several outgoing transitions labeled by the same action. Thus, a naive extension of $\Diamond$, where we study the probability of a formula in the target of a transition [10] does not suffice. Our proposal is to define a new operator that, together with conjunction, can characterize exactly a probability measure. The syntax of the logic $\mathcal{L}^N$ for strong bisimulation is the following:

$$\mathcal{L}^N ::= \top \mid \neg\varphi \mid \bigwedge_I \varphi_i \mid \Diamond a\varphi \mid [\varphi]_p.$$

The semantics of this logic is given in terms of probability measures over states rather than single states. Specifically, the satisfaction relation is defined as follows:

- $\mu \models \top$ for each measure μ
- $\mu \models \neg\varphi$ iff $\mu \not\models \varphi$
- $\mu \models \bigwedge_I \varphi_i$ iff for each $i \in I$, $\mu \models \varphi_i$
- $\mu \models \Diamond a\varphi$ iff for each $s \in supp(\mu)$ there exists a transition $s \xrightarrow{a} \eta$ such that $\eta \models \varphi$
- $\mu \models [\varphi]_p$ iff $\mu([\![\varphi]\!]) \geq p$.

The first three clauses are trivial extensions of those of Hennessy-Milner logic. The diamond operator is exactly that of $\mathcal{L}^{hm}$ if we restrict our study to Dirac distributions, and the operator $[\cdot]_p$ expresses the probability of a set of states with respect to a given probability measure.

The rest of this section is dedicated to the proofs of soundness and completeness of $\mathcal{L}^N$. We start with a few preliminary lemmas that are used to prove the soundness and completeness of $\mathcal{L}^N$. This first lemma states that, when considering logics with negation, if two states satisfy two different sets of formulas, then none of these sets is contained in the other.

Lemma 1. *Given a logic with negation, for each pair of states s and r of a probabilistic automaton, if $\mathcal{F}(s) \neq \mathcal{F}(r)$ then $\mathcal{F}(s) \not\subseteq \mathcal{F}(r)$.*

The second lemma shows that the states of a probabilistic automaton satisfy the same sets of formulas of depth zero.

Lemma 2. *For each pair of states s, r, $\mathcal{F}_0(s) = \mathcal{F}_0(r)$.*

The third lemma relates diamond formulas with probability measures and the states in their support.

Lemma 3. *For each measure μ, $\mu \models \Diamond a\varphi$ iff for each $s \in supp(\mu)$, $s \models \Diamond a\varphi$.*

The last lemma states that the lifting of $\bowtie_n$ preserves the sets of formulas satisfied by two probability distributions μ and μ'.

Lemma 4. *Let $\mathcal{R}$ be a subset of $\bowtie_n$. Then, for each pair of distributions μ, μ', $\mu \mathrel{\mathcal{R}} \mu'$ implies $\mathcal{F}_n(\mu) = \mathcal{F}_n(\mu')$.*

Proof. Without loss of generality, let $\mu \models \varphi$, where $\varphi \in \mathcal{F}_n$. We prove that $\mu' \models \varphi$ by induction on the structure of φ.

– If $\varphi = \top$, then the result is trivial.
– If $\varphi = \Diamond a\psi$, then, by Lemma 3, for each $s \in supp(\mu)$, $s \models \varphi$. We show that $s' \models \varphi$ for each $s' \in supp(\mu')$. Then, $\mu' \models \varphi$ follows again by Lemma 3. Let $s' \in supp(\mu')$, and let $[s']$ be the equivalence class of $\mathcal{R}$ containing s'. Since $\mu'(s') > 0$ and $\mu \mathrel{\mathcal{R}} \mu'$, then $\mu'([s']) > 0$ and $\mu([s']) > 0$. Thus, there exists an element $s \in supp(\mu)$ such that $s \mathrel{\mathcal{R}} s'$. By hypothesis, $\mathcal{F}_n(s) = \mathcal{F}_n(s')$. Since $s \in supp(\mu)$, then $s \models \varphi$ and since $\varphi \in \mathcal{F}_n$, then $s' \models \varphi$ as needed.
– If $\varphi = [\psi]_p$, then $\mu(\llbracket\psi\rrbracket) \geq p$. Since $\psi \in \mathcal{F}_n$, then $\llbracket\psi\rrbracket$ is a union of equivalence classes of $\mathcal{R}$ (by hypothesis, each formula of $\mathcal{F}_n$ is satisfied either by all or none of the states of an equivalence class of $\mathcal{R}$). Since $\mu \mathrel{\mathcal{R}} \mu'$, then each equivalence class of $\mathcal{R}$ has the same measure according to μ and μ'. Thus, $\mu(\llbracket\psi\rrbracket) = \mu'(\llbracket\psi\rrbracket)$. Since $\mu(\llbracket\psi\rrbracket) \geq p$, then also $\mu'(\llbracket\psi\rrbracket) \geq p$ and thus, $\mu' \models [\psi]_p$ as needed.
– If $\varphi = \neg\psi$, then $\mu \not\models \psi$. By induction, since $\psi \in \mathcal{F}_n$, $\mu' \not\models \psi$. Thus, $\mu' \models \neg\psi$.
– If $\varphi = \bigwedge_I \psi_i$, then for each $i \in I$, $\mu \models \psi_i$. Since $\psi_i \in \mathcal{F}_n$ for each $i \in I$, then by induction, $\mu' \models \psi_i$. Thus, $\mu' \models \bigwedge_I \psi_i$.

We are now ready to prove the soundness and completeness of the logic $\mathcal{L}^N$ for probabilistic automata.

Theorem 1. *Given the logic $\mathcal{L}^N$, for each pair of states s, r of a probabilistic automaton, $s \sim r$ iff $\mathcal{F}(s) = \mathcal{F}(r)$.*

Proof. By induction on n, we show that $s \sim_n r$ iff $\mathcal{F}_n(s) = \mathcal{F}_n(r)$ for each $n \geq 0$. The base case follows trivially by Lemma 2 and the definition of $\sim_0$ (all states are related). For the inductive step we prove separately the two directions of our claim.

($\Longrightarrow$). Let $s \sim_{n+1} r$. We show by induction on the structure of a formula $\varphi \in \mathcal{F}_{n+1}$ that $s \models \varphi$ iff $r \models \varphi$. Let $s \models \varphi$ (the case for $r \models \varphi$ is symmetric). If $\varphi = \top$, then $r \models \varphi$ trivially. If $\varphi = \neg\psi$, then $s \not\models \psi$ and by structural induction, $r \not\models \psi$. Thus, $r \models \neg\psi$. If $\varphi = \bigwedge_I \psi_i$, then for each $i \in I$, $s \models \psi_i$. By structural induction, $r \models \psi_i$ for each $i \in I$ and thus, $r \models \varphi$. If $\varphi = [\psi]_p$, then either $p = 0$ or $s \models \psi$. In the first case $r \models \varphi$ trivially; in the second case, by structural induction, $r \models \psi$ and thus, $r \models \varphi$. If $\varphi = \Diamond a\psi$, then $\psi \in \mathcal{F}_n$. By definition, there exists a transition $s \xrightarrow{a} \mu$ such that $\mu \models \psi$. Since $s \sim_{n+1} r$, there exists a distribution μ' and a transition $r \xrightarrow{a} \mu'$ such that $\mu \sim_n \mu'$. By induction on n, $\sim_n \subseteq \bowtie_n$. Thus, by Lemma 4 and since $\mu \sim_n \mu'$, $\mathcal{F}_n(\mu) = \mathcal{F}_n(\mu')$. Since $\psi \in \mathcal{F}_n$, and since $\mu \models \psi$, then also $\mu' \models \psi$. That is, $r \models \Diamond a\psi$.

($\Longleftarrow$). We show that $s \not\sim_{n+1} r$ implies $\mathcal{F}_{n+1}(s) \neq \mathcal{F}_{n+1}(r)$. Let $\{[t_i]_n\}_I$ be an enumeration of the equivalence classes of $\sim_n$. By induction on n and by Lemma 1, for each $i, j \in I$, if $i \neq j$, there exists a formula $\varphi_{ij} \in \mathcal{F}_n$ such that $t_i \models \varphi_{ij}$ and $t_j \not\models \varphi_{ij}$. For each $i \in I$, define $\varphi_i = \bigwedge_{j \in I \setminus \{i\}} \varphi_{ij}$. Then φ_i is satisfied only by the states of $[t_i]_n$. Let $s \not\sim_{n+1} r$ and suppose, for the sake of contradiction, that $\mathcal{F}_{n+1}(s) = \mathcal{F}_{n+1}(r)$. Without loss of generality, there exists a transition $s \xrightarrow{a} \mu$ such that there is no transition $r \xrightarrow{a} \mu'$ with $\mu \sim_n \mu'$. For each $i \in I$ let $p_i = \mu([t_i]_n)$. Now, define $\varphi = \bigwedge_I [\varphi_i]_{p_i}$. By definition, $\mu \models \varphi$. Furthermore, $\varphi \in \mathcal{F}_n$. By the semantics of $\Diamond$, $s \models \Diamond a\varphi$. Since $\Diamond a\varphi \in \mathcal{F}_{n+1}$, by hypothesis, $r \models \Diamond a\varphi$ as well. Thus, there exists a distribution μ'' such that $r \xrightarrow{a} \mu''$ and $\mu'' \models \varphi$. This means that for each $i \in I$, $\mu''([t_i]_n) \geq p_i$. Since $\sum_I p_i = 1$ and since $\sum_I \mu''([t_i]_n) = 1$, then for each $i \in I$, $\mu''([t_i]_n) = p_i$. That is, for each $i \in I$, $\mu([t_i]_n) = \mu''([t_i]_n)$, which means that $\mu \sim_n \mu''$, a contradiction.

4.3 Hennessy-Milner Logic for Strong Probabilistic Bisimulation

In the definition of probabilistic bisimulation, a transition can be matched by combining transitions. Thus, it is reasonable to believe that the diamond for strong probabilistic bisimulation should take into account this possibility. Indeed, the logic $\mathcal{L}_p^N$ for strong probabilistic bisimulation is obtained by replacing the operator $\Diamond$ with $\dot{\Diamond}$. The syntax of $\mathcal{L}_p^N$ is:

$$\mathcal{L}_p^N ::= \top \mid \neg\varphi \mid \bigwedge_I \varphi_i \mid \dot{\Diamond} a\varphi \mid [\varphi]_p.$$

The semantics of the operator $\dot{\Diamond}$ is:

– $\mu \models \dot{\Diamond} a\varphi$ iff for each $s \in \mathit{supp}(\mu)$ there exists a *combined transition* $s \xrightarrow{a} \eta$ such that $\eta \models \varphi$.

Since, as shown in [15], $\sim^p$ is weaker than $\sim$, adding the operator $\Diamonddot$ to $\mathcal{L}^N$ does not change its expressivity. Hence, the operator $\Diamonddot$ is weaker than $\Diamond$. Soundness and completeness of $\mathcal{L}_p^N$ are stated by the following theorem.

Theorem 2. *Given the logic $\mathcal{L}_p^N$, for each pair of states s, r of a probabilistic automaton, $s \sim^p r$ iff $\mathcal{F}(s) = \mathcal{F}(r)$.*

Proof outline. Lemma 1, 2, 3 and 4 hold also here. Then, the proof of this theorem follows the lines of that of Theorem 1, by replacing strong transitions with combined transitions in the appropriate places.

4.4 Logic for Weak Probabilistic Bisimulation

The definition of the logic $\mathcal{L}_w^N$ follows the same ideas of $\mathcal{L}_p^N$. We replace the $\Diamond$ of $\mathcal{L}^N$ with operator $\Diamonddot^w$, and the new syntax is:

$$\mathcal{L}_w^N ::= \top \mid \neg\varphi \mid \bigwedge_I \varphi_i \mid \Diamonddot^w a\varphi \mid [\varphi]_p.$$

The semantics of $\Diamonddot^w$ is:

- $\mu \models \Diamonddot^w a\varphi$ iff for each $s \in supp(\mu)$ there exists a *weak combined transition* $s \overset{a}{\Longrightarrow} \eta$ such that $\eta \models \varphi$.

As observed in the previous section with respect to the new diamond operator $\Diamonddot$ introduced, we can easily infer that the operator $\Diamonddot^w$ is weaker than $\Diamonddot$. As for strong probabilistic bisimulation, we can prove that $\mathcal{L}_w^N$ characterizes weak probabilistic bisimulation.

Theorem 3. *Given the logic $\mathcal{L}_w^N$, for each pair of states s, r of a probabilistic automaton, $s \approx^p r$ iff $\mathcal{F}(s) = \mathcal{F}(r)$.*

Proof outline. Lemma 1, 2, 3 and 4 hold also here. Then, the proof of this theorem follows the lines of that of Theorem 1, by replacing strong transitions with weak combined transitions in the appropriate places.

5 Hennessy-Milner Logic for Reactive Systems

Reactive systems [6] are essentially *deterministic* probabilistic automata, i.e., for each state and for each label, there exists at most one transition. There is already a logical characterization of bisimulation for reactive systems by Larsen and Skou [10]. The syntax of the Larsen and Skou logic is the following:

$$\mathcal{L}^{ls} ::= \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Diamond_p a\varphi,$$

where p is a rational number in $[0, 1]$. The only new operator is $\Diamond_p$, whose semantics is:

- $s \models \Diamond_p a\varphi$ iff there exists a transition $s \xrightarrow{a} \mu$ such that $\mu([\![\varphi]\!]) \geq p$.

Desharnais, Panangaden et al. [4] have shown that negation is not necessary to characterize bisimulation for reactive systems. Moreover, they have shown that infinitary conjunction is not needed even if the state space is uncountable, and can be replaced by a finite conjunction operator. The syntax of their logic is the following:

$$\mathcal{L}^R ::= \top \mid \varphi_1 \wedge \varphi_2 \mid \Diamond_p a\varphi.$$

It is immediate to see that $\mathcal{L}^R$ is a sublogic of $\mathcal{L}^{ls}$. We can see the logic $\mathcal{L}^R$ as a sublogic of $\mathcal{L}^N$ as well. Also the operator $[\cdot]_p$ is necessary when nondeterminism is present. However, as shown in the next section, this operator can be dropped in favor of $\Diamond_p$ if nondeterminism is partially restricted.

6 Hennessy-Milner Logic for Alternating Models

In this section, we define two logics for strong and weak probabilistic bisimulation for a restriction of probabilistic automata that embeds the alternating models and we show that the logical characterization of [5] continues to hold.

We say that a probabilistic automaton is *alternating* if the states that enable a non-Dirac transition enable only one transition. We call *probabilistic* those states that enable non-Dirac transitions, and *nondeterministic* all the other states. This definition of alternating probabilistic automaton, indeed, describes a class of systems that is more general than the labeled concurrent Markov chains of Hansson [7] and of Philippou et al. [11], since it does not impose any alternation between nondeterministic and probabilistic states, and it allows probabilistic transitions to be labeled by external actions. As shown in [15], the notion of bisimulation of this paper coincides with those of Hansson and of Philippou et al. when applied to their models. Thus, the logical characterization given for alternating probabilistic automata is also a logical characterization for the alternating models [7,11]. We do not handle strong probabilistic bisimulation explicitly since it coincides with strong bisimulation [15].

6.1 Hennessy-Milner Logic for Strong Bisimulation

Since in alternating probabilistic automata each probabilistic transition is described by some state, then intuitively the target measure of a probabilistic transition that leaves from a state s can be derived just by observing the probability of reaching each equivalence class from s. For this reason the operator $\Diamond_p$ should suffice. Indeed, bisimulation relations can be characterized in terms of maximal probabilities of reaching equivalence classes (Lemma 5), and thus, an operator $\Diamond_p$ suffices. The overall idea of maximal probabilities is taken from [11]. The syntax of the logic $\mathcal{L}^A$ for strong bisimulation is the following:

$$\mathcal{L}^A ::= \top \mid \neg\varphi \mid \bigwedge_I \varphi_i \mid \Diamond_p a\varphi.$$

The semantics of the operator $\Diamond_p$ is exactly the same as for reactive systems. Note that if we drop the operator $[\cdot]_p$, it is no more necessary to define the satisfaction relation on measures, since it can be defined on single states. In a similar way as in [11], for each action a and for each equivalence class $[t]$ of $\sim$, we define $\overline{\mu}_{s,a}([t])$ as the maximal probability to reach $[t]$ from s with a strong transition labeled a.

Lemma 5. *For each pair of states s and r of an alternating automaton, if $\overline{\mu}_{s,a}([t]) = \overline{\mu}_{r,a}([t])$ for each action a and each equivalence class $[t]$ of $\sim$, then $s \sim r$.*

Proof. Let $\{[t_i]\}_I$ be an enumeration of the equivalence classes of $\sim$. Without loss of generality, we distinguish the following cases.

1. s is nondeterministic. Let $s \xrightarrow{a} \mu$. Then, μ is a Dirac measure $\delta(s')$. Let $k \in I$ be the index of the class such that $s' \in [t_k]$. Then, $\mu([t_k]) = 1$. This means that $\mu([t_k]) = \overline{\mu}_{s,a}([t_k])$. Then, by hypothesis, there exists μ' such that $\mu'([t_k]) = \overline{\mu}_{r,a}([t_k])$ and $\mu'([t_k]) = 1$. By definition, for each equivalence class $[t_j]$, $\mu'([t_j]) = 0$. Thus, $\mu \sim \mu'$.
2. s is probabilistic. Then, let $s \xrightarrow{a} \mu$ be the only transition from s. By definition, for each $i \in I$, $\mu([t_i]) = \overline{\mu}_{s,a}([t_i])$. If r is probabilistic, then by hypothesis, $r \xrightarrow{a} \mu'$ is the only transition from r, and μ, μ' agree on all the equivalence classes. If r is nondeterministic, then each transition is of the form $r \xrightarrow{a} \delta(r')$ and for each $i \in I$, $\delta([t_i]) = \overline{\mu}_{r,a}([t_i]) = 1$ only if $r' \in [t_i]$. By hypothesis, there exists an equivalence class $[t_k]$ such that $\mu([t_k]) = 1$. This also implies that all the target Dirac distributions $\delta(r')$ reached from r are related, since by hypothesis, each r' must belong to $[t_k]$. Thus, for each transition $r \xrightarrow{a} \delta(r')$, $\mu \sim \delta(r')$.

Now we can prove the completeness of $\mathcal{L}^A$ using the result of the previous lemma.

Theorem 4. *Given the logic $\mathcal{L}^A$, for each pair of states s, r of an alternating automaton, $s \sim r$ iff $\mathcal{F}(s) = \mathcal{F}(r)$.*

Proof. $(\Longrightarrow)$. Soundness follows by Theorem 1, by the fact that alternating automata are a special case of probabilistic automata via embedding [15] and since $\mathcal{L}^A$ is a sublogic of $\mathcal{L}^N$.

$(\Longleftarrow)$. Let $s \bowtie r$, and let $\{\langle t_i \rangle\}_I$ be an enumeration of the equivalence classes of $\bowtie$. We show that $\bowtie$ is a bisimulation between s and r that is, by Lemma 5, for each action a and for each $i \in I$, $\overline{\mu}_{s,a}(\langle t_i \rangle) = \overline{\mu}_{r,a}(\langle t_i \rangle)$. By hypothesis and by Lemma 1, for each pair of classes $\langle t_i \rangle, \langle t_j \rangle$, there exists a formula φ_{ij} such that $t_i \models \varphi_{ij}$ and $t_j \not\models \varphi_{ij}$. For each $i \in I$, let $\varphi_i = \bigwedge_{I \setminus \{i\}} \varphi_{ij}$. Then, φ_i is satisfied only by the states of $\langle t_i \rangle$. For the sake of contradiction, suppose that there exists an action a and a class $\langle t_k \rangle$ such that $\overline{\mu}_{s,a}(\langle t_k \rangle) \neq \overline{\mu}_{r,a}(\langle t_k \rangle)$. For each $i \in I$, let $\mu(\langle t_i \rangle) = \overline{\mu}_{s,a}(\langle t_i \rangle)$ and $\mu'(\langle t_i \rangle) = \overline{\mu}_{s,a}(\langle t_i \rangle)$. Without loss of generality, let $\mu'(\langle t_k \rangle) < \mu(\langle t_k \rangle)$ Then, there exists a rational p such that $\mu'(\langle t_k \rangle) < p < \mu(\langle t_k \rangle)$. By definition, $s \models \Diamond_p a \varphi_k$. By hypothesis, $r \models \Diamond_p a \varphi_k$ as well. This means that there exists a measure μ'' such that $r \xrightarrow{a} \mu''$ and $\mu''([\![\varphi_k]\!]) \geq p$. By hypothesis, $[\![\varphi_k]\!] = \langle t_k \rangle$, and by definition, $\mu''(\langle t_k \rangle) \leq \mu'(\langle t_k \rangle)$, a contradiction.

6.2 Hennessy-Milner Logic for Weak Probabilistic Bisimulation

The additivity of the maximal probabilities reachable via weak transitions is assured if we restrict to *compact* systems. The notion of compactness (see Section 2) can be applied to the alternating models introducing an adequate metric d on measures and defining an alternating automaton to be *compact* if for each state s the set of all measures reachable from s via weak transitions is compact under the given metric [5]. Thus, each sequence of weak transitions leaving s converge to a limit measure and a transition reaching that measure exists and starts from s. This implies that for each maximal probability reachable, there is a weak transition yielding that maximal probability.

In the following, we will implicitly assume that the alternating models considered are compact, thus allowing us to handle maximal probabilities in our proofs. However, our logical characterization holds also for non-compact systems, since alternating automata are a special case of probabilistic automata [15].

The logic characterizing weak probabilistic bisimulation for labeled concurrent Markov chains [5] is reported in the following:

$$\mathcal{L}_w^A ::= \top \mid \neg\varphi \mid \bigwedge_I \varphi_i \mid \Diamond_p^w a\varphi.$$

In [5], disjunction is also used since it simplifies their proof of completeness, but it is not necessary. This logic is exactly that of Larsen and Skou, except for the diamond operator, whose semantics $\Diamond_p^w$ is defined as follows:

– $s \models \Diamond_p^w a\varphi$ iff $\overline{\mu}_{s,a}([\![\varphi]\!]) \geq p$.

This definition underlines the strict correlation between weak probabilistic bisimulation for alternating models and maximal probabilities. Philippou et al. [11] restrict their study to deterministic schedulers, while Desharnais et al. [5] permit linear combination of deterministically scheduled paths to reach maximal probabilities. These linear combinations reflect the concept of *convex combinations* for probabilistic automata. Anyway, as stated in Lemma 6, deterministic schedulers are enough to reach maximal probabilities, allowing us to simplify some proofs using deterministic schedulers instead of randomized ones.

Keeping the same notation of the previous section, for each action a and for each formula φ, $\overline{\mu}_{s,a}([\![\varphi]\!])$ is defined as the maximal probability (over all schedulers) of $[\![\varphi]\!]$ over all the distributions reached via *weak combined transitions* from s with label a. Like for strong bisimulation, for each action a and for each formula φ, we can state that $\Diamond_p^w a\varphi \equiv \Diamond^w a[\varphi]_p$. In the following, we prove the completeness of the logic $\mathcal{L}_w^A$, extending some results of [5] to our definition of alternating model.

We define a new relation $\doteq$ such that, for each pair of states s, r of an alternating automaton, $s \doteq r$ iff

– s and r are nondeterministic and for each action a and for each $\doteq$-closed set E, $\overline{\mu}_{s,a}(E) = \overline{\mu}_{r,a}(E)$, or

– s is probabilistic with action τ and for each $\doteq$-closed set E such that $s \notin E$, $\overline{\mu}_{s,\tau}(E) = \overline{\mu}_{r,\tau}(E)$, or
– s is probabilistic with external action and for each $\doteq$-closed set E, $\overline{\mu}_{s,a}(E) = \overline{\mu}_{r,a}(E)$.

The following lemma states a property of deterministic schedulers [2], and permits to use indifferently deterministic or randomized schedulers when calculating maximal probabilities. The assumption is to work in compact systems, since this result requires that each set considered must be reachable by a weak transition.

Lemma 6. *In compact systems, maximal probabilities can be reached with deterministic schedulers.*

The next lemma shows that the relation $\bowtie$ implies the relation $\doteq$, which directly talks about the maximal probabilities reachable. This result is basilar to prove the soundness of the logic $\mathcal{L}_w^N$.

Lemma 7. *For each pair of states s, r of an alternating automaton, if $s \bowtie r$ then $s \doteq r$.*

Proof. Let $s \bowtie r$, and let $\{\langle t_i \rangle\}_I$ be an enumeration of the equivalence classes of $\bowtie$. We prove a stronger result, showing that for each action a and for each equivalence class $\langle t_k \rangle$, $\overline{\mu}_{s,a}(\langle t_k \rangle) = \overline{\mu}_{r,a}(\langle t_k \rangle)$, which directly implies that $s \doteq r$. By hypothesis and by Lemma 1, for each pair of classes $\langle t_i \rangle$, $\langle t_j \rangle$, there exists a formula φ_{ij} such that $t_i \models \varphi_{ij}$ and $t_j \not\models \varphi_{ij}$. For each $i \in I$, let $\varphi_i = \bigwedge_{I \setminus \{i\}} \varphi_{ij}$. Then, φ_i is satisfied only by the states of $\langle t_i \rangle$. For the sake of contradiction, suppose that there exists an action a and a class $\langle t_k \rangle$ such that $\overline{\mu}_{s,a}(\langle t_k \rangle) \neq \overline{\mu}_{r,a}(\langle t_k \rangle)$. Without loss of generality, let $\overline{\mu}_{s,a}(\langle t_k \rangle) < \overline{\mu}_{r,a}(\langle t_k \rangle)$. Then, there exists a rational p such that $\overline{\mu}_{s,a}(\langle t_k \rangle) < p < \overline{\mu}_{r,a}(\langle t_k \rangle)$. By definition, $r \models \diamondsuit_p^w a \varphi_k$ since $\overline{\mu}_{r,a}([\![\varphi_k]\!]) \geq p$. By hypothesis, $s \models \diamondsuit_p^w a \varphi_k$, that is, $\overline{\mu}_{s,a}([\![\varphi_k]\!]) \geq p$. Then, $\overline{\mu}_{s,a}(\langle t_k \rangle) \geq p$, a contradiction.

Theorem 5 extends a result by [5] to alternating automata. This theorem is necessary to prove the completeness of the logic for weak probabilistic bisimulation in Theorem 6.

Theorem 5. *The relation $\doteq$ is a weak probabilistic bisimulation.*

Proof outline. The proof follows the lines of Theorem 2 of [5], considering also the possibility for probabilistic states to enable transitions labeled by external actions

Theorem 6. *Given the logic $\mathcal{L}_w^A$, for each pair of states s, r of an alternating automaton, $s \approx^p r$ iff $\mathcal{F}(s) = \mathcal{F}(r)$.*

Table 1. Hennessy-Milner logics for discrete probabilistic systems

Model	Logic	Syntax	Bisimulation
Non-Alternating	$\mathcal{L}^N$	$\top \mid \neg\varphi \mid \wedge_I \varphi_i \mid \Diamond a\varphi \mid [\varphi]_p$	$\sim$
	$\mathcal{L}^N_p$	$\top \mid \neg\varphi \mid \wedge_I \varphi_i \mid \dot{\Diamond} a\varphi \mid [\varphi]_p$	$\sim^p$
	$\mathcal{L}^N_w$	$\top \mid \neg\varphi \mid \wedge_I \varphi_i \mid \dot{\Diamond}^w a\varphi \mid [\varphi]_p$	$\approx^p$
Alternating	$\mathcal{L}^A$	$\top \mid \neg\varphi \mid \wedge_I \varphi_i \mid \Diamond_p a\varphi$	$\sim$
	$\mathcal{L}^A_w$	$\top \mid \neg\varphi \mid \wedge_I \varphi_i \mid \dot{\Diamond}^w_p a\varphi$	$\approx^p$
Reactive	$\mathcal{L}^R$	$\top \mid \varphi_1 \wedge \varphi_2 \mid \Diamond_p a\varphi$	$\sim$

Proof outline. Soundness follows by Theorem 1, by the fact that alternating automata are a special case of probabilistic automata via embedding [15] and since $\mathcal{L}^A_w$ is a sublogic of $\mathcal{L}^N$. Completeness follows by Theorem 5 and Lemma 7.

7 Concluding Remarks

We have studied logical characterizations, in terms of Hennessy-Milner style logics, of strong, strong probabilistic and weak probabilistic bisimulations for probabilistic automata [12]. Our three logics are defined on measures over states rather than on single states, and add a new operator $[\varphi]_p$ to the classical Hennessy-Milner logic, that measures the probability of the set of states that satisfy a formula. Compared to other existing logics for reactive and alternating systems [10,5], our logics keep the $\Diamond$ operator of Hennessy-Milner rather than replacing it with $\Diamond_p$, at the cost of adding a more powerful operator to measure probabilities and to handle coexistent probability and nondeterminism.

We have then studied restrictions on probabilistic automata that embed the alternating models and at the same time can be characterized by the logics of [5]. These restrictions impose that each state that enables a probabilistic transition enables only one transition, which is the key property to keep alternative characterizations of bisimulation relations in terms of maximal probabilities [11,5]. This result is important because it explains what are the key features of the alternating models that make them more tractable from the algorithmic point of view. Recall indeed that weak bisimulations are decidable in polynomial time in the alternating models [11] due to their alternative characterization in terms of maximal probabilities, while they are decidable in exponential time for probabilistic automata [2].

Our long term goal is to extend all the theory of probabilistic automata to non-discrete probability measures. The logical characterizations studied in this paper will provide us important guidelines for the definitions to propose in this more general setting.

References

1. M. Bravetti and P.R. D'Argenio. Tutte le algebre insieme: Concepts, discussions and relations of stochastic process algebras with general distributions. In B Haverkort et al, editor, *Proc. of GI/Dagstuhl Research Seminar Validation of Stochastic Systems,* Dagstuhl Germany, volume 2925 of *Lecture Notes in Computer Science*, pages 44–88. Springer, 2004.
2. S. Cattani and R. Segala. Decision algorithms for probabilistic bisimulation. In P. Jankar and M. Kretinsky, editors, *Proceedings of CONCUR 2002,* Brno, Czech Republic, volume 2421 of *Lecture Notes in Computer Science*, pages 371–385. Springer-Verlag, 2002.
3. Y. Deng. *Axiomatisations and Types for Probabilistic and Mobile Processes.* PhD thesis, Ecole des Mines de Paris, 2005.
4. J. Desharnais, A. Edalat, and P. Panangaden. A logical chracterization of bisimulation for labelled markov processes. *Proceedings of the 13th IEEE Symposium On Logic In Computer Science*, 179(2):478–489, 1998.
5. J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Weak bisimulation is sound and complete for PCTL. In M. Ketnsky L. Brim, P. Janar and A. Kuera, editors, *Proc. CONCUR 2002*, volume 2421 of *LNCS*, pages 355–370, 2002.
6. R.J. van Glabbeek, S.A. Smolka, B. Steffen, and C.M.N. Tofts. Reactive, generative, and stratified models of probabilistic processes. In *Proceedings* 5^{th} *Annual Symposium on Logic in Computer Science,* Philadelphia, USA, pages 130–141. IEEE Computer Society Press, 1990.
7. H. Hansson. *Time and Probability in Formal Design of Distributed Systems.* PhD thesis, Department of Computer Science, Uppsala University, 1991.
8. M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985.
9. B. Jonsson and K.G. Larsen. Specification and refinement of probabilistic processes. In *Proceedings of the 6th IEEE Symposium on Logic in Computer Science*, pages 266–277, Amsterdam, July 1991.
10. K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, September 1991.
11. A. Philippou, I. Lee, and O. Sokolsky. Weak bisimulation for probabilistic systems. In C. Palamidessi, editor, *Proceedings of CONCUR 2000,* University Park, PA, USA, volume 1877 of *Lecture Notes in Computer Science*, pages 334–349. Springer-Verlag, 2000.
12. R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, MIT, Dept. of Electrical Engineering and Computer Science, 1995. Also appears as technical report MIT/LCS/TR-676.
13. R. Segala. Probability and nondeterminism in operational models for concurrency. In *Proceedings of CONCUR'2006*, volume 4137 of *Lecture Notes in Computer Science*, pages 64–78. Springer-Verlag, 2006.
14. R. Segala and N.A. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.
15. R. Segala and A. Turrini. Comparative analysis of bisimulation relations on alternating and non-alternating probabilistic models. In *Proceedings of QEST 2005*, September 2005.
16. A. Sokolova and E.P. de Vink. Probabilistic automata: system types, parallel composition and comparison. In C. Baier, B.R. Haverkort, H. Hermanns, J.-P. Katoen, and M. Siegle, editors, *Validation of Stochastic Systems: A Guide to Current Research*, pages 1–43. LNCS 2925, 2004.

Semantic Barbs and Biorthogonality

Julian Rathke, Vladimiro Sassone, and Paweł Sobociński*

ECS, University of Southampton

Abstract. We use the framework of biorthogonality to introduce a novel semantic definition of the concept of barb (basic observable) for process calculi. We develop a uniform basic theory of barbs and demonstrate its robustness by showing that it gives rise to the correct observables in specific process calculi which model synchronous, asynchronous and broadcast communication regimes.

1 Introduction

Labelled transition systems and structural operational semantics [17] have been the idiomatic approach to the semantics of communicating concurrent systems for many years [13,9]. Such semantics naturally yield many notions of equivalence based on traces or bisimulations. As communication patterns in these calculi grew more complex, there was a need to justify the ad-hoc labelled transition semantics being provided with respect to simpler, more canonical equivalences.

Following the seminal contribution of Berry and Boudol [3], calculi began to be equipped with a reduction relation meaning that widely accepted techniques from the theory of lambda-calculi, such as the definition of a contextually defined reduction congruence, were able to be studied in the setting of calculi for concurrency and mobility. However, even in the setting of CCS [13], reduction congruence is coarser than standard bisimilarity; this led Milner and Sangiorgi to introduce the concept of a basic observable of a process, which came to be dubbed a *barb* [14]. Together with a reduction semantics, barbs yield a canonical notion of process equivalence for most modern calculi.

Barbs are notable in that they are perhaps the most well-known concept of formal concurrent semantics which, despite being frequently used, do *not* actually have a general formal definition. In [14], a barb is understood simply to be a predicate on processes which captures the intuitive notion of basic observable. In many other settings specific barbs are precisely defined but no account is taken of whether these definitions are appropriate. For example, in the calculus, CCS, the choice of barb as being the ability to synchronise on a given name is uncontentious. However, even moving to such a simple setting as an asynchronous version of CCS leads to questions about the suitability of certain synchronisations as barbs. In this setting, it is accepted [2] for instance that the ability to synchronise as a receive action is *not* suitable as a barb. To date though, there is no formal definition which justifies this.

* Research partially supported by EPSRC grant EP/D066565/1.

The goal of this paper is to provide an abstract, semantic definition of what it means to be a barb. We aim to make the definition as general as possible, whilst ensuring that we encompass the intuitive definitions of barbs in various well-known examples. The benefits of this approach will be that in complex languages or formal systems of communication, we will have a canonical definition with which to identify the basic patterns of interaction. The need for more thorough foundational insights is becoming increasingly important with the advent of complex communication patterns in calculi for mobility [5,19] and biological modelling [4].

A defining feature of our approach is that we obtain a notion of *observable* for our calculi based solely on their reduction semantics. Moreover, the observables are required to be suitably minimal so as to allow them to be considered basic. In this sense, our work is related to Leifer and Milner's attempts to obtain a labelled semantics from a reduction semantics [11], however we focus on static capability for interaction between agents and imbue the framework with a notion of 'successful' interaction which is key to understanding the nature of observability.

Our definition of barb is based around the notion of a closure operator, which, given a process (or a set of processes), will construct the set of all processes which offer the immediate interactions of that process (or set of processes). For example, if we take the process: $a!p_0 \parallel b?q_0$, (where $a!$ and $b?$ denote, respectively, the ability to output on a channel a and the ability to receive on a channel b) then its closure is the set of all processes of the form

$$a!p \parallel b?q \parallel r \qquad \text{for any} \qquad p, q, r$$

as these are the processes which offer *both* an $a!$ and $b?$ interaction. This closed set is considered to represent the abstract concept of a $a!$ and $b?$ interaction. Similarly, the closure of $\{a!p_0, b?q_0\}$ is the set of all processes of the form

$$a!p \parallel r \qquad \text{or} \qquad b?q \parallel r$$

that is, all processes which offer an $a!$ *or* a $b?$ interaction. Similar ideas have been used both in logic and computer science, for example in Girard's phase semantics of linear logic [8], Krivine's realisability [6], Pitts' $\top\top$-closure [16] and formal concept theory [20]. In particular, we shall show how all of the aforementioned examples fit within a basic general framework.

To achieve a decent account of minimality in the barbs we need to consider the closed sets that represent the least possible interactions. In the examples above, we should not consider either of these minimal as they are made up of combinations of distinct interaction capabilities in CCS. To capture this we make use of the notion of *irreducibility* from algebraic geometry, see [18] for an introductory account. Indeed, we only consider those closed sets of processes which cannot be decomposed into a combination of two separate smaller closed sets. This certainly rules out the closure of $\{a!p_0, b?q_0\}$ from being a barb, but we also wish to rule out the closure of $a!p_0 \parallel b?q_0$. This is done in a similar, and dual, way of considering the closed sets of contexts which this process can successfully interact with and demanding that this closed set of contexts cannot be decomposed either. In this example, the contexts which successfully interact are of the form

$$a?p \parallel r \quad \text{or} \quad b!q \parallel r$$

and we see that these are not minimal in the sense suggested.

This use of irreducibility in both the closed sets of processes and the contexts for them is, to our knowledge, novel and is a main contribution of the paper. The remainder of the paper is arranged so that we present our general notion of interaction frameworks and biorthogonality along with familiar examples of such frameworks. We then tailor the setting to specifically allow us to study interaction in processes and we introduce three paradigmatic example process calculi. The definition of irreducibility and the main definition of barb is then given. After this we study properties of barbs in example languages, including full process languages such as π-calculus.

2 Biorthogonality

The notion of *biorthogonality* is the device underpinning our entire approach. It is at the same time a conceptual and a technical tool; its strength resides in the simplicity and elegance with which it captures mathematically –amongst other things– the notion of test as relationship between processes and contexts. We will use it to understand the concept and the role of 'observation' at a foundational level. We shall omit the proofs in this section because all of the results are basic and well-known, even if we are unaware of previous work which collected all of the examples listed at the conclusion of this section as instances of a single framework.

We assume the following basic ingredients:

- sets $\mathbf{T}$, $\mathbf{\Gamma}$ and $\mathbf{\Pi}$ which we shall self-evidently refer to as *terms*, *contexts* and *processes*, respectively;
- a function $@\colon\ \mathbf{T} \times \mathbf{\Gamma} \to \mathbf{\Pi}$, representing the insertion of a term in a context to yield a process;
- a subset $\perp\ \subseteq \mathbf{\Pi}$ of *successful* processes, which we think of as a unary predicate.

Informally, it is we think of $t \mathbin{@} \gamma \in \perp$ as stating that t passes the test (represented by context) γ, and vice versa; we shall say that t is successful for γ and that γ is successful for t. We shall usually write the shorthand form $t \perp \gamma$ to mean $t \mathbin{@} \gamma \in \perp$. From these basic notions we derive the maps:

$$\begin{array}{l} (-)^{\perp} : \mathcal{P}(\mathbf{T}) \to \mathcal{P}(\mathbf{\Gamma}) \\ T \mapsto \{\, \gamma \in \mathbf{\Gamma} \,:\, \forall t \in T.\, t \perp \gamma \,\} \end{array} \quad \text{and} \quad \begin{array}{l} (-)^{\perp} : \mathcal{P}(\mathbf{\Gamma}) \to \mathcal{P}(\mathbf{T}) \\ \mathit{\Gamma} \mapsto \{\, t \in \mathbf{T} \,:\, \forall \gamma \in \mathit{\Gamma}.\, t \perp \gamma \,\}. \end{array}$$

Thus, given a set T of terms, $T^{\perp}$ is the set of contexts which are successful for every $t \in T$, and similarly, given a set $\mathit{\Gamma}$ of contexts, $\mathit{\Gamma}^{\perp}$ is the set of terms which are successful for every $\gamma \in \Gamma$. The reader is asked to tolerate the abuse of notation, justified by the symmetry of the definitions, as the type of the argument to $(-)^{\perp}$ will always be made clear.

The following lemma proves some basic properties of the functions, and strengthens the intuition of their combination $(-)^{\perp\perp}$ as a closure operation.

Lemma 1

(i) $T \subseteq U$ *implies* $U^{\perp} \subseteq T^{\perp}$*;*
(ii) $T \subseteq T^{\perp\perp}$*;*
(iii) $T^{\perp} = T^{\perp\perp\perp}$ *(and therefore* $(T^{\perp\perp})^{\perp\perp} = T^{\perp\perp}$*).*

This allows us to define the central notion of a biorthogonal set of terms. Due to the symmetry in the definitions, one can also define a biorthogonal set of contexts, we leave this fact implicit. The lemma which follows the definition illustrates some intuitively equivalent formulations of biorthogonality.

Definition 2 (Biorthogonals). We shall call a subset T' of $\mathbf{T}$ a *biorthogonal* if there exists $T \subseteq \mathbf{T}$ such that $T' = T^{\perp\perp}$.

Lemma 3. *The following are equivalent:*

(i) T *is a biorthogonal;*
(ii) $T = T^{\perp\perp}$*;*
(iii) there exists $\Gamma \subseteq \mathbf{\Gamma}$ *such that* $T = \Gamma^{\perp}$*.*

The basic algebraic properties of biorthogonals are expressed by the following two lemmas.

Lemma 4. $T^{\perp\perp}$ *is the smallest biorthogonal containing* T*, for all* $T \subseteq \mathbf{T}$*.*

Lemma 5. *Biorthogonals are closed under arbitrary intersections.* [1]

Two sets of terms T and U are said to be *logically congruent* when $T^{\perp} = U^{\perp}$. Logical congruence is an equivalence relation. The remainder of this section is devoted to illustrating several examples of the basic framework.

Example 6 (Girard's phase semantics for linear logic [8]). Let $\langle P, \cdot, 1\rangle$ be a commutative monoid with identity; let $\mathbf{T} = \mathbf{\Gamma} = \mathbf{\Pi}$ be P and @ be the action '$\cdot$'. Let $\perp \subseteq P$. Then P is a phase space and the biorthogonals are its *facts*.

Example 7 (Krivine's realizability). For simplicity, let $\mathbf{T}$ be the set of terms of the simply-typed lambda calculus, and $\mathbf{\Gamma}$ a set of stacks, see [6] for details. Then let $\mathbf{\Pi}$ be the set of syntactic expressions $\langle t \mid \gamma\rangle$, for $t \in \mathbf{T}$ and $\gamma \in \mathbf{\Gamma}$. The expected reduction semantics reduction semantics on is defined on $\mathbf{\Pi}$ and $\perp \subseteq \mathbf{\Pi}$ is taken be left-closed with respect to it:

$$\langle t \mid \gamma\rangle \in \perp \quad \text{and} \quad \langle t' \mid \gamma'\rangle \rightarrow^{*} \langle t \mid \gamma\rangle \quad \text{implies} \quad \langle t' \mid \gamma'\rangle \in \perp.$$

Under the obvious interpretation of $\mid$ as @, the biorthogonals are called *truth values*. See also [12] for a recent application of this technique.

[1] Biorthogonals are *not* in general closed under even finite unions, as we shall illustrate in Example 19.

Example 8 ($\top\top$-closed relations). We'll use Abadi's 'semantic' interpretation [1] of Pitts' $\top\top$-closed relations [16]. Let A, B be CPOs with bottom $\bot$. Let 2 be the poset $\{\{\bot, \top\}, \{\bot \sqsubseteq \top\}\}$, and let 2^A (resp. 2^B) denote the set of monotonic, strict and continuous functions $A \to 2$ (resp. $B \to 2$). Let $\mathbf{T} = A \times B$, $\mathbf{\Gamma} = 2^A \times 2^B$, and $\mathbf{\Pi} = 2 \times 2$, and define $@ : (A \times B) \times (2^A \times 2^B) \to 2 \times 2$ to be the obvious evaluation function $@((a, b), (f, g)) = (fa, gb)$. Let $\bot$ be the equality relation on 2. The biorthogonals are the $\top\top$*-closed relations.*

Example 9 (Formal concept theory). Introduced by Wille [20], this subject shares our notion of frameworks for biorthogonality and was proposed as a way of restructuring lattice theory to account for the interaction between what was called *objects* and *attributes.* The notion of 'concept' corresponds to our notion of biorthogonal. The goal of this research effort seems to be one of useful representations of lattices and efficient computations on these.

Example 10 (Classical algebraic geometry). Let $\mathbf{T}$ be an n-dimensional affine space over an algebraically closed field k (say k^n). Let $\mathbf{\Gamma}$ be the ring $k[x_1, \ldots, x_n]$ of polynomials with n variables. Let $\mathbf{\Pi}$ be the field k, and $@ : k^n \times k[x_1, \ldots, x_n] \to k$ be the evaluation map. Let $\bot = \{0\}$. Then the biorthogonals are the *affine varieties.* Varieties which are *irreducible*, that is, cannot be written as a nontrivial union of two other varieties, are of particular interest. We shall make use of irreducibility in §5.

3 A Refined Model: $\bot$ as an Ideal

In preparation for applications of biorthogonality to sets of terms with richer structure, we now refine our framework with a specialised notion of composition. Recall that for $\langle M, \cdot, 1\rangle$ a commutative monoid, an ideal I is a subset of M closed under the action of M, i.e., $\forall i \in I\, \forall m \in M.\ i \cdot m \in I$. We shall write $[M']$ to mean the ideal generated by a set $M' \subseteq M$. We shall now extend the basic framework as follows:

- $\mathbf{T} \subseteq \mathbf{\Pi}$ and $\mathbf{\Gamma} \subseteq \mathbf{\Pi}$;
- there exists a binary operator $\|$ and $\epsilon \in \mathbf{\Pi}$ such that $\langle \mathbf{\Pi}, \|, \epsilon\rangle$ is a commutative monoid, and for all $t \in \mathbf{T}$ and $\gamma \in \mathbf{\Gamma}$ we define $t\,@\,\gamma = t \parallel \gamma$; moreover, $\mathbf{T}$ and $\mathbf{\Gamma}$ are submonoids, ie $\epsilon \in \mathbf{T}$, $\epsilon \in \mathbf{\Gamma}$ and for all $t, t' \in \mathbf{T}$, $t \parallel t' \in \mathbf{T}$ and for all $\gamma, \gamma' \in \mathbf{\Gamma}$, $\gamma \parallel \gamma' \in \mathbf{\Gamma}$;
- $\bot$ is an ideal of $\langle \mathbf{\Pi}, \|, \epsilon\rangle$.

Example 11 (Phase semantics for affine linear logic). With reference to Example 6, observe that collapsing $\mathbf{T} = \mathbf{\Gamma} = \mathbf{\Pi}$ *and* additionally requiring $\bot$ to be an ideal, we obtain phase semantics for affine linear logic; that is, the logic obtained from linear logic by introducing weakening.

The following lemma lists some basic properties of the refined framework. Let $\bot_{\mathbf{\Gamma}} = \bot \cap \mathbf{\Gamma}$ and $\bot_{\mathbf{T}} = \bot \cap \mathbf{T}$.

Lemma 12

(i) For every $T \subseteq \mathbf{T}$, we have $\perp_{\mathbf{\Gamma}} \subseteq T^{\perp}$;
(ii) for every $\Gamma \subseteq \mathbf{\Gamma}$, we have $\perp_{\mathbf{T}} \subseteq \Gamma^{\perp}$;
(iii) every biorthogonal $V \subseteq \mathbf{T}$ is an ideal of $\mathbf{T}$;
(iv) every biorthogonal $\Gamma \subseteq \mathbf{\Gamma}$ is an ideal of $\mathbf{\Gamma}$.

Proof. (i) Obvious, since $\perp$ is an ideal of $\mathbf{\Pi}$, for any $\pi \in \perp_{\mathbf{\Gamma}}$ and $t \in T$ we have $\pi \parallel t \in \perp$, so $\pi \in T^{\perp}$. (ii) Is immediate by duality. (iii) For arbitrary $t \in \mathbf{T}$, $v \in V$, $\gamma \in V^{\perp}$, we have that $t \parallel v \parallel \gamma \in \perp$ since $v \parallel \gamma \in \perp$. (iv) Is immediate by duality. □

4 Idealised Process Calculi

Idealised process calculi, introduced here, will be the main examples of the extended framework defined in the previous section. Although the calculi we consider do not cover the entire realm of process models systematically, they are carefully chosen to span a significant spectrum of cases. In particular, our idealised process calculi have only two constructs, action prefix and parallel composition. They are equipped with different reduction semantics which specifies the communication regime used.

The set of 'ordinary' processes P can be generated freely by a simple grammar, given in Fig 1 for some fixed countably infinite set of channel names A. Basic actions $a?$ and $a!$ represent action/co-action synchronisation pairs à la CCS. We consider the set of processes to be quotiented by structural congruence $\equiv$ which makes $\langle P, \parallel, \epsilon\rangle$ a commutative monoid. More concretely, $\equiv$ is the smallest congruence which includes the equations:

$$P \parallel Q \equiv Q \parallel P \quad \text{and} \quad P \parallel \epsilon \equiv P.$$

Ordinary processes will form the set $\mathbf{T}$ of terms.

The set of contexts, denoted C, is obtained in a similar way. A typical context is a finite parallel composition of $\checkmark$s prefixed by a single name. Here, the syntactic entity $\checkmark$ represents success. Our contexts have a simple structure because we shall always be interested in the top level structure of a term; indeed, we shall test only for a process's immediate capabilities for interaction. Finally, $P_{\checkmark}$ is the set of extended processes. An extended process is a finite parallel composition of ordinary processes, contexts and $\checkmark$s. As done for P, we also quotient C and $P_{\checkmark}$ by the structural congruence $\equiv$.

In order to simplify notation, we shall often denote action prefixing with mere juxtaposition and also write simply $a!$ or $a?$ to mean $a!\epsilon$ or $a?\epsilon$, respectively.

Different communication regimes are specified with individual sets of reduction schemas over extended processes. The *reduction semantics* is obtained by closing the reduction rules with respect to $\parallel$ in the sense that if $t \to t'$ then $t \parallel \sigma \to t' \parallel \sigma$ for any $\sigma \in P_{\checkmark}$.

In the asynchronous case, instead of following the tradition of allowing an output to prefix only the null process, we simply only include the reductions

$$\begin{array}{rcl}
P & ::= & \epsilon \;\mid\; P \parallel P \;\mid\; M.P \\
M & ::= & a? \;\mid\; a! \qquad (a \in A) \\
C & ::= & \epsilon \;\mid\; C \parallel C \;\mid\; M_{\checkmark} \\
M_{\checkmark} & ::= & M\checkmark \\
P_{\checkmark} & ::= & P_{\checkmark} \parallel P_{\checkmark} \;\mid\; P \;\mid\; C \;\mid\; \checkmark
\end{array}$$

Fig. 1. Idealised process calculi: processes P and contexts C

where this is the case. Thus in the asynchronous calculus a term of the form $a!.P$ where $P \neq \epsilon$ is operationally indistinguishable from ϵ as it cannot take active part in any reduction.

Example 13 (Synchrony). The reduction rules are given by the schema below where P and Q range over arbitrary extended processes.

$$a!P \parallel a?Q \;\rightarrow\; P \parallel Q \quad (a \in A)$$

Example 14 (Asynchrony). The reduction rules are given by the schema below where P ranges over arbitrary extended processes.

$$a! \parallel a?P \quad \rightarrow \quad P \qquad (a \in A)$$

Example 15 (Broadcast). The reduction rules are given by the schema below; I is any finite (possibly empty) set while P and Q_i range over extended processes.

$$a!P \parallel \prod_{i \in I} a?Q_i \quad \rightarrow \quad P \parallel \prod_{i \in I} Q_i$$

In each of the cases, the three simple calculi described above fit within our refined framework: we let $\mathbf{T} = P$, $\mathbf{\Gamma} = C$ and $\mathbf{\Pi}$ be parallel compositions of these. We define application to be parallel composition:

$$\begin{array}{c}
@ : P \times C \rightarrow \mathbf{\Pi} \\
\quad (t, \gamma) \;\mapsto\; t \parallel \gamma.
\end{array}$$

In order to define the success predicate we make use of the extended processes $P_{\checkmark}$. We call an extended process (a member of $P_{\checkmark}$) '*spent*' if has precisely one $\checkmark$ at top level - ie $\checkmark$ is a parallel component. An extended process is deemed to be *successful* if it reduces in one step to a spent extended process. In formulae:

$$\pi \in \perp \quad \text{iff} \quad \exists \pi' \in P_{\checkmark}. \quad \pi' \text{ spent } \wedge\ \pi \rightarrow \pi'.$$

As usual, we write $p \perp \pi$ when $p @ \pi = p \parallel \pi$ is a successful extended process. Essentially, the definition of success ensures that a context has to either engage the term (since a context which would reduce by itself via a non-trivial interaction, using any of our three reduction schemas, would result in two instances of $\checkmark$) or have an atomic parallel component which reduces by itself to $\checkmark$ (for instance, the context $a!\checkmark$ in the broadcast paradigm). It is clear that in all three examples $\perp_{\mathbf{T}} = \varnothing$. In Examples 13 and 14, also $\perp_{\mathbf{\Gamma}} = \varnothing$. In Example 15 $\perp_{\mathbf{\Gamma}} = [\{\, a!\checkmark \;:\; a \in A \,\}]$, since any $a!\checkmark$ can reduce to $\checkmark$ in one step.

We conclude this section by illustrating typical biorthogonals for the simple calculi illustrated above. Recall that we use the square bracket notation to mean the smallest ideal generated by the indicated set of terms.

Example 16 (Synchrony – cf Example 13). Biorthogonals for basic terms with single communication capability are the set of all terms which have that immediate capability. So we have $\{a!\}^{\perp\perp} = ([a?\checkmark] \cup \perp)^{\perp} = [a!P]$, the set of all terms ready to output on a. Symmetrically, $\{a?\}^{\perp\perp} = [a?P]$. In general, starting with a *single* term, the biorthogonal yields all the terms that have the same selection of immediate capabilities for communication. For instance $\{a? \parallel b!c?\}^{\perp\perp} = [a!\checkmark, b?\checkmark]^{\perp} = [a?P \parallel b!Q]$. For *sets* of terms, the biorthogonal is the smallest biorthogonal which contains all of the terms. In the case of Example 13 this is simply the union of the biorthogonals of the individual terms. However as hinted at previously and illustrated by the calculus of Example 19, this need not be so, as the union of biorthogonals is in general not a biorthogonal.

Example 17 (Asynchrony – cf Example 14). We have again $\{a!\}^{\perp\perp} = [a?\checkmark]^{\perp} = [a!P]$. However, since output actions $a!$ cannot guard $\checkmark$ in a reduction, we have $\{a?\}^{\perp} = \perp_{\mathbf{\Gamma}}$ and therefore $\{a?\}^{\perp\perp} = \mathbf{T}$.

Example 18 (Broadcast – cf Example 15). Since $\perp_{\mathbf{\Gamma}} = [\{\, a!\checkmark \;:\; a \in A \,\}]$, we have $\{a?\}^{\perp\perp} = \perp_{\mathbf{\Gamma}}^{\perp} = \mathbf{T}$. Once again, $\{a!\}^{\perp\perp} = [a!P]$.

5 Irreducibility and Barbs

As indicated previously, irreducibility is an important concept of algebraic geometry. Here we shall apply the concept to biorthogonals, which in general are not closed under finite unions:

Example 19. Consider the following reduction rules which capture an interaction pattern reminiscent of the features of the join calculus [7]:

$$\begin{array}{rcll} a?P \parallel a!P' & \to & P \parallel P' & \qquad\qquad (a \in A) \\ ab?P \parallel a!P' \parallel b!P'' & \to & P \parallel P' \parallel P'' & \qquad\qquad (a, b \in A) \end{array}$$

Here the $ab?$ prefix needs the presence of both $a!$ and $b!$ to reduce. Then $\{a?\checkmark\}^{\perp\perp} = [a!P]^{\perp} = [a?\checkmark]$ and similarly $\{b?\checkmark\}^{\perp\perp} = [b!P]^{\perp} = [b?\checkmark]$. However, $\{a?\checkmark, b?\checkmark\}^{\perp\perp} = [a!P \parallel b!Q]^{\perp} = [a?\checkmark, b?\checkmark, ab?\checkmark]$ is strictly larger than $[a?\checkmark] \cup [b?\checkmark]$. Thus we see that the union of the two biorthogonals $[a?\checkmark]$ and $[b?\checkmark]$ is not itself a biorthogonal.

Definition 20 (Sum). Given biorthogonals V_1 and V_2, their *sum* $V_1 + V_2$ is defined to be the smallest biorthogonal which contains both V_1 and V_2.

It is easy to verify that $V_1 + V_2 = (V_1 \cup V_2)^{\perp\perp} = (V_1^{\perp} \cap V_2^{\perp})^{\perp}$. Thus, intuitively, $V_1 + V_2$ consists of all the terms which pass the test suites of each of V_1 and V_2. Sum $+$ is clearly commutative and easily checked to be associative. It has $\varnothing^{\perp\perp}$ as the identity. Furthermore, the binary operations $+$ and $\cap$ on biorthogonals are related by De Morgan equations: $V \cap W = (V^{\perp} + W^{\perp})^{\perp}$ and $V + W = (V^{\perp} \cap W^{\perp})^{\perp}$.

A sum $V = V_1 + V_2$ is said to be *nontrivial* when $V \neq V_1$ and $V \neq V_2$.

Definition 21 (Irreducibility). A biorthogonal is said to be *irreducible* if it cannot be written as a nontrivial sum of two biorthogonals.

The following definition is one of the central contributions of this paper.

Definition 22 (Barb). A *barb* is defined to be an proper irreducible biorthogonal B, whose orthogonal $B^{\perp}$ is also proper irreducible. For $T \subseteq \mathbf{T}$ any set of terms, T is said to *barb* on B, written $T{\downarrow_B}$, if $T^{\perp\perp} \subseteq B$. In particular, a single term t barbs on B when $\{t\}^{\perp\perp} \subseteq B$. A term $t \in \mathbf{T}$ is said to weakly barb on B, written $t{\Downarrow_B}$, if there exists t' such that $t \to^{*} t'$ and $t' {\downarrow_B}$.

The definition identifies barbs abstractly as 'replete' or 'maximal' sets of terms that exhibit a given basic behaviour in tests (biorthogonality). Such behaviour must be nontrivial (properness), and 'atomic' (irreducibility). The irreducibility condition on $B^{\perp}$ means that barbs are testable by suitably atomic set of contexts.

Definition 22 allows the immediate possibility of defining the standard notions of (strong and weak) barbed bisimilarity and (strong and weak) barb congruence, which provide canonical notions of equivalence. Note that there is a choice in how one defines the congruence, one can either follow Milner and Sangiorgi's original definition [14] of the largest congruence contained in barbed bisimilarity, or to take the largest congruence which is also a bisimulation. The latter equivalences are sometimes described as *dynamic*, see [15,10].

The definition of barb which we have formulated is widely applicable; however, in any given framework, it may take some effort to identify the irreducible biorthogonals. For this reason we now seek to find a straightforward characterisation of the barbs in our range of example calculi. For now, in order to do this we have tailored the success predicate towards handshaking synchronisation and we shall also make further restrictions on the type of calculi considered. These restrictions, while intuitive and natural, disallow some more complex examples of interaction (eg Example 19), which will be the subject of future work.

6 Simple Calculi

In this section we shall require additional structure on the algebra of biorthogonals. A calculus is said to be *simple* when its algebra of biorthogonals enjoys the extra structure.

Definition 23. A biorthogonal V is said to be *finitely generated* (fg) when there exist $v_1, \ldots, v_n \in V$ such that $V = \{v_1, \ldots, v_n\}^{\perp\perp}$.

In any framework, the sum of two fg biorthogonals is fg: if $V = \{v_1, \ldots, v_k\}^{\perp\perp}$ and $W = \{w_1, \ldots, w_l\}^{\perp\perp}$ then $V + W = \{v_1, \ldots, v_k, w_1, \ldots, w_l\}^{\perp\perp}$.

Lemma 24. *Any irreducible fg biorthogonal is generated by one of its elements. Thus if V is fg and irreducible then there exists $v \in V$ such that $V = \{v\}^{\perp\perp}$.*

Proof. Suppose that V is irreducible and fg. Then there exist $v_1, \ldots, v_n \in V$ such that $V = \{v_1, \ldots, v_n\}^{\perp\perp}$. Indeed, clearly $V = \{v_1\}^{\perp\perp} + \{v_2, \ldots, v_n\}^{\perp\perp}$. By irreducibility, either $V = \{v_1\}^{\perp\perp}$ and we are finished or $V = \{v_2, \ldots, v_n\}$, where we repeat the procedure. □

In particular, if all biorthogonals are fg then all irreducible biorthogonals can be generated by a single element. If biorthogonals are closed under binary union, then the converse, that all single element generated biorthogonals are irreducible, is also true.

Lemma 25. *Suppose that biorthogonals are closed under finite union, in other words $V + W = V \cup W$. Then any biorthogonal generated by a single element is irreducible.*

Proof. Suppose that $V = \{v\}^{\perp\perp} = V_1 + V_2 = V_1 \cup V_2$. Then either $v \in V_1$ or $v \in V_2$. In the first case $V = \{v\}^{\perp\perp} \subseteq V_1$, meaning that $V = V_1$; similarly, in the second case $V = V_2$. □

We shall now show that the calculi of Examples 13, 14 and 15 have biorthogonals which are closed under unions. We shall need two technical lemmas.

Lemma 26. *The examples satisfy the following dual properties:*

(i) for all contexts $\gamma \in \mathbf{\Gamma}$ and terms t_1, $t_2 \in \mathbf{T}$:

$$t_1 \parallel t_2 \perp \gamma \quad \textit{iff} \quad t_1 \perp \gamma \textit{ or } t_2 \perp \gamma;$$

(ii) for all terms $t \in \mathbf{T}$ and contexts γ_1, $\gamma_2 \in \mathbf{\Gamma}$:

$$t \perp \gamma_1 \parallel \gamma_2 \quad \textit{iff} \quad t \perp \gamma_1 \textit{ or } t \perp \gamma_2.$$

Proof. The 'if' direction is obvious for both cases.

Suppose that $t_1 \parallel t_2 \perp \gamma$. If γ reduces to a spent process with no need for interaction then clearly both $t_1 \perp \gamma$ and $t_2 \perp \gamma$. Otherwise, since all reduction rules of Examples 13 and 14 have at most two parallel components in the redex, and γ has to provide at least one (since both t_1 or t_2 are ordinary processes and thus have no occurrences of $\checkmark$), it follows that $t_1 \perp \gamma$ or $t_2 \perp \gamma$. In Example 15, it is enough to consider γ of the form $a?\checkmark$. Clearly then either t_1 or t_2 must have output capability on a, and thus $t_1 \perp \gamma$ or $t_2 \perp \gamma$.

Now suppose that $t \perp \gamma_1 \parallel \gamma_2$. Notice that $\gamma_1 \parallel \gamma_2$ reduces to a spent process without interaction if and only if either γ_1 or γ_2 (or both) can do so independently. Indeed, any interaction between γ_1 and γ_2 results in two instances of $\checkmark$ result in the reactum. Hence t must provide a part of the redex. □

Proposition 27. *The examples satisfy* $\{t_1 \parallel t_2\}^\perp = \{t_1\}^\perp \cup \{t_2\}^\perp$*, for any terms* t_1 *and* t_2*. More generally, for* T_1 *and* T_2 *any sets of terms,* $(T_1 \parallel T_2)^\perp = T_1^\perp \cup T_2^\perp$*, where* $\parallel$ *is extended to sets in the obvious pointwise manner.*

Proof. Clearly the more general second statement implies the first. Also, it is obvious that $A^\perp \cup B^\perp \subseteq (A \parallel B)^\perp$. Now suppose that $\pi \in (A \parallel B)^\perp$. If, for all $a \in A$, $\pi \perp a$, then $\pi \in A^\perp$ and we are finished. Suppose then that there exists $a \in A$ such that not $\pi \perp a$. Then, by the assumption on π, for all $b \in B$, $\pi \perp a \parallel b$. Lemma 26 implies that $\pi \perp b$. Thus $\pi \in B^\perp$. □

The examples satisfy $\{\gamma_1 \parallel \gamma_2\}^\perp = \{\gamma_1\}^\perp \cup \{\gamma_2\}^\perp$, for all contexts γ_1 and γ_2. More generally, $(\Gamma_1 \parallel \Gamma_2)^\perp = \Gamma_1^\perp \cup \Gamma_2^\perp$ for all sets of contexts Γ_1 and Γ_2.

Corollary 28. *In the examples, biorthogonals are closed under finite unions.*

Definition 29 (Simple calculi). We shall say that an idealised process calculus is simple when:

(i) $V + W = V \cup W$, for all biorthogonals V and W;
(ii) every irreducible biorthogonal has a single generating element.

Proposition 30. *The calculi introduced in Examples 13, 14 and 15 are simple.*

Proof. Corollary 28 shows that the calculi satisfy (i). If V is a finitely generated irreducible biorthogonal, then it is generated by a single element as shown in Lemma 24. It remains to show that any irreducible biorthogonal is finitely generated, the proof of this fact is more involved and will appear in a fuller version of this paper. □

Proposition 31. *In a simple idealised process calculus, the barbs coincide with proper biorthogonals generated by a single term whose orthogonal is also proper and generated by a single term.*

Proof. By definition, barbs are proper irreducible biorthogonals B whose orthogonal $B^\perp$ is also proper irreducible. We know that, by simplicity, any irreducible biorthogonal has a single generating element. On the other hand, any biorthogonal which is generated by a single term is irreducible by Lemma 25, thus if both the biorthogonal and its orthogonal are proper and generated by a single term then they are both irreducible, and the biorthogonal is a barb.

Remark 32. Several interesting and reasonable features of calculi fall outside the framework of simple calculi. For instance, the ability of synchronising with two separate processes as illustrated in Example 19, can mean that biorthogonals are not closed under binary unions. The requirement of irreducibility in the definition of 'simple' above is necessary even for the simplest calculi. An example below, written in the synchronous idealised language of Example 13, demonstrates a non finitely generated reducible biorthogonal. This example is useful because it shows that, in a finitary calculus, it is irreducibility that forces finite generation for barbs. We also provide an example of a calculus which contains non finitely generated irreducible biorthogonals. In this case, the calculus contains non-finitary reduction rules.

Example 33 (Non finitely generated reducible biorthogonal). We use the idealised synchronous language of Example 13 and let $T = \{\, t_i \,:\, i \geq 0 \,\}$; it holds that $T^{\perp}$ is $\{\, [\pi_i] \,:\, i \geq 0 \,\}$, where we define each t_i and π_i as below:

$$t_i = \prod_{j<i} b_j! \parallel a_i! \qquad \pi_i = b_i?\checkmark \parallel \prod_{j\leq i} a_j?\checkmark .$$

Consequently $T^{\perp\perp} = \{\, [t_i] \,:\, i \geq 0 \,\}$. We claim that $T^{\perp\perp}$ is not finitely generated but is reducible as we can express $T^{\perp\perp}$ as $T^{\perp\perp} = \{t_0\}^{\perp\perp} + \{\, t_i \,:\, i \geq 1 \,\}^{\perp\perp}$ with $\{t_0\}^{\perp\perp} \neq \{\, t_i \,:\, i \geq 1 \,\}^{\perp\perp}$.

Example 34 (Non finitely generated irreducible biorthogonals). Suppose that the set of prefixes is

$$M \quad ::= \quad a_i? \quad | \quad b_i! \quad | \quad b_\omega \qquad (i \in \mathbb{N})$$

and that there is an infinite reaction rule schema of the form:

$$a_i?P \parallel b_j!Q \rightarrow P \parallel Q \quad (i \leq j \vee j = \omega).$$

For each i, $\{a_i?\}^{\perp\perp} = \{\, b_j!\checkmark \,:\, i \leq j \vee j = \omega \,\}^{\perp} = [\{\, a_k P \,:\, P \in \mathbf{T}, k \leq i \,\}]$. In particular, there is an infinite ascending chain

$$\{a_0\}^{\perp\perp} \subset \{a_1\}^{\perp\perp} \subset \ldots$$

Now the biorthogonal $\{b_\omega\}^{\perp} = \{\, a_i \,:\, i \in \mathbb{N} \,\}^{\perp\perp}$ is irreducible and not finitely generated.

We are now in a position to show the barbs for each of our three main examples. Before we do this, it is helpful to consider why certain biorthogonal are *not* barbs. Firstly, any non irreducible biorthogonal is not a barb, for instance, in the synchronous case $\{a?\}^{\perp\perp} \cup \{b?\}^{\perp\perp}$. Intuitively, this is so because the biorthogonal does not capture a single set of capabilities for interaction, its terms have either one type ($a?$) *or* the other ($b?$). An irreducible biorthogonal may also fail to be a barb because its orthogonal is reducible, meaning that no suitable single test exists for it – for instance, $\{a! \parallel b!\}^{\perp} = \{a?\}^{\perp\perp} \cup \{b?\}^{\perp\perp}$. Intuitively, this is because each of the terms in the biorthogonal has two distinct possibilities for interaction, here both $a?$ *and* $b?$. Finally, when a biorthogonal is not proper (it contains all the terms), it is not a barb. In this case, the term does not have non-trivial observations (cf $\{a?\}^{\perp\perp}$ in the asynchronous case).

We know because of Propositions 30 and 31 that in identifying barbs in our simple calculi it is sufficient to examine biorthogonals of singletons:

Proposition 35 (Synchronous barbs). *In Example 13, the barbs are:*

1. *for any $a \in A$, the processes which can output on a: $\{a!\}^{\perp\perp}$;*
2. *for any $a \in A$, the processes which can input on a: $\{a?\}^{\perp\perp}$.*

Proof. Both $\{a!\}^{\perp\perp}$ and $\{a?\}^{\perp\perp}$ are proper biorthogonals with orthogonals $\{a?\checkmark\}^{\perp\perp}$ and $\{a!\checkmark\}^{\perp\perp}$, respectively. Both are thus barbs by Proposition 31.

If the term has (immediate) capability to communicate on two separate channels then its orthogonal is reducible, hence it cannot be a barb. It thus suffices to notice that $\{a! \parallel a!\}^{\perp\perp} = \{a!\}^{\perp\perp}$ and $\{a? \parallel a?\}^{\perp\perp} = \{a?\}^{\perp\perp}$. □

Proposition 36 (Asynchronous barbs). *For the calculus of Example 14, the barbs are, for any $a \in A$, the processes which can output on a, $\{a!\}^{\perp\perp}$.*

Proof. As before, it suffices to check terms which have a communication capability on a single name. But $\{a!\}^{\perp\perp} = \{[a?\checkmark]\}^{\perp} = [a!P]$, a proper biorthogonal, while $\{a?\}^{\perp\perp} = \perp^{\perp} = \mathbf{T}$. □

Proposition 37 (Broadcast barbs). *For the calculus of Example 15, the barbs are, for any $a \in A$, the processes which can output on a, $\{a!\}^{\perp\perp}$.*

Proof. Again, $\{a!\}^{\perp\perp} = \{[a?\checkmark]\}^{\perp} = [a!P]$ while $\{a?\}^{\perp\perp} = \mathbf{T}$. □

7 Extension to Full Process Calculi

We have presented a general notion of barb which behaves well in our idealised calculi; this does not, however, allow us to characterise barbs in full process calculi. In this section we remedy this by observing that since the nature of barbs in their various settings is closely tied to the basic pattern of interaction between processes, and the idealised calculi are sufficient to model these interactions, then barbs in full process calculi can be obtained via translation in to the idealised languages. We shall show that, given well-behaved translations, the biorthogonals of an extended framework derived from a full process calculus are preserved through translation.

Let us assume a process calculus $\mathcal{C}$ with an inert process ϵ and parallel composition $\parallel$. This calculus can be construed as a testing framework by augmenting it with $\checkmark$ in an identical way to the idealised calculi. We will use P and C to range over terms and contexts of this framework.

Definition 38 (Translations). An interaction-preserving translation into the idealised calculus $\mathcal{I}$ consists of a pair of maps

$$[\![\ ]\!] : \mathcal{C} \to \mathcal{I} \qquad\qquad [\![\ ]\!]^{-1} : \mathcal{I} \to \mathcal{C}$$

which preserve $\epsilon, \parallel$ and $\checkmark$ and moreover satisfy:

– $[\![\ ]\!]$ is surjective
– $[\![[\![P]\!]]\!]^{-1}$ is logically congruent to P.
– $[\![P]\!]@\pi \in\perp$ iff $P@[\![\pi]\!]^{-1} \in\perp$ and, dually, $[\![p]\!]^{-1}@C \in\perp$ iff $p@[\![C]\!] \in\perp$.

Lemma 39. *For any interaction-preserving translation, $[\![\ ]\!]^{-1}$ is surjective up to logical congruence.*

Proposition 40 (Translation correctness). *For any interaction-preserving translation and any set of terms/contexts A of $\mathcal{C}$, $[\![A^\perp]\!] = [\![A]\!]^\perp$.*

This tells us that interaction-preserving translations preserve (irreducible) biorthogonals, thus barbs are preserved. To find the barbs of a full process calculus then, it suffices to provide an interaction preserving translation and identify the barbs in the idealised language. We give an example of such a translation for the π-calculus below.

Example 41. We shall translate the π-calculus in to the synchronous idealised calculus. We define the mapping $[\![\ ]\!]^{-1}$ as a simple embedding which preserves, $\epsilon, \checkmark$ and $\|$ and

$$[\![a?p]\!]^{-1} = a(n).\epsilon \qquad [\![a!p]\!]^{-1} = \bar{a}n.\epsilon$$

where n is a reserved fixed name and we include special cases of the translation to preserve prefixed ticks. For the forward mapping, we also preserve $\epsilon, \checkmark$ and $\|$:

$$\begin{array}{ll} [\![\bar{a}nP]\!] = a!\epsilon & [\![!P]\!] = [\![P]\!] \\ [\![a(n)P]\!] = a?\epsilon & [\![\nu nP]\!] = \prod_{m_i \neq n?,n!} m_i\epsilon \ \text{ if } \ [\![P]\!] = \prod_I m_i\epsilon \end{array}$$

and also allow for special cases to preserve prefixed ticks. Note that, as we are interested solely in initial reductions, the role of replication, dynamic scoping and dynamically received names do not impact upon the translation. We leave it to the reader to check that this does indeed form an interaction preserving translation.

8 Conclusions and Future Work

We have introduced a formal definition of the well-known notion of barb, a basic observable of a process calculus. The definition relies only on the presence of a suitable underlying reduction semantics and relies on *biorthogonality*, a simple framework with deep roots in logic and computer science, and *irreducibility*, a concept from algebraic geometry.

We have shown that our definition yields the commonly accepted observables in idealised calculi for synchronous, asynchronous and broadcast communication. The latter fact was made possible by a characterisation of barbs in a particular class of *simple* calculi whose algebra of biorthogonals satisfies additional axioms. We have also shown how to use our idealised calculi to compute the barbs for standard calculi.

Finally, we have identified some synchronisation mechanisms which do not fit within the framework of simple calculi, but which still fit into the general framework. Our future research will concern understanding barbs in such calculi.

References

1. M. Abadi. ⊤⊤-closed relations and admissibility. *Mathematical Structures in Computer Science*, 10:313–320, 2000.
2. R. M. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous pi-calculus. *Theoretical Computer Science*, 195(2):291–324, 1998.

3. G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.
4. L. Cardelli. Brane calculi: Interactions of biological membranes. In *Computational Methods in Systems Biology CMSB '04*, volume 3082 of *LNCS*, pages 257–280, 2005.
5. L. Cardelli and A. D. Gordon. Mobile ambients. In *Foundations of Software Science and Computation Structures, FoSSaCS '98*. Springer Verlag, 1998.
6. V. Danos and J.-L. Krivine. Disjunctive tautologies as synchronisation schemes. In *Computer Science Logic CSL '00*, volume 1862 of *LNCS*, pages 292–301, 2000.
7. C. Fournet and G. Gonthier. The reflexive chemical abstract machine and the join-calculus. In *Proceedings of Symposium on Principles of Programming Languages, POPL'96*, pages 372–385. ACM Press, 1996.
8. J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–101, 1987.
9. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.
10. K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 151(2):437–486, 1995.
11. J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. In *International Conference on Concurrency Theory Concur '00*, volume 1877 of *LNCS*, pages 243–258. Springer, 2000.
12. P.-A. Melliès and J. Vouillon. Recursive polymorphic types and parametricity in an operational framework. In *Logic in Computer Science LICS '05*, pages 82–91. IEEE, 2005.
13. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
14. R. Milner and D. Sangiorgi. Barbed bisimulation. In *9th Colloquium on Automata, Languages and Programming, ICALP '92*, volume 623 of *LNCS*, pages 685–695. Springer, 1992.
15. U. Montanari and V. Sassone. Dynamic congruence vs. progressing bisimulation for CCS. *Fundamenta Informaticae*, XVI:171–199, 1992.
16. A. M. Pitts. Parametric polymorphism and operational equivalence. *Mathematical Structures in Computer Science*, 10:321–359, 2000.
17. G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981. Reprinted in Journal of Logic and Algebraic Programming 60–61 (2004) 17-139.
18. M. Reid. *Undergraduate Algebraic Geometry*, volume 12 of *London Mathematical Society Student Texts*. Cambridge University Press, 1988.
19. J. Vitek and G. Castagna. A calculus of secure mobile computations. In *IEEE Workshop on Internet Programming Languages*, 1998.
20. R. Wille. Restructuring lattice theory: an approach based on hierarchies of concepts. In I. Rival, editor, *Ordered Sets*, pages 445–470. Reidel, Dordrecht-Boston, 1982.

On the Stability by Union of Reducibility Candidates

Colin Riba

INPL & LORIA[⋆], Nancy, France
riba@loria.fr

Abstract. We investigate some aspects of proof methods for the termination of (extensions of) the second-order λ-calculus in presence of union and existential types.

We prove that Girard's reducibility candidates are stable by union iff they are exactly the non-empty sets of terminating terms which are downward-closed w.r.t. a weak observational preorder.

We show that this is the case for the Curry-style second-order λ-calculus. As a corollary, we obtain that reducibility candidates are exactly the Tait's saturated sets that are stable by reduction. We then extend the proof to a system with product, co-product and positive iso-recursive types.

1 Introduction

Since their introduction in [17], union and existential types with type assignment rules are present in many type systems. From a foundational perspective, they are interesting as dual of respectively intersection and second-order types. The paper [3] provides detailed investigations on syntactic as well as semantics issues of union types. As a theoretical tool, they have been used in [8, 9] to prove that a kind of Böhm trees, called Lévy-Longo trees, distinguishes pure λ-terms exactly as does their observation in the lazy concurrent λ-calculus.

Interesting applications of union types are the XML processing languages XDuce [14] and ℂDuce [11]. They describe types of XML documents by means of regular expressions whose internal representation relies on union types.

Existentials and unions are also interesting tools for representing abstractions of programs. In the context of strictness analysis, unions are used in [15] to represent disjunctive properties of programs.

Frédéric Blanqui and the author proposed in [6] a termination criterion for higher-order conditional rewriting that use constrained types. Existential constraints arise naturally, for example when proving that some implementations of QuickSort preserves the size of its argument. This work relies on proof methods for the termination of typed λ-calculus plus rewriting in presence of existential types.

[⋆] UMR 7503 CNRS-INPL-INRIA-Nancy2-UHP, Campus Scientifique, BP 239, 54506 Vandoeuvre-lès-Nancy Cedex, France.

Usual proofs of strong normalization (*i.e.* termination) for typed λ-calculi assign to each type T a set of strongly normalizing terms $[\![T]\!]\rho$ (the *interpretation* of T, with an assignment ρ of its free variables). Then, they use *soundness* of this interpretation w.r.t. the type system: typable terms belong to the interpretation of their types. But soundness requires that types are not interpreted by arbitrary sets of terms. They must satisfy some *closure* conditions. The two most popular ones are Girard's reducibility candidates and Tait's saturated sets. See [12] for a detailed discussion and historical references. A comparison of Girard's and Tait's closure conditions can also be found in [23].

In order to handle elimination rules of union and existential types, it is convenient to interpret these types by using unions of interpretations: $[\![T_1 \cup T_2]\!]\rho = [\![T_1]\!]\rho \cup [\![T_2]\!]\rho$ and $[\![\exists X.T]\!]\rho = \bigcup_{C \in \mathcal{C}l} [\![T]\!]\rho[C/X]$ ($\mathcal{C}l$ is the collection of closed sets under consideration). This making requires *stability by union* of closed sets: if $\mathcal{C}$ is a family of closed sets, then $\bigcup \mathcal{C} =_{\mathrm{def}} \bigcup_{C \in \mathcal{C}} C$ is closed. Approaches not relying on stability by union are briefly discussed in Sec. 8.

It is well known (see e.g. [23]) that Tait's saturated sets for β-reduction are stable by union. However, their extension with additional computational rules can by cumbersome. On the other hand, Girard's reducibility candidates are more adjustable, in particular for dealing with rewriting [5]. However, their stability by union is known to be problematic [23].

In this paper, we prove the stability by union of reducibility candidates for the polymorphic λ-calculus $\boldsymbol{\lambda 2}$. To our knowledge, this property was hitherto unknown, and even often believed to be false. We extend the proof to $\boldsymbol{\lambda 2 U^+}$, the extension of $\boldsymbol{\lambda 2}$ with product, co-product and positive iso-recursive types (it is shown in [21] that it is a proper extension of $\boldsymbol{\lambda 2}$).

The key observation is that reducibility candidates are stable by union iff they are exactly the non-empty sets of strongly-normalizing terms which are downward-closed w.r.t. a weak observational preorder (see [16] for a presentation and references on related topics). This is a very simple structure compared to the one appearing in the definition of reducibility candidates, which is not trivial and somehow mysterious. Hence, studying stability by union of reducibility candidates reveals important facts on their fundamental nature.

We show that the above condition is equivalent to say that some terms, called neutral, have a maximal reduct w.r.t. to that preorder. For the case of $\boldsymbol{\lambda 2}$ and $\boldsymbol{\lambda 2 U^+}$, we prove it by using a simple syntactic property called weak-standardization. It relies on the *orthogonality* of $\boldsymbol{\lambda 2}$ and $\boldsymbol{\lambda 2 U^+}$: computational rules are non-ambiguous and left-linear (no equality tests between open terms). As a by-product, we obtain that Girard's sets correspond exactly to the Tait's sets that are stable by reduction.

We present the syntax of $\boldsymbol{\lambda 2}$ in Sec. 2, and reducibility in Sec. 3. Our analysis of stability by union of reducibility candidates is presented in Sec. 4. Finally, the system $\boldsymbol{\lambda 2 U^+}$ is presented in Sec. 6, and stability by union of its reducibility candidates in Sec. 7.

We assume familiarity with λ-calculus and types, and refer to [13, 4] for detailed introductions.

2 The Polymorphic $\boldsymbol{\lambda}$-Calculus $\boldsymbol{\lambda 2}$

The core language of the paper is the Curry-style second-order λ-calculus $\boldsymbol{\lambda 2}$, as presented in [4]. We recall the main notations below, and then discuss union and existential types.

Let $\mathcal{X}$ be a countable set of variables and Λ be the set of untyped λ-terms:

$$t, u \in \Lambda \quad ::= \quad x \in \mathcal{X} \quad | \quad \lambda x.t \quad | \quad t\,u \ .$$

They are considered equals modulo α-conversion. Terms come with the usual notion of β-reduction, namely $(\lambda x.t)u \mapsto_\beta t[u/x]$, where $t[u/x]$ is the capture-avoiding substitution of x by u in t. We let $\rightarrow$ be the smallest rewrite relation on Λ containing $\mapsto_\beta$. In the following, we refer to $\mapsto_\beta$ as the *top reduction* and denote by $\mathcal{TNF}$ the set of terms in $\mapsto_\beta$-normal form. We let $(t)_{\rightarrow} =_{\text{def}} \{u \mid t \rightarrow u\}$ and $(t)^*_{\rightarrow} =_{\text{def}} \{u \mid t \rightarrow^* u\}$, where $\rightarrow^*$ is the reflexive-transitive closure of $\rightarrow$. We write $(t_1, \dots, t_n) \rightarrow (t'_1, \dots, t'_n)$ iff there is i such that $t_i \rightarrow t'_i$ and $t_j = t'_j$ for all $j \neq i$. A term t is *strongly normalizing* iff every reduction sequence issued from t is finite. Let $\mathcal{SN}$ be the set of strongly-normalizing terms. Note that $t \in \mathcal{SN}$ iff either t is not reducible or all its reducts are in $\mathcal{SN}$. It follows that $\mathcal{SN}$ is the *smallest* set such that for all t,

$$(\forall u \ (t \rightarrow u \ \Rightarrow \ u \in \mathcal{SN})) \ \Rightarrow \ t \in \mathcal{SN} \ .$$

Types are the formulas of second-order minimal logic, with variables in $\mathcal{V}$:

$$T, U \in \mathcal{T} \quad ::= \quad X \in \mathcal{V} \quad | \quad T \Rightarrow U \quad | \quad \forall X.T \ .$$

We denote by $\mathcal{X}(t)$ (resp. $\mathcal{V}(T)$) the set of free variables of t (resp. T). An *environment* Γ is a finite set of declarations $x : A$ such that $x \neq y$ whenever $(x : A), (y : B) \in \Gamma$. Typing judgments are sequents of the form $\Gamma \vdash t : A$, derived with the following rules:

$$(\text{Ax}) \ \frac{}{\Gamma, x : T \vdash x : T}$$

$$(\Rightarrow \text{I}) \ \frac{\Gamma, x : U \vdash t : T}{\Gamma \vdash \lambda x.t : U \Rightarrow T} \qquad (\Rightarrow \text{E}) \ \frac{\Gamma \vdash t : U \Rightarrow T \qquad \Gamma \vdash u : U}{\Gamma \vdash tu : T}$$

$$(\forall \text{I}) \ \frac{\Gamma \vdash t : T}{\Gamma \vdash t : \forall X.T} \ (X \notin \mathcal{V}(\Gamma)) \qquad (\forall \text{E}) \ \frac{\Gamma \vdash t : \forall X.T}{\Gamma \vdash t : T[U/X]}$$

Existential and Union Types. Our main point is to prove the soundness (w.r.t. some closure operator) of elimination rules of union and implicit existential types. Such types are manipulated with type-assignment rules, in the spirit of [17, 3]. Typical rules for these systems are:

$$(\exists \text{I}) \ \frac{\Gamma \vdash t : T[U/X]}{\Gamma \vdash t : \exists X.T} \qquad (\exists \text{E}) \ \frac{\Gamma \vdash t : \exists X.T \quad \Gamma, x : T \vdash u : U}{\Gamma \vdash u[t/x] : U} \ (X \notin \mathcal{V}(\Gamma, U))$$

$$(\cup\text{I}) \frac{\Gamma \vdash t : T_i}{\Gamma \vdash t : T_1 \cup T_2} (i \in \{1, 2\}) \qquad (\cup\text{E}) \ \frac{\Gamma \vdash t : T_1 \cup T_2 \quad \forall i \in \{1, 2\}, \ \Gamma, x : T_i \vdash u : U}{\Gamma \vdash u[t/x] : U}$$

It is worth noting that such types are not subject to the Curry-Howard *propositions-as-types* isomorphism, in the sense that proofs trees do not corresponds to terms. It would require to reflect all types constructions at the term level, and this leads to use explicit constructs for disjunction and existential quantification, as discussed for e.g. in [13].

This is precisely what we want to avoid in [6], where existential quantifications are used in a constrained type system. We want terms typed in this system to be used in a constrained-free type system. Hence, constraints should not appear at the term level, and we are thus interested in implicit existential types.

On the other-hand, it is clear that the impredicative codings of unions and existential quantification given in [13] do not define union and implicit existential types. This motivates us to handle the soundness of rules ($\cup$E) and ($\exists$ E) by tools lying inside the reducibility framework. A first step is to study the behavior of Girard's reducibility candidates w.r.t. union. Since type quantifications of [6] are not the usual ones, presented above, that operate on the type structure, we study stability by union in a generic way, without committing to specific typing rules.

3 Reducibility

Because of its second order-type quantification, strong normalization proofs of $\boldsymbol{\lambda 2}$ must use a third-order predicate. This is achieved by interpreting types $T \in \mathcal{T}$ as sets of strongly normalizing terms $[\![T]\!]\rho \subseteq \mathcal{SN}$. Then strong normalization of typable terms follows from the soundness of the interpretation:

$$\textit{If } \Gamma \vdash t : T \textit{ and } \sigma(x) \in [\![A]\!]\rho \textit{ for all } (x : A) \in \Gamma\textit{, then } \sigma(t) \in [\![T]\!]\rho. \qquad (\star)$$

As said in the introduction, ($\star$) does not hold when types are interpreted by arbitrary subsets of $\mathcal{SN}$. In this section, we focus on a well-known collection of suitable subsets of $\mathcal{SN}$ called Girard's reducibility candidates.

Definition 3.1 (Neutral Terms). *Terms not headed by an abstraction are called* neutral. *Let $\mathcal{N}$ be the set of neutral terms.*

Let $\mathcal{HN}$, the set of hereditary neutral terms, *be the smallest set such that for all $t \in \mathcal{N}$, $(\forall u\ (t \to u \Rightarrow u \in \mathcal{HN})) \Rightarrow t \in \mathcal{HN}$.*

Note that $\mathcal{HN} \subseteq \mathcal{SN}$. Let *elimination contexts* be $E[\,] ::= [\,] \mid E[\,]t$. They correspond to call-by-name evaluation contexts, dual to call-by-value evaluation contexts of [10]. We borrowed from [1] their use in reducibility.

Remark 3.2. The general intuition behind neutral terms is linked to the duality between introductions and eliminations in natural deduction. Since neutral terms are not headed by introductions, they do not interact with elimination contexts: if $t \in \mathcal{N}$ and $E[t] \to v$ then $v = E'[t']$ with $(E[\,], t) \to (E'[\,], t')$. It follows that for $t \in \mathcal{N}$, we have $E[t] \in \mathcal{N}$ and $E[t] \in \mathcal{SN}$ as soon as $\{E[u] \mid t \to u\} \subseteq \mathcal{SN}$.

Definition 3.3 (Reducibility Candidates). *The set of* reducibility candidates, *denoted by $\mathcal{CR}$, is the set of all $C \subseteq \mathcal{SN}$ such that*

($\mathcal{CR}0$) *if* $t \in C$ *and* $t \to u$ *then* $u \in C$,
($\mathcal{CR}1$) *if* $t \in \mathcal{N}$ *and* $(\forall u\ (t \to u \ \Rightarrow\ u \in C))$ *then* $t \in C$.

The definition of $\mathcal{SN}$ directly implies that $\mathcal{SN} \in \mathcal{CR}$. Now, let us see why $\mathcal{CR}$ is stable by intersection ($\mathcal{C} \subseteq \mathcal{CR} \Rightarrow \bigcap \mathcal{C} \in \mathcal{CR}$). Recall that a *closure operator* on a partial order $(D, \leq)$ is a function $\overline{\cdot} : D \to D$ which is idempotent: $\overline{\overline{X}} = \overline{X}$; extensive: $X \leq \overline{X}$; and monotone: $X \leq Y \Rightarrow \overline{X} \leq \overline{Y}$. It is well-known that the greatest lower bound of a family of closed elements is closed.

The *shape* of the clauses of $\mathcal{CR}$ is sufficient for the existence of a closure operator $\overline{\cdot} : \mathcal{P}(\mathcal{SN}) \to \mathcal{P}(\mathcal{SN})$ such that $\overline{X}$ is the least reducibility candidate containing X. Indeed, clauses ($\mathcal{CR}0$) and ($\mathcal{CR}1$) are *closure rules* in the sense of [19], p.17, and the existence of $\overline{\cdot}$ is insured by Thm. 2.6, see pages 16–18 of [19]. We thus have $X \in \mathcal{CR}$ iff $X = \overline{X}$.

It follows that $(\mathcal{CR}, \subseteq)$ has all greatest lower bounds, and they are given by $\bigcap$. As a consequence, it has $\bigcap \mathcal{CR}$ as least element. It is thus an inf-semi lattice with greatest element $\mathcal{SN}$, hence a complete lattice.

Proposition 3.4. *$\mathcal{HN}$ is the least element of $\mathcal{CR}$.*

Proof. We obviously have $\mathcal{HN} \subseteq C$ for all $C \in \mathcal{CR}$, hence $\mathcal{HN} \subseteq \bigcap \mathcal{CR}$. For the converse, it suffices to remark that $\mathcal{HN} \in \mathcal{CR}$. $\square$

We now define the interpretation of arrow types.

Proposition 3.5 (Arrow Type Constructor). *The* arrow type constructor $\Rightarrow : \mathcal{P}(\Lambda) \times \mathcal{P}(\Lambda) \to \mathcal{P}(\Lambda)$, *defined as* $\mathcal{A} \Rightarrow \mathcal{B} =_{def} \{t \mid \forall u (u \in \mathcal{A} \ \Rightarrow\ tu \in \mathcal{B})\}$, *maps reducibility candidates $\mathcal{A}$ and $\mathcal{B}$ to a reducibility candidate.*

Proof. Strong normalization and stability by reduction follows directly from that of $\mathcal{B}$. For ($\mathcal{CR}1$), we have to show that if $t \in \mathcal{N}$, then $(t)_{\to} \subseteq \mathcal{A} \Rightarrow \mathcal{B}$ implies $t \in \mathcal{A} \Rightarrow \mathcal{B}$, *i.e.* $u \in \mathcal{A} \ \Rightarrow\ tu \in \mathcal{B}$. The crucial point is given by Rem. 3.2: since $tu \in \mathcal{N}$ and t does not interact with the context $[\ ]u$, we are done with ($\mathcal{CR}1$) applied to $\mathcal{B}$, using an induction on $u \in \mathcal{SN}$. $\square$

Now, given $\rho : \mathcal{V} \to \mathcal{CR}$, we can interpret types $T \in \mathcal{T}$ as reducibility candidates $[\![T]\!]\rho \in \mathcal{CR}$, with $[\![X]\!]\rho =_{\mathrm{def}} \rho(X)$, $[\![U \Rightarrow T]\!]\rho =_{\mathrm{def}} [\![U]\!]\rho \Rightarrow [\![T]\!]\rho$ and $[\![\forall X.T]\!]\rho =_{\mathrm{def}} \bigcap\{[\![T]\!]\rho[C/X] \mid C \in \mathcal{CR}\}$. We can then prove soundness ($\star$), and strong normalization is a consequence of $\mathcal{X} \subseteq \mathcal{HN}$ and Prop. 3.4. See [13, 12].

4 A General Study of Stability by Union of $\mathcal{CR}$

We have seen that stability by intersection of $\mathcal{CR}$ is a consequence of the *shape* of the clauses ($\mathcal{CR}0$) and ($\mathcal{CR}1$). Such shallow observations do not imply stability by union, which must therefore be proved through a deeper analysis of $\mathcal{CR}$. Indeed, given $\mathcal{C} \subseteq \mathcal{CR}$, in order to get $\bigcup \mathcal{C} \in \mathcal{CR}$ we must show that if t is a neutral term with $(t)_{\to} \subseteq \bigcup \mathcal{C}$, then every one-step reduct of t must be in the *same* $C \in \mathcal{C}$.

We begin by stability by union of closure operators. In the next Proposition, we assume given a set D and a closure operator $\overline{\cdot} : \mathcal{P}(D) \to \mathcal{P}(D)$. Write $\overline{x}$ for $\overline{\{x\}}$ and $\overline{\mathcal{P}(D)}$ for $\{\overline{X} \mid X \subseteq D\}$.

Proposition 4.1 (Topological Closure of $\overline{\cdot}$). *Given a closure operator $\overline{\cdot} : \mathcal{P}(D) \to \mathcal{P}(D)$, let $\overline{\Omega}$ be the set of non-empty $X \subseteq D$ such that $X = \bigcup\{\overline{x} \mid x \in X\}$. Then $\overline{\Omega}$ is the smallest set such that $\overline{\mathcal{P}(D)} \subseteq \overline{\Omega}$ and $\mathcal{C} \subseteq \overline{\Omega} \Rightarrow \bigcup \mathcal{C}, \bigcap \mathcal{C} \in \overline{\Omega}$.*

Proof. For all $X \in \overline{\mathcal{P}(D)}$ we have $X = \bigcup\{\overline{x} \mid x \in X\}$ since for all $x \in X$, $\overline{x} \subseteq X$ and $x \in \overline{x}$. It follows that $\overline{\mathcal{P}(D)} \subseteq \overline{\Omega}$. If $\mathcal{C} \subseteq \overline{\Omega}$, then $C = \bigcup\{\overline{x} \mid x \in C\}$ for all $C \in \mathcal{C}$. Hence $\bigcup \mathcal{C} = \bigcup\{\overline{x} \mid x \in \bigcup \mathcal{C}\}$, and $\bigcup \mathcal{C} \in \overline{\Omega}$. Moreover, if $t \in \bigcap \mathcal{C}$, then for all $C \in \mathcal{C}$, $t \in C$ hence $\overline{t} \subseteq C$. Therefore $\overline{t} \subseteq \bigcap \mathcal{C}$, and it follows that $\bigcap \mathcal{C} = \bigcup\{\overline{t} \mid t \in \bigcap \mathcal{C}\}$, *i.e.* $\bigcap \mathcal{C} \in \overline{\Omega}$.

Now, let $\Omega \supseteq \overline{\mathcal{P}(D)}$ be stable by union and intersection. If $X \in \overline{\Omega}$ then $X = \bigcup\{\overline{x} \mid x \in X\}$. But $\{\overline{x} \mid x \in X\} \subseteq \Omega$, hence $\bigcup\{\overline{x} \mid x \in X\} \in \Omega$. □

In other words, $(D, \overline{\Omega} \cup \{\emptyset\})$ is a topological space where the set of *opens* $\overline{\Omega} \cup \{\emptyset\}$ contains all $\overline{\cdot}$-closed sets. Proposition 4.1 implies that it is moreover the *coarser* topology with this property: if Ω is a collection of opens that contains $\overline{\cdot}$-closed sets, then $\overline{\Omega} \subseteq \Omega$. We now use these facts to study the stability by union of reducibility candidates.

Since we are concerned with properties independent from the calculus, we work with an extension $\boldsymbol{\lambda}\mathbf{ext}$ of $\boldsymbol{\lambda}\mathbf{2}$. We assume that every term typable in $\boldsymbol{\lambda}\mathbf{2}$ is typable in $\boldsymbol{\lambda}\mathbf{ext}$ and that $\boldsymbol{\lambda}\mathbf{ext}$ is equipped with a rewrite relation, denoted by $\to$, that is finitely branching and contains $\mapsto_\beta$. Notations of Sec. 2 are imported in $\boldsymbol{\lambda}\mathbf{ext}$. Finally, we assume given a set of neutral terms of $\boldsymbol{\lambda}\mathbf{ext}$, still denoted by $\mathcal{N}$, that contains the neutral terms of $\boldsymbol{\lambda}\mathbf{2}$.

Definition 4.2 ($\boldsymbol{\lambda}$ext-Reducibility Candidates). *The set of* $\boldsymbol{\lambda}\mathbf{ext}$-reducibility candidates*, denoted by $\mathcal{CR}$, is the set of all $C \subseteq \mathcal{SN}$ such that*
($\mathcal{CR}0$) *if $t \in C$ and $t \to u$ then $u \in C$,*
($\mathcal{CR}1$) *if $t \in \mathcal{N}$ and $(\forall u\ (t \to u \ \Rightarrow\ u \in C))$ then $t \in C$.*

As with $\boldsymbol{\lambda}\mathbf{2}$, $\mathcal{CR}$ is the set of closed sets for a closure operator $\overline{\cdot} : \mathcal{P}(\mathcal{SN}) \to \mathcal{P}(\mathcal{SN})$. An explicit inductive definition of $\overline{\cdot}$ is useful:

Lemma 4.3 (The Closure Operator $\overline{\cdot}$ of $\mathcal{CR}$). *Given $X \subseteq \mathcal{SN}$, let $\overline{X}_0 =_{def} (X)^*_{\to}$, and for all $i \geq 0$, $\overline{X}_{i+1} =_{def} \overline{X}_i \cup \{t \in \mathcal{N} \mid (t)_{\to} \subseteq \overline{X}_i\}$. Then, for all $X \subseteq \mathcal{SN}$, $\overline{X} =_{def} \bigcup_{i \geq 0} \overline{X}_i$ is the smallest reducibility candidate containing X.*

Proof. First, we show that $\overline{X} \in \mathcal{CR}$.
($\mathcal{CR}0$) Let $t \to v$ and $t \in \overline{X}_i$ with i as small as possible. If $i = 0$, then $t \in (X)^*_{\to}$, hence $v \in (X)^*_{\to} = \overline{X}_0$. Otherwise, $i = j + 1$, and $v \in \overline{X}_j$.
($\mathcal{CR}1$) Let $t \in \mathcal{N}$ such that for all $v \in (t)_{\to}$, there is i_v with $v \in \overline{X}_{i_v}$. Since $\to$ is finitely branching, there is a j greater than every i_v, thus $t \in \overline{X}_{j+1}$.

Second, by induction on i, we prove that if $C \in \mathcal{CR}$ and $X \subseteq C$, then $\overline{X}_i \subseteq C$. We have $\overline{X}_0 = (X)^*_{\to} \subseteq C$ by ($\mathcal{CR}0$). For $i \geq 0$, if $t \in \overline{X}_{i+1} \setminus \overline{X}_i$, then $t \in \mathcal{N}$ and by induction hypothesis, $(t)_{\to} \subseteq \overline{X}_i \subseteq C$. Hence $t \in C$ by ($\mathcal{CR}1$). □

Thanks to Lem. 4.3, we have an explicit definition of the closure operator of $\mathcal{CR}$, denoted by $\overline{\cdot}$. Now, Prop. 4.1 gives us $\overline{\Omega}$, which is the *topological closure* of $\overline{\cdot}$.

We let $\overline{\mathcal{CR}} =_{\mathrm{def}} \overline{\Omega}$. Proposition. 4.1 implies that $\mathcal{CR} \subseteq \overline{\mathcal{CR}}$ and that $\mathcal{CR}$ is stable by union iff $\mathcal{CR} = \overline{\mathcal{CR}}$.

Before studying what insures the equality of $\mathcal{CR}$ and $\overline{\mathcal{CR}}$, let us have a look on the structure of the latter. We begin by a characterization of the membership $t \in \overline{u}$ that uses the *weak observational preorder* $\sqsubseteq$.

Definition 4.4. *Let $t \sqsubseteq u$ iff for all $v \notin \mathcal{N}$, $t \to^* v \;\Rightarrow\; u \to^* v$. We denote by $\sqsubseteq_{\mathcal{SN}}$ the restriction of $\sqsubseteq$ to $\mathcal{SN}$.*

Observational preorders where introduced to characterize behavioral equivalence: two pieces of programs are observationally equivalent iff when plugged in a program context, the obtained programs both diverges or evaluates to the same value. Contexts are arbitrary terms with a hole, including those of the form $t[\,]$ and $\lambda x.[\,]$. With closed terms, thanks to Milner's Context Lemma, this is equivalent to observation in applicative contexts (our elimination contexts). Of course, this fails for open terms. See [18, 16] for a presentation and references on the subject.

In $\boldsymbol{\lambda 2}$, non-neutral terms are abstractions, hence closed non-neutral terms correspond to the usual notion of value. Moreover, we have $t \sqsubseteq u$ iff for all $E[\,]$, for all $v \notin \mathcal{N}$, $(E[t] \to^* v \Rightarrow E[u] \to^* v)$. Thus, with $\sqsubseteq$ we observe the reduction to values of open terms plugged in elimination contexts. Hence the name *weak* observational preorder. We generalize these ideas to the system $\boldsymbol{\lambda 2U^+}$ in Sec. 7.

In order to characterize $t \in \overline{u}$ with $\sqsubseteq$, we need a few properties. First, note that $t \to u$ implies $u \sqsubseteq t$.

Proposition 4.5. *Let $X \subseteq \mathcal{SN}$. Then $t \in \overline{X}$ iff either $t \in (X)^*_{\to}$ or ($t \in \mathcal{N}$ and $(t)_{\to} \subseteq \overline{X}$).*

Proof. The "if" direction directly follows from ($\mathcal{CR}0$) and ($\mathcal{CR}1$). Conversely, let i be as small as possible such that $t \in \overline{X}_i$. Thus, either $i = 0$ and $t \in (X)^*_{\to}$ or $i = j+1$, $t \in \overline{X}_{j+1} \setminus \overline{X}_j$ and by definition $t \in \mathcal{N}$ and $(t)_{\to} \subseteq \overline{X}_j$. □

As a consequence, all non-neutral terms of $\overline{X}$ are in $(X)^*_{\to}$. In other words, the values of $\overline{X}$ are entirely determined by X.

Proposition 4.6. *Let $t \in \mathcal{N} \cap \mathcal{SN}$ and $X \subseteq \mathcal{SN}$. Then $(t)_{\to} \subseteq \overline{X}$ iff for all $v \notin \mathcal{N}$, $t \to^* v \;\Rightarrow\; v \in (X)^*_{\to}$.*

Proof. For the "only-if" direction, we reason by induction on $t \in \mathcal{SN}$. Assume that $(t)_{\to} \subseteq \overline{X}$ and let $t \to^* v' \notin \mathcal{N}$. Since $t \in \mathcal{N}$, we have $t \to v \to^* v'$ with $v \in \overline{X}$. If $v \notin \mathcal{N}$, then by Prop. 4.5 we have $v \in (X)^*_{\to}$, hence $v' \in (X)^*_{\to}$. Otherwise, since $(v)_{\to} \subseteq \overline{X}$ we can apply the induction hypothesis and get $v' \in (X)^*_{\to}$.

For the converse, we also reason by induction on $t \in \mathcal{SN}$. If $t \in (X)^*_{\to}$ then we are done since $(t)^*_{\to} \subseteq (X)^*_{\to} \subseteq \overline{X}$. Assume that $t \notin (X)^*_{\to}$. Let $v \in (t)_{\to}$. If $v \notin \mathcal{N}$, then by assumption $v \in (X)^*_{\to}$, hence $v \in \overline{X}$. Otherwise, using ($\mathcal{CR}1$) it suffices to show that $(v)_{\to} \subseteq \overline{X}$. But by assumption, every $v' \in (v)^*_{\to} \setminus \mathcal{N}$ belongs to $(X)^*_{\to}$. Hence by induction hypothesis $(v)_{\to} \subseteq \overline{X}$. □

Thanks to Proposition 4.6, each $\overline{t}$ is in fact an *initial segment* w.r.t. $\sqsubseteq_{\mathcal{SN}}$.

Lemma 4.7. *For all $t \in \mathcal{SN}$, $\overline{t} = \{u \mid u \sqsubseteq_{\mathcal{SN}} t\}$.*

Proof. By Prop. 4.6, $u \in \overline{t}$ iff $t, u \in \mathcal{SN}$ and either $t \to^* u$ or $u \in \mathcal{N}$ and $(u)^*_{\to} \setminus \mathcal{N} \subseteq (t)^*_{\to}$. But this is exactly $u \sqsubseteq_{\mathcal{SN}} t$. □

This gives a nice straight structure to the elements of $\overline{\mathcal{CR}}$.

Theorem 4.8. *Let $\mathcal{O}$ be the set of non empty subsets of $\mathcal{SN}$ which are downward closed w.r.t. $\sqsubseteq_{\mathcal{SN}}$,* i.e. *$X \in \mathcal{O}$ iff $\emptyset \neq X \subseteq \mathcal{SN}$ and for all t, u, if $t \in X$ and $u \sqsubseteq_{\mathcal{SN}} t$ then $u \in X$. Then we have $\overline{\mathcal{CR}} = \mathcal{O}$.*

Proof. Thanks to Lem. 4.7, we have $X \in \overline{\mathcal{CR}}$ iff it is a non-empty subset of $\mathcal{SN}$ such that $X = \bigcup\{\overline{t} \mid t \in X\} = \{u \mid \exists t(u \sqsubseteq_{\mathcal{SN}} t \ \wedge \ t \in X)\}$, *i.e.* $X \in \mathcal{O}$. □

It is surprising that we can give such a simple structure to $\overline{\mathcal{CR}}$. The most important, however, is the consequence on the stability by union of $\mathcal{CR}$.

Corollary 4.9. *$\mathcal{CR}$ is stable by union iff $\mathcal{CR} = \mathcal{O}$.*

The structure of reducibility candidates is at a first sight not trivial and somehow mysterious. It is therefore extremely interesting to understand what allows $\mathcal{CR} = \mathcal{O}$, giving them a so simple structure. Hence the question of their stability by union reveals important facts on their fundamental nature.

The next step is to characterize what implies $\mathcal{CR} = \mathcal{O}$.

Proposition 4.10. *1. $\mathcal{CR} \subseteq \mathcal{O}$.*
2. $\mathcal{O} \subseteq \mathcal{CR}$ iff for all $t \in \mathcal{N} \cap \mathcal{SN}$, there is $u \in (t)_{\to}$ such that $t \sqsubseteq_{\mathcal{SN}} u$.

Proof. 1. Let $C \in \mathcal{CR}$. By Prop. 3.4, $C \neq \emptyset$. If $t \in C$ and $u \sqsubseteq_{\mathcal{SN}} t$, then by Lem. 4.7 we have $u \in \overline{t} \subseteq C$. Hence $C \in \mathcal{O}$.
2. Let $C \in \mathcal{O}$. Since it is stable by reduction, it suffices to check that if $t \in \mathcal{N}$ and $(t)_{\to} \subseteq C$, then $t \in C$. But there is $u \in (t)_{\to}$ such that $t \sqsubseteq_{\mathcal{SN}} u$, and $u \in C$ implies $t \in C$. Conversely, let $C \in \mathcal{CR}$, $t \in C$ and $u \sqsubseteq_{\mathcal{SN}} t$. Then, by Lem. 4.7, $u \in \overline{t} \subseteq C$. □

The property of $\mathcal{SN}$ neutral terms expressed in Prop. 4.10.(2) is that a reduct u of a neutral term t such that $t \sqsubseteq_{\mathcal{SN}} u$ is in some sense a *principal reduct of* t: the values of t are exactly those of u. Moreover, $u \in \max_{\sqsubseteq_{\mathcal{SN}}} (t)_{\to}$, *i.e.* a principal reduct is maximal among all possible reducts.

For $\boldsymbol{\lambda 2}$, this has to be linked with call-by-name languages, in which terms to be evaluated are neutral, and the evaluation preserves possible values.

Definition 4.11. *A term $u \in (t)_{\to}$ is a* principal reduct *(written p.r.) of t iff $t \sqsubseteq u$ and t is said to have the* principal reduct property *(written p.r.p.) when either such a u exists or t is a normal form.*

We reduce the stability by union of $\mathcal{CR}$ to the principal reduct property for neutral terms.

Corollary 4.12. *We have $\mathcal{C} \subseteq \mathcal{CR} \Rightarrow \bigcup \mathcal{C} \in \mathcal{CR}$ iff every $t \in \mathcal{N} \cap \mathcal{SN}$ has the principal reduct property.*

As pointed out by Prop. 3.5, in order for $t \in \mathcal{N}$ to belong to $C \in \mathcal{CR}$, C uses information on the behavior of t in elimination contexts. This information is given by the values of t, and clause ($\mathcal{CR}1$) relies on the fact since $t \in \mathcal{N}$, its values are that of $(t)_{\rightarrow}$. But if $t \sqsubseteq_{\mathcal{SN}} u$, the values of t are also values of u. That is, if $u \in C$ and $t \sqsubseteq_{\mathcal{SN}} u$ then $t \in C$, and it follows that $t \sqsubseteq_{\mathcal{SN}} u$ implies $t \in \overline{u}$. Moreover, Lem. 4.7 says that this exactly characterizes $\overline{u}$: if it contains t, then the values of t are values of u.

Now, recall that given $\mathcal{C} \subseteq \mathcal{CR}$, in order to get $\bigcup \mathcal{C} \in \mathcal{CR}$ we must show that if $t \in \mathcal{N}$ has $(t)_{\rightarrow} \subseteq \bigcup \mathcal{C}$, then every $v \in (t)_{\rightarrow}$ must be in fact the *same* $C \in \mathcal{C}$. This amounts to say that $(t)_{\rightarrow}$ contains an u such that $t \sqsubseteq_{\mathcal{SN}} u$, hence that t has a principal reduct.

5 Stability by Union of $\mathcal{CR}$ in $\boldsymbol{\lambda}$2

In this section, we show the p.r.p. for $\boldsymbol{\lambda 2}$. This leads to observe that reducibility candidates are exactly the Tait's saturated sets which are stable by reduction. We do not give proofs since they are subsumed by those for $\boldsymbol{\lambda 2 \mathrm{U}^{+}}$, presented in Sec. 7.

The p.r.p. is a consequence of the standardization theorem of $\boldsymbol{\lambda 2}$. However, since we only need this property on $\mathcal{SN}$ terms, it can be proved using a weaker property, called weak standardization.

This property was coined in [2] as a consequence of standardization. However, it admits a much direct proof, which rely on the orthogonality of the calculus. By the way, it is less a *weak standardization* property than a standardization for the *weak head reduction*, which corresponds to reduction in elimination contexts.

Definition 5.1 (Weak Head Reduction). *The relation $\rightarrow_{\mathcal{H}}$ of* weak head reduction *is defined as $t \rightarrow_{\mathcal{H}} u$ iff $t = E[t']$, $t' \mapsto_{\beta} u'$ and $E[u'] = u$. We denote by $\mathcal{HNF}$ the set of terms in $\rightarrow_{\mathcal{H}}$-normal form.*

Lemma 5.2 (Weak Standardization). *Let $t \mapsto_{\beta} u$ and assume that $E[t] \rightarrow v$ with $v \neq E[u]$. Then $v = E'[t']$ with $(E[\ ], t) \rightarrow (E'[\ ], t')$ and there exists u' such that $t' \mapsto_{\beta} u'$ and $E[u] \rightarrow^{*} E'[u']$.*

In order to show that strongly normalizing neutral terms have the p.r.p., we use in addition the fact that $\mathcal{HNF} \cap \mathcal{N}$ (*i.e.* the set of terms of the from $E[x]$), is stable by reduction. We then obtain stability by union of $\mathcal{CR}$, thanks to Cor. 4.12.

Lemma 5.3 (Principal Reduct Property). *Let $t \in \mathcal{N} \cap \mathcal{SN}$. If there is u such that $t \rightarrow_{\mathcal{H}} u$ then it is a p.r. of t, otherwise every $u \in (t)_{\rightarrow}$ is a p.r. of t.*

Theorem 5.4. *If $\mathcal{C} \subseteq \mathcal{CR}$ then $\bigcup \mathcal{C} \in \mathcal{CR}$.*

The principal reduct property of neutral terms corresponds to the fact that reducibility candidates are stable by strongly normalizing weak head expansions.

Lemma 5.5 (Weak Head Expansion and Reducibility Candidates). *Let $C \in \mathcal{CR}$. If $E[s] \in C$, $t \mapsto_\beta s$ and $t \in \mathcal{SN}$ then $E[t] \in C$.*

Weak head reduction is the main notion of the Krivine Abstract Machine, and stability by weak head expansion is the main property required by truth values of Krivine and Danos [7].

Thanks to Cor. 4.9, we have shown that $\mathcal{CR} = \mathcal{O}$, thus giving a nice straight structure to $\mathcal{CR}$. Moreover, Lem. 5.3, 4.7 and 5.5 imply that for all $C \in \mathcal{CR}$,
($\mathcal{SAT}1$) if $E[x] \in \mathcal{SN}$ then $E[x] \in C$; and
($\mathcal{SAT}2$) if $E[s] \in C$, $t \mapsto_\beta s$ and $t \in \mathcal{SN}$, then $E[t] \in C$.

Subsets of $\mathcal{SN}$ satisfying ($\mathcal{SAT}1$) and ($\mathcal{SAT}2$) are Tait's *saturated sets.*

Definition 5.6. *Let $\mathcal{SAT}$ be the set of all subsets $S \subseteq \mathcal{SN}$ satisfying ($\mathcal{SAT}1$) and ($\mathcal{SAT}2$). We denote by $\mathcal{SAT}_\rightarrow$ the collection of $S \in \mathcal{SAT}$ that are stable by reduction: If $t \in S$ and $t \rightarrow u$ then $u \in S$ ($\mathcal{SAT}0$).*

The definition makes sense since in ($\mathcal{SAT}1$) we have $E[x] \in \mathcal{SN}$, and, in ($\mathcal{SAT}2$) we have $E[t] \in \mathcal{SN}$ by Lem. 5.5.

The stability by union of $\mathcal{CR}$ is therefore linked with the fact that every $C \in \mathcal{CR}$ is saturated. This inclusion has been coined in [12]. The converse is false, as shown by a counter-example given in [23] (Lem. 3.16 pp. 87–88). It relies on the fact that saturated sets are not stable by reduction. However, we can show that saturated sets that are stable by reduction are exactly the reducibility candidates. This seems to have not been remarked before.

Theorem 5.7. *$\mathcal{SAT}_\rightarrow = \mathcal{CR}$.*

6 The System $\boldsymbol{\lambda}$2U$^+$ of Product, Co-product and Positive Iso-recursive Types

We now extend results of the previous section by applying Cor. 4.12 to a system with more elaborated types. The system considered, called $\boldsymbol{\lambda}\mathbf{2U}^+$, features product, co-product, positive iso-recursive types and the final type 1. It is a proper extension of $\boldsymbol{\lambda}\mathbf{2}$ [21]. Our presentation is inspired by that of [1]. Types are extended with:

$$T, U \in \mathcal{T} \quad ::= \quad \dots \quad | \quad T \times U \quad | \quad T + U \quad | \quad \mu X.T \quad | \quad 1$$

where, in $\mu X.T$, X occurs only positively in T: any path from the root of T to an occurrence of X chooses the left argument of $\Rightarrow$ an even number of times. The syntax of terms is enriched with corresponding introductions and eliminations:

$$\begin{array}{rlll} s, t \in \Lambda ::= \dots & | & () & \\ | \ \langle t, u \rangle & | & \pi_i t & i \in \{1, 2\} \\ | \ \mathsf{inj}_i\, t & | & \mathsf{case}\,(u, x_1.t_1, x_2.t_2) & i \in \{1, 2\} \\ | \ \mathsf{in}\, t & | & \mathsf{out}\, t & \end{array}$$

We consider the following extension of β-reduction:

$$\pi_i\langle t_1, t_2\rangle \mapsto_\beta t_i \qquad \mathsf{case}\,(\mathsf{inj}_i\, u, x_1.t_1, x_2.t_2) \mapsto_\beta t_i[u/x_i] \qquad \mathsf{out}\,(\mathsf{in}\, t) \mapsto_\beta t\ .$$

We let $\to$ be the smallest rewrite relation on Λ containing $\mapsto_\beta$. The type system is enriched with the following additional rules:

$$(1\,\mathrm{I})\ \frac{}{\Gamma \vdash () : 1}$$

$$(\times\mathrm{I})\ \frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash \langle t_1, t_2\rangle : T_1 \times T_2} \qquad (\times\mathrm{E})\ \frac{\Gamma \vdash t : T_1 \times T_2}{\Gamma \vdash \pi_i t : T_i}\ (i \in \{1,2\})$$

$$(+\mathrm{I})\ \frac{\Gamma \vdash t : T_i}{\Gamma \vdash \mathsf{inj}_i\, t : T_1 + T_2}\ (i \in \{1,2\}) \qquad (+\mathrm{E})\ \frac{\Gamma \vdash t : T_1 + T_2 \quad \Gamma, x : T_i \vdash t_i : U}{\Gamma \vdash \mathsf{case}\,(t, x_1.t_1, x_2.t_2) : U}$$

$$(\mu\mathrm{I})\ \frac{\Gamma \vdash t : T[\mu X.T/X]}{\Gamma \vdash \mathsf{in}\, t : \mu X.T} \qquad (\mu\mathrm{E})\ \frac{\Gamma \vdash t : \mu X.T}{\Gamma \vdash \mathsf{out}\, t : T[\mu X.T/X]}$$

Example 6.1. Our motivation to consider $\boldsymbol{\lambda 2}\mathbf{U}^+$ is that it is a very atomic calculus for inductives types. For example, polymorphic lists can be encoded as follows:

$$\begin{array}{rcl} \mathsf{List} & =_{\mathrm{def}} & \forall Y.\mu X.1 + Y \times X \\ \mathsf{nil} & =_{\mathrm{def}} & \mathsf{in}\,(\mathsf{inj}_1\,()) \\ \mathsf{cons}\,(x, xs) & =_{\mathrm{def}} & \mathsf{in}\,(\mathsf{inj}_2\,\langle x, xs\rangle) \end{array}$$

In [21], it is shown that higher-order primitive recursion can be defined using iso-recursives types and the rule $\mathsf{out}\,(\mathsf{in}\, t) \mapsto_\beta t$, and moreover that this rule can not be simulated in $\boldsymbol{\lambda 2}$ by means of β-reductions.

7 Reducibility and Stability by Union for $\boldsymbol{\lambda 2}\mathbf{U}^+$

We now introduce tools for reducibility in $\boldsymbol{\lambda 2}\mathbf{U}^+$. We then prove the p.r.p. and equivalence of $\mathcal{CR}$ and a suitable version of $\mathcal{SAT}_\to$.

We define the *top reduction* as $\mapsto_\beta$, and still denote by $\mathcal{TNF}$ the set of terms in top-normal form. It is convenient to factorize the introduction-elimination duality of natural deduction with the following atomic contexts: *atomic eliminations contexts* (aec) denoted by $\epsilon[\ ]$ and *atomic introduction contexts* (aic) denoted by $\iota[\ ]$ are defined as follows:

$$\epsilon[\ ] ::= \ [\ ]t \ \mid \ \pi_1[\ ] \ \mid \ \pi_2[\ ] \ \mid \mathsf{case}\,([\ ], x_1.t_1, x_2.t_2) \mid \mathsf{out}\,[\ ]\ ;$$

$$\iota[\ ] ::= \lambda x.[\ ] \mid \langle [\ ], t\rangle \mid \langle t, [\ ]\rangle \mid \ \mathsf{inj}_1\,[\ ] \ \mid \ \mathsf{inj}_2\,[\ ] \ \mid \ \mathsf{in}\,[\ ] \ \mid ()\ .$$

Note that the introduction context for the terminal type constructor is not linear. The columns of the above array define a relation written $\epsilon[\ ] \perp \iota[\ ]$. We have $\epsilon[\ ] \perp \iota[\ ]$ when $\epsilon[\iota[\ ]]$ is a β-redex, except for the product, where we let

$$\begin{array}{lccc} & \pi_1[\ ] \perp \langle [\ ], t\rangle & \text{and} & \pi_2[\ ] \perp \langle t, [\ ]\rangle \\ \text{but} & \pi_2[\ ] \not\perp \langle [\ ], t\rangle & \text{and} & \pi_1[\ ] \not\perp \langle t, [\ ]\rangle\ . \end{array}$$

Then, to each aec $\epsilon[\ ]$ corresponds the set $\epsilon[\ ]^{\perp}$ of aic $\iota[\ ]$ such that $\epsilon[\ ] \perp \iota[\ ]$, and conversely, $\iota[\ ]^{\perp}$ is the set of aec $\epsilon[\ ]$ such that $\epsilon[\ ] \perp \iota[\ ]$. In the following, we will sometime write unambiguously $\iota[\ ] \perp \epsilon[\ ]$ instead of $\epsilon[\ ] \perp \iota[\ ]$. The extension of elimination contexts $E[\ ]$ is obvious: $E[\ ] ::= [\ ] \mid \epsilon[E[\ ]]$.

Proposition 7.1. *If $E[\epsilon[t]] \to v$, then $v = E'[t']$ with $(E[\ ], \epsilon[t]) \to (E'[\ ], t')$.*

Proof. By structural induction on $E[\ ]$. The case $E[\ ] = [\ ]$ is obvious. In the other cases, we have $E[\ ] = \epsilon'[F[\ ]]$. First, the reduction $\epsilon'[F[\epsilon[t]]] \to v$ can not be a top reduction. Hence we have $v = \epsilon''[v']$ with $(\epsilon'[\ ], v) \to (\epsilon''[\ ], v')$, and by induction hypothesis $v' = F'[t']$ with $(F[\ ], t) \to (F'[\ ], t')$. Take $E'[\ ] = \epsilon''[F'[\ ]]$. □

Definition 7.2. *We say that terms not of the form $\iota[t]$ are neutral and denote by $\mathcal{N}$ the set of neutral terms.*

Hence, the terms nil and $\mathsf{cons}(x, l)$ of Ex. 6.1 are not neutral. This strengthen our intuition that non-neutral terms corresponds to values. Note that elimination contexts are a generalization of applicative contexts to product, coproduct and iso-recursive types. We can now substantiate our claim that $\sqsubseteq$ is a weak observational preorder.

Proposition 7.3. *$t \sqsubseteq u$ iff $\forall E[\ ]\ (E[t] \to^* \iota[h]\ \Rightarrow\ E[u] \to^* \iota[h])$.*

Proof. Obviously, $\forall E[\ ]\ (E[t] \sqsubseteq E[u])$ implies $t \sqsubseteq u$ (take the empty context). Conversely, assume that $t \sqsubseteq u$. If $E[t] \to^* \iota[h]$, we are in the case that $E[t] \to^* E'[\iota'[h']] \to^* \iota[h]$ with $(E[\ ], t) \to^* (E'[\ ], \iota'[t'])$. Since $t \sqsubseteq u$, we have $u \to^* \iota'[h']$, hence $E[u] \to^* E'[\iota'[h']] \to^* \iota[h]$. □

We obtain $\mathcal{CR}$ by instantiating Def. 4.2 with $\boldsymbol{\lambda 2U^+}$ and $\mathcal{N}$. Notions of weak head reduction and weak head normal forms are directly adapted from Sec. 5, with the obvious update of elimination contexts. Recall that $u \in \mathcal{HNF}$ iff for all $E[\ ]$, t such that $u = E[t]$, we have $t \in \mathcal{TNF}$. It follows that untyped terms in $\mathcal{HNF} \cap \mathcal{N}$ need not to be of the form $E[x]$. However, stability by reduction of $\mathcal{HNF} \cap \mathcal{N}$ still holds.

Proposition 7.4. *If $t \in \mathcal{HNF} \cap \mathcal{N}$ and $t \to u$, then $u \in \mathcal{HNF} \cap \mathcal{N}$.*

Proof. We reason by structural induction on t. The case $t \in \mathcal{X}$ is trivial. Assume $t = \epsilon[t_1]$ and let $t \to t'$. Since $t \in \mathcal{HNF}$, we have $t' = \epsilon'[t_1']$ with $(t_1, \epsilon[\ ]) \to (t_1', \epsilon'[\ ])$, hence $t' \in \mathcal{N}$. If $t_1 \in \mathcal{N}$, then $t_1 \in \mathcal{N} \cap \mathcal{HNF}$ and by induction hypothesis $t_1' \in \mathcal{N} \cap \mathcal{HNF}$. It follows that $t' = \epsilon'[t_1'] \in \mathcal{HNF}$. Otherwise, $t_1 = \iota[t_2]$ with $\epsilon[\ ] \not\perp \iota[\ ]$. Hence, $t_1' = \iota'[t_2']$ with $(t_2, \iota[\ ]) \to (t_2', \iota'[\ ])$, and $\epsilon'[\ ] \not\perp \iota'[\ ]$. Hence $t' = \epsilon'[\iota'[t_2']] \in \mathcal{HNF}$. □

We now turn to weak standardization. It is stated and used for $\boldsymbol{\lambda 2U^+}$ in [1].

Proposition 7.5. *If $t \mapsto_\beta u$ and $t \to v$, then either $v = u$ or there exists u' such that $v \mapsto_\beta u' \leftarrow^* u$.*

Proof. By cases on $t \mapsto_\beta u$.

$t = (\lambda x.t_1)t_2$. In this case, $u = t_1[t_2/x]$ and if $v \neq u$, then $v = (\lambda x.t'_1)t'_2$ with $(t_1, t_2) \to (t'_1, t'_2)$, and $v \mapsto_\beta t'_1[t'_2/x] \leftarrow^* u$.

$t = \pi_i\langle t_1, t_2\rangle$. In this case, $u = t_i$, and if $v \neq u$, then $v = \pi_i\langle t'_1, t'_2\rangle$ with $(t_1, t_2) \to (t'_1, t'_2)$, and $v \mapsto_\beta t'_i \leftarrow^* u$.

$t = \mathsf{case}\,(\mathsf{inj}_i\, r, x_1.t_1, x_2.t_2)$. In this case, $u = t_i[r/x_i]$ and if $v \neq u$, then $v = \mathsf{case}\,(\mathsf{inj}_i\, r', x_1.t'_1, x_2.t'_2)$ with $(r, t_1, t_2) \to (r', t'_1, t'_2)$ and $v \mapsto_\beta t'_i[r'/x_i] \leftarrow^* u$.

$t = \mathsf{out}\,(\mathsf{in}\, r)$. In this case, $u = r$ and if $v \neq u$, then $v = \mathsf{out}\,(\mathsf{in}\, r')$ with $r \to r'$ and $v \mapsto_\beta r' \leftarrow u$. □

Lemma 7.6 (Weak Standardization). *If $t \mapsto_\beta u$ and $E[t] \to v$, then either $v = E[u]$ or $v = E'[t']$ for some $E'[\,]$, t' such that $(E[\,], t) \to (E'[\,], t')$ and there exists u' such that $t' \mapsto_\beta u'$ and $E[u] \to^* E'[u']$.*

Proof. Let $E[t] \to v$ with $v \neq E[u]$. Since $t \mapsto_\beta u$, t is an elimination and by Prop. 7.1, $v = E'[t']$ where $(E[\,], t) \to (E'[\,], t')$. The case $E[\,] \to E'[\,]$ with $t = t'$ is trivial. Otherwise, we have $t \to t'$ with $E'[\,] = E[\,]$ and we conclude by Prop. 7.5. □

It follows that strongly normalizing neutral terms have the p.r.p..

Lemma 7.7 (Principal Reduct Property). *Let $t \in \mathcal{N} \cap \mathcal{SN}$. If there is u such that $t \to_{\mathcal{H}} u$, then it is a p.r. of t, otherwise every $u \in (t)_\to$ is a p.r. of t.*

Proof. Let $t \in \mathcal{N} \cap \mathcal{SN}$. If $t \in \mathcal{HNF}$, since $\mathcal{HNF} \cap \mathcal{N}$ is stable by reduction by Prop. 7.4, t never reduces to a non-neutral term. It follows that $t \sqsubseteq_{\mathcal{SN}} u$ for all $u \in (t)_\to$. Otherwise, $t \to_{\mathcal{H}} u$ and by induction on $t \in \mathcal{SN}$, we show that u is the p.r. of t. If $t \to^* v \notin \mathcal{N}$, since $t \in \mathcal{N}$, there is t' such that $t \to t' \to^* v$. By Lem. 7.6, if $t' \neq u$, there is u' such that $t' \to_{\mathcal{H}} u' \leftarrow^* u$. Therefore, $t' \in \mathcal{N} \cap \mathcal{SN}$ and by induction hypothesis $t' \sqsubseteq_{\mathcal{SN}} u'$, hence $u \to^* u' \to^* v$. □

Using Cor. 4.12, we have thus proved:

Theorem 7.8. *If $\mathcal{C} \subseteq \mathcal{CR}$ then $\bigcup \mathcal{C} \in \mathcal{CR}$.*

As in Sec. 5, we get stability of reducibility candidates by weak head expansion.

Lemma 7.9 (Weak Head Expansion and Reducibility Candidates). *Let $C \in \mathcal{CR}$. If $E[s] \in C$, $t \mapsto_\beta s$ and $t \in \mathcal{SN}$, then $E[t] \in C$.*

Proof. We obtain easily that $E[t] \in \mathcal{SN}$, using Lem. 7.6 and an induction on $(E[\,], t) \in \mathcal{SN} \times \mathcal{SN}$. Now, by Lem. 4.7, $E[t] \in \overline{E[s]}$ iff $E[t] \sqsubseteq_{\mathcal{SN}} E[s]$. But this follows from Lem. 7.7, and we get $E[t] \in \overline{E[s]} \subseteq C$. □

Unlike reducibility candidates, saturated sets are modified. They use elimination contexts in an essential way. The new clauses for $\mathcal{SAT}$ are the following:

$(\mathcal{SAT}1)$ $\mathcal{HNF} \cap \mathcal{SN} \cap \mathcal{N} \subseteq S$.

$(\mathcal{SAT}2)$ If $s \in S$, $t \to_{\mathcal{H}} s$ and $t \in \mathcal{SN}$ then $t \in S$.

As in Def. 5.6, we denote by $\mathcal{SAT}_{\rightarrow}$ the collection of saturated sets that are stable by reduction (axiom $(\mathcal{SAT}0)$).

Non-emptiness of $\mathcal{SAT}$ follows from Lem. 7.9. The clause $(\mathcal{SAT}1)$ is unusual. Indeed, we use it for correspondence with $\mathcal{CR}$, but for strong normalization, it is sufficient to require $E[x] \in \mathcal{SN} \;\Rightarrow\; E[x] \in S$ $(\mathcal{SAT}1')$. Moreover, well-typed terms $\mathcal{HNF} \cap \mathcal{N}$ are of the form $E[x]$, hence $(\mathcal{SAT}1')$ would have been sufficient with typed candidates (see [12]).

Theorem 7.10. *$\mathcal{SAT}_{\rightarrow} = \mathcal{CR}$.*

Proof. We begin by showing that $\mathcal{CR} \subseteq \mathcal{SAT}_{\rightarrow}$. Let $C \in \mathcal{CR}$. $(\mathcal{SAT}0)$ follows from $(\mathcal{CR}0)$. For $(\mathcal{SAT}1)$, we can reason by induction on $\rightarrow$, since by Prop. 7.4, $\mathcal{HNF} \cap \mathcal{N} \cap \mathcal{SN}$ is stable by reduction. Finally, the satisfaction $(\mathcal{SAT}2)$ directly follows from Lem. 7.9.

Conversely, we show that $\mathcal{SAT}_{\rightarrow} \subseteq \mathcal{CR}$. Let $S \in \mathcal{SAT}_{\rightarrow}$. As above, $(\mathcal{CR}0)$ follows from $(\mathcal{SAT}0)$. For $(\mathcal{CR}1)$, we have to show that for every neutral term v such that $(v)_{\rightarrow} \subseteq S$, then $v \in S$. First, $v \in \mathcal{SN}$ since $(v)_{\rightarrow} \subseteq S \subseteq \mathcal{SN}$. Thus, we conclude by $(\mathcal{SAT}1)$ if $v \in \mathcal{HNF}$. Otherwise, $v = E[t]$ with $t \mapsto_{\beta} s$ and $E[s] \in S$. Thus $v \in S$ by $(\mathcal{SAT}2)$. $\square$

8 Conclusion and Related Works

Related Works. There are interesting alternatives approaches, which may not rely on stability by union. Using bi-orthogonality (see [7]), Melliès and Vouillon present a semantic of types that is not stable by union, and instead relies on a closer adequation between the interpretations and the typing rules [22].

Without unions and existentials, bi-orthogonals work well for strong normalization, especially for classical logic [20]. With non-ambiguous left-linear calculus, it is not unreasonable to expect that they can deal with existentials and union types, but the proof would rely on non-trivial properties of the calculus.

Concluding Remarks. We have shown that, with some well-behaved calculi, Girard's reducibility candidates are stable by union. This was commonly believed to be false. Moreover, and maybe more important, we have shown that their definition hide a very simple structure, namely that candidates are exactly the non empty subsets of $\mathcal{SN}$ that are downward-closed w.r.t. the weak observational preorder $\sqsubseteq_{\mathcal{SN}}$.

This shed new light on the semantics of strong normalization. In particular, we hope that this can lead to precise comparisons of bi-orthogonal and candidates. A related question is to know when the soundness of elimination rules of union and existentials types can be proved without stability by union of some type interpretation.

Acknowledgments. The author thanks Frédéric Blanqui and Claude Kirchner for useful advices, support and comments, and Marco Gaboardi for a plenty of interesting discussions. Antoine Reilles suggested interesting improvements in the presentation, and the anonymous referees made interesting insightful comments.

References

[1] A. Abel. Termination Checking with Types. *RAIRO – Theoretical Informatics and Applications*, 38(4):277–319, 2004. Special Issue (FICS'03).

[2] T. Altenkirch. *Constructions, Inductive Types and Strong Normalization.* PhD thesis, University of Edinburgh, 1993.

[3] F. Barbanera, M. Dezani-Ciancaglini, and U. de'Liguoro. Intersection and Union Types: Syntax and Semantics. *Information and Computation*, 119:202–230, 1995.

[4] H. Barendregt. Lambda Calculi with Types. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2. Oxford University Press, 1992.

[5] F. Blanqui, J.-P. Jouannaud, and M. Okada. Inductive-Data-Types Systems. *Theoretical Computer Science*, 271, 2002.

[6] F. Blanqui and C. Riba. Combining Typing and Size Constraints for Checking the Termination of Higher-Order Conditional Rewrite Systems. In *LPAR'06*, LNCS, 2006.

[7] V. Danos and J.-L. Krivine. Disjunctive Tautologies as Synchronisation Schemes. In *CSL'00*, volume 1862 of *LNCS*, pages 292–301, 2000.

[8] M. Dezani-Ciancaglini, U. de' Liguoro, and P. Piperno. A Filter Model for Concurrent Lambda-Calculus. *Siam Journal on Computing*, 27(5):1376–1419, 1998.

[9] M. Dezani-Ciancaglini, J. Tiuryn, and P. Urzyczyn. Discrimination by Parallel Observers. In *LICS'97*, 1997.

[10] M. Felleisen and R. Hieb. The Revised Report on the Syntactic Theories of Sequential Control and State. *Theoretical Computer Science*, 103(2):235–271, 1992.

[11] A. Frisch, G. Castagna, and V. Benzaken. Semantic Subtyping. In *LICS'02*, 2002.

[12] J.H. Gallier. On Girard's "Candidats de Reducibilité". In P. Odifredi, editor, *Logic and Computer Science*. Academic Press, 1989.

[13] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types.* Cambridge University Press, 1989.

[14] H. Hosoya, J. Vouillon, and B. Pierce. Regular Expression Types for XML. In *ICFP'00*, 2000.

[15] T. Jensen. Disjunctive Strictness Analysis. In *LICS'92*, 1992.

[16] T. Jim and A. R. Meyer. Full Abstraction and the Context Lemma. *Siam Journal on Computing*, 25(3):663–696, 1996.

[17] D. MacQueen, G. Plotkin, and R. Sethi. An Ideal Model for Recursive Polymorphic Types. *Information and Control*, 71(1-2):95–130, 1986.

[18] R. Milner. Fully Abstract Models of Typed λ-Calculi. *Theoretical Computer Science*, 4:1–22, 1977.

[19] J.-B. Nation. Notes on Lattice Theory. Available at `http://www.math.hawaii.edu/~jb/books.html`.

[20] M. Parigot. Proofs of Strong Normalization for Second Order Classical Natural Deduction. *Journal of Symbolic Logic*, 62(4):1461–1479, 1997.

[21] Z. Spławski and P. Urzyczyn. Type Fixpoints: Iteration vs. Recursion. In *ICFP'99*, pages 102–113. ACM, 1999.

[22] J. Vouillon and P.-A. Melliès. Semantic Types: A Fresh Look at the Ideal Model for Types. In *POPL'04*. ACM, 2004.

[23] B. Werner. *Une Théorie des Contructions Inductives.* PhD thesis, Université Paris 7, 1994.

An Effective Algorithm for the Membership Problem for Extended Regular Expressions

Grigore Roşu

Department of Computer Science,
University of Illinois at Urbana-Champaign, USA
grosu@cs.uiuc.edu

Abstract. By adding the complement operator ($\neg$), extended regular expressions (*ERE*) can encode regular languages non-elementarily more succinctly than regular expressions. The *ERE* membership problem asks whether a word w of size n belongs to the language of an *ERE* R of size m. Unfortunately, the best known membership algorithms are either non-elementary in m or otherwise require space $\Omega(n^2)$ and time $\Omega(n^3)$; since in many practical applications n can be very large, these space and time requirements could be prohibitive. In this paper we present an *ERE* membership algorithm that runs in space $O(n \cdot (\log n + m) \cdot 2^m)$ and time $O(n^2 \cdot (\log n + m) \cdot 2^m)$. The presented algorithm outperforms the best known algorithms when n is exponentially larger than m.

1 Introduction

Regular expressions can compactly specify patterns in strings. Extended regular expressions (*ERE*s), which add complementation ($\neg R$) to the usual union ($R_1 + R_2$), concatenation ($R_1 \cdot R_2$), and repetition ($R^\star$) operators, make the description of regular languages more convenient and more succinct. The membership problem for an *ERE* R and a word w is to decide whether w is in the regular language generated by R.

Due to their simplicity and popularity, regular expressions, and implicitly the membership problem, have many applications. There are programming and/or scripting languages, such as Perl, which are mostly based on efficient implementations of pattern matching via regular expressions. Many languages either have builtin efficient regular expression membership algorithms or provide libraries for them. Testing is another application area; events produced by the execution of physical processes or computer programs can be logged and then searched for property violations. Also, [5] suggests applications in molecular biology. Since many properties are more naturally expressed as what should *not* happen or as *intersection* of several policies, *ERE*s are particularly desirable. Moreover, since the input words can be quite large (e.g., a chromosome can have hundreds of millions of nucleobases, or a log file can have billions of events), *ERE* membership algorithms that are efficient in the length of the word are highly preferred.

The simplest-minded solution would be generate a DFA or an NFA from R, and then to check the membership of w in linear time with n by simply traversing w letter-by-letter once. Unfortunately, this may not always be practical. This is because the size of the NFA or DFA can be non-elementarily larger than R [10]. Even if one succeeded to store such an immense automaton, running it would still be non-elementary on each letter in the input, because one needs non-elementarily long labels for each state. There could admittedly be practical situations in which one can quickly generate a DFA or an NFA from R; if this is the case, then one should definitely use this simple algorithm. From a practical perspective, the work in this paper can be seen as an alternative to the simple-minded algorithm, when generating a standard automaton from R is not plausible.

There are many other *ERE* membership algorithms in precisely the same category. The first such algorithm was introduced in [3] in 1979; it runs in space $O(n^2 \cdot m)$ and time $O(n^3 \cdot m)$. A technique for speeding up membership algorithms by a factor of $\log n$ is presented in [7]. Several *ERE* membership algorithms have been published since 1979, such as [2,12,13,14,6,4], improving slightly[1] the complexity of the now classic algorithm in [3]. More precisely, they reduced the space requirements to $O(n^2 \cdot k + n \cdot m)$ and the time to $O(n^3 \cdot k + n^2 \cdot m)$ or worse, where k is the number of complement operators in R. An *ERE* membership algorithm was presented in [9], which "rewrites" or "derives" the *ERE* by each letter in the input word; the lower-bound result in [10] tells that this algorithm is also worst-case non-elementary in the *ERE*, but, unlike in the simplistic NFA/DFA generation algorithm, the worst-case penalty is not paid upfront. At our knowledge, there are no *ERE* algorithms that are asymptotically better than the non-elementary-in-m one based on generation of NFA/DFA or than the dynamic-programming 27-year-old algorithm in [3].

In this paper we present an *ERE* membership algorithm that is not polynomial, but which avoids the non-elementary explosion in the size of the *ERE*. More precisely, it runs in space $O(n \cdot (\log n + m) \cdot 2^m)$ and in time $O(n^2 \cdot (\log n + m) \cdot 2^m)$. When n is exponentially larger than m, in which case the "polynomial" algorithms would be exponential in the *ERE* anyway, our algorithm asymptotically outperforms all the known algorithms.

The basic idea of our algorithm is to repeatedly cut the EREs at complement operators to obtain a data-structure of nested NFAs. Formally, this is performed by introducing novel notions of *contextual regular expressions* and *automata*. To achieve the effect of complementation at each cut point, special novel data-structures, called *jumping machines* and implemented using priority queues, are introduced; these encode information needed to "jump" to the next subword which is *not* in the corresponding language. The advantage of jumping machines is that one does not need to store (via indexes) all the subwords which are not in the language, but only the next one; so we drop a factor of n in storage. The price to pay is that we need to store additional information to be able to jump to the next subword.

[1] Some of these algorithms originally claimed better improvements, but they turned out to be misanalysed – see [8] for a detailed discussion of this issue.

2 Preliminaries, Notations and Assumptions

Numbers and memory. To simplify the analysis and explanation of our algorithms, we adopt standard conventions about numbers and memory: numbers take constant time to store to and retrieve from memory, either independently or as elements of arrays or matrices. The companion report [8] gives an analysis of the algorithms presented in this paper considering that number storage/retrieval and operations on them (comparison, additions, etc.) and on memory require logarithmic time etc.; such pessimistic assumptions increase the time complexity of our algorithm by a logarithmic factor [8] (in the size of the input word and *ERE*). However, we still assume that logarithmic space is required to store a "large" number when it appears in more complex data-structures. For example, we build tables in our algorithm whose cells store pairs consisting of an index between 1 and n (n is the length of the input word) and a subset of states in an automaton having $O(m)$ states (m is the size of the input *ERE*); in this case, we will assume that it takes $O(\log n + m)$ space to store each cell in the table – this is what actually gives the "$\log n + m$" factor in the analysis of our algorithm. If one thinks that we are over-conservative here since n and m are small enough in practice that the number $n \cdot 2^m$ fits in a constant number of memory units (say, e.g., in two 64-bit words), then one can consider that our algorithm takes $O(n \cdot 2^m)$ space and $O(n^2 \cdot 2^m)$ time.

Languages. In this paper, Σ is a finite set called *alphabet* whose elements are called *letters*, and X is a set of *variables*. The elements of $\Sigma^\star$, i.e., finite sequences of letters in Σ, are called *Σ-words* or simply *words*. We let ϵ denote the empty word. If $w \in \Sigma^\star$ then we let $|w|$ denote the *length* of w and w_i the ith letter of w. If w has n letters then we can also write w as $w_1 w_2 \cdots w_n$. If $1 \leq i \leq j \leq n$ then $w_i w_{i+1} \cdots w_j$ is the *subword* of w between i and j. If $i > j$ then $w_i w_{i+1} \cdots w_j$ is ϵ by convention. A language over Σ is a subset of $\Sigma^\star$. We let $\mathcal{L}_\Sigma$ denote the set of languages over Σ, i.e., the powerset $\mathcal{P}(\Sigma^\star)$. Let $\emptyset$ denote the empty language. If $L_1, L_2 \in \mathcal{L}_\Sigma$ then $L_1 \cdot L_2$ is the language $\{\alpha_1\alpha_2 \mid \alpha_1 \in L_1 \text{ and } \alpha_2 \in L_2\}$. If $L \in \mathcal{L}_\Sigma$ then $L^\star$ is $\{\alpha_1\alpha_2\cdots\alpha_n \mid n \geq 0 \text{ and } \alpha_1, \alpha_2, \ldots, \alpha_n \in L\}$ and $\neg L$ is $\Sigma^\star - L$.

Extended regular expressions. (*ERE*s) define languages by inductively applying *union* ($+$), *concatenation* ($\cdot$), *Kleene Closure* ($\star$), *intersection* ($\cap$), and *complementation* ($\neg$). The language of an *ERE* R, denoted by $L(R)$, is defined inductively as follows, where a is any letter in Σ: $L(\emptyset) = \emptyset$, $L(\epsilon) = \{\epsilon\}$, $L(a) = \{a\}$, $L(R_1 + R_2) = L(R_1) \cup L(R_2)$, $L(R_1 \cdot R_2) = L(R_1) \cdot L(R_2)$, $L(R^\star) = (L(R))^\star$, $L(R_1 \cap R_2) = L(R_1) \cap L(R_2)$, $L(\neg R) = \neg L(R)$. One can define a procedure to check $\epsilon \in L(R)$ by just traversing R once. If R does not contain $\neg$ and $\cap$ then it is a *regular expression* (*RE*). By applying De Morgan's law $R_1 \cap R_2 \equiv \neg(\neg R_1 + \neg R_2)$, *ERE*s can be linearly (in both time and size) translated into equivalent *ERE*s without intersection. Hence, in the sequel we consider expressions without intersection. If Σ is not understood from context, then we let ERE_Σ denote the set of *ERE*s over letters in Σ and let RE_Σ denote

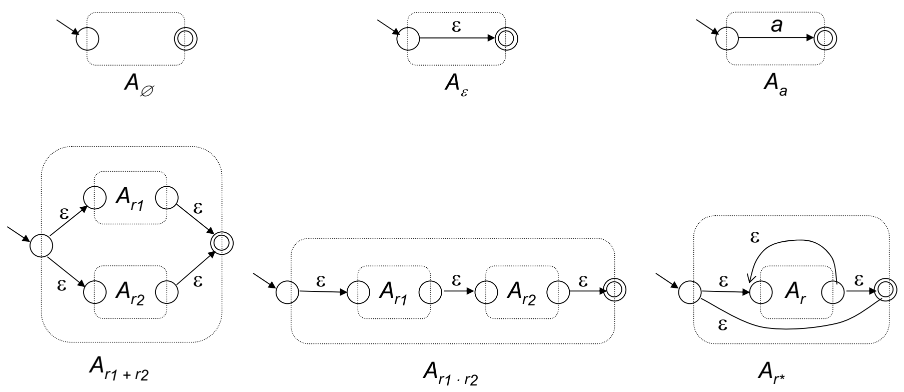


Fig. 1. Thompson's translation

the set of *RE*s over Σ. We use R, R_1, R_2, R', etc., for *ERE*s, and r, r_1, r_2, r', etc., for *RE*s.

The *size* of an *ERE* is the total number of occurrences of letters and composition operators ($+$, $\cdot$, $\star$, and $\neg$) that it contains. We will frequently need to check if $\epsilon \in L(R)$ for various subexpressions R of the original *ERE*; we can calculate all these ϵ-memberships in linear time in the original *ERE*, by a simple DFS traversal, updating the ϵ-membership of each subexpression from the corresponding ϵ-memberships of its subexpressions.

For any map $\varphi : X \to ERE_\Sigma$, we let $\varphi : ERE_{\Sigma \cup X} \to ERE_\Sigma$ also denote its unique extension to a *morphism*, that is, the map with $\varphi(\emptyset) = \emptyset$, $\varphi(\epsilon) = \epsilon$, $\varphi(a) = a$ for any $a \in \Sigma$, $\varphi(R_1+R_2) = \varphi(R_1)+\varphi(R_2)$, $\varphi(R_1 \cdot R_2) = \varphi(R_1)\cdot\varphi(R_2)$, $\varphi(R^\star) = (\varphi(R))^\star$, and $\varphi(\neg R) = \neg\varphi(R)$; also, we let $\varphi_\neg : X \to ERE_\Sigma$ denote the map defined by $\varphi_\neg(x) = \neg\varphi(x)$.

Automata. *Non-deterministic finite automata (NFA) with ϵ-transitions* are used in this paper, i.e., tuples $(S, \Sigma, \delta, s_0, F)$, where S is a finite set of *states*, Σ is an alphabet, $\delta : S \times (\Sigma \cup \{\epsilon\}) \to 2^S$ is a *transition function*, s_0 is an *initial state*, and F is a set of *final states*. We let NFA_Σ denote the set of such automata. It is well-known that one can associate an *NFA* A_r to any regular expression r. Moreover, the number of nodes and edges of A_r is linear with the size of r. A common *RE*-to-*NFA* translation, due to Thompson [11], is shown in Figure 1. The resulting *NFA* is linear with the size of the original *RE*. An important observation for this paper is that a letter x occurs exactly once in r iff x occurs on exactly one edge in A_r.

We assume a linear-time linear-space procedure GEN-NFA taking *RE*s to *NFA*s, using Thompson's construction. There are two important *NFA* operations that can be performed in linear space/time, namely *ϵ-closure* and the *global step*. Given $Q \subseteq S$ and a letter a, the *ϵ-closure* of Q is the set $\overline{\delta}(Q, \epsilon)$ of states that can be reached starting with a state in Q and applying only ϵ-transitions, and the *global step* $\delta(Q, a)$ is the set of states $\cup_{s \in Q}\delta(s, a)$. We can encode sets of states in an *NFA* of m states as vectors of m bits: 1 means that the corresponding state is in the set. The implementation of these operations is simple. The first, e.g., maintains a queue T of states, originally equal to Q, that still need to be processed; then it picks and removes a state from T and considers each of its ϵ-transitions. If a new state is found, add it to both T and the result set (the latter is initially empty). Repeat until T is empty. Also, *intersection*, *union* and *emptiness* test on sets of states represented as vectors of size m take $O(m)$.

Priority queues. [1] are structures useful to maintain sets, supporting insertion and extraction of elements, as well as access to a "highest priority" element. They are routinely implemented in linear space using heaps flattened in vectors, which can be initialized in linear time; we assume an $O(|\mathcal{E}|)$ procedure $\textsc{Initialize}(\mathcal{Q}, \mathcal{E})$ that initializes queue $\mathcal{Q}$ to hold the elements $\mathcal{E}$ (also called "heapify"). The appealing aspect of priority queues is that insertion and extraction take log time, while accessing the highest priority element takes constant time. These numbers, however, assume that elements take constant space/time to store, access and compare. We will accordingly tune these numbers (conservatively) when the elements in $\mathcal{E}$ require more than constant space to be stored.

3 Contextual Regular Expressions and Automata

Definition 1. *A* ***contextual regular expression over letters*** Σ ***and variables*** X *is a regular expression in* $RE_{\Sigma\cup X}$ *containing exactly one occurrence of each variable in* X*. We let* $RE_\Sigma[X]$ *denote the set of contextual regular expressions over* Σ *and* X*.*

The restriction to one variable does *not* apply to the language of a contextual *RE*. Indeed, if $r \in RE_\Sigma[X]$ then $\alpha \in L(r)$ can have zero, one or more occurrences of any $x \in X$. The motivation for contextual *RE*s comes from the fact that any *ERE* can be decomposed in a "root" contextual regular expression, together with an *ERE* with fewer complement operators associated to each variable. This well-founded decomposition of *ERE*s is a crucial step in our membership algorithm.

Proposition 1. *For any* $R \in ERE_\Sigma$*, there is a set of variables* X*, an* $r \in RE_\Sigma[X]$*, and a map* $\varphi : X \to ERE_\Sigma$*, such that* $R = \varphi_\neg(r)$*. Moreover, for any* $x \in X$*, the ERE* $\varphi(x)$ *contains strictly fewer complement operators than* R*. We call* r *the* ***root*** *of* R*.*

In what follows we assume a procedure $\textsc{Decompose}$ that takes *ERE*s R to triples (X, r, φ) as above. If one uses pointers to refer to regular (sub)expressions, then one can decompose an *ERE* R into (X, r, φ) in $O(m_r)$ space and time, where m_r is the size of r.

Definition 2. *Automata in* $NFA_{\Sigma\cup X}$ *containing for each* $x \in X$ *exactly one edge labeled with* x *are called* ***contextual automata over letters*** Σ ***and variables*** X*. Let* $NFA_\Sigma[X]$ *denote the set of such automata. To emphasize their* *contextual nature, we write such automata as tuples* $(S, \Sigma, X, \delta, s_0, F)$ *rather then* $(S, \Sigma \cup X, \delta, s_0, F)$*. In any contextual automaton, let* $in_x, out_x \in S$ *denote, respectively, the source and the target states of the edge labeled* x*, for each* $x \in X$*.*

Note that Thompson's construction takes contextual *RE*s in $RE_\Sigma[X]$ to contextual *NFA*s in $NFA_\Sigma[X]$. One can associate any $R \in ERE_\Sigma$ a contextual automaton by first decomposing it into some (X, r, φ) and then taking Gen-NFA(r). Continuing this automata generation process for each $\varphi(x)$, one eventually gets a structure of "nested" *NFA*s, one for each complement operator in the original *ERE*. To ease the task of calculating ϵ-closures in such automata, we prefer to shortcut a nested *NFA* by an ϵ-transition whenever it contains ϵ in its language:

Definition 3. *Given* $R \in ERE_\Sigma$ *decomposing to* (X, r, φ)*, the* ***root*** *NFA* ***of*** R *is the NFA returned by* Gen-NFA(r) *in which a new edge* $\delta(in_x, \epsilon) = out_x$ *is added for each* $x \in X$ *with* $\epsilon \in L(\neg\varphi(x))$*.*

With this, note that $\epsilon \in L(R)$ iff $\overline{\delta}(\{s_0\}, \epsilon) \cap F \neq \emptyset$. Let us next give an automata-based characterization for the membership of any w to $L(R)$.

Definition 4. *Let* $w = w_1 w_2 \cdots w_n \in \Sigma^\star$*, let* $R \in ERE_\Sigma$ *decompose to* (X, r, φ)*, and let* $(S, \Sigma, X, \delta, s_0, F)$ *be the root NFA of* R*. Then we define* Z_0, Z_1, Z_2, ..., Z_n *as the smallest sets of states closed under the following:*

- $s_0 \in Z_0$;
- $\overline{\delta}(Z_i, \epsilon) \subseteq Z_i$ *for each* $i \in \{0, 1, ..., n\}$;
- $\delta(Z_i, w_{i+1}) \subseteq Z_{i+1}$ *for each* $i \in \{0, 1, ..., n-1\}$;
- *if* $in_x \in Z_i$ *for some* $i \in \{0, 1, ..., n\}$ *and* $x \in X$ *then* $out_x \in Z_j$ *for all* $j \in \{i+1, ..., n\}$ *with* $w_{i+1} \cdots w_j \in L(\neg\varphi(x))$*.*

Note that the "smallest sets" in the definition above makes sense, because sequences of sets closed under the operations above are also closed under component-wise intersection.

Proposition 2. *With the notation above,* $w \in L(R)$ *iff* $Z_n \cap F \neq \emptyset$*.*

The proposition above immediately implies that $w \notin L(R)$ iff $Z_n \cap F = \emptyset$. Since the definition of Z_0, Z_1, ..., Z_n is based on memberships of the subwords $w_{i+1} \cdots w_j$ to the languages $L(\varphi(x))$, which can be iteratively reduced to generating the root *NFA* of $\varphi(x)$ and then checking for emptiness the intersection of its final states with some corresponding Z set obtained like Z_n, one can now derive a membership algorithm based on root automata. In what follows we present an algorithm which, considering the information "$w_{i+1} \cdots w_j \in L(\neg\varphi(x))$" encoded in some convenient way, calculates all the sets Z_0, Z_1, ..., Z_n and then checks for membership.

Definition 5. *Given* $w = w_1 w_2 \cdots w_n \in \Sigma^\star$ *and* $L \subseteq \Sigma^\star$*, a map* $t : \{0, 1, ..., n-1\} \times \{1, 2, ..., n\} \to \{0, 1\}$ ***is a table for*** w ***and*** L *if and only if for any* $0 \leq i < j \leq n$*,* $t[i][j] = 1$ *iff* $w_{i+1} \cdots w_j \in L$*.*

```
MEMB-WITH-TABLES$(w, R)$
Input: $w = w_1 w_2 \cdots w_n \in \Sigma^\star,\ R \in ERE$
Output: true / false
Globals: $Z_0, Z_1, ..., Z_n$
 1. %GEN-TABLE-STRUCTURES
 2. $Z_0 \leftarrow \{s_0\}$
 3. for $i \leftarrow 1, 2, ..., n$ do $Z_i \leftarrow \emptyset$ endfor
 4. for $i \leftarrow 0, 1, ..., n$ do
 5.   %STEP-WITH-TABLES
 6. endfor
 7. return $Z_n \cap F \neq \emptyset$

GEN-TABLE$(w, R)$
Input: $w = w_1 w_2 \cdots w_n \in \Sigma^\star,\ R \in ERE$
Output: table $t$
 1. %GEN-TABLE-STRUCTURES
 2. for $l = 0, 1, ..., n-1$ do
 3.   $Z_l \leftarrow \{s_0\}$
 4.   for $i \leftarrow l+1, ..., n$ do
 5.     $Z_i \leftarrow \emptyset$
 6.     $t[l][i] \leftarrow 0$
 7.   endfor
 8.   for $i \leftarrow l, ..., n$ do
 9.     %STEP-WITH-TABLES
10.     if $Z_i \cap F = \emptyset$ and $(i > l)$ then
11.       $t[l][i] \leftarrow 1$
12.     endif
13.   endfor
14. endfor
15. return $t$

macro %GEN-TABLE-STRUCTURES
 1. $(X, r, \varphi) \leftarrow$ DECOMPOSE$(R)$
 2. for all $x \in X$ do
 3.   $t_x \leftarrow$ GEN-TABLE$(w, \varphi(x))$
 4. endfor
 5. $(S, \Sigma, X, \delta, s_0, F) \leftarrow$ GEN-NFA$(r)$
 6. for all $x \in X$ do
 7.   if $\epsilon \notin L(\varphi(x))$ then
 8.     $\delta(in_x, \epsilon) \leftarrow out_x$
 9.   endif
10. endfor

macro %STEP-WITH-TABLES
 1. $Z_i \leftarrow \overline{\delta}(Z_i, \epsilon)$
 2. if $i < n$ then
 3.   for all $x \in X$ do
 4.     if $in_x \in Z_i$ then
 5.       for $j \leftarrow i+1, ..., n$ do
 6.         if $t_x[i][j]$ then
 7.           $Z_j \leftarrow Z_j \cup \{out_x\}$
 8.         endif
 9.       endfor
10.     endif
11.   endfor
12.   $Z_{i+1} \leftarrow Z_{i+1} \cup \delta(Z_i, w_{i+1})$
13. endif
```

Fig. 2. Membership algorithm using tables

The simplest way to represent a table is as (half) an $n \times n$ matrix of boolean values. As far as the calculation of Z_0, Z_1, ..., Z_n and the membership of w to R are concerned, a set of tables $\{t_x$ table for w and $\neg\varphi(x) \mid x \in X\}$ would *contain all the necessary information* regarding the map $\varphi : X \to ERE_\Sigma$. Figure 2 shows an *ERE* membership algorithm that generates the table of each *ERE*-subexpression occurring under a complement from the tables of its subexpressions.

Proposition 3. *The algorithm* MEMB-WITH-TABLES(w, R) *in Figure 2 returns* true *if and only if* $w \in L(R)$*. If* $|w| = n$, $|R| = m$, *and* R *contains* k *complement operators, then this algorithm runs in space* $O(n^2 \cdot k + n \cdot m)$ *and time* $O(n^3 \cdot k + n^2 \cdot m)$.

Proof. To simplify its presentation and analysis, the algorithm in Figure 2 is split into two procedures and 2 macros. The macros should be regarded "ad literam", that is, one should simply replace their "invocation" by their pseudocode, character-by-character. %GEN-TABLE-STRUCTURES assumes some *ERE* R and some word w, and first decomposes R into (X, r, φ), then generates the corresponding tables for each $\neg\varphi(x)$ (in fact, for (non-asymptotic) efficiency, the procedure GEN-TABLE is passed $\varphi(x)$, but note that at its Steps 10-11 it actually sets the table bits to 1 when the subword is *not* in the language), and finally generates the root automaton of R. The macro %STEP-WITH-TABLES performs a "global step" in a root automaton. It assumes some step number i, corresponding to the latest processed letter in w, for which all sets Z_0, Z_1, ..., Z_{i-1} are already completely calculated and for which the sets Z_i, Z_{i+1}, ..., Z_n are only partially calculated, and finishes the calculation of Z_i, which only needs an ϵ-closure, and then updates the remaining Z_{i+1}, ..., Z_n as follows: if Z_i contains any special state in_x then the table t_x is consulted on its level $t_x[i]$ and all the sets Z_j with $w_{i+1} \cdots w_j \in L(\neg\varphi(x))$ are updated with the special state out_x; finally, the set Z_{i+1} is also updated by processing the next letter, w_{i+1}, in the current global state, Z_i. The procedure GEN-TABLE(w, R) will always be called on a sub-*ERE* R occurring under a complement in the original *ERE*, for which a table therefore needs to be generated. For each $0 \leq l \leq n-1$, it needs to set to 1 all the entries $t[l][i]$ for which $w_{l+1} \cdots w_i \in L(\neg R)$ (note that R is always some $\varphi(x)$ in its "parent" *ERE*). This can be done by first setting Z_l to $\{s_0\}$ and then simply traversing all the i's, completing Z_i and updating Z_{i+1}, ..., Z_n, and also checking whether Z_i contains any final state. The main procedure, MEMB-WITH-TABLES, is now self-explanatory. This algorithm follows more or less blindly Definition 4, so its correctness follows by Proposition 2.

Let us next calculate the complexity of this algorithm. Note that the sets Z_0, Z_1, ..., Z_n can be reused at each invocation of MEMB-WITH-TABLES and/or GEN-TABLE, so we define them as global; these sets of states are represented as vectors of bits of size m, so they take total space $O(n \cdot m)$.

Let us first analyze %GEN-TABLE-STRUCTURES, both with respect to space and time. Note that this macro is invoked by both MEMB-WITH-TABLES and GEN-TABLE, and both of these have a current *ERE* R; let m_r be the size of the *RE* root r of R. Step 1 takes space and time $O(m_r)$, including the time to update the bits stating the membership of ϵ to the language of each subexpression of r (and thus R). Steps 2-4 take space $O(\sum_{x\in X} space_{\mathrm{GT}(x)})$ and time $O(\sum_{x\in X} time_{\mathrm{GT}(x)})$, where $space_{\mathrm{GT}(x)}$ and $time_{\mathrm{GT}(x)}$ are the space and the time of GEN-TABLE$(w, \varphi(x))$. The $O(n^2)$ space needed to store the table t_x will be counted as part of $space_{\mathrm{GT}(x)}$; what is assigned to t_x is a pointer to the table already generated by GEN-TABLE$(w, \varphi(x))$. Step 5 takes space and time $O(m_r)$; assume the worst case space here, so adding new edges (at most one per node) to the automaton later will not require additional space. Since $\varphi(x)$ already contains the information $\epsilon \in L(\varphi(x))$ and since no new space is needed to add a new edge to a node in the automaton, Steps 6-10 take constant space and $O(m_r)$ time. Summing all these up, we obtain that %GEN-TABLE-STRUCTURES takes space $O(m_r + \sum_{x\in X} space_{\mathrm{GT}(x)})$ and time $O(m_r + \sum_{x\in X} time_{\mathrm{GT}(x)})$.

Let us now analyze %STEP-WITH-TABLES. The space for the global sets Z_0, Z_1, ..., Z_n has already been counted, and the space for the other operators can be reused, so this macro should take constant space in a good implementation. Anyhow, we can afford to assume, conservatively, that the space needed by the various operators is not reused, so the total space of %STEP-WITH-TABLES is $O(m_r)$. Steps 1 and 12 take time $O(m_r)$ and Step 7 takes $O(1)$, so the total time taken by %STEP-WITH-TABLES is $O(|X| \cdot n + m_r)$.

Let us next analyze GEN-TABLE. Since it needs to create the table t of size $O(n^2)$, one can readily see that it takes space $O(n^2 + m_r + \sum_{x \in X} space_{\mathrm{GT}(x)})$. Step 1 takes time $O(m_r + \sum_{x \in X} time_{\mathrm{GT}(x)})$. Steps 8-12, taking the major time in the outmost loop, take time $O(n \cdot (|X| \cdot n + m_r))$, so the total time taken by GEN-TABLE is $O(n^3 \cdot |X| + n^2 \cdot m_r + \sum_{x \in X} time_{\mathrm{GT}(x)})$.

We can now analyze the main procedure, MEMB-WITH-TABLES. Without making explicit the space and time consumed by GEN-TABLE, one can readily see that MEMB-WITH-TABLES takes space $O(m_r + \sum_{x \in X} space_{\mathrm{GT}(x)})$ and time $O(n^2 \cdot |X| + n \cdot m_r \cdot + \sum_{x \in X} time_{\mathrm{GT}(x)})$. To complete the analysis, note that GEN-TABLE is eventually invoked exactly once on every *ERE* R' with $\neg R'$ a subterm of the original *ERE* R. Since the sum of all the sizes $m_{r'}$ of the *RE* roots of these *ERE*s R' is $O(m)$, one can relatively easily see that the total space of MEMB-WITH-TABLES is $O(n^2 \cdot k + m)$ plus the total space $O(n \cdot m)$ to store Z_0, Z_1, ..., Z_n, that is, $O(n^2 \cdot k + n \cdot m)$. One can similarly calculate the total time of MEMB-WITH-TABLES to $O(n^3 \cdot k + n^2 \cdot m)$.

The space above can be non-asymptotically improved, by noting that once a table is calculated for an *ERE*, the tables of its subexpressions are not necessary anymore, so their space can be reused. Like the algorithms in [2,12,13,14,6,4], the algorithm in Figure 2 provides only a slight improvement over the classic one in [3]. Unfortunately, all known membership algorithms, including the one above, still require space $\Omega(n^2)$, which can be prohibitively large in many applications of interest. The problem here comes from storing the tables t_x for $x \in X$, each requiring $\Theta(n^2)$ space. We will next see that one can significantly reduce the required space as a function of n, namely from n^2 to $n \cdot \log n$. The idea is to encode the languages of $\varphi(x)$ for $x \in X$ in a more space effective fashion.

4 An Effective ERE Membership Algorithm

Definition 6. *A* ***jumping machine*** $\mathcal{P} = (P, p_0, \pi)$ *consists of set* P *of* ***states****, an* ***initial state*** p_0*, and a* ***jumping map*** $\pi : \{0, 1, ..., n-1\} \times P \to (\{1, 2, ..., n\} \times P) \cup \{\bot\}$ *with the property that for any* $0 \leq i < n$ *and any* $p \in P$*, if* $\pi(i, p) = (j, p')$ *then* $i < j$*. Given* $0 \leq i < n$*, we let* $\pi(i)$ *denote the set* $\{j_1, j_2, ..., j_{n_i}\}$ *with* $\pi(i, p_0) = (j_1, p_1)$*,* $\pi(j_1, p_1) = (j_2, p_2)$*, ...,* $\pi(j_{n_i - 1}, p_{n_i - 1}) = (j_{n_i}, p_{n_i})$*,* $\pi(j_{n_i}, p_{n_i}) = \bot$*. Given word* $w = w_1 w_2 \cdots w_n$ *and language* L*, we say that* (P, p_0, π) ***is a jumping machine for*** w ***and*** L *if and only if* $\pi(i) = \{j \mid j > i,\ w_{i+1} \cdots w_j \in L\}$*.*

Therefore, a jumping machine provides a mechanism to generate the sets $\pi(i)$ in a stepwise manner. A jumping machine for w and L can therefore eventually produce the same information as a table for w and L. However, the advantage of

```
MEMB-WITH-MACHINES($w, R$)
Input: $w = w_1 w_2 \cdots w_n \in \Sigma^\star$
       $R \in ERE$
Output: true / false
Globals: $Z', Z$
1. %GEN-MACHINE-STRUCTURES
2. %INITIALIZE-QUEUES
3. $Z' \leftarrow \{s_0\}$
4. for $i \leftarrow 0, 1, ..., n$ do
5.    %STEP-WITH-MACHINES
6. endfor
7. return $Z \cap F \neq \emptyset$

GEN-MACHINE($w, R$)
Input: $w = w_1 w_2 \cdots w_n \in \Sigma^\star$
       $R \in ERE$
Output: machine $(P, p_0, \pi)$
1. %GEN-MACHINE-STRUCTURES
2. $P \leftarrow 2^S$; $p_0 \leftarrow \{s_0\}$
3. for $l = 0, 1, ..., n-1$ do
4.    for all $p \in P$ do
5.       %INITIALIZE-QUEUES
6.       $Z' \leftarrow p$
7.       for $i \leftarrow l, ..., n$ do
8.          %STEP-WITH-MACHINES
9.          if $Z \cap F = \emptyset$ and $(i > l)$ then
10.            $\pi_x[l][p] \leftarrow i$; break-loop
11.         endif
12.      endfor
13.   endfor
14. endfor
15. return $(P, p_0, \pi)$

macro %GEN-MACHINE-STRUCTURES
1. $(X, r, \varphi) \leftarrow$ DECOMPOSE($R$)
2. for all $x \in X$ do
3.    $(P_x, p_0^x, \pi_x) \leftarrow$ GEN-MACHINE($w, \varphi(x)$)
4. endfor
5. $(S, \Sigma, X, \delta, s_0, F) \leftarrow$ GEN-NFA($r$)
6. for all $x \in X$ do
7.    if $\epsilon \notin L(\varphi(x))$ then
8.       $\delta(in_x, \epsilon) \leftarrow out_x$
9.    endif
10. endfor

macro %INITIALIZE-QUEUES
1. for all $x \in X$ do
2.    INITIALIZE($\mathcal{Q}_x, \{1, 2, ..., n\} \times P_x$)
3. endfor

macro %STEP-WITH-MACHINES
1. for all $x \in X$ do
2.    if $key$(TOP($\mathcal{Q}_x$)) equals $i$ then
3.       $Z' \leftarrow Z' \cup \{out_x\}$
4.       while $key$(TOP($\mathcal{Q}_x$)) equals $i$ do
5.          $(i, p_x) \leftarrow$ EXTRACT-TOP($\mathcal{Q}_x$)
6.          INSERT($\mathcal{Q}_x, \pi_x[i][p_x]$)
7.       endwhile
8.    endif
9. endfor
10. $Z \leftarrow \overline{\delta}(Z', \epsilon)$
11. if $i < n$ then
12.    for all $x \in X$ do
13.       if $in_x \in Z$ then
14.          INSERT($\mathcal{Q}_x, \pi_x[i][p_0^x]$)
15.       endif
16.    endfor
17.    $Z' \leftarrow \delta(Z, w_{i+1})$
18. endif
```

Fig. 3. Membership algorithm using jumping machines

jumping machines in contrast to tables is that they may require *much less space* to be stored. Indeed, a machine (P, p_0, π) can be encoded in space $\Theta(n \cdot |P| \cdot (\log n + \log |P|))$, namely when encoded as a $n \times |P|$ matrix storing in each cell an element in $(\{1, 2, ..., n\} \times P) \cup \{\bot\}$. This space can be roughly approximated with $\Theta(n \cdot \log n)$ when n is significantly larger than $|P|$, as opposed to $\Theta(n^2)$ as required by tables.

Figure 3 shows an *ERE* membership algorithm based on jumping machines, that modifies the one in Figure 2 correspondingly.

Theorem 1. MEMB-WITH-MACHINES(w, R) *in Figure* 3 *returns* true *iff* $w \in L(R)$. *If* $|w| = n$ *and* $|R| = m$ *then* MEMB-WITH-MACHINES(w, R) *runs in space* $O(n \cdot (\log n + m) \cdot 2^m)$ *and in time* $O(n^2 \cdot (\log n + m) \cdot 2^m)$.

Proof. One may show the correctness of this algorithm by analogy with the table-based algorithm in Figure 2, which is the reason for which we actually presented the table-based algorithm. In the table-based algorithm, given an *ERE* R that decomposed to (X, r, φ), we maintained a table t_x listing *explicitly* the entire "future" of each $\varphi(x)$ w.r.t. the remaining suffix of w (i.e., the set of future indexes $1 \leq j \leq n$ for which the special state out_x needs to be added to the current set of states Z_j at that moment). We now maintain a jumping machine $\mathcal{P}_x = (P_x, p_0^x, \pi)$ instead, which, at any "moment", i.e., index $0 \leq i \leq n-1$, "knows" explicitly only the first future moment when out_x needs to be considered, namely the one given by the first component of $\pi[i][\{p_0^x\}]$. However, the jumping machine also "freezes" its corresponding state at that future moment (the second component of $\pi[i][\{p_0^x\}]$), so that it *implicitly* "knows" how to generate the entire information in the corresponding table in the table-based algorithm; but this will be done on a *by-need* basis.

Like in the table-based algorithm, the ultimate purpose of the data-structures, jumping machines in this case, is to detect the future indexes at which the special states out_x need to be included in the set of (future) current states. In the table-based algorithm, the sets Z_0, Z_1, ..., Z_n accumulated this information progressively, by simply transferring it from the tables. Since the tables are not available anymore, when the special state in_x is encountered during the global step of the root automaton, we need to store somewhere the first future moment, say i, that out_x needs to be considered. That informal "somewhere" can be effectively replaced by a *priority queue* data-structure, $\mathcal{Q}_x$. Since the state in_x can be encountered several times before that moment i, each time starting a new "jumping session" in $\mathcal{P}_x$, we need to store *all* the first future moments to consider out_x of all the "sessions" that the jumping machine $\mathcal{P}_x$ can be in. Then at any global step of the algorithm, one needs to check whether any of the jumping machine sessions "predicted" the current moment as one to include out_x. If that is the case then, besides including out_x in the current global state, one also needs to advance the corresponding session in the jumping machine to its next "predicted" moment to include the state out_x. This is what Steps 1-9 in %STEP-WITH-MACHINES do. To accomplish this task properly, we store not only the first future moments of each session in the priority queue, but also the corresponding jumping machine session. Since several different sessions in $\mathcal{P}_x$ could have predicted the same current moment, all these sessions need to be advanced to their next predicted future moments to consider out_x (Steps 4-7 in %STEP-WITH-MACHINES). Making the intuitions above rigorous, the algorithm MEMB-WITH-MACHINES in Figure 3 flows in a one-to-one analogy to the table-based algorithm in Figure 2. As a "synchronization" point in this analogy, note that Z at Step 10 in %STEP-WITH-MACHINES corresponds to Z_i at Step 1 in %STEP-WITH-TABLES.

Let us next analyze the space and time complexity of this algorithm. Following a similar analysis to that of %Gen-Table-Structures, one immediately gets that %Gen-Machine-Structures requires space $O(m_r + \sum_{x \in X} \mathit{space}_{\mathrm{GM}(x)})$ and time $O(m_r + \sum_{x \in X} \mathit{time}_{\mathrm{GM}(x)})$. Gen-Machine tells us that P_x will have size 2^{m_x}, where m_x is the size of the root of $\varphi(x)$. Since we need to insert $|\{1, 2, ..., n\} \times P_x|$, which is $n \cdot 2^{m_x}$, in the priority queue $\mathcal{Q}_x$, and since each element requires space $\log(n \cdot 2^{m_x})$ to be stored as part of the Initialize step of the queue, one obtains that %Initialize-Queues takes space and time $O(n \cdot \sum_{x \in X} 2^{m_x} \cdot \log(n \cdot 2^{m_x}))$. %Step-With-Machines is invoked at places where all the memory it needs is allocated, so it takes constant space. The crucial observation in the time analysis of %Step-With-Machines is that the loop at Steps 4-7 executes at most 2^{m_x} times, because there can be at most that many pairs (i, p) in total and because we do not allow duplicates in queues. Therefore, Steps 1-9 take time $O(\sum_{x \in X} 2^{m_x} \cdot \log(n \cdot 2^{m_x}))$. Steps 10-18 only add time $O(m_r)$, where m_r is the size of r, so the total time of %Step-With-Machines is $O(\sum_{x \in X} 2^{m_x} \cdot \log(n \cdot 2^{m_x}) + m_r)$.

Let us now analyze the remaining two procedures. Step 1 in each of them takes space $O(m_r + \sum_{x \in X} \mathit{space}_{\mathrm{GM}(x)})$. Gen-Machine needs to allocate a jumping machine, whose space is dominated by the matrix π of size $n \times 2^{m_r}$ keeping elements in $\{1, 2, ..., n\} \times 2^S$, so each element of size $\log(n \cdot 2^{m_r})$. Therefore, the total space required by π is $O(n \cdot 2^{m_r} \cdot \log(n \cdot 2^{m_r}))$. Since %Initialize-Queues at Step 5 can reuse the same space for each iteration of the loop at Steps 4-13, we conclude that the total space required by Gen-Machine is $O(n \cdot (2^{m_r} \cdot \log(n \cdot 2^{m_r}) + \sum_{x \in X} 2^{m_x} \cdot \log(n \cdot 2^{m_x})) + \sum_{x \in X} \mathit{space}_{\mathrm{GM}(x)})$. Time-wise, note that the loops at Steps 3 and 4, respectively, add a factor of $n \cdot 2^{m_r}$ to the time of Steps 5-12. After calculations, we get that the total time of Gen-Machine is $O(n^2 \cdot 2^{m_r} \cdot (m_r + \sum_{x \in X} 2^{m_x} \cdot \log(n \cdot 2^{m_x})) + \sum_{x \in X} \mathit{time}_{\mathrm{GM}(x)})$. Without making explicit the space and time of the invoked Gen-Machine, one can quickly see that Memb-With-Machines takes space $O(m_r + n \cdot \sum_{x \in X} 2^{m_x} \cdot \log(n \cdot 2^{m_x}) + \sum_{x \in X} \mathit{space}_{\mathrm{GM}(x)})$ and time $O(n \cdot (m_r + \sum_{x \in X} 2^{m_x} \cdot \log(n \cdot 2^{m_x})) + \sum_{x \in X} \mathit{time}_{\mathrm{GM}(x)})$.

Let us now put all these together by iteratively expanding all the $\mathit{space}_{\mathrm{GM}(x)}$ and $\mathit{time}_{\mathrm{GM}(x)}$. Let us first calculate the space. Note that if one iteratively expands the terms $\mathit{space}_{\mathrm{GM}(x)}$ that occur in the space complexity of Memb-With-Machines, then each term of the form $n \cdot 2^{m_x} \cdot \log(n \cdot 2^{m_x})$ will occur exactly twice. The resulting space then will be $O(m_r + n \cdot \sum_{r'} 2^{m_{r'}} \cdot \log(n \cdot 2^{m_{r'}}))$, which is $O(m_r + n \cdot \log n \cdot \sum_{r'} 2^{m_{r'}} + n \cdot \sum_{r'} m_{r'} \cdot 2^{m_{r'}})$, where r' ranges over all the *RE* roots of all *ERE*s R' occurring under a $\neg$ operator in the original *ERE*, and $m_{r'}$ is the size of r'. Since $m_{r'} \leq m$ and since $\sum_{r'} 2^{m'_r} \leq 2^{\sum_{r'} m_{r'}} = 2^m$, by overestimation we get that the space required by Memb-With-Machines is $O(n \cdot (\log n + m) \cdot 2^m)$. The total time of Memb-With-Machines can be calculated in a similar manner to $O(n^2 \cdot (\log n + m) \cdot 2^m)$.

Corollary 1. *If $n > 2^m$, then our ERE-membership algorithm above runs in space $O(n \cdot \log n \cdot 2^m)$ and time $O(n^2 \cdot \log n \cdot 2^m)$.*

Hence, if $n > 2^m$, our algorithm above runs in space $O(n \cdot \log n \cdot 2^m)$, compared to $O(n^2 \cdot k + n \cdot m)$ (k is the number of complements in the *ERE*), the space required by the best known algorithms to solve the same problem; all algorithms, including ours, run in time a factor of n larger than their corresponding space. When does the algorithm proposed in this paper outperform the other algorithms? Roughly speaking, if $k = \Theta(m)$ then by the monotonicity of the function $x/\log x$ when $x > 2$, one gets $n/\log n > 2^m/m$, that is, that $n \cdot \log n \cdot 2^m$ is asymptotically better than $n^2 \cdot k$, so our algorithm wins. On the other hand, if $k = \Theta(1)$ then our algorithm again wins, but this time when $n > m \cdot 2^m$; however, if $k = \Theta(1)$ then one is likely better off using the standard NFA-to-DFA-then-complement algorithm.

5 Conclusion

Previous known algorithms to test whether a word of size n is in the language of an *ERE* of size m are either space/time non-elementary in m or otherwise space $\Omega(n^2)$ and time $\Omega(n^3)$. In the 27 years that passed since the first non-elementary algorithm has been given in [3], several algorithms for the *ERE* membership problem have been proposed. Unfortunately, none of them improved significantly the original algorithm in [3]. In particular, all the current non-elementary-in-m algorithms require space $\Omega(n^2)$, which is prohibitive in the context of some applications of interest. For example, $\Omega(n^2)$ means more than 1TB of memory when n is 1 million, and more than what today's technology can offer when n is larger than 1 billion. In this paper we presented an algorithm which is simply exponential in m but is in the order of $n \cdot \log n$ space-wise and $n^2 \cdot \log n$ time-wise. The proposed algorithm outperforms the known polynomial algorithms when n is exponentially larger than m.

A novel data-structure, called jumping machine, was also introduced in this paper and played a crucial technical role in our algorithm. It would be interesting to investigate to what extent the jumping machines can be used for improving other automata-theoretic constructions (that use two dimensional tables).

References

1. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.
2. S. Hirst. A new algorithm solving membership of extended regular expressions. Technical report, The University of Sydney, 1989.
3. J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.
4. L. Ilie, B. Shan, and S. Yu. Fast algorithms for extended regular expression matching and searching. In *Proceedings of STACS'03*, volume 2607 of *LNCS*, pages 179–190, 2003.
5. J.R. Knight and E.W. Myers. Super-pattern matching. *Algorithmica*, 13(1/2): 211–243, 1995.

6. O. Kupferman and S. Zuhovitzky. An improved algorithm for the membership problem for extended regular expressions. In *Proc. of MFCS'02*, volume 2420 of *LNCS*, pages 446–458, 2002.
7. G. Myers. A four russians algorithm for regular expression pattern matching. *Journal of the ACM*, 39(4):430–448, 1992.
8. G. Roşu. An effective algorithm for the membership problem for extended regular expressions. Technical Report UIUCDCS-R-2005-2964, Department of Computer Science, University of Illinois at Urbana-Champaign, 2005.
9. G. Roşu and M. Viswanathan. Testing extended regular language membership incrementally by rewriting. In *Proceedings of RTA'03*, volume 2706 of *LNCS*, pages 499–514. Springer, 2003.
10. L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time (preliminary report). pages 1–9. ACM Press, 1973.
11. K. Thompson. Regular expression search algorithm. *CACM*, 11(6):419–422, 1968.
12. H. Yamamoto. An automata-based recognition algorithm for semi-extended regular expressions. In *Proc. of MFCS'00*, volume 1893 of *LNCS*, pages 699–708, 2000.
13. H. Yamamoto. A new recognition algorithm for extended regular expressions. In *Proceedings of ISAAC'01*, volume 2223 of *LNCS*, pages 257–267, 2001.
14. H. Yamamoto and T. Miyazaki. A fast bit-parallel algorithm for matching extended regular expressions. In *Proc. of COCOON'03*, volume 2697 of *LNCS*, pages 222–231, 2003.

Complexity Results on Balanced Context-Free Languages

Akihiko Tozawa[1] and Yasuhiko Minamide[2]

[1] IBM Research,
Tokyo Research Laboratory, IBM Japan, Ltd.
[2] Department of Computer Science
University of Tsukuba

Abstract. Some decision problems related to balanced context-free languages are important for their application to the static analysis of programs generating XML strings. One such problem is the balancedness problem which decides whether or not the language of a given context-free grammar (CFG) over a paired alphabet is balanced. Another important problem is the validation problem which decides whether or not the language of a CFG is contained by that of a regular hedge grammar (RHG). This paper gives two new results; (1) the balancedness problem is in PTIME; and (2) the CFG-RHG containment problem is 2EXPTIME-complete.

1 Introduction

The study of balanced context-free languages or parenthesis context-free languages dates back to the 1960s and 1970s [McN67, Knu67, Tak75]. Recently balanced context-free languages attract new interests because of their application to XML-related problems [BB02b, BB02a, KM06], e.g., the static analysis of programs generating XML strings. In the previous work [MT06], we give algorithms to two such problems. This paper continues that study and gives answers to problems previously left open.

Let A be a base alphabet. Then, we introduce a paired alphabet consisting of two sets $\acute{A}$ and $\grave{A}$:

$$\acute{A} = \{\, \acute{a} \mid a \in A \,\} \qquad \grave{A} = \{\, \grave{a} \mid a \in A \,\}$$

where $\acute{A}$ and $\grave{A}$ correspond to the set of start tags and the set of end tags, respectively. We consider that $\acute{a}$ and $\grave{a}$ match. We write Σ for $\acute{A} \cup \grave{A}$. Then the fundamental notion on a string over a paired alphabet is whether it is balanced. For example, $\acute{a}\acute{b}\grave{b}\acute{c}\grave{c}\grave{a}$ and $\acute{a}\grave{a}\acute{b}\grave{b}$ are balanced, but $\acute{a}\grave{b}$ and $\acute{a}\acute{b}\grave{b}$ are not. This notion of balanced strings corresponds to well-formed documents in XML. We call the set of all balanced strings $B(\Sigma)$ the Dyck set over Σ [Ber79].

We consider context-free grammars over paired alphabets. The first problem to ask is the balancedness problem. Namely, whether or not the language of a given context-free grammar (CFG) G is balanced, or whether or not $L(G) \subseteq B(\Sigma)$. To our knowledge, the best known algorithm for this problem requires exponential time. However this is not optimal. We will prove that this problem in actually in PTIME.

This PTIME algorithm consists of two steps. The first step is to check the balancedness of the language as a grammar of a single kind of parentheses. We here use a fixpoint algorithm based on the algorithms by Knuth [Knu67], and Berstel and Boasson [BB02b]. However, we give a finer analysis of the number of iterations needed to reach fixpoint, which at first glance seems to be exponential to the size of the grammar, but in fact it is linear. The second step is to check each matched letters are of the same kind. Consider a CFG G with a singleton language. Such a CFG is sometimes called a straight line program (SLP). Assume that in this G, we have a production rule $I \to XY$ whose X and Y derive strings $\phi \in \acute{A}^*$ and $\psi \in \grave{A}^*$, respectively, of the same length. Now clearly, the singleton language of G is balanced if ϕ is identical to the reverse of ψ by ignoring $\acute{\_}$ and $\grave{\_}$ signs. Plandowski has shown that the problem of deciding the equivalence of two SLPs, and hence the balancedness of this $L(G)$, is in PTIME [Pla94]. We later show how to apply Plandowski's algorithm to the problem for general CFGs.

The second problem is the validation problem. Our previous work discussed the problem whether $L(G) \subseteq L(G')$ holds where G is a CFG and G' is either an (i) XML grammar or (ii) regular hedge grammar (RHG). In particular, the RHG defines an important language class corresponding to regular languages over trees. Indeed, any regular hedge language can be defined only from the following two kinds of productions.

$$X \to \acute{a}Y\grave{a}Z \quad \text{or} \quad X \to \epsilon.$$

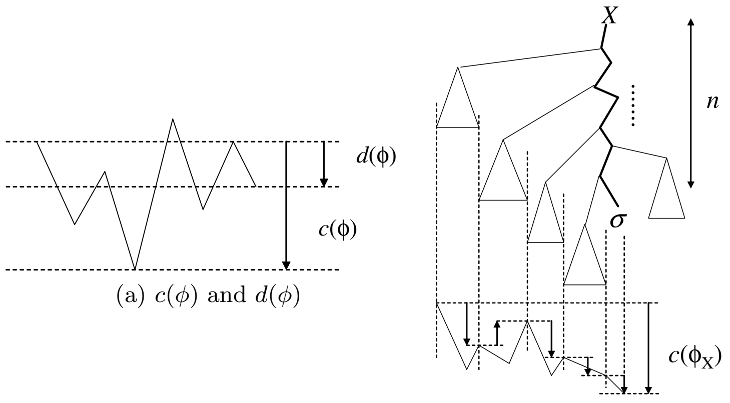
This corresponds to non-deterministic tree automata (NTA) on binary trees. The best known time complexity for the CFG-RHG containment is doubly exponential. This is actually optimal. In this paper, we prove that this problem is 2EXPTIME-hard.

A parenthesis grammar (PG) [McN67, Knu67] is a grammar with a single kind of parentheses, e.g., [and], and with production rules restricted to the following form.

$$X \to [Y_1 \cdots Y_k] \quad \text{or} \quad X \to a$$

Here letters a can be considered as the abbreviation of $\acute{a}\grave{a}$. It is then easy to see that the class of the PG is a subclass of the class of the RHG. We actually prove that the containment $L(G) \subseteq L(G')$ where G' is PG is already 2EXPTIME-hard. The idea of the proof comes from the observation of the gap between derivation trees and parse trees of balanced languages generated from CFGs. Namely, a single node on a parse tree, i.e., matching parentheses, can be split to two $2^{O(n)}$-distant nodes in the corresponding derivation tree.

The rest of the paper is organized as follows. Section 2 explains the algorithm for the balancedness problem. Section 3 proves 2EXPTIME-hardness of the CFG-RHG containment problem. Section 4 summarizes the related work.



(a) $c(\phi)$ and $d(\phi)$

(b) derivation tree of ϕ_X

Fig. 1. Illustration of $c(\phi)$, $d(\phi)$ and ϕ_X

2 PTIME Balancedness Check

We develop an algorithm which decides in polynomial time whether or not the language of a context-free grammar is balanced. The algorithm has two steps. In the first step, we only check *shapes* of strings. The language of a grammar is shape-balanced iff it is balanced by treating it as if using a single pair of parentheses, e.g., $\acute{a}\grave{b}$ is not balanced, but shape-balanced. In this step, we also pick up a string with the deepest valley in the set of strings produced from each nonterminal. The second step is the check of *color-balancedness*, i.e., we never see corresponding $\acute{a}$ and $\grave{b}$ such that $a \neq b$.

In this section, we assume a grammar $G = (\Sigma, V, R, I)$ such that all production rules are in the form $X \to \alpha$ or $X \to \alpha\beta$ where $\alpha, \beta \in \Sigma \uplus V$. We can convert arbitrary CFGs into this form in linear size[1]. We also assume that G is reduced. That is, every nonterminal is accessible from the start symbol and every nonterminal produces at least one terminal string. By a notation $C[]$, we mean a string containing a hole, which is filled with a string ϕ as $C[\phi]$.

2.1 Checking Balancedness of Shapes

The shape of a string is intuitively understood by reading this string from left to right as a line graph. Each letter $\grave{a}$ corresponds to a descending slope of the graph, and each $\acute{a}$ to a climbing slope of the same unit, respectively. Now we obtain the shape of a string by

– Keeping the levels of both ends of this line graph.
– Erase all valleys other than the one deepest in the graph.

[1] More precisely, we obtain G such that $L(G) = L(G') \setminus \{\epsilon\}$ from G'. Removing ϵ does not change the balancedness.

Formally, we say a string ϕ is shape-balanced if repeatedly removing all matching $\acute{a}\grave{b}$ from ϕ gives an empty string. Any string $\phi \in \Sigma^*$ is reduced to the following form with this reduction.

$$\grave{a}_i \cdots \grave{a}_1 \acute{b}_1 \cdots \acute{b}_j$$

We identify the shape of ϕ by $c(\phi)$ and $d(\phi)$ defined as $c(\phi) = i$ and $d(\phi) = i - j$. Here $c(\phi)$ is a nonnegative integer denoting the depth of the deepest valley measured from the left end, and $d(\phi)$ is a possibly-negative integer denoting the level of the right end measured from the left end (cf. Fig. 1(a)). A string ϕ is shape-balanced iff $c(\phi) = d(\phi) = 0$. The language $L(G)$ of a grammar G is shape-balanced if all strings in the language are shape-balanced.

If a grammar has a shape-balanced language, each language corresponding to its nonterminal X has constant $d(\phi)$, i.e., $d(\phi) = d(\psi)$ if $X \overset{*}{\Rightarrow} \phi, \psi$. Furthermore, for each nonterminal X, there is a bound m such that $X \overset{*}{\Rightarrow} \phi$ implies $c(\phi) \leq m$. To see this, consider the derivation $I \overset{*}{\Rightarrow} C[X]$. This $C[]$ must be in the form $\psi_0 \acute{a}_1 \psi_1 \cdots \acute{a}_i \psi_i [] \zeta_j \grave{b}_j \cdots \grave{b}_1 \zeta_0$ with $\psi_0, \ldots, \psi_i, \zeta_0, \ldots, \zeta_j$ shape-balanced. Since $c(C[\phi]) = d(C[\phi]) = 0$ for any $X \overset{*}{\Rightarrow} \phi$, we have $c(\phi) \leq i$ and $d(\phi) = i - j$.

The bound of $c(\phi)$ implies the existence of, not necessarily unique, element ϕ_X such that $X \overset{*}{\Rightarrow} \phi_X$ with maximum depth $c(\phi_X)$ $(= m)$. Here is the main proposition about this ϕ_X.

Proposition 1. *Assume $G = (\Sigma, V, R, I)$ such that $L(G)$ is shape-balanced. For each $X \in V$, we have ϕ_X with maximum $c(\phi_X)$, such that the height of the derivation tree is bounded by $2n + 1$ where $n = |V|$.*

Note that $L(G)$ is shape-balanced iff $c(\phi_I) = d(\phi_I) = 0$. This proposition bounds the number of iterations to find out ϕ_I, which is linear to the size of the grammar. To prove this proposition, we need analysis on primary paths of derivation trees.

See Fig. 1(b). This figure explains how given a derivation tree for $X \overset{*}{\Rightarrow} \phi$, we compute the depth of each valley in ϕ. Assume a path $t_0, t_1, \ldots, t_k (= t)$ in the derivation tree where t_0 is the root, and t is a leaf labeled either as $\acute{a}$ or $\grave{a}$. We would like to compute the depth $c(t)$ of the valley found around this occurrence of $\acute{a}$ or $\grave{a}$. Intuitively, this value is computed from the sum of all $d(\psi)$ where ψ is a substring corresponding to each branch appearing in the left of the path to t.

In this path, each non-leaf node t_i is labeled by a nonterminal $X \in V$ and associated with a rule $X \to \alpha$ or $X \to \alpha\beta \in R$. If t_{i+1} is the first successor of t_i labeled by α, the derivation by $X \to \alpha$ or $X \to \alpha\beta$ does not contribute to deepening the valley around t. On the other hand, if t_{i+1} is the second successor labeled by β, $c(t)$ is increased by $d(\psi)$ where $\alpha \overset{*}{\Rightarrow} \psi$ is a sub-derivation for the first successor of t_i. If the leaf node t is labeled by $\acute{a}$, the valley exists on the left of this occurrence of $\acute{a}$. On the other hand, if it is $\grave{a}$, the valley exists on the right, so that we increase $c(t)$ by 1. Now the *primary path* of ϕ is a path to a leaf t in the derivation tree with maximum $c(t)$. Then clearly this $c(t)$ is equal to $c(\phi)$.

Now assume that the grammar has a shaped-balanced language. Then the problem of finding out ϕ_X maximizing $c(\phi_X)$ becomes the problem of finding out a primary path, in a certain derivation tree for X, to the leaf t maximizing $c(t)$.

This problem can be considered as a graph problem. We can draw a graph whose vertices correspond to $V \uplus \Sigma$, and whose edge from X to γ corresponds to $X \to \gamma$ or $X \to \alpha\beta \in R$ such that either $\gamma = \alpha$ or β. Now we assign the weight corresponding to the increase in the depth $c(t)$ to each edge of this graph. The grammar has a shaped-balanced language, so that we can determine constant $d(\phi_X)$ for each nonterminal X. Let $d(\phi_{\acute{a}}) = -1$ and $d(\phi_{\grave{a}}) = 1$.

- For each edge from X to α where $\alpha \in V \uplus \acute{A}$,
  - if this edge corresponds to $X \to \alpha$ or $X \to \alpha\beta$, we give the weight 0,
  - if this edge corresponds to $X \to \beta\alpha$, we give the weight $d(\phi_\beta)$.
- For each edge from X to $\grave{a}$,
  - if this edge corresponds to $X \to \grave{a}$ or $X \to \grave{a}\beta$, we give the weight 1,
  - if this edge corresponds to $X \to \beta\grave{a}$, we give the weight $d(\phi_\beta) + 1$.

For example, consider the following grammar with the shape-balanced language.

$$I \to Z_0 Z_1 \qquad Z_0 \to \acute{a}\acute{b} \qquad Z_1 \to Z_1 I \qquad Z_1 \to \grave{b}\grave{a}$$

In this grammar, we always have $d(\phi) = -2$ if $Z_0 \overset{*}{\Rightarrow} \phi$, so that the edge from I to Z_1 is given weight -2.

$$\begin{tikzcd}
\acute{b} & Z_0 \arrow[l, "-1"'] \arrow[d, "0"] & I \arrow[l, "0"'] \arrow[r, bend right, "-2"'] & Z_1 \arrow[l, "2"'] \arrow[r, "2"'] \arrow[d, "1"] \arrow[loop above, "0"] & \grave{a} \\
& \acute{a} & & \grave{b} &
\end{tikzcd}$$

The problem of finding out a primary path with maximum weight first corresponds to the detection of *positive cycles* in the graph. If there is no such cycle, the problem reduces to the longest (maximum-weight) path problem.

Indeed, if the grammar has a shaped-balanced language, the graph constructed as such has no positive cycles. If there is such a cycle, clearly we fail to find any primary path with maximum weight, contradicting the shape-balancedness. On the other hand, if there is no such cycle, then for any path containing the same vertex twice, we can always find another path with at least the same weight, and without such duplicated occurrence of vertices. Now this proves that we can always find a maximum-weight path of length at most $n + 1$.

Fig. 1(b) illustrates how we construct ϕ_X. The length of the primary path in ϕ_X can be made at most $n+1$. We can use arbitrary derivation trees for subtrees not related to the primary path, whose height can be made at most $n + 1$. To sum up, the height of the derivation of ϕ_X can be made at most $2n+1$, proving Proposition 1.

2.2 Straight Line Programs for ϕ_X

If only concerning the shape-balancedness, it is enough to compute the shape of this ϕ_X, i.e., $c(\phi_X)$ and $d(\phi_X)$. This is rather easy. However for the color-balancedness, we need to compute ϕ_X themselves. Unfortunately, in the worst case, ϕ_X are not of polynomial length, so that we cannot obtain a PTIME algorithm if they are explicitly represented as strings. We here consider the compressed string representation using sharing graphs.

Definition 1. *(Straight Line Program) A straight line program (SLP) is an acyclic CFG without alternatives in production rules.*

In other words, an SLP is a CFG generating a singleton set. We give an algorithm to find each ϕ_X as an SLP in Fig. 2. This algorithm combines the fixpoint algorithm of shape-balancedness check with Bellman-Ford's algorithm for the longest path problem where the graph does not contain positive cycles [Law76].

This algorithm needs to be repeated $2n+1$ times. After S_{2n+1} is computed, we first check $S_{2n+1}(X) = S_{2n}(X)$ for all $X \in V$. If so, this means that the algorithm reaches the fixpoint, so that it could find ϕ_X with maximum depth for each X. Otherwise, the algorithm could not find ϕ_X with the height of derivation $2n+1$, so that the shape-balancedness has failed from Proposition 1. The first and second component of $S_{2n}(X)$ denote $c(\phi_X)$ and $d(\phi_X)$, respectively, so that we then check that $S_{2n}(I) = (0, 0, \_)$.

After the shape-balancedness check has passed, we construct ϕ_X for each nonterminal X of the grammar. The third component of $S_{2n}(X)$ corresponds to primary paths in the SLP for ϕ_X. We introduce a set of nonterminals $\overline{V} = \{\overline{X} \mid X \in V\}$ and define the set of rules P in the form $\overline{X} \to \theta$ where $\theta = \overline{\alpha}, \overline{\alpha}\beta$ or $\alpha\overline{\beta}$.

$$P = \{\overline{X} \to \theta \mid S_{2n}(X) = (\_, \_, \overline{X} \to \theta),\ X \in V\}$$

This P corresponds to the longest paths trees of Bellman-Ford algorithm computing a collection of longest paths for any source-target pair in the graph. For example, the following trees show the solution of the longest path problem in the previous section, where target vertices correspond to terminals.

$$\begin{tikzcd}
\acute{b} & Z_0 \arrow[d, "0"] & I \arrow[l, "0"'] & Z_1 \arrow[r, "2"'] & \grave{a} \\
 & \acute{a} & & \grave{b} &
\end{tikzcd}$$

This means that we use the same set of rules P to compute any ϕ_X. The following P corresponds to the above solution where $\overline{\acute{a}} = \acute{a}$ and $\overline{\grave{a}} = \grave{a}$.

$$\overline{I} \to \overline{Z}_0 Z_1 \qquad \overline{Z}_0 \to \overline{\acute{a}}\acute{b} \qquad \overline{Z}_1 \to \grave{b}\overline{\grave{a}}$$

Finally, we construct a set of rules U. This is by induction on the depth k of the derivation. Let $U_0 = \{\}$, and at $k+1$-th step, choose and add exactly one rule $Y \to \alpha$ or $Y \to \alpha\beta$ in R to U_{k+1} such that $Y \notin dom(U_k)$, and $\alpha, \beta \in dom(U_k)$. Here $dom(U) = \Sigma \uplus \{Y \in V \mid Y \to \_ \in U\}$. We let $U = U_n$. By construction, $dom(U) = \Sigma \uplus V$, for each $Y \in V$ we have exactly one rule in U, and U induces no cycles.

The following SLP only has the size linear to the original grammar G.

Algorithm 1. *We obtain the SLP for ϕ_X as $(\Sigma, V \uplus \overline{V}, U \uplus P, \overline{X})$.*

Input: CFG (Σ, V, R, I).
Output: A mapping $S_k \in (\Sigma \uplus V) \to (\mathbb{N}\times\mathbb{N}\times(\{\overline{X} \to \theta \mid \theta = \overline{\alpha},\ \overline{\alpha}\beta,\ \alpha\overline{\beta}\}\uplus\{\bullet\})\uplus\{\bot\})$.

1 We first initialize $S_0(X)$ for $X \in V$ and $S_k(\sigma)$ for $\sigma \in \Sigma$ as follows.

$$S_0(X) = \bot,\ S_k(\acute{a}) = (0, -1, \bullet),\ S_k(\grave{a}) = (1, 1, \bullet)$$

2 We iteratively compute $S_{k+1}(X)$ for $X \in V$ as follows.

– For each of (i) $X \to \alpha$ or (ii) $X \to \alpha\beta$, such that $S_k(\alpha)$ and $S_k(\beta)$ (if (ii)) are not $\bot$. Assume that $S_k(\alpha), S_k(\beta)$ and $S_k(X)$ (if not $\bot$) are as follows.

$$S_k(\alpha) = (c_1, d_1, \_),\ S_k(\beta) = (c_2, d_2, \_),\ S_k(X) = (c_0, d_0, \_)$$

In case (ii) with $S_k(X) \neq \bot$, we first confirm that $d_0 = d_1 + d_2$. If this fails, the shape-balancedness has failed. We then compute $S_{k+1}(X)$ as follows. Let $d_3 = d_1 + d_2$, $c_3 = d_1 + c_2$, and $\overline{\sigma} = \sigma$.

$$S_{k+1}(X) = \begin{cases} (c_1, d_1, \overline{X} \to \overline{\alpha}) & ((\text{i}),\ c_1 > c_0 \text{ if } S_k(X) \neq \bot) \\ (c_1, d_3, \overline{X} \to \overline{\alpha}\beta) & ((\text{ii}),\ c_1 \geq c_3, \text{ and } c_1 > c_0 \text{ if } S_k(X) \neq \bot) \\ (c_3, d_3, \overline{X} \to \alpha\overline{\beta}) & ((\text{ii}),\ c_1 < c_3, \text{ and } c_3 > c_0 \text{ if } S_k(X) \neq \bot) \\ S_k(X) & (\text{otherwise}) \end{cases}$$

Fig. 2. Combined algorithm for computing primary paths

2.3 Checking Color-Balancedness

The remaining step of the balancedness check is to confirm that each pair of coupled parentheses in strings is of the same color, i.e., of the same base letter in A. If so we call such strings color-balanced, or partially-balanced since they can be defined as substrings of balanced strings. A string is balanced iff it is both shape-balanced and color-balanced.

A color-balanced string ϕ is factorized as $\phi_{-i}\grave{a}_i\phi_{i-1}\cdots\grave{a}_1\phi_0\acute{b}_1\phi_1\cdots\acute{b}_j\phi_j$ such that $\phi_{-i}, \ldots, \phi_j$ are balanced. Even an arbitrary string can be similarly factorized by allowing $\phi_{-i}, \ldots, \phi_j$ to be shape-balanced. According to this factorization, we define

$$\rho(\phi) = \grave{a}_i\cdots\grave{a}_1\acute{b}_1\cdots\acute{b}_j$$

Next we define an ordering $\sqsubseteq$ on $\grave{A}^*\acute{A}^*$ as the minimal one satisfying

$$\phi\psi \sqsubseteq \phi\grave{a}\acute{a}\psi$$

for $\phi \in \grave{A}^*$ and $\psi \in \acute{A}^*$. We again extend this to a quasi-ordering $\phi \sqsubseteq \psi \Leftrightarrow \rho(\phi) \sqsubseteq \rho(\psi)$. Note that $\phi \sqsubseteq \psi$ implies $c(\phi) \leq c(\psi)$. Now, the remaining part of the algorithm is fairly simple.

Algorithm 2. *(Balancedness Check) Assume that we already computed ϕ_X with maximum $c(\phi_X)$ for each nonterminal of the given grammar $G = (\Sigma, V, R, I)$. We let $\phi_\sigma = \sigma$. For each $X \in V$, we check the following:*

- *ϕ_X is color-balanced.*
- *$\phi_X \sqsupseteq \phi_\alpha$, if $X \to \alpha \in R$.*
- *$\phi_\alpha\phi_\beta$ is color-balanced and $\phi_X \sqsupseteq \phi_\alpha\phi_\beta$, if $X \to \alpha\beta \in R$.*

It is easy to see that a grammar with a balanced language satisfies these conditions. This follows from the fact that under the balancedness, any ϕ_X with maximum $c(\phi_X)$ bounds all strings generated from X also according to $\sqsubseteq$. The other direction is shown by the following proposition; the success of Algorithm 2 implies the color-balancedness of the language of the grammar which, together with the shape-balanced check, proves the balancedness of the language.

Proposition 2. *If the check succeeds, $X \overset{*}{\Rightarrow} \phi$ implies (i) ϕ is color-balanced, and (ii) $\phi_X \sqsupseteq \phi$.*

The proof is by induction on the length of the derivation of ϕ. The base case is about terminal symbols and easy. For inductive step, assume that the proposition holds for strings obtained by derivation whose height is not greater than k. Now consider the derivation $X \overset{*}{\Rightarrow} \phi$ with its height $k+1$. If $X \to \alpha\beta$ is used, we have $\phi_1\phi_2 = \phi$ such that $\alpha \overset{*}{\Rightarrow} \phi_1$ and $\beta \overset{*}{\Rightarrow} \phi_2$. Note that it is safe to replace a substring ψ of a color-balanced string with another color-balanced string ψ' compatible to ψ. Formally, if (a) ψ' and $C[\psi]$ are color-balanced, and (b) $\psi \sqsupseteq \psi'$, we have (c) $C[\psi']$ color-balanced and (d) $C[\psi] \sqsupseteq C[\psi']$. Now we prove the case as follows. By assumption, (a) $\phi_\alpha\phi_\beta$, ϕ_1 and ϕ_2 are color-balanced, we also have (b) $\phi_\alpha \sqsupseteq \phi_1$ and $\phi_\beta \sqsupseteq \phi_2$. Hence (c) ϕ is color-balanced, and (d) $\phi_X \sqsupseteq \phi_\alpha\phi_\beta \sqsupseteq \phi_1\phi_\beta \sqsupseteq \phi_1\phi_2 = \phi$. The case $X \to \alpha$ is easy.

2.4 SLP and CS Equivalence

Finally, we need to confirm that for strings given as SLPs, both the color-balancedness and the check of $\phi \sqsubseteq \psi$ are decidable in PTIME. Fortunately, as noted in the introduction, we can use Plandowski's algorithm deciding the equivalence of two SLPs in PTIME.

First we give some definitions. Let ϕ^o be strings created from ϕ just by taking letters in $\acute{A}$ and removing $\acute{}$ sign. Let ϕ^c be strings created from ϕ similarly for $\grave{A}$ but in addition, by reversing the obtained string. For example, $(\grave{a}\grave{b}\acute{c})^o = c$ and $(\grave{a}\grave{b}\acute{c})^c = ba$. We use $\phi[j,k]$ to denote a substring of ϕ starting from its j-th letter and ending before the k-th letter.

Proposition 3. *Let $c_j = |\rho(\phi_j)^c|$, $o_j = |\rho(\phi_j)^o|$, and $m = min(o_1, c_2)$.*

(i) We have $\phi_1\phi_2$ color-balanced iff ϕ_1 and ϕ_2 are color-balanced, and

$$\rho(\phi_1)^o[o_1 - m, o_1] = \rho(\phi_2)^c[c_2 - m, c_2]$$

(ii) We have $\phi_1 \sqsubseteq \phi_2$ iff $s = c_2 - c_1 = o_2 - o_1 \geq 0$, and

$$\begin{aligned}\rho(\phi_1)^c &= \rho(\phi_2)^c[s, c_2]\\ \rho(\phi_1)^o &= \rho(\phi_2)^o[s, o_2]\\ \rho(\phi_2)^o[0, s] &= \rho(\phi_2)^c[0, s]\end{aligned}$$

Input: SLP (Σ, V, R, I)
Output: CS $(A, V^c \uplus V^o, R', I^c \text{ or } I^o)$

1 For a rule $X \to \alpha\beta \in R$, we let $o_1 = c(\phi_\alpha) - d(\phi_\alpha)$, $c_2 = c(\phi_\beta)$, and $m = min(c_2, o_1)$.

$$X^o \to \alpha^o[0, o_1 - m]\beta^o$$
$$X^c \to \beta^c[0, c_2 - m]\alpha^c$$

2 For a rule $X \to \alpha$, we just define

$$X^c \to \alpha^c, X^o \to \alpha^o$$

3 Finally for letters we define

$$\acute{a}^c = \grave{a}^o = \epsilon,\ \acute{a}^o = \grave{a}^c = a,$$

Fig. 3. Translation of SLP for ϕ_X into CS for $\rho(\phi_X)$

(iii) A composition system (CS) is an SLP which allows occurrences of nonterminals in the form $X[j,k]$ in the rhs of productions. Given an SLP generating ϕ, the translation in Fig. 3 gives CS such that $I^c \overset{*}{\Rightarrow} \rho(\phi)^c$ and $I^o \overset{*}{\Rightarrow} \rho(\phi)^o$.

It is known that the equivalence problem of two CSs also has a PTIME algorithm, since a CS can always be converted back into a polynomial-size SLP [Hag00, Sch06]. We use this algorithm with the property (ii) to determine $\phi_1 \sqsubseteq \phi_2$. We also use it with the property (i) to create a proof tree showing that each ϕ_X is (or is not) color-balanced by checking, for each production $Y \to \alpha\beta$ needed in constructing ϕ_X, that two CS, i.e., those using start symbols $\alpha^o[o_1 - m, o_1]$ and $\beta^c[c_2 - m, c_2]$, are equivalent.

Now the following theorem is immediate from Proposition 2 and 3.

Theorem 1. *The balancedness problem is in PTIME.*

3 2EXPTIME-Completeness of CFG-RHG Containment

We show that the CFG-RHG containment problem is 2EXPTIME-complete. In our previous work, we developed a decision algorithm for the problem which has doubly exponential time complexity [MT06]. Here we prove that this algorithm is actually optimal by showing that the problem is 2EXPTIME-hard.

As noted in the introduction, we actually show the 2EXPTIME-hardness of the CFG-PG containment problem. The result for the CFG-RHG containment is immediately obtained by regarding PGs as RHGs. For this, we distinguish a single pair of parenthesis [and]. We use an abbreviation $a = \acute{a}\grave{a}$ and use A to denote the set of strings in this form.

3.1 The Key Observation

Seidl [Sei90] showed that the containment between the languages of two nondeterministic tree automata (NTA) is EXPTIME-complete. In fact, NTA defines

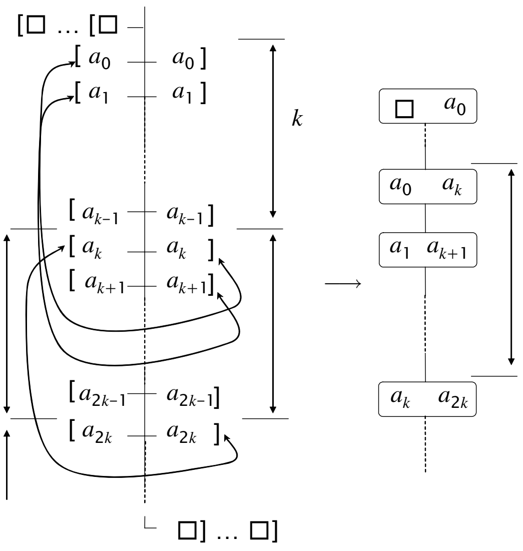


Fig. 4. A derivation tree and parse tree

the class of tree languages corresponding to languages of parse trees obtained from string languages defined by PG. If the problem is PG-PG or RHG-RHG containment, or even if the lhs of the containment is a balanced grammar such that $c(\phi_X) = d(\phi_X) = 0$ for all $X \in V$, the complexity relaxes to EXPTIME. In the case of NTA-containment, the size of the lhs is merely a polynomial factor. On the other hand, in the CFG-PG containment problem, the primary factor of the complexity is CFG on lhs. We here explain the high 2EXPTIME complexity from the gap between a derivation tree and parse tree of each string in the language of this CFG. Each node in a derivation tree corresponds to a non-terminal, while each node in a parse tree corresponds to matched parentheses. In the case of balanced grammars, two trees are not so different in the sense that each matching parentheses also exist closely, i.e., as siblings, in a derivation tree. This is generally not the case for a grammar just with a balanced language.

First note that for arbitrary $k \in \mathbb{N}$ and $\phi \in \Sigma^*$, we can construct a CFG (SLP) I_ϕ^k of size $O(\log k)$ that accepts ϕ^k. For this, we define $X_0 \to \phi$, $X_{i+1} \to X_i X_i$, and $I_\phi^k \to X_{i_1} \ldots X_{i_n}$ where $i_1, .., i_n$-th bits are set in the binary encoding of k. Now let $\Box \in A$ and define a grammar G as

$$I \to I_{[\Box}^k X,\ X \to I_{\Box]}^k \text{ and } X \to [aXa] \text{ for } a \in A$$

generating the following strings.

$$\overbrace{[\Box \cdots [\Box}^{k}[a_0[a_1 \cdots \overbrace{\Box] \cdots \Box]}^{k} \cdots a_1]a_0]$$

See also a derivation tree for this grammar described as the left tree of Fig. 4. In the figure, double-ended arrows between parentheses indicate matched

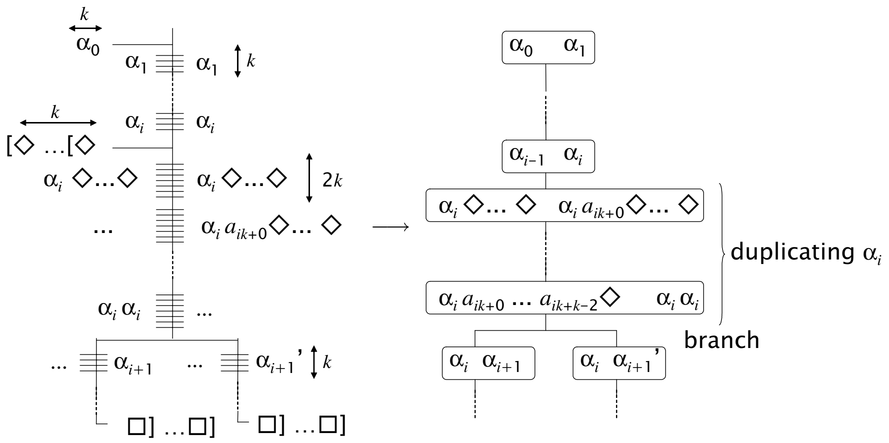


Fig. 5. Simulation of a branch in ATM

parentheses. We can see that each matched parentheses exist in k-distant positions in the derivation tree. Hence, a parse tree of the language of this grammar looks like the right tree of Fig. 4.

Now the procedure that decides $L(G) \subseteq L(G_{\mathrm{pg}})$ can be considered as a procedure which checks whether all parse trees above are contained in $L(G_{\mathrm{pg}})$ as trees. This PG compares the sequence $a_0 \cdots a_{k-1}$ with $a_k \cdots a_{2k-1}$, $a_k \cdots a_{2k-1}$ with $a_{2k} \cdots a_{3k-1}$, and so on. The first idea for the hardness proof is to use this fact to simulate a Turing machine (TM) using k-space. Let $\alpha_i = a_{ik} \cdots a_{(i+1)k-1}$ be a configuration of TM. It is possible to construct a PG to check if each transition from α_i to α_{i+1} is a valid single computation step of the given TM.

Although this idea indeed works as we will later formally discuss, we need one more trick to prove the 2EXPTIME-hardness. Note here that the size of G is $O(\log k)$. This means that the above TM can only solve EXPSPACE problems. To obtain the 2EXPTIME-hardness, we consider alternating Turing machines (ATM). An ATM is a Turing machine with conjunctive transitions in the form $q \vdash q_1 \wedge q_2$[2]. A computation of an ATM is thus a tree with branching degrees at most 2, and each 2-degree branch corresponds to this $\wedge$-transition. It is known that the class of 2EXPTIME is identical to the problems solvable by ATM using exponential space [CKS81].

The idea to simulate ATM is to add the following production rules to the previous grammar.

$$X \to I^k_{[\diamond} Y,\ Y \to XX \text{ and } Y \to [aYa] \text{ for } a \in A$$

A derivation tree of this grammar may look like the left tree of Fig. 5. We simplify the figure by omitting most of [and], and we also group each length-k

[2] In the literature, conjunctive transitions $q \vdash q_1 \wedge q_2$ are often expressed by using $\forall$-states, which correspond to branches in the computation tree, of possibly more than degree 2. It is easy to see that such an ATM using many-degreed branches can be easily converted into an equivalent ATM using at most 2-degree branches.

sequence of leaves $a_{ik}, \cdots, a_{(i+1)k-1}$ along the main branch as α_i. Now, we at some timing start to use Y instead of X in the main branch of the derivation, and this branch by Y eventually ends with two branches of derivations by X. Now the right tree of Fig. 5 explains how such derivation is parsed. In this figure, each node corresponding to the derivation by Y is illustrated as twice as wide as nodes for X. This reflects $d(\phi_Y) = 2k$ and $d(\phi_X) = k$. This also means that the PG can now simulate a machine with $2k$-space here. We would like to use this machine to duplicate the information of α_i before the branch. Such a duplication can be done by $O(k)$-computation steps, whose each step can be checked by constant size PG. After successfully creating $\alpha_i\alpha_i$, we can simulate alternating transitions α_i to α_{i+1}, and α_i to α'_{i+1}, by checking two child branches independently.

Using this idea, we will prove the following theorem in the next section.

Theorem 2. *The CFG-PG containment problem is 2EXPTIME-hard.*

3.2 Proof of 2EXPTIME-Hardness

Let P be a 2EXPTIME problem and M be an ATM solving this problem using exponential space. This ATM M is a tuple $(Q, \Gamma, \vdash, q_0, \Box, H)$ where

- Q is a set of states,
- Γ is a set of symbols,
- $\vdash \subseteq (\Gamma \times Q \times \{-1,0,1\} \times \Gamma \times Q) \uplus (Q \times \{q \wedge q' \mid q, q' \in Q\})$ is a transition relation,
- q_0 is an initial state,
- $\Box \in \Gamma$ is a blank symbol, and
- H is a set of accepting states.

A configuration of M is $\alpha = s_0 \cdots s_{i-1} s_i^q s_{i+1} \cdots \in \Gamma^*(\Gamma \times Q)\Gamma^\omega$ where s_i^q indicates that the current state is q and the head is at i-th position. The computation of M starts from $\alpha_0 = x_0^{q_0} x_1 \cdots x_{n-1} \Box \cdots$ where $x = x_0 \cdots x_{n-1}$ is an input. If $|x| = n$, the computation of M uses no more than $k = 2^{p(n)}$ space for some polynomial p. A computation history of M is a finite tree $T = (T, \alpha)$ associated with a function $\alpha \in T \to \Gamma^*(\Gamma \times Q)\Gamma^\omega$. This T is of branching degrees at most 2, so that the set of nodes T is given as a finite downward-closed subset of $\{1,2\}^*$ under lexicographic ordering on $\{1,2\}^*$. For each node $t \in T$, $\alpha(t)$ represents the configuration at t. Each degree 2 branch in T corresponds to alternating transitions $q \vdash q_1 \wedge q_2$. If $x \in P$, we can find T whose all leaves t satisfy $s_i^q \in \alpha(t)$ for some $q \in H$. Otherwise, any T has some leaves with non-accepting states.

Given an input $x = x_0 \cdots x_{n-1}$, we can construct the following CFG G in deterministic polynomial time and with its size polynomial to the input:

$$\begin{aligned}
&I \to [\#[x_0^{q_0} \cdots [x_{n-1} I_{[\Box}^{k-n} X \\
&X \to [aXa] \quad \text{for } a \in A' \\
&X \to [\# I_{[\Diamond}^{k} Y \\
&X \to \#] I_{\Box]}^{k} \\
&Y \to [aYa] \quad \text{for } a \in A \\
&Y \to XX
\end{aligned}$$

where we let $k = 2^{p(n)}$, $Q^\bullet = \{\bullet\} \uplus Q$, $A' = \{\#\} \uplus \Gamma \times Q^\bullet = \{\#\} \uplus \Gamma \uplus \Gamma \times Q$, $A = \{\#\} \uplus (\Gamma \times Q^\bullet \uplus \{\diamond\}) \times \{!\}^\bullet$. We add the symbol # to recognize boundaries of each configuration.

We would like to create a PG G_{pg} or equivalently NTAs such that any parse tree in $L(G) \setminus L(G_{\text{pg}})$ corresponds to an accepting computation history in the sense as explained in the last section. We model the parse tree as in Fig. 4 by a tree $U = (U, \lambda)$ associated with a labeling function $\lambda \in U \to A \times A$. Any $\phi \in L(G)$ can be parsed as such U. This U also has branching degrees at most 2. Any NTA running on U can be efficiently converted into an equivalent PG on ϕ.

First, we rule out ill-formed trees such that either # is not inserted appropriately, or 2-degree branches occur at inappropriate positions. For this, we use NTA N_1 which accepts any tree U with $u \in U$ such that either (i) $\lambda(u) = (\#, s), (s, \#)$ where $s \neq \#$, or (ii) $\lambda(u) \neq (\#, \#)$ and u is of branching degree 2.

Before simulating an alternation step of the ATM, we need to copy a configuration. By the production $X \to [\# I^k_{[\diamond} Y$, we prepare the copy and obtain the configuration below.

$$\# \; s_0 \; s_1 \cdots s_i^q \cdots s_{k-1} \; \# \; \diamond \; \diamond \quad \cdots \quad \diamond \; \#$$

At the first step, we place two markers denoted by ! at the two positions after #, then repeatedly move two markers to the right and copy the contents.

$$\begin{array}{l} \# \; s_0 \; s_1 \cdots s_i^q \cdots s_{k-1} \; \# \; \diamond \; \diamond \quad \cdots \quad \diamond \; \# \\ \# \; s_0^! \; s_1 \cdots s_i^q \cdots s_{k-1} \; \# \; \diamond^! \; \diamond \quad \cdots \quad \diamond \; \# \end{array}$$

$$\begin{array}{c} \# \; s_0^! \; s_1 \cdots s_i^q \cdots s_{k-1} \; \# \; \diamond^! \; \diamond \quad \cdots \quad \diamond \; \# \\ \# \; s_0 \; s_1^! \cdots s_i^q \cdots s_{k-1} \; \# \; s_0 \; \diamond^! \quad \cdots \quad \diamond \; \# \\ \vdots \\ \# \; s_0 \; s_1 \cdots s_i^q \cdots s_{k-1}^! \; \# \; s_0 \; s_1 \quad \cdots \quad \diamond^! \; \# \\ \# \; s_0 \; s_1 \cdots s_i^q \cdots s_{k-1} \; \# \; s_0 \; s_1 \quad \cdots \quad s_{k-1} \; \# \end{array}$$

This process is checked by an NTA N_2, which checks each sequence of nodes $u_0, \ldots, u_n, \ldots, u_m \in U$ such that (i) each u_{i+1} is the successor of u_i, (ii) $\lambda(u_i) = (\#, \#)$ iff $i = 0, n, m$, (iii) $\lambda(u_i) = (\diamond, \_)$ or $(\diamond^!, \_)$ for some $n < i < m$. Note that any tree in $L(G) \setminus L(N_1)$ contains such a sequence only when $n = k+1$ and $m = 2k+2$. N_2 accepts a tree which contains an incorrect copying sequence, e.g., $\lambda(u_i) = (s^!, s)$, $\lambda(u_j) = (\diamond^!, s')$ and $s \neq s'$ for some i, j. This N_2 also accepts a tree if one of (i) $\lambda(u_{m-1}) = (\diamond^!, \_)$ and (ii) the node u_m is of branching degree 2, exclusively holds.

Finally, we construct an NTA N_3 for checking the transitions of M. This is similar to N_2 except that we check a sequence $u_0, \ldots u_n$ such that $\lambda(u_0) = \lambda(u_n) = (\#, \#)$ without occurrence of ! and # on nodes $u_1 \ldots u_{n-1}$. This N_3 accepts U whenever it finds a sequence $u_0, \ldots, u_n$ such that u_0 is of branching degree 1, but the sequence is either (i) not obeying $\vdash$ when u_n is also of branching-degree 1, or (ii) not containing accepting state, i.e., no u_i such that $\lambda(u_i) = (s^q, \square)$ for some $q \in H$, when u_n is a leaf. This N_3 also accepts U whenever it finds $u_0 \in U$ of branching degree 2, followed by two sequences $u_0, u_1, \ldots, u_n$ and $u_0, u'_1, \ldots, u'_m$, which does not have i and j such that (i) $\lambda(u_{i'})$, $\lambda(u'_{j'})$ are in the form (s, s) if $i' \neq i$ and $j' \neq j$, and (ii) $\lambda(u_i) = (s^q, s^{q_1})$ and $\lambda(u'_j) = (s^q, s^{q_2})$ for some alternating transition $q \vdash q_1 \wedge q_2$.

We obtain the parenthesis grammar G_{pg} from an NTA accepting the union of $L(N_1)$, $L(N_2)$ and $L(N_3)$. The size of this G_{pg} is independent of x. Now for any x, we construct G in deterministic polynomial time from x so that we have $x \in P$ iff $\neg L(G) \subseteq L(G_{\mathrm{pg}})$. Hence the CFG-PG containment is 2EXPTIME-hard.

4 Related Work

In the same paper as the SLP equivalence [Pla94], Plandowski has also shown the existence of a polynomial-size *test set* for given CFG G. A test set $T \subseteq L(G)$ is such that given two morphisms $h, h' \in \Sigma \to M$ for a free group M, if $h(\phi) = h'(\phi)$ for all $\phi \in T$ then this holds for all $\phi \in L(G)$. He computed this T as a set of SLPs. Hence if we can efficiently decide whether or not $h(\phi) = \epsilon$ for all $\phi \in T$, by letting $h'(\sigma) = \epsilon$ (= unit of M) for all $\sigma \in \Sigma$, we obtain the PTIME algorithm to a problem similar to the balancedness problem which decides whether or not $h(L(G)) = \{\epsilon\}$. One difference here is that $aa^{-1} = a^{-1}a = \epsilon$ holds in a free group, while $\grave{a}\acute{a}$ is irreducible in the balancedness problem. We are not sure if we have another polynomial time algorithm for the balancedness problem in this direction.

Meyer and Stockmeyer showed that the equivalence problem for regular expressions extended with squaring is EXPSPACE-hard and further investigated the complexity of the problems for various variants of regular expressions [MS72, SM73]. For tree languages, Seidl showed that the equivalence of non-deterministic tree automata is EXPTIME-complete using an alternating Turing Machine as our discussion [Sei90]. The proofs of these completeness results are based on the hardness of the corresponding universality problems for languages of the rhs of the containment. On the other hand, in our proof of 2EXPTIME-completeness of CFG-RHG containment, a CFG in the lhs of the containment is essential.

References

[BB02a] Jean Berstel and Luc Boasson. Balanced grammars and their languages. In *Formal and Natural Computing: Essays Dedicated to Grzegorz*, volume 2300 of *LNCS*, pages 3–25, 2002.

[BB02b] Jean Berstel and Luc Boasson. Formal properties of XML grammars and languages. *Acta Informatica*, 38(9):649–671, 2002.

[Ber79] Jean Berstel. *Transductions and Context-Free Languages.* Teubner Studienbucher, 1979.

[CKS81] Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.

[Hag00] Christian Hagenah. *Gleichungen mit regulären Randbedingungen über freien Gruppen.* PhD thesis, Universität Stuttgart, Fakultät Informatik, 2000.

[KM06] Christian Kirkegaard and Anders Møller. Static analysis for Java Servlets and JSP. In *Proc. 13th International Static Analysis Symposium, SAS '06*, volume 4134 of *LNCS*, August 2006.

[Knu67] Donald E. Knuth. A characterization of parenthesis languages. *Information and Control*, 11(3):269–289, 1967.

[Law76] Eugene Lawler. *Combinatorial Optimization: Networks and Matroids*, chapter 3. 1976.

[McN67] Robert McNaughton. Parenthesis grammars. *Journal of the Association for Computing Machinery*, 14(3):490–500, 1967.

[MS72] A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Conf. Rec. 13th IEEE Symp. on Switching and Automata Theory*, pages 125–129, 1972.

[MT06] Yasuhiko Minamide and Akihiko Tozawa. XML validation for context-free grammars. In *Proc. of The Fourth ASIAN Symposium on Programming Languages and Systems*, volume 4279 of *LNCS*, pages 357–373, 2006.

[Pla94] Wojciech Plandowski. Testing equivalence of morphisms on context-free languages. In *Algorithms – ESA '94 (Utrecht)*, volume 855 of *LNCS*, pages 460–470. Springer, 1994.

[Sch06] Saul Schleimer. Polynomial-time word problems, 2006. `http://front.math.ucdavis.edu/math.GR/0608563`.

[Sei90] Helmut Seidl. Deciding equivalence of finite tree automata. *SIAM Journal on Computing*, 19(3):424–437, 1990.

[SM73] Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC*, pages 1–9, 1973.

[Tak75] Masako Takahashi. Generalizations of regular sets and their application to a study of context-free languages. *Information and Control*, 21(1):1–36, 1975.

Logical Reasoning for Higher-Order Functions with Local State

Nobuko Yoshida[1], Kohei Honda[2], and Martin Berger[1]

[1] Department of Computing, Imperial College, London
[2] Department of Computer Science, Queen Mary, University of London

Abstract. We introduce an extension of Hoare logic for call-by-value higher-order functions with ML-like local reference generation. Local references may be generated dynamically and exported outside their scope, may store higher-order functions and may be used to construct complex mutable data structures. This primitive is captured logically using a predicate asserting reachability of a reference name from a possibly higher-order datum and quantifiers over hidden references. The logic enjoys three completeness properties: relative completeness, a logical characterisation of the contextual congruence and derivability of characteristic formulae. The axioms for reachability and local invariants play a fundamental role in reasoning about non-trivial programs combining higher-order procedures and dynamically generated references.

1 Introduction

New reference generation, embodied for example in ML's `ref`-construct, is a highly expressive programming primitive. The key functionality of this construct is, firstly, to induce local state by generating a fresh reference inaccessible from the outside. Consider the following program:

$$\mathtt{Inc} \overset{\text{def}}{=} \mathtt{let}\ x = \mathtt{ref}(0)\ \mathtt{in}\ \lambda().(x := !x + 1;\ !x) \tag{1}$$

where "$\mathtt{ref}(M)$" returns a fresh reference whose content is the value which M evaluates to; "$!x$" means dereferencing the imperative variable x; and ";" is sequential composition. In (1), a reference with content 0 is newly created, but never exported to the outside. When the anonymous function in `Inc` is invoked, it increments the content of a local variable x, and returns the new content. The procedure returns a different result at each call, whose source is hidden from external observers. This is different from $\lambda().(x := !x + 1;\ !x)$ where x is globally accessible.

Secondly, local references may be exported outside of their original scope and be shared. Consider the following program from [25, § 6]:

$$\mathtt{incShared} \overset{\text{def}}{=} a := \mathtt{Inc}; b := !a; z_1 := (!a)(); z_2 := (!b)(); (!z_1 + !z_2) \tag{2}$$

This program returns 3. To understand the behaviour of `incShared`, we must capture the sharing of x between the procedures assigned to a and b. The scope of x is originally restricted to $!a$ but gets extruded to $!b$. If we replace $b := !a$ by $b := \mathtt{Inc}$, two separate instances of Inc are assigned to a and b, and the final result is 2. Controlling sharing by local reference is essential to writing concise algorithms that manipulate mutable data structures, but complicates formal reasoning, even for relatively small programs [8, 18].

Thirdly, through information hiding, local references can be used for efficient implementation of highly regular observable behaviour. The following program, taken from [25, § 1], called memFact, is a simple memoised factorial.

$$\mathtt{let}\, a = \mathtt{ref}(0)\;\; b = \mathtt{ref}(1)\;\mathtt{in}\;\lambda x.\mathtt{if}\; x = !a\;\mathtt{then}\;!b\;\mathtt{else}\;(a := x; b := \mathtt{fact}(x)\,;\,!b)$$

Here fact is the standard factorial function. To external observers, memFact behaves purely functionally. The program implements a simple case of memoisation: when memFact is called with a stored argument in a, it immediately returns the stored value $!b$ without calculation. If x differs from what is stored at a, the factorial fx is calculated and the new pair is stored. The reason why memFact is indistinguishable from the pure factorial function can be understood through the following *local invariant* [25]:

> *Throughout all possible invocations of* memFact*, the content of b is the factorial of the content of a.*

Such local invariants capture one of the basic patterns in programming with local state, and play a key role in preceding studies of operational reasoning about program equivalence in the presence of local state [15, 23, 25, 30].

As a further example of local invariants, this time involving mutually recursive stored functions, consider the following program:

$$\begin{aligned}\mathtt{mutualParity} \overset{\text{def}}{=} & \; x := \lambda n.\mathtt{if}\; n{=}0\;\mathtt{then}\;\mathtt{f}\;\mathtt{else}\;\mathtt{not}((!y)(n{-}1));\\ & \; y := \lambda n.\mathtt{if}\; n{=}0\;\mathtt{then}\;\mathtt{t}\;\mathtt{else}\;\mathtt{not}((!x)(n{-}1))\end{aligned}$$

After running mutualParity, the application $(!x)n$, returns true if n is odd, false if not, and $(!y)n$ acts dually. But since x and y are free, another program may prevent mutualParity from functioning correctly by inappropriate assignment to x or y. With local state, we can avoid unexpected interference at x and y.

$$\mathtt{safeOdd} \overset{\text{def}}{=} \mathtt{let}\; x = \mathtt{ref}(\lambda n.\mathtt{t})\; y = \mathtt{ref}(\lambda n.\mathtt{t})\;\mathtt{in}\;(\mathtt{mutualParity}; !x) \quad (3)$$
$$\mathtt{safeEven} \overset{\text{def}}{=} \mathtt{let}\; x = \mathtt{ref}(\lambda n.\mathtt{t})\; y = \mathtt{ref}(\lambda n.\mathtt{t})\;\mathtt{in}\;(\mathtt{mutualParity}; !y) \quad (4)$$

(Here $\lambda n.\mathtt{t}$ can be any initialising value.) Now that x, y are inaccessible, the programs behave like pure functions, e.g. safeOdd(3) always returns true without any side effects. In this case, the invariant says that *throughout all possible invocations,* $!x$ *is a procedure which checks if its argument is odd, provided y stores a procedure which does the dual, whereas* $!y$ *is a procedure which checks if its argument is even, whenever x stores a dual procedure*. Later we present general reasoning principles for local invariants which can verify these two and many other non-trivial examples [14, 15, 18, 23, 25].

This paper studies a Hoare logic for imperative higher-order functions with dynamic reference generation, a core part of ML-like languages. Our aim is to identify basic logical primitives needed to capture precisely the semantics of local state, on the basis of a stratification of logics for sequential higher-order functions in our preceding works [2, 10, 12, 13]. For this purpose we introduce two new logical primitives, one for reachability of references from an arbitrary datum and another for quantifying hidden references (§ 2.2). This leads to a simple proof system for reference generation, which can assert and derive desired properties for programs with significant use of local state from the literature [14, 15, 18, 23, 25] (§ 2.4). The status of these new logical primitives is clarified through soundness and three completeness results, including relative completeness (§ 3). Basic axioms for reachability, hiding and local invariants are studied in § 4. The local invariance axioms capture a common pattern in reasoning about local state, and enable us to verify the examples in [14, 15, 18, 23, 25], including programs discussed above (§5). Comparisons with related work are found in §6. Detailed derivations, large examples and proofs are found in the full version [1].

2 Assertions for Local State

2.1 A Programming Language

Our target programming language is call-by-value PCF with unit, sums, products and recursive types, augmented with imperative constructs. Let $x, y, \dots$ range over an infinite set of variables, and $\mathrm{X}, \mathrm{Y}, \dots$ over an infinite set of type variables. Then types, values and programs are given by:

$$\begin{aligned}
\alpha, \beta \ &::= \mathsf{Unit} \mid \mathsf{Bool} \mid \mathsf{Nat} \mid \alpha \Rightarrow \beta \mid \alpha \times \beta \mid \alpha + \beta \mid \mathsf{Ref}(\alpha) \mid \mathrm{X} \mid \mu\mathrm{X}.\alpha \\
V, W \ &::= \mathsf{c} \mid x^{\alpha} \mid \lambda x^{\alpha}.M \mid \mu f^{\alpha \Rightarrow \beta}.\lambda y^{\alpha}.M \mid \langle V, W \rangle \mid \mathtt{inj}_i^{\alpha+\beta}(V) \\
M, N \ &::= V \mid MN \mid M := N \mid \mathtt{ref}(M) \mid !M \mid \mathsf{op}(\tilde{M}) \mid \pi_i(M) \mid \langle M, N \rangle \mid \mathtt{inj}_i^{\alpha+\beta}(M) \\
&\quad \mid \ \mathtt{if}\, M \,\mathtt{then}\, M_1 \,\mathtt{else}\, M_2 \mid \mathtt{case}\, M \,\mathtt{of}\, \{\mathtt{in}_i(x_i^{\alpha_i}).M_i\}_{i \in \{1,2\}}
\end{aligned}$$

We use standard notation [22] like constants c (unit $()$, booleans $\mathtt{t}$, $\mathtt{f}$, numbers $\mathtt{n}$ and locations $l, l', \dots$) and first-order operations op ($+$, $-$, $\times$, $=$, $\neg$, $\wedge$, $\dots$). Locations only appear at runtime when references are generated. $\tilde{M}$ etc. denotes a vector and ε the empty vector. A program is *closed* if it has no free variables. We freely use shorthands like $M; N$, $\lambda().M$, and $\mathtt{let}\, x = M \,\mathtt{in}\, N$. Typing is standard: we take the equi-isomorphic approach [22] for recursive types. Nat, Bool and Unit are *base types*. We leave illustration of each construct to standard textbooks [22], except for reference generation $\mathtt{ref}(M)$, the focus of the present study, which behaves as: first M of type α is evaluated and becomes a value V; then a *fresh* reference of type $\mathsf{Ref}(\alpha)$ with initial content V is generated. This behaviour is formalised by the following reduction rule:

$$(\mathtt{ref}(V),\ \sigma) \longrightarrow (\nu\, l)(l,\ \sigma \uplus [l \mapsto V]) \qquad (l \text{ fresh})$$

Above σ is a store, a finite map from locations to closed values, denoting the initial state, whereas $\sigma \uplus [l \mapsto V]$ is the result of disjointly adding a pair (l, V) to σ. The resulting configuration uses a ν-binder, which denotes l fresh. The general form $(\nu\, \tilde{l})(M, \sigma)$ means $\tilde{l}$ (a vector of distinct locations) occur in M and σ (the order is irrelevant). We write (M, σ) for $(\nu\, \varepsilon)(M, \sigma)$. The one-step reduction $\longrightarrow$ over configurations is defined using standard rules [22] except for closure under ν-bindings. A *basis* $\Gamma; \Delta$ is a pair of finite maps, one from variables to non-reference types ($\Gamma, \Gamma', \ldots$), the other from locations and variables to reference types ($\Delta, \Delta', \ldots$). $\Theta, \Theta', \ldots$ combine two kinds of bases. The typing rules are standard. Sequents have form $\Gamma; \Delta \vdash M : \alpha$, to be read: M has type α under $\Gamma; \Delta$. We omit empty Γ or Δ . A store σ is typed under Δ, written $\Delta \vdash \sigma$, when, for each l in its domain, $\sigma(l)$ is a closed value which is typed α under Δ, where we assume $\Delta(l) = \mathsf{Ref}(\alpha)$. A configuration (M, σ) is *well-typed* if for some $\Gamma; \Delta$ and α we have $\Gamma; \Delta \vdash M : \alpha$ and $\Delta \vdash \sigma$. Standard type safety holds for well-typed configurations. *Henceforth we only consider well-typed programs and configurations.*

2.2 A Logical Language

The logical language is based on standard first-order logic with equality [17, § 2.8]. It extends the logic [2] with two new primitives. The grammar follows, letting $\star \in \{\wedge, \vee, \supset\}$, $\mathcal{Q} \in \{\exists, \forall, \nu, \overline{\nu}\}$ and $\mathcal{Q}' \in \{\exists, \forall\}$.

$$\begin{array}{lcl} e & ::= & x \mid \mathsf{c} \mid \mathsf{op}(\tilde{e}) \mid \langle e, e' \rangle \mid \pi_i(e) \mid \mathsf{inj}_i(e) \mid !e \\ C & ::= & e = e' \mid \neg C \mid C \star C' \mid \mathcal{Q}x^{\alpha}.C \mid \mathcal{Q}'\mathrm{X}.C \mid \{C\} e \bullet e' = x \{C\} \mid [!e]C \mid e \hookrightarrow e' \end{array}$$

The first grammar ($e, e', \ldots$) defines *terms*; the second *formulae* ($A, B, C, E, \ldots$). Terms include variables, constants c (unit $()$, numbers n, booleans t, f and locations $l, l', \ldots$), pairing, projection, injection and standard first-order operations. $!e$ denotes the dereference of a reference e. Formulae include standard logical connectives and first-order quantifiers [17], and following [2, 12], quantification over type variables.

Introduced in [13], $\{C\}\ e \bullet e' = x\ \{C'\}$ is the *evaluation formula*, which intuitively says: *If we apply a function e to an argument e′ starting from an initial state satisfying C, then it terminates with a resulting value (name it x) and a final state together satisfying C′*. We shall also use a refined form of evaluation formulae, introduced in §2.3. $[!e]C$ is *universal content quantification*, introduced in [2] for treating aliasing. $[!e]C$ (with e of a reference type) says: *Whatever value a program may store in a reference e, the assertion C continues to be valid.*

There are two new logical primitives. First, $\nu x.C$ (*for some hidden reference x, C holds*) and $\overline{\nu}x.C$ (*for each hidden reference x, C holds*) are *hiding-quantifiers* which quantify over reference variables, i.e. x above is of the form $\mathsf{Ref}(\beta)$. They range over hidden references, such as x generated by `Inc` in (1) in § 1. The need for adding these quantifiers is illustrated in §4.1, Proposition 12. The second new primitive is $e_1 \hookrightarrow e_2$ (with e_2 of a reference type), which is the *reachability predicate*. It says: *We can reach the reference denoted by e_2 from a datum denoted by e_1*. We then set its dual [6, 29] as $e \# e' \equiv \neg e' \hookrightarrow e$, which says: *One can never reach a reference e starting from a datum denoted by e′*. # is used for representing freshness of new references.

Convention. Logical connectives are used with standard precedence/association, using parentheses as necessary to resolve ambiguities. We use truth T (definable as $1 = 1$) and falsity F (which is $\neg\mathsf{T}$). $x \neq y$ stands for $\neg(x = y)$. $\mathsf{fv}(C)$ (resp. $\mathsf{fl}(C)$) denotes the set of free variables (resp. locations) in C. Note that x in $[!x]C$ occurs free, while in $\{C\} e \bullet e' = x \{C'\}$ x occurs bound with scope C'. We often write $[!x_1..x_n]C$ for $[!x_1]..[!x_n]C$. $C_1 \equiv C_2$ stands for $(C_1 \supset C_2) \wedge (C_2 \supset C_1)$. We write $\tilde{e} \# e$ for $\wedge_i e_i \# e$; $e \# \tilde{e}$ for $\wedge_i e \# e_i$; and $\tilde{e} \# \tilde{e}'$ for $\wedge_{ij} e_i \# e'_j$. Terms are typed starting from variables. A formula is well-typed if all occurring terms are well-typed. *Hereafter we assume all terms and formulae we use are well-typed.* Type annotations are often omitted.

2.3 Assertions for Local State

We explain assertions for programs with local state with examples.

1. The assertion $x = 6$ says x of type Nat is equal to 6. Assuming x has type $\mathsf{Ref}(\mathsf{Nat})$, $!x = 2$ means x stores 2. Consider $x := y; y := z; w := 1$. After its run, we can reach z by dereferencing y, and y by dereferencing x. Hence z is reachable from y, y from x, hence z from x. So the final state satisfies $x \hookrightarrow y \wedge y \hookrightarrow z \wedge x \hookrightarrow z$.
2. Next, assuming w is newly generated, we may wish to say w is *unreachable* from x, to ensure freshness of w. For this we assert $w\#x$, which, as noted, stands for $\neg(x \hookrightarrow w)$. $x\#y$ always implies $x \neq y$. Note that $x \hookrightarrow x$ and $x \hookrightarrow !x$ are tautologies whereas $x\#x \equiv \mathsf{F}$. But $!x \hookrightarrow x$ may or may not hold (since there may be a cycle between x's content and x in the presence of recursive types).
3. We consider reachability in procedures. Assume $\lambda().(x := 1)$ is named f_w and $\lambda().!x$, f_r. Since f_w can write to x, we have $f_w \hookrightarrow x$. Similarly $f_r \hookrightarrow x$. Next suppose $\mathtt{let}\ x = \mathtt{ref}(z)\ \mathtt{in}\ \lambda().x$ has name f_c and z's type is $\mathsf{Ref}(\mathsf{Nat})$. Then $f_c \hookrightarrow z$ (e.g. consider $!(f_c()) := 1$). However x is *not* reachable from $\lambda().((\lambda y.())(\lambda().x))$ since semantically it never touches x.
4. $\lambda().(x := !x + 1; !x)$ named u satisfies: $\forall i^{\mathsf{Nat}}.\{!x = i\}u \bullet () = z\{!x = z \wedge !x = i + 1\}$ saying: *invoking the function u increments the content of x and returns that content.*
5. We often wish to say that the write effects of an application are restricted to specific locations. The *located assertion* introduced in [2] is used for this purpose: $\{C\}e \bullet e' = x\{C'\}@\tilde{e}$ where each e_i is of a reference type and does not contain a dereference. $\tilde{e}$ is called *write set.* As an example: $\mathsf{inc}(u,x) \stackrel{\text{def}}{=} \forall i.\{!x = i\}u \bullet () = z\{z = !x = i + 1\}@x$ is satisfied by $\lambda().(x := !x + 1; !x)$ named u, saying this function, when invoked, only touches x.
6. Assuming u denotes the result of evaluating `Inc` in the Introduction, we can assert, using the existential hiding quantifier:

$$\nu x.(!x = 0\ \wedge\ \forall i^{\mathsf{Nat}}.\{!x = i\}u \bullet () = z\{z = !x\ \wedge\ !x = i + 1\}@x) \tag{5}$$

   which says: there is a hidden reference x storing 0 such that whenever u is invoked, it stores to x and returns the increment of the value in x at the time of invocation.
7. $\lambda n^{\mathsf{Nat}}.\mathtt{ref}(n)$, named u, meets the following specification. Let i, X be fresh.

$$\forall n^{\mathsf{Nat}}.\forall \mathrm{X}.\forall i^{\mathrm{X}}.\{\mathsf{T}\}u \bullet n = z\{\nu x.(!z = n\ \wedge\ z\#i \wedge z = x)\}@\emptyset. \tag{6}$$

   This says that u, when applied to n, will return a hidden reference z whose content is n and which is unreachable from any existing datum; and it has no writing effects to the existing state. Since i ranges over arbitrary data, unreachability of x from each such i indicates x is freshly generated and is not stored in any existing reference.

2.4 Proof Rules

Following Hoare [7], a judgement consists of a program and a pair of formulae, but augmented with a fresh name called an *anchor* [10, 12, 13].

$$\{C\}\, M :_u \{C'\}$$

The judgement is about total correctness and reads: *If we evaluate M in the initial state satisfying C, then it terminates with a value named by u and a final state, which together satisfy C'.* The same sequent is used for both validity and provability. If we wish to be specific, we prefix it with either $\vdash$ (for provability) or $\models$ (for validity). Let $\Gamma;\Delta$ be the minimum basis of M. In $\{C\}\ M :_u \{C'\}$, the name u is the *anchor* of the judgement, which should *not* be in $\mathsf{dom}(\Gamma,\Delta)\cup\mathsf{fv}(C)$; and C is the *pre-condition* and C' is the *post-condition*. The *primary names* are $\mathsf{dom}(\Gamma,\Delta)\cup\{u\}$, while the *auxiliary names* (ranged over by $i, j, k, \ldots$) are those free names in C and C' which are not primary. An anchor is used for naming the value from M and for specifying its behaviour.

The full compositional proof rules are given in Appendix A. Despite our semantic enrichment, all compositional proof rules in [2] syntactically stay as they are, except for adding the following rule for reference generation, with fresh i, X:

$$[\mathit{Ref}]\ \frac{\{C\}\, M :_m \{C'\}}{\{C\}\ \mathtt{ref}(M) :_u \{\nu x.(C'[!u/m]\wedge u\#i^{\mathrm{X}}\wedge u = x)\}}$$

In this rule, $u\#i$ indicates that the newly generated cell u is unreachable from any i of arbitrary type X in the initial state: then the result of evaluating M is stored in that cell.

Reachability is a stateful property: for this reason it is generally not invariant under state change. For example, suppose x is unreachable from y; after running $y := x$, x becomes reachable from y. Hence a rule such as "if $\{C\}\, M :_m \{C'\}$, then $\{C\wedge e\#e'\}\, M :_m \{C'\wedge e\#e'\}$" is unsound. However from the general invariance rule [*Inv*] from [2] below (on the left), which uses the located form of judgement $\{C\}\ M :_u \{C'\}@\tilde{e}$ (understood as located evaluation formulae), we can derive an invariance rule for $\#$, [*Inv*-#].

$$[\mathit{Inv}]\ \frac{\{C\}\, M :_m \{C'\}@\tilde{w}}{\{C\wedge[!\tilde{w}]C_0\}\, M :_m \{C'\wedge C_0\}@\tilde{w}} \qquad [\mathit{Inv}\text{-}\#]\ \frac{\{C\}\, M :_m \{C'\}@x \quad \text{no dereference occurs in } \tilde{e}}{\{C\wedge x\#\tilde{e}\}\, M :_m \{C'\wedge x\#\tilde{e}\}@x}$$

In [*Inv*], unlike the existing invariance rules as found in [28], we need no side condition "M does not modify stores mentioned in C_0": C and C_0 may even overlap in their mentioned references, and C does not have to mention all references M may read or write. For [*Inv*-#], we note $[!x]x\#\tilde{e}\equiv x\#\tilde{e}$ is always valid if $\tilde{e}$ contains no dereference $!e$, cf. Proposition 7 3-(5) later. The side condition is indispensable: consider $\{\mathsf{T}\}x := x\{\mathsf{T}\}@x$, which does not imply $\{x\#!x\}x := x\{x\#!x\}@x$.

3 Models, Soundness and Completeness

3.1 Models

We introduce the semantics of the logic based on term models. For capturing local state, models incorporate hidden locations using ν-binders [20]. For example, the Introduction's

`Inc`, named u, is modelled as: $(\nu l)(\{u : \lambda().(l := !l+1; !l)\}, \{l \mapsto 0\})$, which says that the appropriate behaviour at is at u, in addition to a hidden reference l storing 0.

Definition 1. An *open model of type* $\Theta = \Gamma;\Delta$, with $\mathsf{fv}(\Delta) = \emptyset$, is a tuple (ξ, σ) where:

- ξ, called *environment*, is a finite map from $\mathsf{dom}(\Theta)$ to closed values such that, for each $x \in \mathsf{dom}(\Gamma)$, $\xi(x)$ is typed as $\Theta(x)$ under Δ, i.e. $\Delta \vdash \xi(x) : \Theta(x)$.
- σ, called *store*, is a finite map from labels to closed values such that for each $l \in \mathsf{dom}(\sigma)$, if $\Delta(l)$ has type $\mathsf{Ref}(\alpha)$, then $\sigma(l)$ has type α under Δ, i.e. $\Delta \vdash \sigma(l) : \alpha$.

When Θ includes free type variables, ξ maps them to closed types, with the obvious corresponding typing constraints. A *model* of type $(\Gamma;\Delta)$ is a structure $(\nu\tilde{l})(\xi,\sigma)$ with (ξ,σ) being an open model of type $\Gamma;\Delta\cdot\Delta'$ with $\mathsf{dom}(\Delta') = \{\tilde{l}\}$. $(\nu\tilde{l})$ act as binders. $\mathcal{M}, \mathcal{M}', \ldots$ range over models.

An open model maps variables and locations to closed values: a model then specifies part of the locations as “hidden”. Since assertions in the present logic are intended to capture observable program behaviour, the semantics of the logic uses models quotiented by an observationally sound equivalence. Below $(\nu\tilde{l})(M,\sigma) \Downarrow$ means $(\nu\tilde{l})(M,\sigma) \longrightarrow^n (\nu\tilde{l}')(V,\sigma')$ for some n.

Definition 2. Assume $\mathcal{M}_i \stackrel{\text{def}}{=} (\nu\tilde{l}_i)(\tilde{x} : \tilde{V}_i, \sigma_i)$ typable under $\Gamma;\Delta$. Then we write $\mathcal{M}_1 \approx \mathcal{M}_2$ if the following clause holds for each closing typed context $C[\,\cdot\,]$ which is typable under Δ and in which no labels from $\tilde{l}_{1,2}$ occur: $(\nu\tilde{l}_1)(C[\langle\tilde{V}_1\rangle],\sigma_1) \Downarrow$ iff $(\nu\tilde{l}_2)(C[\langle\tilde{V}_2\rangle],\sigma_2) \Downarrow$ where $\langle\tilde{V}\rangle$ is the n-fold pairings of a vector of values.

3.2 Semantics of Reachability and Hiding

Let σ be a store and $S \subset \mathsf{dom}(\sigma)$. Then the *label closure of S in* σ, written $\mathsf{lc}(S,\sigma)$, is the minimum set S' of locations such that: (1) $S \subset S'$ and (2) If $l \in S'$ then $\mathsf{fl}(\sigma(l)) \subset S'$.

Lemma 3. *For all* σ*, we have:*

1. $S \subset \mathsf{lc}(S,\sigma)$*;* $S_1 \subset S_2$ *implies* $\mathsf{lc}(S_1,\sigma) \subset \mathsf{lc}(S_2,\sigma)$*; and* $\mathsf{lc}(S,\sigma) = \mathsf{lc}(\mathsf{lc}(S,\sigma),\sigma)$
2. $\mathsf{lc}(S_1,\sigma) \cup \mathsf{lc}(S_2,\sigma) = \mathsf{lc}(S_1 \cup S_2,\sigma)$
3. $S_1 \subset \mathsf{lc}(S_2,\sigma)$ *and* $S_2 \subset \mathsf{lc}(S_3,\sigma)$*, then* $S_1 \subset \mathsf{lc}(S_3,\sigma)$
4. *there exists* $\sigma' \subset \sigma$ *such that* $\mathsf{lc}(S,\sigma) = \mathsf{fl}(\sigma') = \mathsf{dom}(\sigma')$.

(1,2) are direct from the definition, and (3,4) follow from (1,2). Now set $\Gamma;\Delta \vdash e : \alpha$, $\Gamma;\Delta \vdash \mathcal{M}$ and $\mathcal{M} = (\xi,\sigma)$. The *interpretation of e under* $\mathcal{M}$, denoted $[\![e]\!]_{\xi,\sigma}$ is given by:

$$[\![x]\!]_{\xi,\sigma} = \xi(x) \quad [\![!e]\!]_{\xi,\sigma} = \sigma([\![e]\!]_{\xi,\sigma}) \quad [\![\mathsf{c}]\!]_{\xi,\sigma} = \mathsf{c} \quad [\![\mathsf{op}(\tilde{e})]\!]_{\xi,\sigma} = \mathsf{op}([\![\tilde{e}]\!]_{\xi,\sigma})$$

$$[\![\langle e,e'\rangle]\!]_{\xi,\sigma} = \langle [\![e]\!]_{\xi,\sigma}, [\![e']\!]_{\xi,\sigma}\rangle \quad [\![\pi_i(e)]\!]_{\xi,\sigma} = \pi_i([\![e]\!]_{\xi,\sigma}) \quad [\![\mathtt{inj}_i(e)]\!]_{\xi,\sigma} = \mathsf{inj}_i([\![e]\!]_{\xi,\sigma})$$

We now set the satisfaction of the reachability which says that the set of hereditarily reachable names from e_1 includes e_2 up to $\approx$.

$$\mathcal{M} \models e_1 \hookrightarrow e_2 \quad \text{if } [\![e_2]\!]_{\xi,\sigma} \in \mathsf{lc}(\mathsf{fl}([\![e_1]\!]_{\xi,\sigma}),\sigma) \text{ for each } (\nu\tilde{l})(\xi,\sigma) \approx \mathcal{M}$$

For the programs in § 2.3 (3), we can check $f_w \hookrightarrow x$, $f_r \hookrightarrow x$ and $f_c \hookrightarrow z$ hold under $f_w : \lambda().(x := 1)$, $f_r : \lambda().!x$, $f_c : \mathtt{let}\ x = \mathtt{ref}(z)\ \mathtt{in}\ \lambda().x$ (regardless of the store part).

The following characterisation of # is often useful for justifying axioms. Below $\sigma = \sigma_1 \uplus \sigma_2$ indicates that σ is the union of σ_1 and σ_2, assuming $\mathsf{dom}(\sigma_1) \cap \mathsf{dom}(\sigma_2) = \emptyset$.

Proposition 4 (partition). *$\mathcal{M} \models x\#u$ iff for some $\tilde{l}$, V, l and $\sigma_{1,2}$, we have $\mathcal{M} \approx (\nu\tilde{l})(\xi \cdot u : V \cdot x : l,\ \sigma_1 \uplus \sigma_2)$ such that $\mathsf{lc}(\mathsf{fl}(V), \sigma_1 \uplus \sigma_2) = \mathsf{fl}(\sigma_1) = \mathsf{dom}(\sigma_1)$ and $l \in \mathsf{dom}(\sigma_2)$.*

The characterisation says that if x is unreachable from u then, up to $\approx$, the store can be partitioned into one covering all reachable names from u and another containing x.

The existential hiding-quantifier has the following semantics.

$$\mathcal{M} \models \nu x.C \quad \text{if} \quad \exists \mathcal{M}'.((\nu l)\mathcal{M}' \approx \mathcal{M} \ \wedge\ \mathcal{M}'[x:l] \models C)$$

where l is fresh, i.e. $l \notin \mathsf{fl}(\mathcal{M})$ where $\mathsf{fl}(\mathcal{M})$ denotes free labels in $\mathcal{M}$. The notation $(\nu l)\mathcal{M}'$ denotes addition of the hiding of l to $\mathcal{M}'$, as well as indicating that l occurs free in $\mathcal{M}'$. $\mathcal{M}[x:l]$ adds $x:l$ to the environment part of $\mathcal{M}$. This says that x denotes a hidden reference, say l, and the result of taking it off from $\mathcal{M}$ satisfies C. $\overline{\nu}x.C$ is defined dually. As an example of satisfaction, let: $\mathcal{M} \stackrel{\text{def}}{=} (\nu l)(\{u : \lambda().(l := !l + 1; !l)\},\ \{l \mapsto 0\})$ then we have $\mathcal{M} \models \nu x.C$ with $C = (!x = 0 \ \wedge\ \forall i^{\mathsf{Nat}}.\{!x = i\}u \bullet () = z\{z = !x \wedge\ !x = i+1\})$ using the above definition. To see this, let $\mathcal{M}' \stackrel{\text{def}}{=} (\{u : \lambda().(l := !l + 1; !l)\},\ \{l \mapsto 0\})$ then we surely have $(\nu l)\mathcal{M}' = \mathcal{M}$ and $\mathcal{M}'[x:l] \models C$. Here $\mathcal{M}$ represents a situation where l is hidden and u denotes a function which increments and returns the content of l; whereas $\mathcal{M}'$ is the result of taking off this hiding, exposing the originally local state.

3.3 Soundness and Completeness

The definition of satisfiability $\mathcal{M} \models C$ for the remaining formulae is given in [1]. where logical connectives are interpreted classically and type variables are treated syntactically [12]. Let $\mathcal{M}$ be a model $(\nu\tilde{l})(\xi, \sigma)$ of type $\Gamma; \Delta$, and $\Gamma; \Delta \vdash M : \alpha$ with u fresh.

Then *validity* $\models \{C\}M :_u \{C'\}$ is given by $\forall \mathcal{M}.(\mathcal{M} \models C \ \supset\ (\mathcal{M}[u{:}M] \Downarrow \mathcal{M}' \ \wedge\ \mathcal{M}' \models C')$ with $\mathcal{M}$ including all variables in M, C and C' except u, where we write $\mathcal{M}[u{:}N] \Downarrow \mathcal{M}'$ when $(N\xi, \sigma) \Downarrow (\nu\tilde{l'})(V, \sigma')$ and $\mathcal{M}' = (\nu\tilde{l}\tilde{l'})(\xi \cdot u{:}V,\ \sigma')$.

Theorem 5 (soundness). *$\vdash \{C\}M :_u \{C'\}$ implies $\models \{C\}M :_u \{C'\}$.*

We next discuss the completeness properties of the logic. A strong completeness property is *descriptive completeness* studied in [11], which is provability of a characteristic assertion for each program (i.e. assertions characterising programs' behaviour). In [11], we have shown that, for our base logic, this property directly leads to two other completeness properties, *relative completeness* (which says that provability and validity of judgements coincide) and *observational completeness* (which says that validity precisely characterises the standard contextual equivalence).

The proof of descriptive completeness closely follows [11]. Relative and observational completeness are its direct consequences. Descriptive completeness is established for a refinement of the present logic where evaluation formulae and content quantification are decomposed into fine-grained operators [1]. For the space sake, we only state observational completeness, which we regard as a basic semantic property of the logic.

Write $\cong$ for the standard contextual congruence for programs [22]; further write $M_1 \cong_{\mathcal{L}} M_2$ to mean ($\models \{C\}M_1 :_u \{C'\}$ iff $\models \{C\}M_2 :_u \{C'\}$). We have:

Theorem 6 (observational completeness). *For each* $\Gamma;\Delta \vdash M_i : \alpha$ *(*$i = 1,2$*), we have* $M_1 \cong_{\mathcal{L}} M_2$ *iff* $M_1 \cong M_2$.

4 Axioms for Reachability, Hiding and Local Invariant

4.1 Basic Axioms for Reachability and Hiding

We start from the axioms for reachability. Note that our types include recursive types.

Proposition 7 (axioms for reachability). *The following assertions are valid.*

1. $(1)\, x \hookrightarrow x$; $(2)\, x \hookrightarrow y \wedge y \hookrightarrow z \supset x \hookrightarrow z$;
2. $(1)\, y\#x^{\alpha}$ *with* $\alpha \in \{\mathsf{Unit}, \mathsf{Nat}, \mathsf{Bool}\}$; $(2)\, x\#y \Rightarrow x \neq y$; $(3)\, x\#w \wedge w \hookrightarrow u \supset x\#u$.
3. $(1)\, \langle x_1, x_2 \rangle \hookrightarrow y \equiv x_1 \hookrightarrow y \vee x_2 \hookrightarrow y$; $(2)\, \mathtt{inj}_i(x) \hookrightarrow y \equiv x \hookrightarrow y$; $(3)\, x \hookrightarrow y^{\mathsf{Ref}(\alpha)} \supset x \hookrightarrow !y$; $(4)\, x^{\mathsf{Ref}(\alpha)} \hookrightarrow y \wedge x \neq y \supset !x \hookrightarrow y$; $(5)\, [!x]x\#y \equiv x\#y$.

The proofs use Lemma 3. 3-(5) says that altering the content of x does not affect reachability *to* x. Note $[!x]y\#x \equiv y\#x$ is not valid at all. 3-(5) was already used for deriving $[\mathit{Inv}\text{-}\#]$ in §2.4 (we cannot substitute $!x$ for y in $[!x]x\#y$ to avoid name capture).

Let us say α is *finite* if it does not contains an arrow type or a type variable. We say $e \hookrightarrow e'$ is *finite* if e has a finite type. Then by Proposition 7 2-(1) and 3:

Theorem 8 (elimination). *Suppose all reachability predicates in* C *are finite. Then there exists* C' *such that* $C \equiv C'$ *and no reachability predicate occurs in* C'.

A straightforward coinductive extension of the above axioms gives a complete axiomatisation with recursive types [1], but not function types. For analysing reachability, we define the following "one-step" reachability predicate. Below e_2 is of a reference type.

$$\mathcal{M} \models e_1 \rhd e_2 \quad \text{if } [\![e_2]\!]_{\xi,\sigma} \in \mathsf{fl}([\![e_1]\!]_{\xi,\sigma}) \text{ for each } (\nu\tilde{l})(\xi,\sigma) \approx \mathcal{M} \tag{7}$$

We can show $(\nu\tilde{l})(\xi,\sigma) \models x \rhd l'$ is equivalent to $l' \in \bigcap\{\mathsf{fl}(V) \mid V \cong \xi(x)\}$, (the latter says that l' is in the support [6, 24, 30] of f). We set: $x \rhd^1 y \equiv x \rhd y$; $x \rhd^{n+1} y \equiv \exists z.(x \rhd z \wedge !z \rhd^n y)$ $(n \geq 1)$. We also set $x \rhd^0 y \equiv x = y$. By definition:

Proposition 9. $x \hookrightarrow y \equiv \exists n.(x \rhd^n y) \equiv (x = y \vee x \rhd y \vee \exists z.(x \rhd z \wedge z \neq y \wedge z \hookrightarrow y))$.

Proposition 9, combined with Theorem 8, suggests that if we can clarify one-step reachability at function types then we will be able to clarify the reachability relation as a whole. Unfortunately this relation is inherently intractable.

Proposition 10. $(1)\, \mathcal{M} \models f^{\alpha \Rightarrow \beta} \rhd x$ *is undecidable.* $(2)\, \mathcal{M} \models f^{\alpha \Rightarrow \beta} \hookrightarrow x$ *is undecidable.*

The same result holds for call-by-value $\beta\eta$-equality. The result also implies that the validity of $\forall xf.(A \supset f \rhd x)$ is undecidable, since we can represent any PCFv-term as a formula using the method [11]. However Proposition 10 does not imply that we cannot obtain useful axioms for (un)reachability for function types. We discuss a collection of basic axioms in the following.

Proposition 11. *For an arbitrary C, the following is valid with i, X fresh:* $\{C \wedge x\#fy\tilde{w}\}f \bullet y = z\{C'\}@\tilde{w} \supset \forall \mathrm{X}, i^{\mathrm{X}}.\{C \wedge x\#fiy\tilde{w}\}f \bullet y = z\{C' \wedge x\#fiyz\tilde{w}\}@\tilde{w}$.

The above axiom says that if x is unreachable from f, y and $\tilde{w}$, then the application of f to y with the write set $\tilde{w}$ never exports x. Next we list basic axioms for hiding.

Proposition 12. (1) $C \supset \nu x.C$ *if* $x \notin \mathsf{fv}(C)$; $\nu x.C \equiv C$ *if* $x \notin \mathsf{fv}(C)$ *and no evaluation formula occurs in* C; (2) $\nu x.(C \wedge u = x) \equiv C \wedge \nu x.u = x$ *where* $x \notin \mathsf{fv}(C)$; *and* (3) $\nu x.(C_1 \vee C_2) \equiv (\nu x.C_1) \vee (\nu x.C_2)$; $\nu x.(C_1 \wedge C_2) \supset (\nu x.C_1) \wedge (\nu x.C_2)$

For (1), it is notable that we do *not* generally have $C \supset \nu x.C$. Neither $\nu x.C \supset C$ with $x \notin \mathsf{fv}(C)$ holds generally, see [1]. This shows that integrating these quantifiers into the standard $\forall$ and $\exists$-quantifiers let the latter lose their standard axioms, motivating the introduction of ν-operator. (2,3) list some of useful axioms for moving the scope of x.

4.2 Local Invariant

We now introduce an axiom for local invariants. Let us first consider a function which writes to a local reference of a base type. Even programs of this kind pose fundamental difficulties in reasoning, as shown in [18]. Take the following program:

$$\mathtt{compHide} \stackrel{\text{def}}{=} \mathtt{let}\, x = \mathtt{ref}(7)\ \mathtt{in}\ \lambda y.(y > !x) \tag{8}$$

The program behaves as a pure function $\lambda y.(y > 7)$. Clearly, the obvious local invariant $!x = 7$ is preserved. We demand this assertion to hold under arbitrary invocations of `compHide`: thus (naming the function u) we arrive at the following invariant:

$$C_0 \;=\; !x = 7 \;\wedge\; \forall y.\{!x = 7\}u \bullet y = z\{!x = 7\}@\emptyset \tag{9}$$

Assertion (9) says: (1) the invariant $!x = 7$ holds now; and that (2) once the invariant holds, it continues to hold for ever (note x can never be exported due to the type of y and z, so that only u will touch x). `compHide` is easily given the following judgement:

$$\{\mathsf{T}\}\mathtt{compHide} :_u \{\nu x.(x\#i^{\mathrm{X}} \;\wedge\; C_0 \;\wedge\; C_1)\} \qquad (i \text{ fresh}) \tag{10}$$

with $C_1 = \forall y.\{!x = 7\}u \bullet y = z\{z = (y > 7)\}@\emptyset$. Thus, noting C_0 is only about the content of x, we conclude C_0 continues to hold automatically. Hence we cancel C_0 together with x:

$$\{\mathsf{T}\}\mathtt{compHide} :_u \{\forall y.\{\mathsf{T}\}u \bullet y = z\{z = (y > 7)\}\} \tag{11}$$

which describes a purely functional behaviour. Below we stipulate the underlying reasoning principle as an axiom. Let y, z be fresh. For simplicity of presentation, we assume y has a base type.[1]

$$\mathsf{Inv}(u, C_0, \tilde{x}) \;=\; C_0 \;\wedge\; (\forall yi.\{C_0\}u \bullet y = z\{\mathsf{T}\} \supset \forall yi.\{C_0\}u \bullet y = z\{C_0 \;\wedge\; \tilde{x}\#z\}) \tag{12}$$

where we assume $C_0 \supset \tilde{x}\#i$. $\mathsf{Inv}(u, C_0, x)$ says that, first, currently C_0 holds; and that second if C_0 holds, then applying u to y results in, if it ever converges, C_0 again and the returned z is disjoint from $\tilde{x}$. Below we say C is *stateless* if $\mathcal{M} \models C$ and $\mathcal{M}[u : N] \Downarrow \mathcal{M}'$ imply $\mathcal{M}' \models C$ (its syntactical characterisation can be found in Appendix A).

[1] That is sufficient for all examples in this paper: The refinement allows arbitrary types [1].

Proposition 13 (axiom for information hiding). *Assume* $C_0 \supset \tilde{x}\#i$ *and* $[!\tilde{x}]C_0$ *is stateless. Suppose* i,m *are fresh,* $\{\tilde{x},\tilde{g}\} \cap (\mathsf{fv}(C,C') \cup \{\tilde{w}\}) = \emptyset$ *and* y *has a base type. Let* $E_1 = \mathsf{Inv}(u,C_0,\tilde{x}) \wedge \forall yi.\{C_0 \wedge [!\tilde{x}]C\}u\bullet y = z\{C'\}@\tilde{w}\tilde{x}$ *and* $E_2 = \forall y.\{C\}u\bullet y = z\{C'\}@\tilde{w}$. *Then the following assertion is valid.*

$$(\mathsf{AIH}) \quad \{E\}m\bullet() = u\{\nu\tilde{x}.\exists\tilde{g}.(E_1 \wedge E')\} \supset \{E\}m\bullet() = u\{E_2 \wedge E'\}$$

(AIH) is used with the consequence rule (Appendix A) to simplify from E_1 to E_2. Its validity is proved using Proposition 4. The axiom says: *if a function* u *with a fresh reference* x_i *is generated, and if it has a local invariant* C_0 *on the content of* x_i*, then we can cancel* C_0 *together with* x_i. The statelessness of $[!\tilde{x}]C_0$ ensures that satisfiability of C_0 is not affected by state change except at $\tilde{x}$; and $[!\tilde{x}]C$ says that whether C holds does not depend on $\tilde{x}$. Finally $\exists\tilde{g}$ in E_1 allows the invariant to contain free variables, extending applicability as we shall use in §5 for `safeEven`.

Coming back to `compHide`, we take C_0 to be $!x = 7 \wedge x\#i$, $\tilde{w}$ empty, both C and E' to be T and C' to be $z = (y > 7)$ in (AIH), to reach the desired assertion. [1] lists the axioms of the higher-order version of Proposition 11 and apply to the examples in [18].

5 Reasoning Examples

5.1 Shared Stored Function

This section presents concrete reasoning examples. We show the key ideas; the detailed derivations can be found in [1]. We first present a simple example of hiding-quantifiers and unreachability using `incShared` in (2) from § 1. We use a derived rule for the combination of "let" and new reference generation.

$$[\mathit{LetRef}]\ \frac{\{C\}\,M :_m \{C_0\} \quad \{C_0[!x/m] \wedge x\#\tilde{e}\}\,N :_u \{C'\} \quad x \notin \mathsf{fpn}(\tilde{e})}{\{C\}\,\mathtt{let}\,x = \mathtt{ref}(M)\ \mathtt{in}\,N :_u \{\nu x.C'\}}$$

where $\mathsf{fpn}(e)$ denotes the set of *free plain names* of e which are reference names in e that do not occur dereferenced, defined as: $\mathsf{fpn}(x) = \{x\}$, $\mathsf{fpn}(\mathsf{c}) = \mathsf{fpn}(!e) = \emptyset$, $\mathsf{fpn}(\langle e,e'\rangle) = \mathsf{fpn}(e) \cup \mathsf{fpn}(e')$, and $\mathsf{fpn}(\pi_i(e)) = \mathsf{fpn}(\mathsf{inj}_i(e)) = \mathsf{fpn}(e)$. We also restrict C' above to a *thin formula* given in Appendix A (this does not limit the usability of this rule, at least for the reasoning examples we shall treat). The notation $x\#\tilde{e}$ appeared in § 2.3. The rule reads: *Assume* (1) M *with pre-condition* C *leads to post-condition* C_0*, with the resulting value named* m*; and* (2) *running* N *from* C_0 *with* m *as the content of* x *together with the assumption* x *is unreachable from each* e_i*, leads to* C' *with the resulting value named* u*. Then running* $\mathtt{let}\ x = \mathsf{Ref}(M)\ \mathtt{in}$ N *from* C *leads to* C' *whose* x *is fresh and hidden.* The side condition $x \notin \mathsf{fpn}(e_i)$ is essential for consistency (e.g. without it, we could assume $x\#x$, i.e. F). The rule directly gives a proof rule for new reference declaration [18, 23, 28], $\mathtt{new}\,x := M\ \mathtt{in}\,N$, which has the same operational behaviour as $\mathtt{let}\ x = \mathtt{ref}(M)\ \mathtt{in}\,N$. Note also that the original Hoare and Wirth [9]'s rule for local variable declaration is a special case of this rule.

Let $\mathsf{inc}(x,u,n) = \forall j.\{!x = j\}u\bullet() = j+1\{!x = j+1\}@x \wedge !x = n$ and $\mathsf{inc}'(n,m) = \mathsf{inc}(!a,x,n) \wedge \mathsf{inc}(!b,y,m) \wedge x \neq y$. The left derivation is for `incShared`, while that on the right is for a program where "$b := !a$" has been replaced by "$b :=$ `Inc`" in `incShared`. We assume and use pairwise distinctness of a, b, z_1, z_2, and omit anchors of unit type.

$1.\{\mathsf{T}\}\ a := \texttt{Inc}\ \{\nu x.\mathsf{inc}(!a,x,0)\}$	$\{\mathsf{T}\}\ a := \texttt{Inc}\ \{\nu x.\mathsf{inc}(!a,x,0)\}$
$2.\{\mathsf{inc}(!a,x,0)\}\ b := !a\ \{\mathsf{inc}(!a,x,0) \wedge \mathsf{inc}(!b,x,0)\}$	$\{\mathsf{inc}(!a,x,0)\}\ b := \texttt{Inc}\ \{\nu y.\mathsf{inc}'(0,0)\}$
$3.\{\mathsf{inc}(!a,x,0)\}\ z_1 := (!a)()\ \{\mathsf{inc}(!a,x,1) \wedge !z_1 = 1\}$	$\{\mathsf{inc}'(0,0)\}\ z_1 := ..\ \{\mathsf{inc}'(1,0) \wedge !z_1 = 1\}$
$4.\{\mathsf{inc}(!b,x,1)\}\ z_2 := (!b)()\ \{\mathsf{inc}(!b,x,2) \wedge !z_2 = 2\}$	$\{\mathsf{inc}'(1,0)\}\ z_2 := ..\ \{\mathsf{inc}'(1,1) \wedge !z_2 = 1\}$
$5.\{!z_1 = 1 \wedge !z_2 = 2\}\ (!z_1) + (!z_2) :_u \{u = 3\}$	$\{!z_1 = 1 \wedge !z_2 = 1\}\ (!z_1) + (!z_2) :_u \{u = 2\}$

Line 1 uses [*LetRef*]. In Line 2 on the left, x is automatically shared after "$b := !a$" which leads to scope extrusion, while in the right, $x \neq y$ in $\mathsf{inc}'(0,0)$ is ensured by the ν-binding operator.

Memoised Factorial [25] (from `memFact` in § 1)**.** Our target assertion specifies the behaviour of a pure factorial. The following inference starts from the `let`-body of `memFact`, which we name V. We set: $E_{1a} = C_0 \wedge \forall xi.\{C_0\}u \bullet x = y\{C_0 \wedge ab\#y\}@ab$, and $E_{1b} = \forall xi.\{C_0 \wedge C\}u \bullet x = y\{C'\}@ab$ where we set C_0 to be $ab\#i \wedge !b = (!a)!$, C to be T, and C' to be $y = x!$. Note that $[!ab]C_0$ is stateless by Prop. 7 5; and that, by the type of y being Nat and Prop. 7 2-(1), we have $ab\#y \equiv \mathsf{T}$. We can now reason:

$1.\{\mathsf{T}\}\ V :_u \{\forall xi.\{C_0\}u \bullet x = y\{C_0 \wedge C'\}\}@\emptyset$	
$2.\{ab\#i\}\ V :_u \{E_{1a} \wedge E_{1b}\}$	(1, Conseq, Inv-#)
$3.\{\mathsf{T}\}$ `memFact` $:_u \{\nu ab.(E_{1a} \wedge E_{1b})\}$	(2, LetRef)
$4.\{\mathsf{T}\}$ `memFact` $:_u \{\forall x.\{\mathsf{T}\}u \bullet x = y\{y = x!\}@\emptyset.\}$	(3,(AIH),Conseq)

Line 2 uses the axiom $\{C\}f \bullet x = y\{C_1 \wedge C_2\}@\tilde{w} \supset \wedge_{i=1,2}\{C\}f \bullet x = y\{C_i\}@\tilde{w}$.

5.2 Mutually Recursive Stored Functions

(from (3) in § 1). We first verify [1]:

$$\{\mathsf{T}\}\texttt{mutualParity} :_u \{\exists gh.IsOddEven(gh, !x!y, xy, n)\} \tag{13}$$

where, with $Even(n) \equiv \exists x.(n = 2 \times x)$ and $Odd(n) \equiv Even(n+1)$:

$$IsOddEven(gh, wu, xy, n) = (IsOdd(w, gh, n, xy) \wedge IsEven(u, gh, n, xy) \wedge !x = g \wedge !y = h)$$
$$IsOdd(u, gh, n, xy) = \forall n.\{!x = g \wedge !y = h\}u \bullet n = z\{z = Odd(n) \wedge !x = g \wedge !y = h\}@xy$$

where $IsOdd(u, gh, n, xy)$ says that *x stores a procedure which checks if its argument is odd if y stores a procedure which does the dual, and x does store the behaviour.* $IsEven(u, gh, n, xy)$ is defined dually. Our aim is to derive the following judgement for `safeOdd` starting from (13) (the case for `safeEven` is symmetric).

$$\{\mathsf{T}\}\texttt{safeOdd} :_u \{\forall n.\{\mathsf{T}\}u \bullet n = z\{z = Odd(n)\}@\emptyset\} \tag{14}$$

We first identify the local invariant: $C_0 = !x = g \wedge !y = h \wedge IsEven(h, gh, n, xy) \wedge xy\#ij$. Since C_0 only talks about g, h and the content of x and y, we know $[!xy]C_0$ is stateless. We now observe $IsOddEven(gh, !x!y, xy, n)$ is the conjunction of:

$$Odd_a = C_0 \ \wedge\ \forall n.\{C_0\}u \bullet n = z\{C_0\}@xy \quad Odd_b = \forall n.\{C_0\}u \bullet n = z\{z = Odd(n)\}@xy$$

As Line 3 in `memFact`, we can apply (AIH) to obtain (14).

Higher-Order Invariant [30, p.104]. We move to a program whose invariant behaviour depends on another function. The program instruments an original program with a simple profiling (counting the number of invocations), with α a base type.

$$\texttt{profile} \quad \stackrel{\text{def}}{=} \quad \texttt{let}\, x = \texttt{ref}(0)\ \texttt{in}\, \lambda y^{\alpha}.(x := !x + 1; fy)$$

Since x is never exposed, this program should behave precisely as f. We shall derive:

$$\{\forall y.\{C\}f \bullet y = z\{C'\}@\tilde{w}\}\ \texttt{profile} :_u \{\forall y.\{C\}u \bullet y = z\{C'\}@\tilde{w}\} \qquad (15)$$

with $x \notin \mathsf{fv}(C, C')$ (by the bound name condition). This judgement says: *if f satisfies the specification $E = \forall y.\{C\}f \bullet y = z\{C'\}@\tilde{w}$, then* `profile` *satisfies the same specification E*. Note C and C' are arbitrary. To derive (15), we first set C_0, the invariant, to be $x\#fi\tilde{w}$. As with the previous derivations, we use two subderivations. First, by the axiom in Proposition 11, we can derive:

$$\{\mathsf{T}\}\lambda y.(x := !x + 1; fy) :_u \{\forall yi.\{C_0\}u \bullet y = z\{C_0 \wedge x\#z\}@x\tilde{w}\} \qquad (16)$$

Secondly, again by Prop. 11 we obtain $E \ \supset\ \forall y.\{C \ \wedge\ x\#f\tilde{w}\}f \bullet y = z\{x\#z\tilde{w}\}@\tilde{w}$. By this, E being stateless, Prop.7 3-(5) and [*Inv*-#], we obtain:

$$\{E\}\lambda y.(x := !x + 1; fy) :_u \{\forall yi.\{C_0 \wedge [!x]C\}u \bullet y = z\{C' \wedge x\#z\}@x\tilde{w}\}. \qquad (17)$$

By combining (16) and (17), we can use (AIH), hence done.

6 Related Work and Future Topics

For the sake of space, detailed comparisons with existing program logics and reasoning methods, in particular with Clarke's impossibility result, Caires-Cardelli's spatial logic, recent mechanisations of reachability predicates [16], as well as other logics such as LCF, Dynamic logic, higher-order logic, specification logic, Larch/ML, and Extended ML are left to the long version [1] and our past papers [2, 10, 12, 13]. Below we focus on work that treats locality and recent work on Hoare logics.

Reasoning Principles for Functions with Local State. There is a long tradition of studying equivalences over higher-order programs with local state. Meyer and Sieber [18] present examples and reasoning principles based on denotational semantics. Mason, Talcott and others [14] investigate equational axioms for an untyped version of the language treated in the present paper, including local invariance. Pitts and Stark [23, 25, 30] present powerful operational reasoning principles for the same ML-fragment considered here, including reasoning principle for local invariance at higher-order types [25]. Our axioms for information hiding in § 4, which capture a basic pattern of programming with local state, are closely related with these reasoning principles. Our logic differs in that its aim is to offer a method for describing and validating properties of programs beyond program equivalence. Equational and logical approaches are complimentary: Theorem 6 offers a basis for integration. For example, we may consider deriving a property of the optimised version M' of M: if we can easily verify $\{C\}M :_u \{C'\}$ and if we know $M \cong M'$, we can conclude $\{C\}M' :_u \{C'\}$, which is useful if M is better structured than M'.

Program Logics for Aliasing and Higher-Order Functions. Reynolds et al. [28] present a program logic for aliasing where fresh data generation is represented by a special conjunction denoting spatial disjointness from the original datum. Their method can reason many programs with aliasing. The logic studied in the present paper captures freshness through generic unreachability from arbitrary data in the initial state. Apart from completeness properties discussed in §3.3, the approach enables uniform treatment of known data types, including product, sum, reference, closure, etc. Reasoning examples using the present method include those in the present paper as well as higher-order invariants from [18], objects from [15], circular lists from [16], tree-, dag- and graph-copy from [5], as presented in [1, § 6]. Birkedal et al. [4] present a typing system for a variant of Idealised Algol where types are constructed from formulae of the logic in [28]. Their typing system uses subtyping calculated via categorical semantics, the focus of their study. [3] extends the logic in [28] with higher-order frame rules, and demonstrates reasoning about priority queues. Both works consider neither exportable fresh reference generation nor higher-order procedures in full generality. In particular, it would be difficult to validate the examples in § 5.

Nanevski et al [21] studies Hoare Type Theory (HTT) which combines dependent types and Hoare triples with anchors based on monadic understanding of computation. HTT aims to provide an effective general framework which unifies standard static checking techniques and logical verifications. Local store is not treated and left as an open problem in [21]. Reus and Streicher [27] present a Hoare logic for a simple language with higher-order stored procedures, extended in [26]. Soundness is proved with denotational methods. Completeness is not considered in [26, 27]. Their assertions contain quoted programs, which is necessary to handle recursion via stored functions. Their language does not allow procedure parameters and general reference creation.

The logic studied in the present work aims to capture the behaviour of sequential higher-order programs with local state in the framework of compositional program logics *à* la Hoare, stratified on the basis of simpler program logics [2, 10, 12, 13]. The semantic precision of the logic (cf. Theorem 6), axiomatisation of local invariance, and uniform extensibility to diverse data types are among those features not found in the preceding program logics mentioned above.

Meta-Logical Study on Freshness. Freshness of names has recently been studied from the viewpoint of formalising binding relations in programming languages and computational calculi. Pitts and Gabbay [6, 24] extend first-order logic with constructs to reason about freshness of names based on permutations. The key syntactic additions are the (interdefinable) “fresh” quantifier И and the freshness predicate #, mediated by a swapping (finite permutation) predicate. Miller and Tiu [19] are motivated by the significance of generic (or eigen-) variables and quantifiers at the level of both formulae and sequents, and split universal quantification in two, introduce a self-dual freshness quantifier ∇ and develop the corresponding sequent calculus of Generic Judgements. While these logics are not program logics, their logical machinery may be usable in the present context. As noted in Proposition 9, reasoning about $\hookrightarrow$ or $\#$ is tantamount to reasoning about $\rhd$, which denotes the support (i.e. semantically free locations) of a datum. A characterisation of support by the swapping operation may be interesting from the viewpoint of axiomatisation of reachability.

Acknowledgement. We thank Andrew Pitts and Ian Stark for communications on an early version of this paper. The mutual recursion example is due to Bernhard Reus. This work is partially supported by EPSRC GR/T04236, GR/S55545, GR/S55538, GR/T04724, GR/T03208, GR/T03258 and IST-2005-015905 MOBIUS.

References

1. A full version of this paper. http://www.doc.ic.ac.uk/˜yoshida/local.
2. M. Berger, K. Honda, and N. Yoshida. A logical analysis of aliasing for higher-order imperative functions. In *ICFP'05*, pages 280–293, 2005.
3. B. Biering, L. Birkedal, and N. Torp-Smith. Bi hyperdoctrines and higher-order separation logic. In *ESOP'05*, LNCS, pages 233–247, 2005.
4. L. Birkedal, N. Torp-Smith, and H. Yang. Semantics of separation-logic typing and higher-order frame rules. In *LICS'05*, pages 260–269, 2005.
5. R. Bornat, C. Calcagno, and P. O'Hearn. Local reasoning, separation and aliasing. In *Workshop SPACE*, 2004.
6. M. Gabbay and A. Pitts. A New Approach to Abstract Syntax Involving Binders. In *Proc. LICS '99*, pages 214–224, 1999.
7. C. A. R. Hoare. An axiomatic basis of computer programming. *CACM*, 12, 1969.
8. C. A. R. Hoare. Proof of correctness of data representations. *Acta Inf.*, 1:271–281, 1972.
9. C. A. R. Hoare and N. Wirth. Axiomatic semantics of Pascal. *Toplas*, 1(2):226–244, 1979.
10. K. Honda. From process logic to program logic. In *ICFP'04*, pages 163–174. ACM, 2004.
11. K. Honda, M. Berger, and N. Yoshida. Descriptive and relative completeness for logics for higher-order functions. In *ICALP'06*, volume 4052 of *LNCS*, pages 360–371, 2006.
12. K. Honda and N. Yoshida. A compositional logic for polymorphic higher-order functions. In *PPDP'04*, pages 191–202. ACM, 2004.
13. K. Honda, N. Yoshida, and M. Berger. An observationally complete program logic for imperative higher-order functions. In *LICS'05*, pages 270–279, 2005. Full version is at [1].
14. F. Honsell, I. A. Mason, S. F. Smith, and C. L. Talcott. A variable typed logic of effects. *Inf. Comput.*, 119(1):55–90, 1995.
15. V. Koutavas and M. Wand. Small bisimulations for reasoning about higher-order imperative programs. In *Proc. POPL*, 2006.
16. S. K. Lahiri and S. Qadeer. Verifying properties of well-founded linked lists. In *POPL'06*, pages 115–126. ACM, 2006.
17. E. Mendelson. *Introduction to Mathematical Logic*. Wadsworth Inc., 1987.
18. A. R. Meyer and K. Sieber. Towards fully abstract semantics for local variables. In *POPL'88*, pages 191 – 203, 1988.
19. D. Miller and A. Tiu. A proof theory for generic judgments. *ACM Transactions on Computational Logic*, 6(4):749–783, 2005.
20. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts I and II. *Info. & Comp.*, 100(1):1–77, 1992.

21. A. Nanevski, G. Morrisett, and L. Birkedal. Polymorphism and separation in Hoare type theory. In *ICFP06*, pages 62–73. ACM Press, 2006.
22. B. C. Pierce. *Types and Programming Languages*. MIT Press, 2002.
23. A. M. Pitts. Reasoning about local variables with operationally-based logical relations. In *Algol-Like Languages*, volume 2, chapter 17, pages 173–193. Birkhauser, 1997.
24. A. M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186:165–193, 2003.
25. A. M. Pitts and I. D. B. Stark. Operational reasoning for functions with local state. In *Higher Order Operational Techniques in Semantics*, pages 227–273. CUP, 1998.
26. B. Reus and J. Schwinghammer. Separation logic for higher-order store. In *Proc. CSL*, volume 4207 of *LNCS*, pages 575–590, 2006.
27. B. Reus and T. Streicher. About Hoare logics for higher-order store. In *ICALP*, volume 3580 of *LNCS*, pages 1337–1348, 2005.
28. J. C. Reynolds. Separation logic: a logic for shared mutable data structures. In *LICS'02*.
29. J. C. Reynolds. Idealized Algol and its specification logic. In *Tools and Notions for Program Construction*, 1982.
30. I. Stark. *Names and Higher-Order Functions*. PhD thesis, University of Cambridge, 1994.

A Appendix: Proof Rules

The following presents compositional proof rules. We omit the rules for the sum and products. The rule for the reference can be found in the main section.

$$[\mathit{Var}]\,\frac{-}{\{C[x/u]\}\;x:_u\{C\}}\qquad [\mathit{Const}]\,\frac{-}{\{C[\mathtt{c}/u]\}\;\mathtt{c}:_u\{C\}}\qquad [\mathit{Succ}]\,\frac{\{C\}\,M:_m\{C'[m+1/u]\}}{\{C\}\,\mathit{Succ}(M)\;:_u\{C'\}}$$

$$[\mathit{Abs}]\,\frac{\{C\wedge A^{\text{-}x\tilde{i}}\}\,M:_m\{C'\}}{\{A\}\,\lambda x.M:_u\{\forall x\tilde{i}.\{C\}\,u\bullet x=m\{C'\}\}}\qquad [\mathit{App}]\,\frac{\begin{array}{c}\{C\}\,M:_m\{C_0\}\\ \{C_0\}\,N:_n\{\,C_1\wedge\{C_1\}\,m\bullet n=u\,\{C'\}\,\}\end{array}}{\{C\}\,MN:_u\{C'\}}$$

$$[\mathit{If}]\,\frac{\{C\}\,M:_b\{C_0\}\quad \{C_0[\mathsf{t}/b]\}\,M_1:_u\{C'\}\quad \{C_0[\mathsf{f}/b]\}\,M_2:_u\{C'\}}{\{C\}\;\mathtt{if}\;M\;\mathtt{then}\;M_1\;\mathtt{else}\;M_2:_u\{C'\}}$$

$$[\mathit{Deref}]\,\frac{\{C\}\,M:_m\{C'[!m/u]\}}{\{C\}\;!M:_u\{C'\}}\qquad [\mathit{Assign}]\,\frac{\{C\}\,M:_m\{C_0\}\quad \{C_0\}\,N:_n\{C'\{\!|n/\,!m|\!\}\}}{\{C\}\,M:=N\,\{C'\}}$$

$$[\mathit{Rec}]\,\frac{\{A^{\text{-}xi}\wedge\forall j\lneq i.B(j)[x/u]\}\,\lambda y.M:_u\{B(i)^{\text{-}x}\}}{\{A\}\,\mu x.\lambda y.M:_u\{\forall i.B(i)\}}$$

$$[\mathit{Cons\text{-}Eval}]\,\frac{\{C_0\}\,M:_m\{C_0'\}\quad \forall\tilde{i}.\{C_0\}x\bullet()=m\{C_0'\}\supset\forall\tilde{i}.\{C\}x\bullet()=m\{C'\}\;x\text{ fresh, }\tilde{i}\text{ auxiliary}}{\{C\}\,M:_m\{C'\}}$$

We assume that judgements are well-typed in the sense that, in $\{C\}\;M:_u\{C'\}$ with $\Gamma;\Delta\vdash M:\alpha$, $\Gamma,\Delta,\Theta\vdash C$ and $u\!:\!\alpha,\Gamma,\Delta,\Theta\vdash C'$ for some Θ s.t. $\mathsf{dom}(\Theta)\cap(\mathsf{dom}(\Gamma,\Delta)\cup\{u\})=\emptyset$. In the rules, $C^{\text{-}\tilde{x}}$ indicates $\mathsf{fv}(C)\cap\{\tilde{x}\}=\emptyset$. Symbols $i,j,\ldots$ range over auxiliary names. We demand the postconditions of the proof rules [*App, If*] to be *thin*, where we say C is thin iff for each $\mathcal{M}$ and for each $y\in\mathsf{fv}(\mathcal{M})\backslash\mathsf{fv}(C)$, $\mathcal{M}\models C$ implies $\mathcal{M}/y\models C$ (a syntactic characterisation of thinness is discussed in [1]).

In [*Abs, Rec*], A,B denote *stateless* formulae given in §4.2.Syntactically C is stateless when: (1) each dereference $!y$ only occurs either in pre/post conditions of evaluation formulae or under $[!y]$; (2) (un)reachability predicates occur in pre/post conditions of evaluation formulae; and (3) evaluation formulae and content quantifications never occur negatively (using the standard notion of negative/positive occurrences).

[*Assign*] uses *logical substitution* $C\{\!|e_2/!e_1|\!\}$ which is built with content quantification to represent substitution of content of a possibly aliased reference [2]. This is defined as: $C\{\!|e_2/!e_1|\!\} \overset{\text{def}}{=} \forall m.(m = e_2 \supset [!e_1](!e_1 = m \supset C))$. with m fresh. Intuitively $C\{\!|e_2/!e_1|\!\}$ describes the situation where a model satisfying C is updated at a memory cell referred to by e_1 (of a reference type) with a value e_2 (of its content type), with $e_{1,2}$ interpreted in the current model. [*Cons-Eval*] is a strengthened version of the standard consequence rule [*Conseq*].

The proof rules for the located judgement is given just as [2], adding the following rule for the reference, with i, X fresh.

$$[\mathit{Ref}]\ \frac{\{C\}\, M :_m \{C'\}@\tilde{e} \quad x \notin \mathsf{fpn}(\tilde{e}) \cup \mathsf{fv}(\tilde{e})}{\{C\}\, \mathtt{ref}(M) :_u \{\nu x.(u\#i^{\mathrm{X}} \wedge u = x \wedge C')\}@\tilde{e}}$$

Author Index