LNCS 5504

Luca de Alfaro (Ed.)

Foundations of Software Science and Computational Structures

12th International Conference, FOSSACS 2009
Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009
York, UK, March 2009, Proceedings

European Joint Conferences on Theory And Practice of Software 2009

Springer

Luca de Alfaro (Ed.)

Foundations of Software Science and Computational Structures

12th International Conference, FOSSACS 2009
Held as Part of the Joint European Conferences
on Theory and Practice of Software, ETAPS 2009
York, UK, March 22-29, 2009
Proceedings

Springer

Volume Editor

Luca de Alfaro
University of California
School of Engineering
Dept. of Computer Engineering
1156 High Street MS: SOE3, Santa Cruz, CA 95064, USA
E-mail: luca@soe.ucsc.edu

Library of Congress Control Number: 2009922287

CR Subject Classification (1998): F.3, F.1, F.4, D.3, D.2

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-642-00595-0 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-00595-4 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12632343 06/3180 5 4 3 2 1 0

Foreword

ETAPS 2009 was the 12th instance of the European Joint Conferences on Theory and Practice of Software. ETAPS is an annual federated conference that was established in 1998 by combining a number of existing and new conferences. This year it comprised five conferences (CC, ESOP, FASE, FOSSACS, TACAS), 22 satellite workshops (ACCAT, ARSPA-WITS, Bytecode, COCV, COMPASS, FESCA, FInCo, FORMED, GaLoP, GT-VMT, HFL, LDTA, MBT, MLQA, OpenCert, PLACES, QAPL, RC, SafeCert, TAASN, TERMGRAPH, and WING), four tutorials, and seven invited lectures (excluding those that were specific to the satellite events). The five main conferences received 532 submissions (including 30 tool demonstration papers), 141 of which were accepted (10 tool demos), giving an overall acceptance rate of about 26%, with most of the conferences at around 25%. Congratulations therefore to all the authors who made it to the final programme! I hope that most of the other authors will still have found a way of participating in this exciting event, and that you will all continue submitting to ETAPS and contributing towards making it the best conference on software science and engineering.

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.

ETAPS is a 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 synchronised 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 2009 was organised by the University of York 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)

and with support from ERCIM, Microsoft Research, Rolls-Royce, Transitive, and Yorkshire Forward.

The organising team comprised:

Chair	Gerald Luettgen
Secretariat	Ginny Wilson and Bob French
Finances	Alan Wood
Satellite Events	Jeremy Jacob and Simon O'Keefe
Publicity	Colin Runciman and Richard Paige
Website	Fiona Polack and Malihe Tabatabaie.

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

Vladimiro Sassone (Southampton, Chair), Luca de Alfaro (Santa Cruz), Roberto Amadio (Paris), Giuseppe Castagna (Paris), Marsha Chechik (Toronto), Sophia Drossopoulou (London), Hartmut Ehrig (Berlin), Javier Esparza (Munich), Jose Fiadeiro (Leicester), Andrew Gordon (MSR Cambridge), Rajiv Gupta (Arizona), Chris Hankin (London), Laurie Hendren (McGill), Mike Hinchey (NASA Goddard), Paola Inverardi (L'Aquila), Joost-Pieter Katoen (Aachen), Paul Klint (Amsterdam), Stefan Kowalewski (Aachen), Shriram Krishnamurthi (Brown), Kim Larsen (Aalborg), Gerald Luettgen (York), Rupak Majumdar (Los Angeles), Tiziana Margaria (Göttingen), Ugo Montanari (Pisa), Oege de Moor (Oxford), Luke Ong (Oxford), Catuscia Palamidessi (Paris), George Papadopoulos (Cyprus), Anna Philippou (Cyprus), David Rosenblum (London), Don Sannella (Edinburgh), João Saraiva (Minho), Michael Schwartzbach (Aarhus), Perdita Stevens (Edinburgh), Gabriel Taentzer (Marburg), Dániel Varró (Budapest), and Martin Wirsing (Munich).

I would like to express my sincere gratitude to all of these people and organisations, the Programme Committee Chairs and PC members of the ETAPS conferences, the organisers 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 Organising Chair of ETAPS 2009, Gerald Luettgen, for arranging for us to hold ETAPS in the most beautiful city of York.

January 2009

Vladimiro Sassone, Chair
ETAPS Steering Committee

Preface

The present volume contains the proceedings of the 12th International Conference on the Foundations of Software Science and Computations Structures (FOSSACS) 2009, held in York, UK, March 22-25, 2009. FOSSACS is an event of the Joint European Conferences on Theory and Practice of Software (ETAPS). The previous 11 FOSSACS conferences took place in Lisbon (1998), Amsterdam (1999), Berlin (2000), Genoa (2001), Grenoble (2002),Warsaw (2003), Barcelona (2004), Edinburgh (2005), Vienna (2006), Braga (2007), and Budapest (2008).

FOSSACS presents original papers on foundational research with a clear significance to software science. The Programme Committee (PC) invited submissions on theories and methods to support analysis, synthesis, transformation and verification of programs and software systems.

This proceedings volume contains 30 regular papers, as well as the abstracts of two invited talks. The first invited talk, "Temporal Reasoning About Program Executions", was by Rajeev Alur, the FOSSACS 2009 invited speaker. The second invited talk, "Facets of Synthesis: Revisiting Church's Problem", was by Wolfgang Thomas, one of the two unifying ETAPS 2009 invited speakers.

We received 122 abstracts and 102 full paper submissions; of these, 30 were selected for presentation at FOSSACS and inclusion in the proceedings. The selection process was performed by the PC in the course of a two-week electronic meeting. The PC members, and the external experts they consulted, wrote a total of over 400 paper reviews, and the discussion phase of the meeting involved several hundred messages. I was very impressed by the work done by the PC members, and by the external experts, both before and during the PC meeting. The competition was particularly fierce, and many good papers could not be accepted.

We sincerely thank all the authors of papers submitted to FOSSACS 2009. Moreover, we would like to thank the members of the PC for their excellent job during the selection process, as well as all the sub-reviewers for the expert help and reviews they provided. Through the phases of submission, evaluation, and production of the proceedings we relied on the invaluable assistance of the EasyChair system; we are very grateful to its developer Andrei Voronkov, and to all its maintainers. Last but not least, we would like to thank the ETAPS 2009 Organizing Committee chaired by Gerald Luettgen and the ETAPS Steering Committee chaired by Vladimiro Sassone for their efficient coordination of all the activities leading up to FOSSACS 2009.

January 2009 Luca de Alfaro

Conference Organization

Programme Chair

Luca de Alfaro

Programme Committee

Parosh Abdulla	Uppsala University
Roberto Amadio	Paris Diderot University
Jos Baeten	Eindhoven University of Technology
Luís Caires	New University of Lisbon
Luca de Alfaro (Chair)	UC Santa Cruz and Google
Javier Esparza	Technical University of Munich
Kousha Etessami	University of Edinburgh
Marcelo Fiore	University of Cambridge
Cédric Fournet	Microsoft Research
Dan Ghica	University of Birmingham
Radha Jagadeesan	DePaul University
Alan Jeffrey	Bell Labs
Marcin Jurdziński	University of Warwick
Naoki Kobayashi	Tohoku University
Barbara König	University of Duisburg-Essen
Ugo Montanari	University of Pisa
Catuscia Palamidessi	INRIA and École Polytechnique
Prakash Panangaden	McGill University
Amir Pnueli	New York University
Jean-François Raskin	University of Brussels (ULB)
Grigore Rosu	University of Illinois Urbana-Champaign
Davide Sangiorgi	University of Bologna
Carolyn Talcott	SRI International

External Reviewers

Suzana Andova
Jesus Aranda
Eugene Asarin
Philippe Audebaud
Gergei Bana
Massimo Bartoletti
Josh Berdine
Martin Berger
Nathalie Bertrand
Karthikeyan Bhargavan
Stefano Bistarelli
Simon Bliudze
Filippo Bonchi
Michele Boreale
Richard Bornat
Artur Boronat

Patricia Bouyer
Tomas Brazdil
Franck van Breugel
Thomas Brihaye
Sander H.J. Bruggink
Roberto Bruni
Mikkel Bundgaard
Marzia Buscemi
Zining Cao
Felice Cardone
Krishnendu Chatterjee
Konstantinos Chatzikokolakis
Feng Chen
James Cheney
Yannick Chevalier
Pierre Clairambault
Hubert Comon-Lundh
Brendan Cordy
Ricardo Corin
Andrea Corradini
Véronique Cortier
Pieter Cuijpers
Deepak D'Souza
Arnaud da Costa
Vincent Danos
Stéphanie Delaune
Stéphane Demri
Josée Desharnais
Dino Distefano
Laurent Doyen
Steven Eker
Constantin Enea
Zoltan Esik
Marco Faella
John Fearnley
Jerome Feret
Gianluigi Ferrari
Carla Ferreira
Andrzej Filinski
Emmanuel Filiot
Riccardo Focardi
Wan Fokkink
Luca Fossati
Sibylle Froeschle
David de Frutos-Escrig
Andrew Gacek
Fabio Gadducci
Pierre Ganty
Simon Gay
Gilles Geeraerts
Raffaela Gentilini
Sonja Georgievska
Giorgio Ghelli
Sergio Giro
Gregor Gößler
Jens Chr. Godskesen
Andy Gordon
Alexey Gotsman
Jean Goubault-Larrecq
Davide Grohmann
Serge Haddad
Esfandiar Haghverdi
Jerry den Hartog
Masahito Hasegawa
Tobias Heindel
Lukas Holik
Hans Hüttel
Atsushi Igarashi
Kazuhiro Inaba
Yoshinao Isobe
Neil Jones
Raman Kazhamiakin
Bruce Kapron
Delia Kesner
Jeroen Ketema
Stefan Kiefer
Vijay Anand Korthikanti
Pavel Krcal
Jörg Kreiker
Steve Kremer
Jean Krivine
Ralf Kuesters
Alexander Kurz
Salvatore La Torre
James Laird
Yassine Lakhnech
Ivan Lanese
Sławomir Lasota
Luciano Lavagno
Ranko Lazić

Jean-Jacques Levy
Alberto Lluch Lafuente
Christof Loeding
Michele Loreti
Dorel Lucanu
Denis Lugiez
Michael Luttenberger
Bas Luttik
Ian Mackie
Pasquale Malacaria
Louis Mandel
Giulio Manzonetto
Nicolas Markey
Jasen Markovski
Ian Mason
Thierry Massart
Richard Mayr
Guy McCusker
Annabelle McIver
Massimo Merro
Antoine Meyer
Marino Miculan
Dale Miller
Yasuhiko Minamide
Larry Moss
Till Mossakowski
Mohammad Reza Mousavi
Gopalan Nadathur
Uwe Nestmann
Zhaozhong Ni
Koki Nishizawa
Gethin Norman
Luke Ong
Simona Orzan
Luca Padovani
Miguel Palomino
Jochen Pfalzgraf
Andrew Phillips
Sylvan Pinsky
Nir Piterman
Andrei Popescu
Rosario Pugliese
Shaz Qadeer
Paola Quaglia
António Ravara
Michel Reniers
Tamara Rezk
Noam Rinetzky
Eike Ritter
Michal Rutkowski
Sylvain Salvati
Alexis Saurin
Lutz Schröder
Stefan Schwoon
Traian Serbanuta
Olivier Serre
Natalia Sidorova
Jakob Grue Simonsen
Alex Simpson
Anu Singh
Scott Smith
Pawel Sobocinski
Ian Stark
Kohei Suenaga
Eijiro Sumii
Andrzej Tarlecki
David Teller
Paul van Tilburg
Andrea Turrini
Nikos Tzevelekos
Frank Valencia
Hugo Vieira
Maria Grazia Vigliotti
Erik de Vink
Tomáš Vojnar
Oskar Wibling
Thomas Wilke
Hongseok Yang
Wang Yi
Anna Zaks
Eugen Zalinescu
Marina Zanella
Hans Zantema
Gianluigi Zavattaro
Sami Zhioua
Wieslaw Ziełonka

Table of Contents

Facets of Synthesis: Revisiting Church's Problem

Wolfgang Thomas

RWTH Aachen University, Lehrstuhl Informatik 7, Aachen, Germany
thomas@informatik.rwth-aachen.de

Abstract. In this essay we discuss the origin, central results, and some perspectives of algorithmic synthesis of nonterminating reactive programs. We recall the fundamental questions raised more than 50 years ago in "Church's Synthesis Problem" that led to the foundation of the algorithmic theory of infinite games. We outline the methodology developed in more recent years for solving such games and address related automata theoretic problems that are still unresolved.

1 Prologue

The objective of "synthesis" is to pass from a specification to a program realizing it. This is a central task in computer science, and – unfortunately or fortunately, depending on the point of view – its solution cannot in general be automatized. However, there are classes of specifications for which algorithmic synthesis is possible. In the present paper we deal with a fundamental class of this kind, the regular specifications for nonterminating reactive programs. In another terminology, this amounts to the solution of infinite two-person games where the winning condition is given by a regular ω-language. The central problem on these games (Church's Problem [4,5]) and the first solutions (Büchi and Landweber [3], Rabin [22]) date back decades by now. Starting from this pioneering work, the algorithmic theory of infinite games grew into a very active area of research, and a fascinating landscape of models and algorithmic solutions is developing, covering, for example, timed games, weighted games, games over infinite state spaces, distributed games, stochastic games, concurrent games, and multiplayer games. Rather than trying an overview of these areas we return in this paper to the fundamentals. In an informal style addressing a general audience[1], we discuss three issues. First, we recall the problem, its connection with infinite games, and explain some of its historical context in mathematics (more precisely, set theory). Then the main techniques for solving infinite games are outlined. Finally, we discuss some questions around the concept of uniformization that are directly connected with the original problem and seem worth further study.

[1] We assume, however, that the reader knows the basic concepts about automata and logics over infinite words, like Büchi automata and monadic second-order logic over the structure $(\mathbb{N}, +1)$; see [13] as a reference.

2 Church's Problem

An instance of the synthesis problem is the description of a desired relation $R(X, Y)$ between inputs X (say, from a domain D) and outputs Y (say, from a domain D'). From the description of R we want to construct a program $\mathcal{P}$, computing a function $f_{\mathcal{P}} : D \to D'$ such that

$$\forall X \in D \; R(X, f_{\mathcal{P}}(X))$$

At the "Summer Institute of Symbolic Logic" at Cornell in 1957, Alonzo Church posed the following problem in [4] (see also [5]):

> *Given a requirement which a circuit is to satisfy, we may suppose the requirement expressed in some suitable logistic system which is an extension of restricted recursive arithmetic. The* synthesis problem *is then to find recursion equivalences representing a circuit that satisfies the given requirement (or alternatively, to determine that there is no such circuit).*

Church considers the case that the input X is an infinite sequence $X_0X_1\ldots$ of data (of a finite domain, we consider just bits in this paper) from which *in an on-line mode* the output sequence $Y = Y_0Y_1\ldots$ (again consisting of bits) has to be generated. The desired program is here a "circuit" (which amounts to a finite automaton); it has to produce the output bit Y_i from the input bits $X_0 \ldots X_i$.

The relation $R(X, Y)$ between inputs and outputs can be identified with an ω-language L_R. We associate with each pair $(X, Y) = (X_0X_1X_2\ldots, Y_0Y_1Y_2\ldots)$ of sequences the ω-word

$$X^\wedge Y := X_0Y_0X_1Y_1X_2Y_2\ldots$$

and let $L_R = \{X^\wedge Y | R(X, Y)\}$. A relation R is called *regular* if the associated ω-language L_R is regular, i.e. definable by a Büchi automaton or by a monadic second-order formula (MSO-formula) $\varphi(X, Y)$ over the structure $(\mathbb{N}, +1)$. As solutions we use finite automata with output, more precisely Mealy automata. Over a finite state-space S and with input alphabet Σ_1 and output alphabet Σ_2, a Mealy automaton is equipped with a transition function $\sigma : S \times \Sigma_1 \to S$ and an output function $\tau : S \times \Sigma_1 \to \Sigma_2$; in our case we set $\Sigma_1 = \Sigma_2 = \{0, 1\}$. Given such an automaton $\mathcal{A}$, the definition of the computed function $f_{\mathcal{A}} : \{0, 1\}^\omega \to \{0, 1\}^\omega$ is obvious.

Let us state Church's Problem in precise words for this case of MSO-definable relations and programs in the format of Mealy automata. (Other cases will be addressed later in this paper.)

> *Church's Problem:* Given an MSO-formula $\varphi(X, Y)$, defining the relation $R_\varphi \subseteq \{0, 1\}^\omega \times \{0, 1\}^\omega$, decide whether there is a Mealy automaton $\mathcal{A}$ such that $\forall X R_\varphi(X, f_{\mathcal{A}}(X))$ holds and – if yes – construct such a Mealy automaton from φ.

A dozen years later, Büchi and Landweber [3] solved this problem; Rabin [22] provided independently an alternative solution. The result established an ideal scenario within computer science: For the MSO-definable specifications, we can algorithmically check realizability, and we can in this case algorithmically construct a program (more precisely, an automaton) realizing the specification. In the context of construction of reactive systems, we view the Mealy automaton provided by a solution as a *finite-state controller* that behaves correctly for any behaviour of the environment as represented by input sequences. Since many interesting specifications are in fact regular, this result has great potential for practical applications. (One has to admit, however, that in order to realize this potential more work is necessary regarding the computational complexity of the algorithms involved.) Before we sketch the main ideas that enter the solution of Church's Problem, we introduce a simple connection of the problem with the theory of infinite games and underlying motivations from set theory.

3 Infinite Games, Determinacy, and Set Theory

A treatment of Church's Problem in the framework of infinite games was first proposed by McNaughton in an unpublished technical report [17], based on work of Gale and Stewart [12]. With each ω-language $L \subseteq \{0,1\}^\omega$ one associates an infinite game $G(L)$. (If L is regular, we say that the game $G(L)$ is regular.) It is played between two players. Due to the symmetry between them, we prefer to name them Player 1 and Player 2 rather than "Input" and "Output". The players choose bits in alternation (where Player 1 starts), forming a sequence $\varrho = X_0Y_0X_1Y_1X_2Y_2\dots$ which is called "play" in this context. Player 2 wins the play ϱ if $\varrho \in L$, otherwise Player 1 wins ϱ. A *strategy* for Player 2 is a function $f : \{0,1\}^+ \to \{0,1\}$, mapping a finite sequence $X_0X_1\dots X_i$ to the next chosen bit Y_i. A strategy f induces canonically a function $f_\omega : \{0,1\}^\omega \to \{0,1\}^\omega$ mapping a sequence $X_0X_1\dots$ to a sequence $Y_0Y_1\dots$ If for each sequence $X_0X_1\dots$ the play $X_0Y_0X_1Y_1\dots$ resulting from applying f belongs to L, f is called winning strategy for Player 2. So winning strategies capture the type of function that are asked for in Church's Problem (disregarding however the aspect of finite-state computability). Analogously, one introduces strategies and winning strategies g for Player 1. Now we have $g : Y_0\dots Y_{i-1} \mapsto X_i$, so $g : \{0,1\}^* \to \{0,1\}$. We say that g is a winning strategy g for Player 1 if for each $Y_0Y_1\dots$ the sequence $X_0Y_0X_1Y_1\dots$ obtained by applying g does not belong to L.

By this formulation one introduces some symmetry into the model, which is not present in the formulation as given by Church. In the mathematical theory of infinite games this symmetry is a dominating aspect. The main question in the theory is the following problem: Given L, can we show that either Player 1 or Player 2 has a winning strategy? In this case one calls the game $G(L)$ *determined*. Determinacy is the dichotomy

$$\exists \text{ strategy } f\ \forall X \quad X^\wedge f_\omega(X) \in L \quad \vee \quad \exists \text{ strategy } g\ \forall Y \quad g_\omega(Y)^\wedge Y \notin L$$

Determinacy may at first seem a superfluous aspect in the study of Church's Problem. It looks useless to know that a specification – say modeling the relation between a program and its environment – can either be realized by the program or that the complement of it can be realized by the environment. However, there are several good reasons to address determinacy. In this section we discuss a first such motivation from set theory (and the reader not interested in historical connections can skip this discussion). Later we treat also other uses of determinacy.

Given an ω-language L, we consider a slightly modified game $G^*(L)$, since in this case the connection to set theory comes in very naturally. Here Player 1 picks *finite sequences* of bits (rather than single bits) when it is his turn, whereas Player 2 stays with picking single bits. We write $G^{**}(L)$ if both players pick finite sequences of bits in each move (this is the so-called Banach-Mazur game).

Let us focus on $G^*(L)$. Intuition tells us that for a "very large" set L it will be much easier to guarantee a winning strategy for Player 2 than for a "very small" set L. Now the question of comparing sizes of infinite sets is at the heart of Cantor's set theory. In fact, Cantor conjectured in his "Continuum Hypothesis" CH a dichotomy: that each set $L \subseteq \{0,1\}^\omega$ is either "very small" (and he meant "being at most countable" by this) or "very large" (meaning that its cardinality is the same as that of the full space $\{0,1\}^\omega$). Often, CH is formulated as the claim that between the cardinalities $|\mathbb{N}|$ and $|\mathbb{R}|$ there are no further cardinalities; since $|\{0,1\}^\omega| = |\mathbb{R}|$, this claim is equivalent to the one formulated above.

Cantor could not show this dichotomy – and today we know that CH is indeed independent of the standard set theoretical assumptions as formulated in the axiom system ZFC. However, the so-called Cantor-Bendixson Theorem shows that CH is true at least for all "closed" sets L. A set $L \subseteq \{0,1\}^\omega$ is *closed* if one can infer $X \in L$ from the condition that infinitely many finite prefixes w of X have an extension wZ in L. One calls L *open* if its complement $\overline{L}$ is closed.

We now state the Cantor-Bendixson Theorem and see that it amounts to a game-theoretic dichotomy, namely that for an open set L the game $G^*(L)$ is determined. The statement of the theorem refers to a geometric view of the space $\{0,1\}^\omega$, in the sense that we consider $\{0,1\}^\omega$ as the set of all infinite paths through the binary tree T_2 (where 0 means "branch left" and 1 means "branch right").

A pattern of paths as indicated in Figure 1 below we call *tree copy*; it is given by infinitely many branching points (bullets in the figure), such that from each of the two sons (circles in the figure) of a branching point, a finite path to a next branching point exists. We say that a set $L \subseteq \{0,1\}^\omega$ *allows a tree copy* if

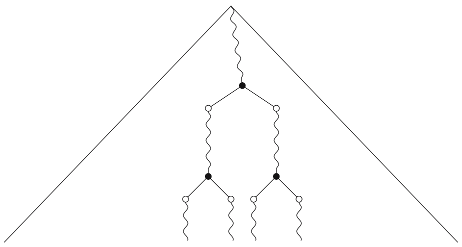


Fig. 1. A tree copy

there is a tree copy such that each path passing through infinitely many of its branching points belongs to L. Obviously a set L that allows a tree copy must have the same cardinality as $\{0,1\}^\omega$. Using this simplifying terminology (instead of the official one involving "perfect sets"), the Cantor-Bendixson Theorem says: *Each closed set L is either at most countable or allows a tree copy.*

Let us infer that for open K, the game $G^*(K)$ is determined. By the Cantor-Bendixson Theorem we know that $L := \overline{K}$ is either countable or allows a tree copy. In the *first case*, Player 2 can apply a diagonalization method: He uses an enumeration $Z_0, Z_1, \ldots$ of L and chooses his i-th bit in a way to make the resulting play different from Z_i. Then the play will be outside L and hence in K, so this is a winning strategy for Player 2 in $G^*(K)$. In the *second case*, Player 1 refers to the tree copy of L and chooses his bit sequences in a way to remain inside this tree copy (he always moves "to the next branching point"). Then the play will be in L and hence outside K, so this is a winning strategy for Player 1 in $G^*(K)$.

For the games $G(L)$, determinacy is not so easily connected with cardinalities of sets. Nevertheless, topological notions (such as "open" and "closed") are the key for showing determinacy results. The fundamental result, due to Martin, rests on the notion of Borel set: An ω-language is *Borel* if it can be built from open and closed sets in an at most countable (possibly transfinite) sequence of steps, where each single step consists of a countable union or a countable intersection of already constructed sets. Martin's Theorem now says that for a Borel set L, the game $G(L)$ is determined (see e.g. [16]). As will be explained later, all regular games are Borel and hence determined.

For the Banach-Mazur games $G^{**}(L)$, nice connections can again be drawn to concepts of richness of sets (like "co-meager"); a recent in-depth analysis of the determinacy problem is Grädel's work [11].

4 Solving Infinite Games

In this section we give a very brief sketch of the techniques that enter (known) solutions of Church's Problem, using game-theoretic terminology. For a detailed development see e.g. the tutorial [25].

4.1 From Logic to Games on Graphs

The start is a conversion of the originally logical problem into an automata theoretic one, by a transformation of the given MSO-formula φ into an equivalent ω-automaton. Here we apply the well-known results due to Büchi [2] and McNaughton [18] that allow to convert an MSO-formula into an equivalent (nondeterministic) Büchi automaton, and then a Büchi automaton into an equivalent (deterministic) Muller automaton. [2] A Büchi automaton has a designated set F of final states; and a run is accepting if some state of F occurs infinitely often in

[2] It should be noted that in the early days of automata theory, the conversion of regular expressions or logical formulas into automata was called "synthesis" and the converse "analysis" (see e.g. the introction of [1]).

it. The run of a Muller automaton with finite state set Q is accepting if the set of states visited infinitely often in it belongs to a predefined collection $\mathcal{F} \subseteq 2^Q$ of state sets.

We can easily build this automaton in a way that the processing of letters X_i contributed by Player 1 is separated from the processing of letters Y_i contributed by Player 2, in the sense that the state set Q is partitioned into two sets Q_1 (from where bits of Player 1 are read) and Q_2 (from where bits of Player 2 are read). Thus a run of the automaton switches back and forth between visits to Q_1- and to Q_2-states. Since the acceptance of a play by the automaton refers only to the visited states, we may drop the input- and output-letters for our further analysis and identify a play with a state sequence through the automaton. The resulting structure is also called *game arena* or *game graph*: We imagine that the two Players 1 and 2 build up a path in alternation – Player 1 picks a transition from a state in Q_1, similarly Player 2 from a state in Q_2. The winning condition (for Player 2) is now no more a logic formula but a very simple requirement, namely that the states visited infinitely often in a play form a set in the predefined collection $\mathcal{F}$. Referring to this "Muller winning condition", we speak of a *Muller game* over a finite graph G.

For solving Church's Problem, it is now sufficient to decide whether for plays beginning in the start state of the graph, Player 2 has a winning strategy, and in this case to construct such a strategy in the form of a Mealy automaton. Due to the deletion of the transition labels, the Mealy automaton now maps a finite play prefix from Q^* to a "next state" from Q. For a vertex v the play ϱ starting from v is won by Player 2 iff

$$\text{for some } F \in \mathcal{F} : \exists^\omega i \ \varrho(i) \in F \wedge \neg \exists^\omega i \ \varrho(i) \notin F$$

(here $\exists^\omega$ is the quantifier "there exist infinitely many"). From this form of the winning condition it is easy to see that the set L_v of plays won by Player 2 from v is Borel, and in fact a Boolean combination of countable intersections of open sets. Hence the Muller game over G with start vertex v is determined.

We add some methodological remarks.

1. The main effect of this transformation is the *radical simplification of the winning condition* from a possibly complex logical formula to the requirement that a play visits certain states infinitely often and others only finitely often. This simplification is made possible by distinguishing finitely many different "game positions" in the form of the automaton states. As mentioned, we can infer that a game as presented in Church's Problem is determined. The cost to be payed for this simplicity is the high number of states; it is known that in the length of (MSO-) formulas this number grows at a rate that cannot be bounded by an elementary function.
2. It should be noted that *all known solutions of Church's Problem involve this reduction from logic to automata (or graphs).* In model-checking, similar remarks apply when the linear-time logic LTL is considered; on the other hand, CTL-model-checking proceeds directly by an induction on formulas, which is one reason for its efficiency. It would be interesting to know logics of

substantial expressive power that allow a similar approach for the solution of games.

3. The introduction of game graphs has another advantage: In modelling reactive systems, it is usually convenient to describe the interaction between a controller and its environment by a game graph, adding a specification in the form of a winning condition (then as an ω-language over the respective state set Q). In practice this winning condition may not be a Muller condition but again an MSO-formula or LTL-formula φ, defining an ω-language $L \subset Q^\omega$. In this case one can also apply the above-mentioned transformation of φ into a Muller automaton (call its state set S), obtaining a game graph over the vertex set $S \times Q$, now together with a Muller condition (this condition just refers to the S-components of visited states).

4.2 Parity Games

The direct solution of Muller games (as presented in the difficult original paper [3]) is rather involved. It is helpful to pass to a different kind of game first, called *parity game*, and then solve this parity game. As for the Muller winning condition, also the parity winning condition is a Boolean combination of statements that certain states are visited infinitely often. But instead of a collection $\mathcal{F}$ of state sets, a uniform coloring of vertices by a finite list of colors is used. We take here natural numbers as colors. A play ϱ is won by Player 2 iff the highest color visited infinitely often during ϱ is even. This amounts to the disjunction over the following statements for all even $i \leq k$: Color i is visited infinitely often but each color $j > i$ only finitely often. This winning condition was proposed first by Mostowski [19], and it has a precursor in the "difference hierarchy" in Hausdorff's *Grundzüge der Mengenlehre* [14], introduced there to structure the levels of the Borel hierarchy (see also [24]).

Technically, one reduces Muller games to parity games in the following sense: For a game graph G and a collection $\mathcal{F}$, defining a Muller winning condition, one constructs a new graph G' with an appropriate coloring c such that each play ϱ in G induces a corresponding play ϱ' in G' with the following property: Player 2 wins ϱ under the Muller condition w.r.t. $\mathcal{F}$ iff Player 2 wins ϱ' under the parity condition w.r.t. c. In the standard construction of G', using the "latest appearance record", one introduces memory about the visited vertices of G in the order of their last visits. Thus a vertex of G' is basically a permutation of vertices in G (the leading entry being the current vertex of G); so the transformation $G \mapsto G'$ involves a serious blow-up. However, the solution of parity games is much easier than that of Muller games. In particular, *memoryless winning strategies* suffice for the respective winner; i.e. strategies that do not involve memory on the past of a play but just depend on the respective currently visited vertex.

As a preparation for the solution of parity games, one considers a very simple winning condition, called *reachability condition.* We are given a set F of "target vertices", and Player 2 wins the play ϱ if a vertex from F occurs in ϱ. To solve a reachability game means to compute those vertices v from which Player 2 has a strategy to force a visit in F.

This problem has a straightforward solution: Over the (finite) vertex set Q, one computes, for $i = 0, 1, 2, \ldots$, the set $A_i^2(F)$ of those states from which Player 2 can guarantee a visit in F within i steps. Obviously $A_0^2(F) = F$, and we have $v \in A_{i+1}^2(F)$ iff one of the two following cases holds:

– $v \in Q_2$ and one edge connects v with a vertex in $A_i^2(F)$,
– $v \in Q_1$ and each edge from v leads to a vertex in $A_i^2(F)$.

We have $A_0^2(F) \subseteq A_1^2(F) \subseteq A_2^2(F) \ldots$ and set $A^2(F) = \bigcup_i A_i^2(F)$; this set is obtained at some stage i_0. Clearly from each vertex in $A^2(F)$, Player 2 has a winning strategy by taking edges which decrease the distance to F with each step (and it is also easy to see that a memoryless strategy suffices). To show that the construction is complete, we verify that from each vertex outside $A^2(F)$ the other player (1) has a winning strategy. Thus we show that reachability games are *determined*. This use of determinacy is a general method: To show completeness of a game solution one verifies a determinacy claim.

How is this done for a reachability game? From the construction of the $A_i^2(F)$ it is clear that for each v in the complement of $A^2(F)$ one of the following cases holds:

– $v \in Q_2 \setminus A^2(F)$ and hence no edge leads from v to some $A_i^2(F)$; so each edge from v leads to a vertex again outside $A^2(F)$,
– $v \in Q_1 \setminus A^2(F)$ and hence not each edge leads from v to some $A_i^2(F)$; so some edge from v leads to a vertex outside $A^2(F)$.

Clearly this yields a strategy for Player 1 to avoid visiting $A^2(F)$ from outside $A^2(F)$, and hence to avoid visiting F; so we have a winning strategy for Player 1.

The set $A^2(F)$ as constructed above for the set F is often called the "attractor of F (for Player 2)". As it turns out, an intelligent iteration of this construction basically suffices also for the solution of parity games. The determinacy claim even holds without the assumption that the game graph is finite: *In a parity game over the game graph G, for each vertex v one of the two players has a memoryless winning strategy for the plays starting from v. If G is finite, one can decide who wins and compute a memoryless winning strategy* ([7]).

The idea to show this is by induction on the number of used colors, and the rough course of the induction step is as follows: Let Q be the set of vertices and W_1 be the set of vertices from which Player 1 has a memoryless winning strategy. We have to show that from each vertex in $Q \setminus W_1$, Player 2 has a memoryless winning strategy. Consider the set C_k of vertices having the maximal color k. We assume that k is even (otherwise one has to switch the players). The case $C_k \subseteq W_1$ gives us a game graph $Q \setminus W_1$ without color k; applying the induction hypothesis to this set we easily get the claim. Otherwise consider $C_k \setminus W_1$ and define its attractor A for Player 2, restricted to $Q \setminus W_1$. (For the case of an infinite game graph, the inductive definition of $A^2(F)$ mentioned above has to be adapted, involving a transfinite induction.) Now the vertex set $Q \setminus (W_1 \cup A)$ defines a game graph without color k, so by induction hypothesis there exists a division into two sets U_1, U_2 from where Player 1, respectively Player 2 has a memoryless winning strategy. It turns out that U_1 must be empty and that on $A \cup U_2$, which is the complement of W_1, Player 2 has a memoryless winning strategy.

Over finite game graphs, also the effectiveness claims can be shown; it is open whether the decision who wins a parity game from a given vertex v is possible in polynomial time.

5 Model-Checking and Tree Automata Theory

We have mentioned two motivations for the question of determinacy, a historical one in set theory, and a technical one regarding the completeness of winning strategies. In this section we address a third type of motivation, originating in logic. In this case determinacy reflects the duality between "true" and "false". We indicate very briefly two kinds of application. Details can be found, e.g., in [13].

5.1 Model-Checking

Model-Checking is the task of evaluating a formula φ in a structure $\mathcal{S}$. It is well-known that the evaluation can be explained in terms of a finite game between the players Proponent (aiming at showing truth of the formula) and Opponent (aiming at the converse). The duality between the connectives $\vee, \wedge$ and between the quantifiers $\exists, \forall$ is reflected in the rules: For example, when treating $\varphi_1 \vee \varphi_2$ then Proponent picks one of φ_1, φ_2 for which the evaluation continues, and when a formula $\varphi_1 \wedge \varphi_2$ is treated, then Opponent picks one of φ_1, φ_2. Proponent wins a play if it ends at an atomic formula which is true in $\mathcal{S}$. Then $\mathcal{S}$ satisfies φ iff in this finite "model-checking game" Proponent has a winning strategy (and falsehood means that Opponent has a winning strategy).

The modal μ-calculus is a logic involving least and greatest fixed points rather than quantifiers; the formulas are interpreted in transition graphs in the form of Kripke structures. Even for finite Kripke structures, the appropriate model-checking game turns out to be infinite, since the semantics depends on infinite paths when fixed point operators enter. As first shown in [8], the model-checking game for the μ-calculus is in fact a parity game; the game graph is a product of the considered structure $\mathcal{S}$ and the collection $\mathrm{SF}(\varphi)$ of subformulas of the given formula φ. Thus the open problem about polynomial time solvability of the μ-calculus model-checking poblem reduces to the (again open) problem of polynomial time solvability of parity games.

An extension of the μ-calculus model-checking game to a "quantitative μ-calculus" was developed in [9].

5.2 Tree Automata

A very strong decidability result of logic is Rabin's Tree Theorem, saying that the MSO-theory of the infinite binary tree is decidable [21]. Rabin's method for showing this was a transformation of MSO-formulas (interpreted over the binary tree T_2) into finite tree automata working over labelled infinite trees (where each node of T_2 gets a label from a finite alphabet Σ). These tree automata are nondeterministic, and a run is a labelling of the input tree nodes with automaton states such that a natural compatibility with the input tree and the automaton's transition relation holds. As acceptance condition for a given run one can require that on each path of the run, a parity condition is satisfied. In this case we speak of a parity tree automaton.

The main problem of this approach is to show complementation for parity tree automata. For this the determinacy of parity games (in the version over infinite game graphs) can be used. In fact, the acceptance of a labelled tree t by a parity tree automaton $\mathcal{A}$ can be captured in terms of a parity game $G_{\mathcal{A},t}$ between two players called "Automaton" and "Pathfinder". The infinite game graph is a kind of product of t and $\mathcal{A}$. Then $\mathcal{A}$ accepts t iff Automaton has a winning strategy in $G_{\mathcal{A},t}$. The complementation proof (following [10]) starts with the negation of the statement "$\mathcal{A}$ accepts t", i.e. in game-theoretical terms with the statement "Automaton does not have a (memoryless) winning strategy in $G_{\mathcal{A},t}$". By memoryless determinacy of parity games, this means that Pathfinder has a memoryless winning strategy in $G_{\mathcal{A},t}$. From this winning strategy it is possible to build (in fact, independently of the given tree t) a new tree automaton $\mathcal{B}$ as the complement automaton for $\mathcal{A}$.

There are further results that belong to the very tight connection between infinite games and automata on infinite trees. In particular, the solution of parity games over finite graphs is reducible to the emptiness problem for parity tree automata and conversely.

Let us comment on the application of tree automata for the solution of games. This use of tree automata appears in Rabin's approach to Church's Problem (developed in [22]). The idea is to code a strategy of Player 2 by a labelling of the nodes of the infinite binary tree: The root has no label, the directions left and right represent the bits chosen by Player 1, and the labels on the nodes different from the root are the bits chosen by Player 2 according to the considered strategy. When Player 1 chooses the bits $b_0, \dots, b_k$, he defines a path to a certain node; the label b of this node is then the next choice of Player 2. Now the paths through a labelled tree t capture all plays that are compatible with Player 2's strategy coded by t. Using analogous constructions to those explained in Section 3 above, one can build a parity tree automaton $\mathcal{A}$ that checks whether t codes a winning strategy. Deciding non-emptiness of $\mathcal{A}$ thus allows to decide whether Player 2 wins the given game. By Rabin's "Basis Theorem" we even know that in this case some regular tree is accepted by $\mathcal{A}$. This regular tree can then be interpreted as a finite-state winning strategy for Player 2.

6 On Uniformization

A class $\mathcal{R}$ of binary relations $R \subseteq D \times D'$ is *uniformizable* by functions in a class $\mathcal{F}$ if for each $R \in \mathcal{R}$ there is a function $f \in \mathcal{F}$ such that

- the graph of f is contained in R,
- the domains of R and f coincide.

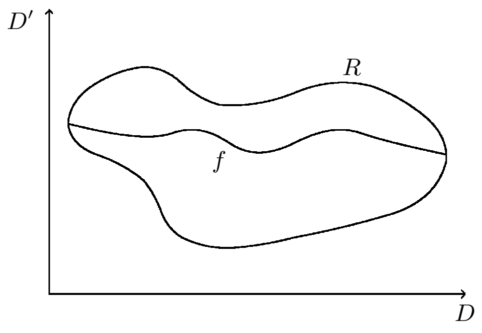


Fig. 2. Uniformization

Two well-known examples from recursion theory and from automata theory are concerned with the recursively enumerable, respectively the rational relations; here we have the "ideal" case that the graphs of the required functions are precisely of the type of the given relations.

Recall that a partial function from $\mathbb{N}$ to $\mathbb{N}$ is recursive iff its graph is recursively enumerable. The Uniformization Theorem of recursion theory says that a binary recursively enumerable relation R is uniformizable by a function whose graph is again recursively enumerable, i.e. by a (partial) recursive function f. A computation of $f(x)$ works as follows: Enumerate R until a pair (y, z) is reached with $x = y$, and in this case produce z as output.

A binary rational relation is defined (for instance) by a finite nondeterministic two-tape automaton that scans a given word pair (u, v) asynchronously, i.e. with two reading heads that move independently from left to right over u, respectively v. Rational relations are uniformizable by rational functions, defined as the functions whose graph is a rational relation (see e.g. [6]).

Church's Problem is a variant of the uniformization problem. It asks for the *decision* whether an MSO-definable relation is uniformizable by a Mealy automaton computable function. In the context of determinacy, the problem is extended to the question whether for an MSO-definable relation $R \subseteq \{0,1\}^\omega \times \{0,1\}^\omega$, either R itself or the complement of its inverse is uniformizable by a Mealy computable function.

There are numerous variants of this problem, either in the unilateral version (as in Church's Problem) or in the determinacy version. For both parameters, the class $\mathcal{R}$ of relations and the class $\mathcal{F}$ of uniformizing functions, there are several other interesting options.[3]

Let us first mention some natural classes of *functions.* A Mealy automaton computable function $f : \{0,1\}^\omega \to \{0,1\}^\omega$ is *continuous* (in Cantor's topology): Each bit of $f(X)$ only depends on a finite prefix of X. If the i-th bit of $f(X)$ only depends on $X_0, \ldots, X_i$ then we call f *causal.* One can show that a function is Mealy automaton computable iff it is causal and its graph is MSO-definable.

[3] In the monograph [26] of Trakhtenbrot and Barzdin one finds an early account of these matters.

The proof rests on the equivalence between MSO-logic and finite automata over *finite* words (the Büchi-Elgot-Trakhtenbrot-Theorem). Note that there are MSO-definable functions that are not causal and not even continuous (a trivial example is the function that maps a bit-sequence with infinitely many 1's to 1^ω and all other sequences to 0^ω).

A more restricted concept of MSO-definability refers to the fact that a strategy $f : \{0,1\}^+ \to \{0,1\}$ can be captured by the language

$$L_f := \{X_0 \dots X_i \in \{0,1\}^+ \mid f(X_0 \dots X_i) = 1\}$$

of finite words. We say that a *strategy is MSO-definable* if L_f is. For MSO-logic this is compatible with the definition given above: A strategy f is MSO-definable iff the graph of the associated causal function f_ω is MSO-definable.

A slightly more extended class of functions (over the causal ones) is given by the condition that the i-th bit of $f(X)$ only depends on the bits $X_0, \dots, X_j$ where $i < j$ and $j - i \leq k$ for some constant k; we then say that f is of *delay* k. In game-theoretic terms, this means that Player 2 can lag behind Player 1 by k moves. The dual type of function g is called "shift k" (where the i-th bit of $g(Y)$ depends on $Y_0, \dots, Y_j$ with $j < i$ and $i - j < k$). It is not difficult to show a determinacy result for MSO-definable relations by finite-state computable functions of delay k, respectively shift k. Hosch and Landweber [15] proved the interesting result that it is decidable whether for an MSO-definable relation the associated game is won by Player 2 with a strategy of bounded delay (by some k), and they also showed that in this case a finite-state winning strategy can be constructed.

Not much is known about more general strategies. One natural option is to consider strategies that induce functions $f : \{0,1\}^\omega \to \{0,1\}^\omega$ with linearly increasing delay, respectively linearly increasing shift. A function of the first type is the "division by 2" of sequences, mapping $X_0X_1X_2X_3X_4 \dots$ to $X_0X_2X_4 \dots$; an example of the second type is the "double function" $X_0X_1 \dots \mapsto X_0X_0X_1X_1 \dots$. Note that gsm-mappings (functions computable by generalized sequential machines) are finite-state computable functions of linearly increasing shift. These functions seem interesting for uniformization or determinacy problems on more general relations than the MSO-definable ones.

Regarding Church's Problem for other classes of *relations*, let us first consider relations that are first-order definable rather than MSO-definable. There are two natural versions of first-order logic, denoted FO(+1) and FO(<), where in brackets we exhibit the available arithmetical signature. (A technical point to be mentioned is that we cannot pass from a relation R to the associated ω-language L_R as explained in Section 2, since the even and the odd positions of an ω-sequence cannot be distinguished in FO(+1) and FO(<). So we consider a pair (X, Y) of bit sequences as an ω-word $(X_0, Y_0), (X_1, Y_1), \dots$ over the alphabet $\{0,1\} \times \{0,1\}$.) In [23], it was shown that a determinacy theorem holds for the FO(+1)-, respectively the FO(<)-definable relations, and that appropriate winning strategies exist which are again FO(+1)-, respectively FO(<)-definable.

There are also interesting logics for which the analogous result fails. We give an example for the unilateral case as addressed in Church's Problem. Consider

Presburger arithmetic, the first-order theory of addition over $\mathbb{N}$. We present a formula $\varphi(X,Y)$ of Presburger arithmetic such that in the game associated with R_φ there is a winning strategy for Player 2, however not a Presburger definable one. First we write down in Presburger arithmetic a formula $\varphi_{\mathrm{squ}}(X)$ which says that X is the set Squ of squares (use the fact that the distances of successive squares increase by 2). Now one invokes the fact that multiplication is FO-definable in $(\mathbb{N},+,\mathrm{Squ})$ ([20]). Hence also each arithmetical set is FO-definable in $(\mathbb{N},+,\mathrm{Squ})$; pick such a set M which is not recursive, and let $\varphi_M(y)$ its FO-definition in $(\mathbb{N},+,\mathrm{Squ})$. Write $\varphi_M(X,y)$ for the formula where the predicate symbol for Squ is replaced by X. Now consider the following Presburger formula over $(\mathbb{N},+)$:

$$\varphi(X,Y) := (\varphi_{\mathrm{squ}}(X) \rightarrow \forall y(Y(y) \leftrightarrow \varphi_M(X,y)))$$

Clearly there is a winning strategy for Player 2 in the game defined by φ, e.g. with $Y = M$ for arbitrary X. As the case $X = \mathrm{Squ}$ shows, the strategy f cannot be recursive. Thus the language L_f coding f is not recursive, so it cannot be Presburger definable.

It seems to this author that a comprehensive theory of effective determinacy and uniformization over infinite words is only in its beginnings. Although this question is raised from a theoretical point of view, results obtained in this research are likely to be interesting also in the context of the synthesis of controllers or reactive programs.

References

1. Büchi, J.R.: Weak second-order arihmetic and finite automata. Z. Math. Logik Grundlagen Math. 6, 66–92 (1960)
2. Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Nagel, E., et al. (eds.) Proc. 1960 International Congress on Logic, Methodology and Philosophy of Science, pp. 1–11. Stanford University Press (1962)
3. Büchi, J.R., Landweber, L.H.: Solving sequential conditions by finite-state strategies. Trans. Amer. Math. Soc. 138, 367–378 (1969)
4. Church, A.: Applications of recursive arithmetic to the problem of circuit synthesis. In: Summaries of the Summer Institute of Symbolic Logic, vol. I, pp. 3–50. Cornell Univ., Ithaca (1957)
5. Church, A.: Logic, arithmetic, and automata. In: Proc. Int. Congr. Math. 1962, Inst. Mittag-Leffler, Djursholm, Sweden, pp. 23–35 (1963)
6. Eilenberg, S.: Automata, Languages, and Machines, vol. A. Academic Press, New York (1974)
7. Emerson, E.A., Jutla, C.S.: Tree automata, mu-calculus, and determinacy. In: Proc. 32nd FoCS 1991, pp. 368–377. IEEE Comp. Soc. Press, Los Alamitos (1991)
8. Emerson, E.A., Jutla, C.S., Sistla, A.P.: On model checking for fragments of the μ-calculus. In: Courcoubetis, C. (ed.) CAV 1993. LNCS, vol. 697, pp. 385–396. Springer, Heidelberg (1993)
9. Fischer, D., Grädel, E., Kaiser, L.: Model checking games for the quantitative μ-Calculus. In: Albers, S., Weil, P. (eds.) Proc. STACS 2008, pp. 301–312 (2008)
10. Gurevich, Y., Harrington, L.: Trees, automata, and games. In: Proc. 14th ACM Symp. on the Theory of Computing, pp. 60–65. ACM Press, New York (1982)

11. Grädel, E.: Banach-Mazur games on graphs. In: Proc. FSTTCS 2008 (2008), http://drops.dagstuhl.de/portals/FSTTCS08
12. Gale, D., Stewart, F.M.: Infinite games with perfect information. Ann. Math. Studies 28, 245–266 (1953)
13. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games. LNCS, vol. 2500. Springer, Heidelberg (2002)
14. Hausdorff, F.: Grundzüge der Mengenlehre, Leipzig (1914)
15. Hosch, F.A., Landweber, L.H.: Finite delay solutions for sequential conditions. In: Nivat, M. (ed.) Proc. ICALP 1972, pp. 45–60. North-Holland, Amsterdam (1972)
16. Kechris, A.S.: Classical Descriptive Set Theory. Springer, New York (1995)
17. McNaughton, R.: Finite-state infinite games, Project MAC Rep., MIT, Cambridge, Mass (September 1965)
18. McNaughton, R.: Testing and generating infinite sequences by a finite automaton. Inf. Contr. 9, 521–530 (1966)
19. Mostowski, A.W.: Regular expressions for infinite trees and a standard form of automata. In: Skowron, A. (ed.) SCT 1984. LNCS, vol. 208, pp. 157–168. Springer, Heidelberg (1984)
20. Putnam, H.: Decidability and essential undecidability. JSL 22, 39–54 (1957)
21. Rabin, M.O.: Decidability of second-order theories and automata on infinite trees. Trans. Amer. Math. Soc. 141, 1–35 (1969)
22. Rabin, M.O.: Automata on infinite objects and Church's Problem. Amer. Math. Soc., Providence (1972)
23. Rabinovich, A., Thomas, W.: Logical Refinements of Church's Problem. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 69–83. Springer, Heidelberg (2007)
24. Selivanov, V.: Fine hierarchies and Boolean terms. J. Smb. Logic 60, 289–317 (1995)
25. Thomas, W.: Solution of Church's Problem: A tutorial. In: Apt, K., van Rooij, R. (eds.) New Perspectives on Games and Interaction. Texts on Logic and Games, vol. 5, pp. 211–236. Amsterdam Univ. Press
26. Trakhtenbrot, B.A., Barzdin, Y.M.: Finite Automata. Behavior and Synthesis. North-Holland, Amsterdam (1973)

Temporal Reasoning about Program Executions

Rajeev Alur

University of Pennsylvania

While temporal verification of programs is a topic with a long history, its traditional basis —semantics based on word languages— is ill-suited for modular reasoning about procedural programs. We address this inadequacy by defining the semantics of procedural (potentially recursive) programs in terms of *languages of nested words* and developing a framework for temporal reasoning around it. This generalization has two benefits. First, this style of reasoning naturally unifies Manna-Pnueli-style temporal reasoning with Hoare-style reasoning about structured programs. Second, it allows verification of "non-regular" properties of specific procedural contexts—for example, "if a procedure acquires a lock, then the same invocation releases it before returning." In this talk, we will first discuss the *Nested Word Temporal Logic*, a temporal logic for infinite nested words, and argue that it is the "right" logic for temporal reasoning about procedural programs based on theoretical results about decidability of the propositional fragment and first-order expressive-completeness. Then, we will present, sound and relatively complete, proof rules for a variety of classes of properties such as *local safety*, *local response*, *global safety*, and *staircase reactivity*.

This talk is based on joint work reported in the following publications:

1. Adding nesting structure to words, full version under journal review (with P. Madhusudan).
2. First-order and temporal logics for nested words, *Logical Methods in Computer Science* (LMCS) **4(4: 11)**, 2008 (with M. Arenas, P. Barcelo, K. Etessami, N. Immerman, and L. Libkin).
3. Temporal reasoning for procedural programs, Pennsylvania State University Technical Report CSE-08-015, 2008 (with S. Chaudhuri).

Least and Greatest Fixpoints in Game Semantics

Pierre Clairambault

PPS, Université Paris 7
pierre.clairambault@pps.jussieu.fr

Abstract. We show how solutions to many recursive arena equations can be computed in a natural way by allowing loops in arenas. We then equip arenas with winning functions and total winning strategies. We present two natural winning conditions compatible with the loop construction which respectively provide initial algebras and terminal coalgebras for a large class of continuous functors. Finally, we introduce an intuitionistic sequent calculus, extended with syntactic constructions for least and greatest fixed points, and prove it has a sound and (in a certain weak sense) complete interpretation in our game model.

1 Introduction

The idea to model logic by game-theoretic tools can be traced back to the work of Lorenzen [21]. The idea is to interpret a formula by a game between two players O and P, O trying to refute the formula and P trying to prove it. The formula A is then valid if P has a *winning strategy* on the interpretation of A. Later, Joyal remarked [18] that it is possible to compose strategies in Conway games [8] in an associative way, thus giving rise to the first category of games and strategies. This, along with parallel developments in Linear Logic and Geometry of Interaction, led to the more recent construction of compositional game models for a large variety of logics [3,23,9] and programming languages [17,4,22,5].

We aim here to use these tools to model an intuitionistic logic with induction and coinduction. Inductive/coinductive definitions in syntax have been defined and studied in a large variety of settings, such as linear logic [6], λ-calculus [1] or Martin-Löf's type theory [10]. Motivations are multiple, but generally amount to increasing the expressive power of a language without paying the price of exponential modalities (as in [6]) or impredicativity (as in [1] or [10]). However, less work has been carried out when it comes to the semantics of such constructions. Of course we have the famous order-theoretic Knaster-Tarski fixed point theorem [25], the nice categorical theory due to Freyd [12], set-theoretic models [10] (for the *strictly positive* fragment) or PER-models [20], but it seems they have been ignored by the current trend for intensional models (*i.e.* games semantics, GoI ...). We fix this issue here, showing that (co)induction admits a nice game-theoretic model which arises naturally if one enriches McCusker's [22] work on recursive types with winning functions inspired by parity games [24].

In Section 2, we first recall the basic definitions of the Hyland-Ong-Nickau setting of game semantics. Then we sketch McCusker's interpretation of recursive types, and show how most of these recursive types can be modelled by means of loops in the arenas. For this purpose, we define a class of functors called *open functors*, including in particular all the endofunctors built out of the basic type constructors. We also present a mechanism of winning functions inspired by [16], allowing us to build a category **Gam** of games and total winning strategies. In section 3, we present μLJ, the intuitionistic sequent calculus with least and greatest fixpoints that we aim to model. We briefly discuss its proof-theoretic properties, then present its semantic counterpart: we show how to build initial algebras and terminal coalgebras to most positive open functors. Finally, we use this semantic account of (co)induction to give a sound and (weakly) complete interpretation of μLJ in **Gam**.

2 Arena Games

2.1 Arenas and Plays

We recall briefly the now usual definitions of arena games, introduced in [17]. More detailed accounts can be found in [22,14]. We are interested in games with two participants: Opponent (O, the *environment*) and Player (P, the *program*). Possible plays are generated by directed graphs called *arenas*, which are semantic versions of *types* or *formulas*. Hence, a play is a sequence of *moves* of the ambient arena, each of them being annotated by a *pointer* to an earlier move — these pointers being required to comply with the structure of the arena. Formally, an **arena** is a structure $A = (M_A, \lambda_A, \vdash_A)$ where:

- M_A is a set of **moves**,
- $\lambda_A : M_A \to \{O, P\} \times \{Q, A\}$ is a **labelling** function indicating whether a move is an Opponent or Player move, and whether it is a question (Q) or an answer (A). We write λ_A^{OP} for the projection of λ_A to $\{O, P\}$ and λ_A^{QA} for its projection on $\{Q, A\}$. $\overline{\lambda_A}$ will denote λ_A where the $\{O, P\}$ part has been reversed.
- $\vdash_A$ is a relation between $M_A + \{\star\}$ to M_A, called **enabling**, satisfying:
  - $\star \vdash m \Longrightarrow \lambda_A(m) = OQ$;
  - $m \vdash_A n \wedge \lambda_A^{QA}(n) = A \Longrightarrow \lambda_A^{QA}(m) = Q$;
  - $m \vdash_A n \wedge m \neq \star \Longrightarrow \lambda_A^{OP}(m) \neq \lambda_A^{OP}(n)$.

In other terms, an arena is a directed bipartite graph, with a set of distinguished **initial** moves (m such that $\star \vdash_A m$) and a distinguished set of **answers** (m such that $\lambda_A^{QA} = A$) such that no answer points to another answer. We now define plays as **justified sequences** over A: these are sequences s of moves of A, each non-initial move m in s being equipped with a pointer to an earlier move n in s, satisfying $n \vdash_A m$. In other words, a justified sequence s over A is such that each reversed pointer chain $s_{\phi(0)} \leftarrow s_{\phi(1)} \leftarrow \ldots \leftarrow s_{\phi(n)}$ is a path on A, viewed as a directed bipartite graph.

The role of pointers is to allow *reopenings* in plays. Indeed, a path on A may be (slightly naively) understood as a linear play on A, and a justified sequence as an interleaving of paths, with possible duplications of some of them. This intuition is made precise in [15]. When writing justified sequences, we will often omit the justification information if this does not cause any ambiguity. $\sqsubseteq$ will denote the prefix ordering on justified sequences. If s is a justified sequence on A, $|s|$ will denote its length.

Given a justified sequence s on A, it has two subsequences of particular interest: the P-view and O-view. The view for P (resp. O) may be understood as the subsequence of the play where P (resp. O) only sees his own duplications. In a P-view, O never points more than once to a given P-move, thus he must always point to the previous move. Concretely, P-views correspond to branches of Böhm trees [17]. Practically, the P-view $\ulcorner s \urcorner$ of s is computed by forgetting everything under Opponent's pointers, in the following recursive way:

– $\ulcorner sm \urcorner = \ulcorner s \urcorner m$ if $\lambda_A^{OP}(m) = P$;
– $\ulcorner sm \urcorner = m$ if $\star \vdash_A m$ and m has no justification pointer;
– $\ulcorner s_1 m s_2 n \urcorner = \ulcorner s \urcorner mn$ if $\lambda_A^{OP}(n) = O$ and n points to m.

The O-view $\llcorner s \lrcorner$ of s is defined dually. Note that in some cases — in fact if s does not satisfies the *visibility condition* introduced below — $\ulcorner s \urcorner$ and $\llcorner s \lrcorner$ may not be correct justified sequences, since some moves may have pointed to erased parts of the play. However, we will restrict to plays where this does not happen. The **legal sequences** over A, denoted by $\mathcal{L}_A$, are the justified sequences s on A satisfying the following conditions:

– **Alternation.** If $tmn \sqsubseteq s$, then $\lambda_A^{OP}(m) \neq \lambda_A^{OP}(n)$;
– **Bracketing.** A question q is **answered** by a if a is an answer and a points to q. A question q is **open** in s if it has not yet been answered. We require that each answer points to the *pending* question, *i.e.* the last open question.
– **Visibility.** If $tm \sqsubseteq s$ and m is not initial, then if $\lambda_A^{OP}(m) = P$ the justifier of m appears in $\ulcorner t \urcorner$, otherwise its justifier appears in $\llcorner t \lrcorner$.

2.2 The Cartesian Closed Category of Innocent Strategies

A **strategy** σ on A is a prefix-closed set of even-length legal plays on A. A strategy is **deterministic** if only Opponent branches, *i.e.* $\forall smn, smn' \in \sigma,\ n = n'$. Of course, if A represents a type (or formula), there are often many more strategies on A than programs (or proofs) on this type. To address this issue we need **innocence**. An innocent strategy is a strategy σ such that

$$sab \in \sigma \wedge t \in \sigma \wedge ta \in \mathcal{L}_A \wedge \ulcorner sa \urcorner = \ulcorner ta \urcorner \Longrightarrow tab \in \sigma$$

We now recall how arenas and innocent strategies organize themselves into a cartesian closed category. First, we build the **product** $A \times B$ of two arenas A and B:

$$\begin{aligned} M_{A\times B} &= M_A + M_B \\ \lambda_{A\times B} &= [\lambda_A, \lambda_B] \\ \vdash_{A\times B} &= \ \vdash_A + \vdash_B \end{aligned}$$

We mention the empty arena $I = (\emptyset, \emptyset, \emptyset)$, which will be terminal for the category of arenas and innocent strategies. We mention as well the arena $\bot = (\bullet, \bullet \mapsto OQ, (\star, \bullet))$ with only one initial move, which will be a weak initial object. We define the **arrow** $A \Rightarrow B$ as follows:

$$\begin{aligned} M_{A \Rightarrow B} &= M_A + M_B \\ \lambda_{A \Rightarrow B} &= [\overline{\lambda_A}, \lambda_B] \\ m \vdash_{A \Rightarrow B} n \Leftrightarrow &\begin{cases} m \neq \star \wedge m \vdash_A n \\ m \neq \star \wedge m \vdash_B n \\ \star \vdash_B m \wedge \star \vdash_A n \\ m = \star \wedge \star \vdash_B n \end{cases} \end{aligned}$$

We define composition of strategies by the usual parallel interaction plus hiding mechanism. If A, B and C are arenas, we define the set of **interactions** $I(A, B, C)$ as the set of justified sequences u over A, B and C such that $u_{\upharpoonright A,B} \in \mathcal{L}_{A \Rightarrow B}$, $u_{\upharpoonright B,C} \in \mathcal{L}_{B \Rightarrow C}$ and $u_{\upharpoonright A,C} \in \mathcal{L}_{A \Rightarrow C}$. Then, if $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$, we define parallel interaction:

$$\sigma || \tau = \{u \in I(A, B, C) \mid u_{\upharpoonright A,B} \in \sigma \wedge u_{\upharpoonright B,C} \in \tau\}$$

Composition is then defined as $\sigma; \tau = \{u_{\upharpoonright A,C} \mid u \in \sigma || \tau\}$. It is associative and preserves innocence (a proof of these facts can be found in [17] or [14]). We also define the identity on A as the copycat strategy (see [22] or [14] for a definition) on $A \Rightarrow A$. Thus, there is a category **Inn** which has arenas as objects and innocent strategies on $A \Rightarrow B$ as morphisms from A to B. In fact, this category is cartesian closed, the cartesian structure given by the arena product above and the exponential closure given by the arrow construction. This category is also equipped with a weak coproduct $A + B$ [22], which is constructed as follows:

$$\begin{aligned} M_{A+B} &= M_A + M_B + \{q, L, R\} \\ \lambda_{A+B} &= [\lambda_A, \lambda_B, q \mapsto OQ, L \mapsto PA, R \mapsto PA] \\ m \vdash_{A+B} n \Leftrightarrow &\begin{cases} m, n \in M_A \wedge m \vdash_A n \\ m, n \in M_B \wedge m \vdash_B n \\ m = \star \wedge n = q \\ (m = q \wedge n = L) \vee (m = q \wedge n = R) \\ (m = L \wedge \star \vdash_A n) \vee (m = R \wedge \star \vdash_B n) \end{cases} \end{aligned}$$

2.3 Recursive Types and Loops

Let us recall briefly the interpretation of recursive types in game semantics, due to McCusker [22]. Following [22], we first define an ordering $\trianglelefteq$ on arenas as follows. For two arenas A and B, $A \trianglelefteq B$ iff

$$\begin{aligned} &M_A \subseteq M_B \\ &\lambda_A = \lambda_{B \upharpoonright M_A} \\ &\vdash_A = \ \vdash_B \cap (M_A + \{\star\} \times M_A) \end{aligned}$$

This defines a (large) dcpo, with least element I and directed sups given by the componentwise union. If $F : \mathbf{Inn} \to \mathbf{Inn}$ is a functor which is continuous with respect to $\trianglelefteq$, we can find an arena D such that $D = F(D)$ in the usual way by setting $D = \bigsqcup_{n=0}^{\infty} F^n(I)$. McCusker showed [22] that when the functors are *closed* (*i.e.* their action can be internalized as a morphism $(A \Rightarrow B) \to (FA \Rightarrow FB)$), and when they preserve inclusion and projection morphisms (*i.e.* partial copycat strategies) corresponding to $\trianglelefteq$, this construction defines *minimal invariants* [12]. Note that the crucial cases of these constructions are the functors built out of the product, sum and function space constructions.

We give now a concrete and new (up to the author's knowledge) description of a large class of continuous functors, that we call **open functors**. These include all the functors built out of the basic constructions, and allow a rereading of recursive types, leading to the model of (co)induction.

Open Arenas. Let $\mathbf{T}$ be a countable set of names. An **open arena** is an arena A with distinguished question moves called **holes**, each of them labelled by an element of $\mathbf{T}$. We denote by $\Box_X$ the holes annotated by $X \in \mathbf{T}$. We will sometimes write $\Box_X^P$ to denote a hole of Player polarity, or $\Box_X^O$ to denote a hole of Opponent polarity. If A has holes labelled by $X_1, \ldots, X_n$, we denote it by $A[X_1, \ldots, X_n]$. By abuse of notation, the corresponding open functor we are going to build will be also denoted by $A[X_1, \ldots, X_n] : (\mathbf{Inn} \times \mathbf{Inn}^{op})^n \to \mathbf{Inn}$.

Image of Arenas. If $A[X_1, \ldots, X_n]$ is an open arena and $B_1, \ldots, B_n$, $B_1', \ldots, B_n'$ are arenas (possibly open as well), we build a new arena $A(B_1, B_1', \ldots, B_n, B_n')$ by replacing each occurrence of $\Box_{X_i}^P$ by B_i and each occurrence of $\Box_{X_i}^O$ by B_i'. More formally:

$$M_{A(B_1,B_1',\ldots,B_n,B_n')} = (M_A \setminus \{\Box_{X_1}, \ldots, \Box_{X_n}\}) + \sum_{i=1}^{n} (M_{B_i} + M_{B_i'})$$

$$\lambda_{A(B_1,B_1',\ldots,B_n,B_n')} = [\lambda_A, \lambda_{B_1}, \overline{\lambda_{B_1'}}, \ldots, \lambda_{B_n}, \overline{\lambda_{B_n'}}]$$

$$m \vdash_{A(B_1,B_1',\ldots,B_n,B_n')} p \Leftrightarrow \begin{cases} m \vdash_A \Box_{X_i}^P \wedge \star \vdash_{B_i} p \\ m \vdash_A \Box_{X_i}^O \wedge \star \vdash_{B_i'} p \\ \star \vdash_{B_i} m \wedge \Box_{X_i}^P \vdash_A p \\ \star \vdash_{B_i'} m \wedge \Box_{X_i}^O \vdash_A p \\ m \vdash_{B_i} p \\ m \vdash_{B_i'} p \\ m \vdash_A p \end{cases}$$

Note that in this definition, we assimilate all the moves sharing the same hole *label* $\Box_{X_i}$ and with the same polarity. This helps to clarify notations, and is justified by the fact that we never need to distinguish moves with the same hole label, apart from when they have different polarity.

Image of Strategies. If A is an arena, we will, by abuse of notation, denote by I_A both the set of initial moves of A and the subarena of A with only these moves. Let $A[X_1, \ldots, X_n]$ be an open arena, $B_1', B_1, \ldots, B_n', B_n$ and $C_1', C_1, \ldots, C_n', C_n$ be arenas. Consider the application ξ defined on moves as follows:

$$\xi(x) = \begin{cases} \Box_{X_i} & \text{if } x \in \bigcup_{i\in\{1,\dots,n\}} (I_{B'_i} \cup I_{B_i} \cup I_{C'_i} \cup I_{C_i}) \\ x & \text{otherwise} \end{cases}$$

and then extended recursively to an application ξ^* on legal plays as follows:

$$\xi^*(sa) = \begin{cases} \xi^*(s) & \text{if } a \text{ is a non-initial move of } B_i, B'_i, C_i \text{ or } C'_i \\ \xi^*(s)\xi(a) & \text{otherwise} \end{cases}$$

ξ^* erases moves in the inner parts of B'_i, B_i, C'_i, C_i and agglomerates all the initial moves back to the holes. This way we will be able to compare the resulting play with the identity on $A[X_1, \dots, X_n]$. Now, if $\sigma_i : B_i \to C_i$ and $\tau_i : C'_i \to B'_i$ are strategies, we can now define the action of open functors on them by stating:

$$s \in A(\sigma_1, \tau_1, \dots, \sigma_n, \tau_n) \Leftrightarrow \begin{cases} \forall i \in \{1, \dots, n\},\ s_{\upharpoonright B_i \Rightarrow C_i} \in \sigma_i \\ \forall i \in \{1, \dots, n\},\ s_{\upharpoonright C'_i \Rightarrow B'_i} \in \tau_i \\ \xi^*(s) \in id_{A[X_1,\dots,X_n]} \end{cases}$$

Proposition 1. *For any $A[X_1, \dots, X_n]$, this defines a functor $A[X_1, \dots, X_n] : (\mathbf{Inn} \times \mathbf{Inn}^{op})^n \to \mathbf{Inn}$, which is monotone and continuous with respect to $\trianglelefteq$.*

Proof (Proof sketch). Preservation of identities and composition are rather direct. A little care is needed to show that the resulting strategy is innocent: this relies on two facts: First, for each Player move the three definition cases are mutually exclusive. Second, a P-view of $s \in A(\sigma_1, \tau_1, \dots, \sigma_n, \tau_n)$ is (essentially) an initial copycat appended with a P-view of one of σ_i or τ_i, hence the P-view of s determines uniquely the P-view presented to one of σ_i, τ_i or $id_{A[X_1,\dots,X_n]}$.

Example. Consider the open arena $A[X] = \Box_X \Rightarrow \Box_X$. For any arena B, we have $A(B) = B \Rightarrow B$ and for any $\sigma : B_1 \to C_1$ and $\tau : C_2 \to B_2$, we have $A(\sigma, \tau) = \tau \Rightarrow \sigma : (B_2 \Rightarrow B_1) \to (C_2 \Rightarrow C_1)$, the strategy which precomposes its argument by τ and postcomposes it by σ.

Loops for Recursive Types. Since these open functors are monotone and continuous with respect to $\trianglelefteq$, solutions to their corresponding recursive equations can be obtained by computing the infinite expansion of arenas (*i.e.* infinite iteration of the open functors). However, for a large subclass of the open functors, this solution can be expressed in a simple way by replacing holes with a loop up to the initial moves. Suppose $A[X_1, \dots, X_n]$ is an open functor, and i is such that $\Box_{X_i}$ appears only in non-initial, positive positions in A. Then we define an arena $\mu X_i.A$ as follows:

$$\begin{aligned} M_{\mu X_i.A} &= (M_A \setminus \Box_{X_i}) \\ \lambda_{\mu X_i.A} &= \lambda_{A\upharpoonright M_{\mu X_i.A}} \end{aligned}$$

$$m \vdash_{\mu X_i.A} n \Leftrightarrow \begin{cases} m \vdash_A n \\ m \vdash_A \Box_{X_i} \wedge \star \vdash_A n \end{cases}$$

A simple argument ensures that the obtained arena is isomorphic to the one obtained by iteration of the functor. For this issue we take inspiration from

Laurent [19] and prove a theorem stating that two arenas are isomorphic in the categorical sense if and only if their set of paths are isomorphic. A **path** in A is a sequence of moves $a_1, \ldots, a_n$ such that for all $i \in \{1, \ldots, n-1\}$ we have $a_i \vdash_A a_{i+1}$. A **path isomorphism** between A and B is a bijection ϕ between the set of paths of A and the set of paths on B such that for any non-empty path p on A, $\phi(ip(p)) = ip(\phi(p))$ (where $ip(p)$ denotes the immediate prefix of p). We have then the theorem:

Theorem 1. *Let A and B be two arenas. They are categorically isomorphic if and only if there is a path isomorphism between their respective sets of paths.*

Now, it is clear by construction that, if $A[X]$ is an open functor such that $\Box_X$ appears only in non-initial positive positions in A, the set of paths of $\bigsqcup_{n=0}^{\infty} A^n(I)$ and of $\mu X.A$ are isomorphic. Therefore $\mu X.A$ is solution of the recursive equation $X = A(X)$, and when $A[X]$ is closed and preserves inclusions and projections, $\mu X.A$ defines as well a minimal invariant for $A[X]$. But in fact, we have the following fact:

Proposition 2. *If $A[X]$ is an open functor, then it is closed and preserves inclusions and projections. Hence $\mu X.A$ is a minimal invariant for $A[X]$.*

This interpretation of recursive types as loops preserves finiteness of the arena, and as we shall see, allows to easily express the winning conditions necessary to model induction and coinduction.

2.4 Winning and Totality

A **total strategy** on A is a strategy $\sigma : A$ such that for all $s \in \sigma$, if there is a such that $sa \in \mathcal{L}_A$, then there is b such that $sab \in \sigma$. In other words, σ has a response to any legal Opponent move. This is crucial to interpret logic because the interpretation of proofs in game semantics always gives total strategies: this is a counterpart in semantics to the cut elimination property in syntax. To model induction and coinduction in logic, we must therefore restrict to total strategies. However, it is well-known that the class of total strategies is not closed under composition, because an infinite chattering can occur in the hidden part of the interaction. This is analogous to the fact that in λ-calculus, the class of strongly normalizing terms is not closed under application: $\delta = \lambda x.xx$ is a normal form, however $\delta\delta$ is certainly not normalizable. This problem is discussed in [2,16] and more recently in [7]. We take here the solution of [16], and equip arenas with winning functions: for every infinite play we choose a loser, hence restricting to winning strategies has the effect of blocking infinite chattering.

The definition of legal plays extends smoothly to infinite plays. Let $\mathcal{L}_A^\omega$ denote the set of infinite legal plays over A. If $\overline{s} \in \mathcal{L}_A^\omega$, we say that $\overline{s} \in \sigma$ when for all $s \sqsubset \overline{s}$, $s \in \sigma$. We write $\overline{\mathcal{L}_A} = \mathcal{L}_A + \mathcal{L}_A^\omega$. A **game** will be a pair $\mathbb{A} = (A, \mathcal{G}_A)$ where A is an arena, and $\mathcal{G}_A$ is a function from infinite **threads** on A (*i.e.* infinite legal plays with exactly one initial move) to $\{W, L\}$. The winning function $\mathcal{G}_A$ extends naturally to potentially finite threads by setting, for each finite s:

$$\mathcal{G}_A(s) = \begin{cases} W \text{ if } |s| \text{ is even ;} \\ L \ \text{ otherwise.} \end{cases}$$

Finally, $\mathcal{G}_A$ extends to legal plays by saying that $\mathcal{G}_A(s) = W$ iff $\mathcal{G}_A(t) = W$ for every thread t of s. By abuse of notation, we keep the same notation for this extended function. The constructions on arenas presented in section 2.2 extend to constructions on games as follows:

– $\mathcal{G}_{A\times B}(s) = [\mathcal{G}_A, \mathcal{G}_B]$ (indeed, a thread on $A \times B$ is either a thread on A or a thread on B) ;
– $\mathcal{G}_{A+B}(s) = W$ iff all threads of $s_{\upharpoonright A}$ are winning for $\mathcal{G}_A$ and all threads of $s_{\upharpoonright B}$ are winning for $\mathcal{G}_B$.
– $\mathcal{G}_{A\Rightarrow B}(s) = W$ iff if all threads of $s_{\upharpoonright A}$ are winning for $\mathcal{G}_A$, then $\mathcal{G}_B(s_{\upharpoonright B}) = W$.

It is straightforward to check that these constructions commute with the extension of winning functions from infinite threads to potentially infinite legal plays. We now define **winning strategies** $\sigma : \mathbb{A}$ as innocent strategies $\sigma : A$ such that for all $s \in \sigma$, $\mathcal{G}_A(s) = W$. Now, the following proposition is satisfied:

Proposition 3. *Let $\sigma : \mathbb{A} \Rightarrow \mathbb{B}$ and $\tau : \mathbb{B} \Rightarrow \mathbb{C}$ be two total winning strategies. Then $\sigma;\tau$ is total winning.*

Proof (Proof sketch.). If $\sigma;\tau$ is not total, there must be infinite s in their parallel interaction $\sigma||\tau$, such that $s_{\upharpoonright A,C}$ is finite. By switching, we have in fact $|s_{\upharpoonright A}|$ even and $|s_{\upharpoonright C}|$ odd. Thus $\mathcal{G}_A(s_{\upharpoonright A}) = W$ and $\mathcal{G}_C(s_{\upharpoonright C}) = L$. We reason then by disjunction of cases. Either $\mathcal{G}_B(s_{\upharpoonright B}) = W$ in which case $\mathcal{G}_{B\Rightarrow C}(s_{\upharpoonright B,C}) = L$ and τ cannot be winning, or $\mathcal{G}_B(s_{\upharpoonright B}) = L$ in which case $\mathcal{G}_{A\Rightarrow B}(s_{\upharpoonright A,B}) = L$ and σ cannot be winning. Therefore $\sigma;\tau$ is total.

$\sigma;\tau$ must be winning as well. Suppose there is $s \in \sigma;\tau$ such that $\mathcal{G}_{A\Rightarrow C}(s) = L$. By definition of $\mathcal{G}_{A\Rightarrow C}$, this means that $\mathcal{G}_A(s_{\upharpoonright A}) = W$ and $\mathcal{G}_C(s_{\upharpoonright C}) = L$. By definition of composition, there is $u \in \sigma||\tau$ such that $s = u_{\upharpoonright A,C}$. But whatever the value of $\mathcal{G}_B(u_{\upharpoonright B})$ is, one of σ or τ is losing. Therefore $\sigma;\tau$ is winning.

It is clear from the definitions that all plays in the identity are winning. It is also clear that all the structural morphisms of the cartesian closed structure of **Inn** are winning (they are essentially copycat strategies), thus this defines a cartesian closed category **Gam** of games and innocent total winning strategies.

3 Fixpoints

3.1 μLJ: An Intuitionistic Sequent Calculus with Fixpoints

Formulas. $S \ ::= \ S \Rightarrow T \mid S \vee T \mid S \wedge T \mid \mu X.T \mid \nu X.T \mid X \mid \top \mid \bot$
A formula F is **valid** if for any subformula of F of the form $\mu X.F'$,

(1) X appears only positively in F',
(2) X does not appear at the root of F' (*i.e.* X appears at least under a $\vee$ or a $\Rightarrow$ in the abstract syntax tree of F').

(2) corresponds to the restriction to arenas where loops allow to express recursive types, whereas *(1)* is the usual positivity condition. We could of course hack the definition to get rid of these restrictions, but we choose not to obfuscate the treatment for an extra generality which is neither often considered in the literature, nor useful in practical examples of (co)induction.

Derivation Rules. We present the rules with the usual dichotomy.

Identity group

$$\frac{}{A \vdash A}\,ax \qquad \frac{\Gamma \vdash A \quad \Delta, A \vdash B}{\Gamma, \Delta \vdash B}\,Cut$$

Structural group

$$\frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B}\,C \qquad \frac{\Gamma \vdash B}{\Gamma, A \vdash B}\,W \qquad \frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C}\,\gamma$$

Logical group

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}\Rightarrow_r \qquad \frac{\Gamma \vdash A \quad \Delta, B \vdash C}{\Gamma, \Delta, A \Rightarrow B \vdash C}\Rightarrow_l \qquad \frac{}{\Gamma, \bot \vdash A}\,\bot_l \qquad \frac{}{\Gamma \vdash \top}\,\top_r$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\wedge_r \qquad \frac{\Gamma, A \vdash C}{\Gamma, A \wedge B \vdash C}\overleftarrow{\wedge_l} \qquad \frac{\Gamma, B \vdash C}{\Gamma, A \wedge B \vdash C}\overrightarrow{\wedge_l}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\overleftarrow{\vee_r} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\overrightarrow{\vee_r} \qquad \frac{\Gamma, A \vdash C \quad \Delta, B \vdash C}{\Gamma, \Delta, A \vee B \vdash C}\vee_l$$

Fixpoints

$$\frac{\Gamma \vdash T[\mu X.T/X]}{\Gamma \vdash \mu X.T}\,\mu_r \qquad \frac{T[A/X] \vdash A}{\mu X.T \vdash A}\,\mu_l \qquad \frac{T[\nu X.T/X] \vdash B}{\nu X.T \vdash B}\,\nu_l \qquad \frac{A \vdash T[A/X]}{A \vdash \nu X.T}\,\nu_r$$

Note that the μ_l, ν_l and ν_r rules are not relative to any context. In fact, the general rules with a context Γ at the left of the sequent are derivable from these ones (even if, for μ_l and ν_r, the construction of the derivation requires an induction on T), and we stick with the present ones to clarify the game model. Cut elimination on the $\Rightarrow, \wedge, \vee$ fragment is the same as usual. For the reduction of μ and ν, we need an additional rule to handle the unfolding of formulas. For this purpose, we add a new rule $[T]$ for each type T with free variables. This method can already be found in [1] for strictly positive functors: no type variable appears on the left of an implication. From now on, $T[A/X]$ will be abbreviated

$T(A)$. This notation implies that, unless otherwise stated, X will be the variable name for which T is viewed as a functor. In the following rules, X appears only positively in T and only negatively in N:

Functors

$$\dfrac{A \vdash B}{T(A) \vdash T(B)}\,[T] \qquad\qquad \dfrac{A \vdash B}{N(B) \vdash N(A)}\,[N]$$

The dynamic behaviour of this rule is to locally perform the unfolding. We give some of the reduction rules. These are of two kinds: the rules for the elimination of $[T]$, and the cut elimination rules. Here are the main cases:

$$\dfrac{\dfrac{\pi}{A \vdash B}}{T \vdash T}\,[T](X \notin FV(T)) \rightsquigarrow \dfrac{}{T \vdash T}\,ax \qquad\qquad \dfrac{\dfrac{\pi}{A \vdash B}}{A \vdash B}\,[X] \rightsquigarrow \dfrac{\pi}{A \vdash B}$$

$$\dfrac{\dfrac{\pi}{A \vdash B}}{N(A) \Rightarrow T(A) \vdash N(B) \Rightarrow T(B)}\,[N \Rightarrow T] \rightsquigarrow \dfrac{\dfrac{\dfrac{\dfrac{\pi}{A \vdash B}}{N(B) \vdash N(A)}\,[N] \qquad \dfrac{\dfrac{\pi}{A \vdash B}}{T(A) \vdash T(B)}\,[T]}{N(A) \Rightarrow T(A), N(B) \vdash T(B)}\,\Rightarrow_l}{N(A) \Rightarrow T(A) \vdash N(B) \Rightarrow T(B)}\,\Rightarrow_r$$

$$\dfrac{\dfrac{\pi}{A \vdash B}}{\mu Y.T(A) \vdash \mu Y.T(B)}\,[\mu Y.T] \rightsquigarrow \dfrac{\dfrac{\dfrac{\dfrac{\pi}{A \vdash B}}{T(A)[\mu Y.T(B)/Y] \vdash T(B)[\mu Y.T(B)/Y]}\,[T[\mu Y.T(B)/Y]]}{T(A)[\mu Y.T(B)/Y] \vdash \mu Y.T(B)}\,\mu_r}{\mu Y.T(A) \vdash \mu Y.T(B)}\,\mu_l$$

We omit the rule for ν, which is dual, and for $\wedge$ and $\vee$, which are simple pairing and case manipulations. Note also that most of these cases have a counterpart where T is replaced by negative N, which has the sole effect of π being a proof of $B \vdash A$ instead of $A \vdash B$ in the expansion rules. With that, we can express the cut elimination rule for fixpoints:

$$\dfrac{\dfrac{\dfrac{\pi_1}{\Gamma \vdash T[\mu X.T/X]}}{\Gamma \vdash \mu X.T}\,\mu_r \qquad \dfrac{\dfrac{\pi_2}{T[A/X] \vdash A}}{\mu X.T \vdash A}\,\mu_l}{\Gamma \vdash A}\,Cut \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\pi_1}{\Gamma \vdash T[\mu X.T/X]} \quad \dfrac{\dfrac{\dfrac{\pi_2}{T[A/X] \vdash A}}{\mu X.T \vdash A}\,\mu_l}{T[\mu X.T/X] \vdash T[A/X]}\,[T]}{\Gamma \vdash T[A/X]}\,Cut \quad \dfrac{\pi_2}{T[A/X] \vdash A}}{\Gamma \vdash A}\,Cut$$

We skip once again the rule for ν, which is dual to μ. We choose consciously not to recall the usual cut elimination rules nor the associated commutation rules, since they are not central to our goals. μLJ, as presented above, does not formally eliminate cuts since there is no rule to reduce the following (and its dual with ν):

$$\dfrac{\dfrac{\dfrac{\pi_1}{T(A) \vdash A}}{\mu X.T \vdash A}\,\mu_l \qquad \dfrac{\pi_2}{\Gamma, A \vdash B}}{\Gamma, \mu X.T \vdash B}\,Cut$$

This cannot be reduced without some prior unfolding of the $\mu X.T$ on the left. This issue is often solved [6] by replacing the rule for μ presented here above by the following:

$$\dfrac{T(A) \vdash A \qquad \Gamma, A \vdash B}{\Gamma, \mu X.T \vdash B}\,\mu'$$

With the corresponding reduction rule, and analogously for ν. We choose here not to do this, first because our game model will prove consistency without the need to prove cut elimination, and second because we want to preserve the proximity with the categorical structure of initial algebras / terminal coalgebras.

3.2 The Games Model

We present the game model for fixpoints. We wish to model a proof system, therefore we need our strategies to be total. The base arenas of the interpretation of fixpoints will be the arenas with loops presented in section 2.3, to which we will adjoin a winning function. While the base arenas will be the same for greatest and least fixpoints, they will be distinguished by the winning function: intuitively, Player loses if a play grows infinite in a least fixpoint (inductive) game, and Opponent loses if this happens in a greatest fixpoint (coinductive) game. The winning functions we are going to present are strongly influenced by Santocanale's work on games for μ-lattices [24]. A **win open functor** is a functor $\mathbb{T} : (\mathbf{Gam} \times \mathbf{Gam}^{op})^n \to \mathbf{Gam}$ such that there is an open functor $T[X_1, \ldots, X_n]$ such that for all games $\mathbb{A}_1, \ldots, \mathbb{A}_{2n}$ of base arenas $A_1, \ldots, A_{2n}$, the base arena of $\mathbb{T}(\mathbb{A}_1, \ldots, \mathbb{A}_{2n})$ is $T(A_1, \ldots, A_n)$. In other terms, it is the natural lifting of open functors to the category of games. By abuse of notation, we denote this by $\mathbb{T}[X_1, \ldots, X_n]$, and $T[X_1, \ldots, X_n]$ will denote its underlying open functor.

Least Fixed Point. Let $\mathbb{T}[X_1, \dots, X_n]$ be a win open functor such that $\square_{X_1}$ appears only positively and at depth higher than 0 in $T[X_1, \dots, X_n]$. Then we define a new win open functor $\mu X_1.\mathbb{T}[X_2, \dots, X_n]$ as follows:

- Its base arena is $\mu X_1.T[X_2, \dots, X_n]$;
- If $\mathbb{A}_3, \dots, \mathbb{A}_{2n} \in$ **Gam**, $\mathcal{G}_{\mu X_1.\mathbb{T}(\mathbb{A}_3,\dots,\mathbb{A}_{2n})}(s) = W$ iff
  - There is $N \in \mathbb{N}$ such that no path of s takes the external loop more that N times, and ;
  - s is winning in the subgame inside the loop, or more formally: $\mathcal{G}_{\mathbb{T}(\mathbb{I},\mathbb{I},\mathbb{A}_3,\dots,\mathbb{A}_{2n})}(s_{\upharpoonright_{\mathbb{T}(\mathbb{I},\mathbb{I},\mathbb{A}_3,\dots,\mathbb{A}_{2n})}}) = W$.

Greatest Fixed Point. Dually, if the same conditions are satisfied, we define the win open functor $\nu X_1.\mathbb{T}[X_1, \dots, X_n]$ as follows:

- Its base arena is $\mu X_1.T[X_2, \dots, X_n]$;
- If $\mathbb{A}_3, \dots, \mathbb{A}_{2n} \in$ **Gam**, $\mathcal{G}_{\nu X_1.\mathbb{T}(\mathbb{A}_3,\dots,\mathbb{A}_{2n})}(s) = W$ iff
  - For any $N \in \mathbb{N}$, there is a path of s crossing the external loop more than N times, or ;
  - s is winning in the subgame inside the loop, or more formally: $\mathcal{G}_{\mathbb{T}(\mathbb{I},\mathbb{I},\mathbb{A}_3,\dots,\mathbb{A}_{2n})}(s_{\upharpoonright_{\mathbb{T}(\mathbb{I},\mathbb{I},\mathbb{A}_3,\dots,\mathbb{A}_{2n})}}) = W$.

It is straightforward to check that these are still functors, and in particular win open functors. There is one particular case that is worth noticing: if $\mathbb{T}[X]$ has only one hole which appears only in positive position and at depth greater than 0, then $\mu X.\mathbb{T}$ is a constant functor, *i.e.* a game. Moreover, theorem 1 implies that it is isomorphic in **Inn** to $\mathbb{T}(\mu X.\mathbb{T})$. It is straightforward to check that this isomorphism $i_{\mathbb{T}} : \mathbb{T}(\mu X.\mathbb{T}) \to \mu X.\mathbb{T}$ is winning (it is nothing but the identity strategy), which shows that they are in fact isomorphic in **Gam**. Then, one can prove the following theorem:

Theorem 2. *If $\mathbb{T}[X]$ has only one hole which appears only in positive position and at depth greater than 0, then the pair $(\mu X.\mathbb{T}, i_{\mathbb{T}})$ defines an* ***initial algebra*** *for $\mathbb{T}[X]$ and $(\nu X.\mathbb{T}, i_{\mathbb{T}}^{-1})$ defines a* ***terminal coalgebra*** *for $\mathbb{T}[X]$.*

Proof. We give the proof for initial alebras, the second part being dual. Let $(\mathbb{A}, \sigma)$ another algebra of $\mathbb{T}[X]$. We need to show that there is a unique $\sigma^\dagger : \mu X.\mathbb{T} \Rightarrow \mathbb{B}$ such that

$$\begin{tikzcd}
\mathbb{T}(\mu X.\mathbb{T}) \arrow[r, "\mathbb{T}(\sigma^\dagger)"] \arrow[d, "i_{\mathbb{T}}"'] & \mathbb{T}(\mathbb{B}) \arrow[d, "\sigma"] \\
\mu X.\mathbb{T} \arrow[r, "\sigma^\dagger"'] & \mathbb{B}
\end{tikzcd}$$

commutes. The idea is to iterate σ:

$$\begin{tikzcd}
\cdots \arrow[r, "\mathbb{T}^3(\sigma)"] & \mathbb{T}^3(\mathbb{B}) \arrow[r, "\mathbb{T}^2(\sigma)"] & \mathbb{T}^2(\mathbb{B}) \arrow[r, "\mathbb{T}(\sigma)"] & \mathbb{T}(\mathbb{B}) \arrow[r, "\sigma"] & \mathbb{B}
\end{tikzcd}$$

and somehow to take the limit. In fact we can give a direct definition of $\sigma^\dagger$:

$$\begin{aligned}\sigma^{(1)} &= \sigma\\ \sigma^{(n+1)} &= \mathbb{T}^n(\sigma);\sigma^{(n)}\\ \sigma^\dagger &= \{s \in \mathcal{L}_{\mu X.\mathbb{T}\Rightarrow\mathbb{B}} \mid \exists n \in \mathbb{N}^*,\ s \in \sigma^{(n)}\}\end{aligned}$$

This defines an innocent strategy, since when restricted to plays of $\mu X.\mathbb{T}$, these strategies agree on their common domain. This strategy is winning. Indeed, take an infinite play $\overline{s} \in \sigma^\dagger$. Suppose $\overline{s}_{\restriction_{\mu X.\mathbb{T}}}$ is winning. By definition of $\mathcal{G}_{\mu X.\mathbb{T}}$, this means that there is $N \in \mathbb{N}$ such that no path of $\overline{s}_{\restriction_{\mu X.\mathbb{T}}}$ takes the external loop more than N times. Thus, $\overline{s} \in \overline{L_{\mathbb{T}^n(\mathbb{I})\Rightarrow\mathbb{B}}}$. But this implies that $\overline{s} \in \sigma^{(n)}$, and $\sigma^{(n)}$ is a composition of winning strategies thus winning, therefore $\overline{s}$ is winning. Moreover, $\sigma^\dagger$ is the unique innocent strategy making the diagram commute: suppose there is another f making this square commute. Since $\mathbb{T}(\mu X.\mathbb{T})$ and $\mu X.\mathbb{T}$ have the same set of paths, $i_\mathbb{T}$ is in fact the identity, thus we have $\mathbb{T}(f);\sigma = f$. By applying T and post-composing by σ, we get:

$$\mathbb{T}^2(f);\mathbb{T}(\sigma);\sigma = \mathbb{T}(f);\sigma = f$$

And by iterating this process, we get for all $n \in \mathbb{N}$:

$$\mathbb{T}^{n+1}(f);\mathbb{T}^n(\sigma);\ldots;\mathbb{T}(\sigma);\sigma = f$$

Thus:

$$\mathbb{T}^{n+1}(f);\sigma^{(n)} = f$$

Now take $s \in f$, and let n be the length of the longest path in s. Since $\mathbb{T}[X]$ has no hole at the root, no path of length n can reach B in $\mathbb{T}^{n+1}(B)$, thus $s \in \sigma^{(n)}$, therefore $s \in \sigma^\dagger$. The same reasoning also works for the other inclusion. Likewise, if $\sigma : \mathbb{B} \to \mathbb{T}(\mathbb{B})$, we build a unique $\sigma^\ddagger : \mathbb{B} \to \nu X.\mathbb{T}$ making the coalgebra diagram commute.

3.3 Interpretation of μLJ

Interpretation of Formulas. As expected, we give the interpretation of valid formulas.

$$\begin{aligned}[\![\top]\!] &= \mathbb{I} & [\![A \Rightarrow B]\!] &= [\![A]\!] \Rightarrow [\![B]\!]\\ [\![\bot]\!] &= \bot & [\![X]\!] &= \square_X\\ [\![A \vee B]\!] &= [\![A]\!] + [\![B]\!] & [\![\mu X.T]\!] &= \mu X.[\![T]\!]\\ [\![A \wedge B]\!] &= [\![A]\!] \times [\![B]\!] & [\![\nu X.T]\!] &= \nu X.[\![T]\!]\end{aligned}$$

Interpretation of Proofs. As usual, the interpretation of a proof π of a sequent $A_1,\ldots,A_n \vdash B$ will be a morphism $[\![\pi]\!] : [\![A_1]\!] \times \ldots \times [\![A_n]\!] \longrightarrow [\![B]\!]$. The interpretation is computed by induction on the proof tree. The interpretation of the rules of LJ is standard and its correctness follows from the cartesian closed structure of **Gam**. Here are the interpretations for the fixpoint and functor rules:

$$\left[\!\!\left[\dfrac{\dfrac{\pi}{\Gamma \vdash T[\mu X.T/X]}}{\Gamma \vdash \mu X.T}\mu_r\right]\!\!\right] = [\![\pi]\!];i_{[\![T]\!]} \qquad \left[\!\!\left[\dfrac{\dfrac{\pi}{T[A/X] \vdash A}}{\mu X.T \vdash A}\mu_l\right]\!\!\right] = [\![\pi]\!]^\dagger$$

$$\left[\!\!\left[\dfrac{\dfrac{\pi}{T[\nu X.T/X] \vdash B}}{\nu X.T \vdash B}\nu_l\right]\!\!\right] = i^{-1}_{[\![T]\!]} ; [\![\pi]\!] \qquad \left[\!\!\left[\dfrac{\dfrac{\pi}{A \vdash T[A/X]}}{A \vdash \nu X.T}\nu_r\right]\!\!\right] = [\![\pi]\!]^{\ddagger}$$

$$\left[\!\!\left[\dfrac{\dfrac{\pi}{A \vdash B}}{T(A) \vdash T(B)}[T]\right]\!\!\right] = [\![T]\!]([\![\pi]\!])$$

We do not give the details of the proof that this defines an invariant of reduction. The main technical point is the validity of the interpretation of the functor rule; more precisely when the functor is a (least or greatest) fixpoint. Given that, we get the following theorem.

Theorem 3. *If* $\pi \rightsquigarrow \pi'$, *then* $[\![\pi]\!] = [\![\pi']\!]$.

In particular, this proves the following theorem which is certainly worth noticing, because μLJ has large expressive power. In particular, it contains Gödel's system T [13].

Theorem 4. μLJ *is consistent: there is no proof of* $\perp$.

Proof. There is no total strategy on the game $\perp$.

Completeness. When it comes to completeness, we run into the issue that the total winning innocent strategies are not necessarily finite, hence the usual definability process does not terminate. Nonetheless, we get a definability theorem in an infinitary version of μLJ. Whether a more precise completeness theorem is possible is a subtle point. First, we would need to restrict to an adequate subclass of the recursive total winning strategies (for example, the Ackermann function is definable in μLJ). Then again, the problem to find a proof whose interpretation is *exactly* the original strategy would be highly non-trivial: if $\sigma : \mu X.T \Rightarrow A$, we have to *guess* an invariant B, a proof π_1 of $T(B) \vdash B$ and a proof π_2 of $B \vdash A$ such that $[\![\pi_1]\!]^{\dagger}; [\![\pi_2]\!] = \sigma$. Perhaps it would be more feasible to look for a proof whose interpretation is *observationally equivalent* to the original strategy, which would be very similar to the universality result in [17].

4 Conclusion and Future Work

We have successfully constructed a games model of a propositional intuitionistic sequent calculus μLJ with inductive and coinductive types. It is striking that the adequate winning conditions on legal plays to model (co)induction are almost identical to those used in parity games to model least and greatest fixpoints, to the extent that the restriction of our winning condition to paths coincides *exactly* with the winning condition used in [24]. It would be worthwile to investigate this connection further: given a game viewed as a bipartite graph along with winning conditions for infinite plays, under which assumptions can these winning conditions be canonically lifted to the set of legal plays on this graph, viewed as an arena? Results in this direction might prove useful, since they would allow to import many game-theoretic results into game semantics, and thus programming languages.

This work is part of a larger project to provide game-theoretic models to total programming languages with dependent types, such as COQ or Agda. In these settings, (co)induction is crucial, since they deliberately lack general recursion. We believe that in the appropriate games setting, we can push the present results further and model Dybjer's Inductive-Recursive[11] definitions.

Acknowledgements. We would like to thank Russ Harmer, Stephane Gimenez and David Baelde for stimulating discussions, and the anonymous referees for useful comments and suggestions.

References

1. Abel, A., Altenkirch, T.: A predicative strong normalisation proof for a lambda-calculus with interleaving inductive types. In: Coquand, T., Nordström, B., Dybjer, P., Smith, J. (eds.) TYPES 1999. LNCS, vol. 1956, pp. 21–40. Springer, Heidelberg (2000)
2. Abramsky, S.: Semantics of interaction: an introduction to game semantics. In: Semantics and Logics of Computation, pp. 1–31 (1996)
3. Abramsky, S., Jagadeesan, R.: Games and full completeness for multiplicative linear logic. J. Symb. Log. 59(2), 543–574 (1994)
4. Abramsky, S., Jagadeesan, R., Malacaria, P.: Full Abstraction for PCF. Info. & Comp. (2000)
5. Abramsky, S., Kohei, H., McCusker, G.: A fully abstract game semantics for general references. In: LICS, pp. 334–344 (1998)
6. Baelde, D., Miller, D.: Least and greatest fixed points in linear logic. In: Dershowitz, N., Voronkov, A. (eds.) LPAR 2007. LNCS, vol. 4790, pp. 92–106. Springer, Heidelberg (2007)
7. Clairambault, P., Harmer, R.: Totality in Arena Games (submitted, 2008)
8. Conway, J.H.: On Numbers and Games. AK Peters, Ltd. (2001)
9. De Lataillade, J.: Second-order type isomorphisms through game semantics. Ann. Pure Appl. Logic 151(2-3), 115–150 (2008)
10. Dybjer, P.: Inductive sets and families in Martin-Löfs Type Theory and their set-theoretic semantics: An inversion principle for Martin-Lös type theory. Logical Frameworks 14, 59–79 (1991)
11. Dybjer, P.: A general formulation of simultaneous inductive-recursive definitions in type theory. J. Symb. Log. 65(2), 525–549 (2000)
12. Freyd, P.: Algebraically complete categories. In: Proc. 1990 Como Category Theory Conference, vol. 1488, pp. 95–104. Springer, Heidelberg (1990)
13. Godel, K.: Über eine bisher noch nicht bentzte Erweiterung des finiten Standpunktes. Dialectica (1958)
14. Harmer, R.: Innocent game semantics. Lecture notes (2004)
15. Harmer, R., Hyland, J.M.E., Melliès, P.-A.: Categorical combinatorics for innocent strategies. In: LICS, pp. 379–388 (2007)
16. Hyland, J.M.E.: Game semantics. Semantics and Logics of Computation (1996)
17. Hyland, J.M.E., Ong, C.H.L.: On full abstraction for PCF: I, II, and III. Inf. Comput. 163(2), 285–408 (2000)

18. Joyal, A.: Remarques sur la théorie des jeux à deux personnes. Gaz. Sc. Math. Qu. (1977)
19. Laurent, O.: Classical isomorphisms of types. Mathematical Structures in Computer Science 15(5), 969–1004 (2005)
20. Loader, R.: Equational theories for inductive types. Ann. Pure Appl. Logic 84(2), 175–217 (1997)
21. Lorenzen, P.: Logik und Agon. Atti Congr. Internat. di Filosofia (1960)
22. McCusker, G.: Games and full abstraction for FPC. Inf. Comput. 160(1-2), 1–61 (2000)
23. Melliès, P.-A.: Asynchronous games 4: A fully complete model of propositional linear logic. In: LICS, pp. 386–395 (2005)
24. Santocanale, L.: Free μ-lattices. J. Pure Appl. Algebra 168(2-3), 227–264 (2002)
25. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics 5(2), 285–309 (1955)

Full Abstraction for Reduced ML

Andrzej S. Murawski[⋆] and Nikos Tzevelekos

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD, UK

Abstract. We present the first effectively presentable fully abstract model for Stark's Reduced ML, the paradigmatic higher-order programming language combining call-by-value evaluation and integer-valued references. The model is constructed using techniques of nominal game semantics. Its distinctive feature is the presence of carefully restricted information about the store in plays, combined with conditions concerning the participants' ability to distinguish reference names. This leads to an explicit characterization of program equivalence.

1 Introduction

Reduced ML is a programming language introduced by Stark [22] as part of his investigations into generative aspects of programming languages. It combines higher-order functions with integer references and is defined simply by extending the call-by-value λ-calculus with primitives for integer and reference manipulation. Despite its economy, Reduced ML manages to embody several important paradigms (imperative programming, functional programming, call-by-value evaluation, scope extrusion), which makes it an attractive object for theoretical study. On the other hand, research into it offers wide scope for applicability, as Reduced ML is intimately related to Standard ML [15] and, in fact, has been designed with faithfulness to the latter in mind.

The first steps in the semantic analysis of Reduced ML were taken by Stark, who has identified a matching categorical framework and considered example categories, albeit without a general full abstraction result[1]. Further progress was possible with the arrival of game semantics [4,9,19]. Although the first papers concerned call-by-name computation, attention soon turned to the call-by-value framework [8,6]. In particular, Abramsky and McCusker presented a fully abstract model for a language called RML [6], which is essentially Reduced ML extended with the "bad-variable" constructor mkvar. Its presence is a consequence of adopting Reynolds's principle of modelling references as objects with read and write methods [21]. Thus, mkvar allows one to define terms of reference type that need not correspond to actual memory locations. Unfortunately, this affects

⋆ Supported by an EPSRC Advanced Research Fellowship (EP/C539753/1).

[1] A denotational model of a programming language is *fully abstract* iff equality of denotations coincides with program equivalence. Programs are *equivalent* iff they can be used interchangeably without observable differences.

the induced notion of program equivalence, so the full abstraction result of [6] does not apply to Reduced ML. More precisely, it can fail at types containing occurrences of $\mathsf{int\,ref}$. Typical counterexamples are the failures of equivalences between $x := !x$ and () (the terminating command), or between $x := 1; x := 1$ and $x := 1$. In the former case the terms are inequivalent in RML, because x may be instantiated with a mkvar-object whose reading or writing method diverges, or causes side effects. Similarly, in the latter case, an assignment to a mkvar-object might trigger a side effect that effectively allows one to count how many assignments took place.

The "bad-variable" phenomenon, also present in the call-by-name setting, has inspired subsequent developments in game semantics. It turned out that, in the call-by-name framework, it could be circumvented by employing suitably crafted (pre)orders on plays [14,18], but no result of this kind has been reported for call-by-value. However, an alternative and general approach to dealing with bad variables seems to have emerged in the form of *nominal game semantics* [10,2,23]. Nominal game semantics advocates a departure from Reynolds's modelling rule and stipulates that reference types be modelled by names rather than objects. Using this approach, Laird showed a full abstraction result for a call-by-value language $\lambda\nu!$ with storable names rather than integers [10]. $\lambda\nu!$ turns out more expressive than Reduced ML in its ability to distinguish reference names and, consequently, the obvious adaptation of the model to Reduced ML results in a failure of full abstraction. This can be illustrated by the terms[2]

$$f : \mathsf{int\,ref} \to \mathsf{unit} \vdash \mathsf{let}\, n_1 = \mathsf{ref}\, 0 \,\mathsf{in}\, \mathsf{let}\, n_2 = \mathsf{ref}\, 0 \,\mathsf{in}\, ((f n_1); (n_2 := !n_1); n_2) : \mathsf{int\,ref}.$$

and $f : \mathsf{int\,ref} \to \mathsf{unit} \vdash \mathsf{let}\, n = \mathsf{ref}\, 0 \,\mathsf{in}\, (f n); n : \mathsf{int\,ref}$, which are equivalent in Reduced ML, but inequivalent[3] in $\lambda\nu!$. This is because a $\lambda\nu!$-context can detect the difference between n_1 from n_2 by storing the names and subsequently comparing them. In contrast, as our results confirm, the same effect cannot be achieved by a context belonging to Reduced ML.

Previous research into ML-like languages has also produced fully abstract game models for more significant extensions of Reduced ML, notably languages with higher-order references [3,23]. The first of these models suffers from the "bad-variable" problem outlined above. The second one, while adaptable to Reduced ML, leans rather too heavily on quotienting in order to achieve full abstraction (information on local state and store update is too explicit in the intensional model and leads to substantial undesirable distinctions). Therefore, it does not lead to an explicit characterization of program equivalence, which is obtained in the present paper.

Our point of departure is the observation that, although a Reduced ML program will in general not be able to keep track of all the names it encounters during the course of interaction with another program, at any given execution point there is a subset of such names that the program may have access to. In game

[2] $\mathsf{let}\, x = M \,\mathsf{in}\, N$ stands for the Reduced ML term $(\lambda x.N)M$.

[3] Strictly speaking, the terms are not in $\lambda\nu!$, but the scenario can be easily recast in $\lambda\nu!$ by replacing $\mathsf{ref}\, 0$ with $\nu n.n$.

semantics, using the notion of P-view, we can describe this set conservatively as one consisting of names that occur in the current P-view as well as those that the program created itself. We call such names P-available. Intuitively, whenever a program returns a name, it will be P-available. A corresponding condition inside our model will be called *P-availability*.

As a consequence, since a program cannot have access to reference names that are not P-available, its immediate behaviour will be independent of the associated values kept in the store, because the program is simply unable to read them. This leads us to found our game model on justified sequences with partial information about the store, restricted to P-available names. Note that this form of representation also conveys the idea that the program might depend on former (possibly outdated) values of currently unavailable references, recorded when the references were still available.

Unfortunately, P-availability and partiality of store alone do not yet suffice to establish a definability result. As our example demonstrates, a Reduced ML context may be unable to distinguish some occurrences of names introduced by the environment. In game semantics, we can capture this oversight in concrete terms: two (occurrences of) such names are indistinguishable to the program iff they have never occurred within the same P-view. Consequently, regardless of whether such occurrences are the same or not, the program's behaviour should remain the same. We formalize this observation via a saturation condition, called *blindness*, and show that any finitary strategy subject to all the conditions discussed above is definable, i.e. is a denotation of a Reduced ML term. This naturally leads to a fully abstract model via the usual *intrinsic quotient* construction.

To obtain a more accessible account of program equivalence we next examine the structure of the quotient in more detail. Crucially, we observe that blind strategies are determined uniquely by plays in which the environment provides a fresh name each time the name cannot be related by the program to any existing names. We call such plays *strict*. Then, by symmetrizing the model, we eventually obtain an explicit characterization of equivalence: terms of Reduced ML are equivalent iff they induce the same *mutually strict complete protoplays* (complete plays where O plays only O-available names and in which stores are restricted to *mutually available* names).

For example, *each* of the two terms introduced above generates the following such plays

$$\begin{tikzcd} * & n_1^{(n_1,0)} \arrow[l] & *^{(n_1,k)} \arrow[l, dash] & n_2^{(n_2,k)} \arrow[lll, bend right] \end{tikzcd},$$

where k ranges over the set of integers. Hence, the terms are indeed equivalent.

Notes. A long version with proofs is available from the authors' webpages. We are grateful to Jim Laird for email discussions.

2 Reduced ML

Reduced ML is the call-by-value λ-calculus over the ground types unit, int, int ref augmented with basic commands (termination, divergence), primitives for inte-

$$\frac{}{u,\Gamma\vdash ():\mathsf{unit}}\qquad \frac{}{u,\Gamma\vdash \Omega:\mathsf{unit}}\qquad \frac{i\in\mathbb{Z}}{u,\Gamma\vdash i:\mathsf{int}}\qquad \frac{l\in u}{u,\Gamma\vdash l:\mathsf{int\,ref}}\qquad \frac{(x:\theta)\in\Gamma}{u,\Gamma\vdash x:\theta}$$

$$\frac{u,\Gamma\vdash M_1:\mathsf{int}\qquad u,\Gamma\vdash M_2:\mathsf{int}}{u,\Gamma\vdash M_1\oplus M_2:\mathsf{int}}\qquad \frac{u,\Gamma\vdash M:\mathsf{int}\quad u,\Gamma\vdash N_0:\theta\quad u,\Gamma\vdash N_1:\theta}{u,\Gamma\vdash \mathsf{if}\,M\,\mathsf{then}\,N_1\,\mathsf{else}\,N_0:\theta}$$

$$\frac{u,\Gamma\vdash M:\mathsf{int\,ref}}{u,\Gamma\vdash !M:\mathsf{int}}\qquad \frac{u,\Gamma\vdash M:\mathsf{int\,ref}\quad u,\Gamma\vdash N:\mathsf{int}}{u,\Gamma\vdash M:=N:\mathsf{unit}}\qquad \frac{u,\Gamma\vdash M:\mathsf{int}}{u,\Gamma\vdash \mathsf{ref}\,M:\mathsf{int\,ref}}$$

$$\frac{u,\Gamma\vdash M:\theta\to\theta'\quad u,\Gamma\vdash N:\theta}{u,\Gamma\vdash MN:\theta'}\qquad \frac{u,\Gamma\cup\{x:\theta\}\vdash M:\theta'}{u,\Gamma\vdash \lambda x^{\theta}.M:\theta\to\theta'}$$

Fig. 1. Syntax of Reduced ML

ger arithmetic (constants, zero-test, binary integer functions) and reference manipulation (locations, dereferencing, assignment, memory allocation). The typing rules are given in Figure 1, where $\mathcal{L}$ stands for a countable set of *locations*, u for a finite subset of $\mathcal{L}$, and $\oplus$ for binary integer functions (e.g. $+$, $-$, $*$, $=$). Their precise choice is to some extent immaterial: for the full abstraction argument to hold it suffices to be able to compare integer variables with integer constants and act on the result. The same can be said about the lack of recursion, which can be added without affecting our results. Note that we did not include reference equality testing, because it is expressible [20]. For instance, one can define $\mathsf{eq}:\mathsf{int\,ref}\to\mathsf{int\,ref}\to\mathsf{int}$ as

$$\begin{aligned}\lambda x^{\mathsf{int\,ref}}.\lambda y^{\mathsf{int\,ref}}.\ &\mathsf{let}\,v_x=\mathsf{ref}\,!x\,\mathsf{in}\\ &\mathsf{let}\,v_y=\mathsf{ref}\,!y\,\mathsf{in}\\ &\mathsf{let}\,b=\mathsf{ref}\,0\,\mathsf{in}\\ &\quad(x:=0;y:=1;(\mathsf{if}\,!x=1\,\mathsf{then}\,b:=1\,\mathsf{else}\,());x:=!v_x;y:=!v_y;!b)\end{aligned}$$

In the above and in what follows, we write $M;N$ for the term $(\lambda z^{\theta}.N)M$, where z does not occur in N and θ matches the type of M.

To define the operational semantics of Reduced ML, we need to introduce a notion of store. A *store* will simply be a function from a finite set of locations to $\mathbb{Z}$. We write $s(l\mapsto i)$ for the store obtained by updating s so that l is mapped to i (this may extend the domain of s). Given a store $s:\{l_1,\cdots,l_n\}\to\mathbb{Z}$ and a term M we say that the pair (s,M) is *compatible* iff all locations occurring in M are from $\{l_1,\cdots,l_n\}$. We say that a term is *canonical* if it is either $()$, an integer constant, a location, a variable or a λ-abstraction. The big-step reduction rules are given as judgements of the shape $s,M\Downarrow s',V$, where (s,M), (s',V) are compatible, $\mathsf{dom}\,s\subseteq\mathsf{dom}\,s'$ and V is canonical. We present them in Figure 2, where we let l range over locations. Most rules take the form

$$\frac{M_1\Downarrow V_1\quad M_2\Downarrow V_2\quad\cdots\quad M_n\Downarrow V_n}{M\Downarrow V}$$

which is meant to abbreviate

$$\frac{s_1,M_1\Downarrow s_2,V_1\quad s_2,M_2\Downarrow s_3,V_2\quad\cdots\quad s_n,M_n\Downarrow s_{n+1},V_n}{s_1,M_1\Downarrow s_{n+1},V}.$$

$$\frac{V \text{ is canonical}}{s, V \Downarrow s, V} \qquad \frac{M \Downarrow 0 \quad N_0 \Downarrow V}{\mathsf{if}\, M\, \mathsf{then}\, N_1\, \mathsf{else}\, N_0 \Downarrow V} \qquad \frac{i \neq 0 \quad M \Downarrow i \quad N_1 \Downarrow V}{\mathsf{if}\, M\, \mathsf{then}\, N_1\, \mathsf{else}\, N_0 \Downarrow V}$$

$$\frac{M_1 \Downarrow i_1 \quad M_2 \Downarrow i_2}{M_1 \oplus M_2 \Downarrow i_1 \oplus i_2} \qquad \frac{M \Downarrow \lambda x.M' \quad N \Downarrow V' \quad M'[V'/x] \Downarrow V}{MN \Downarrow V}$$

$$\frac{s, M \Downarrow s', i \quad l \notin \mathsf{dom}\, s'}{s, \mathsf{ref}\, M \Downarrow s'(l \mapsto i), l} \qquad \frac{s, M \Downarrow s', l \quad s'(l) = i}{s, !M \Downarrow s', i} \qquad \frac{s, M \Downarrow s', l \quad s', N \Downarrow s'', i}{s, M := N \Downarrow s''(l \mapsto i), ()}$$

Fig. 2. Big-step operational semantics of Reduced ML

In particular, this means that the ordering of the hypotheses is significant. We shall write $\Gamma \vdash M : \theta$ iff $\emptyset, \Gamma \vdash M : \theta$ can be derived using the rules of Figure 1. Similarly, $\vdash M : \theta$ is shorthand for $\emptyset, \emptyset \vdash M : \theta$. Given $\vdash M : \mathsf{unit}$ we write $M \Downarrow$ iff $\emptyset, M \Downarrow s, ()$ for some store s.

Definition 1. *We say that the term-in-context* $\Gamma \vdash M_1 : \theta$ **approximates** $\Gamma \vdash M_2 : \theta$ *(written* $\Gamma \vdash M_1 \sqsubseteq\!\!\!\sim M_2$*) iff* $C[M_1] \Downarrow$ *implies* $C[M_2] \Downarrow$ *for any context* $C[-]$ *such that* $\vdash C[M_1], C[M_2] : \mathsf{unit}$*. Two terms-in-context are* **equivalent** *if one approximates the other (written* $\Gamma \vdash M_1 \cong M_2$*).*

The only difference between our definition of Reduced ML and Stark's is the presence of Ω, the divergent constant without a reduction rule. Thanks to it, we can define $\sqsubseteq\!\!\!\sim$ and reason about $\cong$ in a more concise way. At the same time, program equivalence of Ω-free Reduced ML terms remains unaffected, because $C[M] \Downarrow$, where M is Ω-free, is equivalent to $\emptyset, C'[M] \Downarrow s', 0$ where $C'[-] \equiv \mathsf{let}\, x = \mathsf{ref}\, 0\, \mathsf{in}\, C[x := 1/\Omega][M]; !x$ and s' is a state.

3 Nominal Game Semantics

We begin this section with a brief review of the fundamentals of nominal game semantics [2,11,24]. Let us fix a countably infinite set $\mathbb{A}$, the set of *atoms*, the elements of which we denote by a, b, c, n and variants. In nominal game semantics two participants play a game by exchanging moves that might involve atoms. However, when employing such moves, we are not interested in what exactly the names are, though we would like to know how they relate to names that have already been in play. Hence, the objects of study are rather the induced equivalence classes with respect to name-invariance. Since we want all game-semantic notions and constructions to be compatible with name-invariance, their obvious adaptations would repeatedly have to include conditions that enforce closure under name-renamings. Fortunately, this overhead can be dealt with robustly using the language of nominal set theory [7].

Definition 2. *Let us write* PERM($\mathbb{A}$) *for the group of finite permutations of* $\mathbb{A}$*. A* **nominal set** X *is a set* $|X|$ *(usually written* X*) equipped with a group* *action of* PERM($\mathbb{A}$)[4]. *Moreover, each* $x \in X$ *must have* finite support*, that is, there exists a finite set* $S \subseteq \mathbb{A}$ *such that, for all permutations* π*,* $(\forall a \in S.\, \pi(a) = a) \implies \pi \cdot x = x$.

Finite support is closed under intersection, and hence each element x of a nominal set has a least support $\nu(x)$, which we call ***the support of*** x. Intuitively, $\nu(x)$ is the set of names "involved" in x. Accordingly, we say that *a is fresh for x* if $a \notin \nu(x)$. Clearly, $\mathbb{A}$ is a nominal set by taking $\pi \cdot a = \pi(a)$, for each π and a. More interestingly, so is the set $\mathbb{A}^*$ of finite lists of atoms with permutations acting elementwise. If X and Y are nominal sets then so is their cartesian product $X \times Y$, with permutations acting componentwise, and their disjoint union $X \uplus Y$. Moreover, $X' \subseteq X$ is a *nominal subset* of X if X' is closed under permutation actions, these acting as on X. Then we can define $R \subseteq X \times Y$ to be a *nominal relation* iff R a nominal subset of $X \times Y$. A *nominal function* is a function which is also a nominal relation.

In game semantics a particular strengthening of the notion of support, called *strong support*, has turned out necessary to guarantee correct behaviour under strategy composition (see [24] for motivation and a detailed explanation of its significance). Here we consider an even stronger notion of support, one in which the support of each element can be linearly ordered in a canonical, nominal manner. A nominal set X is called a ***linear nominal set*** if, for each element x of X, there exists a linear order $<_x$ on $\nu(x)$ such that, for all $a, b \in \mathbb{A}$, $a <_x b$ implies $\pi(a) <_{\pi \cdot x} \pi(b)$ for any permutation π[5]. It is easy to check that all elements in a linear nominal set have strong support. For example, the nominal set $\mathbb{A}^*$ is linear, whereas $\mathcal{P}_{fin}(\mathbb{A})$ is not. A nominal subset of a linear nominal set is itself linear. Moreover, by straightforward manipulations of the orderings available for linear X, Y we can render the nominal sets $X \times Y$ and $X \uplus Y$ linear.

Finally, in nominal sets we can *define* atom-abstractions. The form of abstraction we will be using is that of complete support abstraction, that is, for a nominal set X and $x \in X$, we define $[x]$ to be $\{y \in X \mid \exists \pi.\, y = \pi \cdot x\}$.

3.1 Nominal Arenas

Here we present nominal arenas (and prearenas), which are essentially the call-by-value arenas of Honda and Yoshida [8] cast inside the theory of nominal sets.

Definition 3. *An* **arena** $A = (M_A, I_A, \vdash_A, \lambda_A)$ *is given by:*

- *a linear nominal set* M_A *of moves,*
- *a nominal subset* $I_A \subseteq M_A$ *of initial moves,*
- *a nominal justification relation* $\vdash_A \subseteq M_A \times (M_A \setminus I_A)$*,*
- *a nominal labelling function* $\lambda_A : M_A \to \{O, P\} \times \{Q, A\}$*;*

[4] A group action of PERM($\mathbb{A}$) on X is a function $\_\cdot\_ : \text{PERM}(\mathbb{A}) \times X \to X$ such that, for all $x \in X$ and $\pi, \pi' \in \text{PERM}(\mathbb{A})$, $\pi \cdot (\pi' \cdot x) = (\pi \circ \pi') \cdot x$ and $\mathsf{id} \cdot x = x$, where id is the identity permutation.

[5] Equivalently, the relation $\{(a, b, x) \mid a, b \in \mathbb{A},\, x \in X,\, a <_x b\}$ is nominal.

satisfying, for each $m, m' \in M_A$, the conditions:

- $m \in I_A \implies \lambda_A(m) = (P, A)$,
- $m \vdash_A m' \wedge \lambda_A^{QA}(m) = A \implies \lambda_A^{QA}(m') = Q$,
- $m \vdash_A m' \implies \lambda_A^{OP}(m) \neq \lambda_A^{OP}(m')$.

The role of λ_A is to label moves as *Opponent* or *Proponent* moves and as *Questions* or *Answers*. The simplest arena is $0 = (\emptyset, \emptyset, \emptyset, \emptyset)$. Other "flat" arenas are 1, $\mathbb{Z}$ and $\mathbb{A}$, defined by $M_1 = I_1 = \{*\}$, $M_{\mathbb{Z}} = I_{\mathbb{Z}} = \mathbb{Z}$, $M_{\mathbb{A}} = I_{\mathbb{A}} = \mathbb{A}$.

We take advantage of the following constructions on arenas. By $\bar{I}_A$ we denote $M_A \setminus I_A$, by $\bar{\lambda}_A$ the OP-complement of λ_A; i_A and i_B range over initial moves in the respective arenas.

$$\begin{aligned}
M_{A\otimes B} &= (I_A \times I_B) \uplus \bar{I}_A \uplus \bar{I}_B \\
I_{A\otimes B} &= I_A \times I_B \\
\lambda_{A\otimes B} &= [((i_A, i_B), PA), \lambda_A \upharpoonright \bar{I}_A, \lambda_B \upharpoonright \bar{I}_B] \\
\vdash_{A\otimes B} &= \{((i_A, i_B), m) \mid i_A \vdash_A m \text{ or } i_B \vdash_B m\} \cup (\vdash_A \upharpoonright \bar{I}_A^{\,2}) \cup (\vdash_B \upharpoonright \bar{I}_B^{\,2})
\end{aligned}$$

$$\begin{aligned}
M_{A\Rightarrow B} &= \{*\} \uplus I_A \uplus \bar{I}_A \uplus M_B \\
I_{A\Rightarrow B} &= \{*\} \\
\lambda_{A\Rightarrow B} &= [(*, PA), (i_A, OQ), \bar{\lambda}_A \upharpoonright \bar{I}_A, \lambda_B] \\
\vdash_{A\Rightarrow B} &= \{(*, i_A), (i_A, i_B)\} \cup \vdash_A \cup \vdash_B
\end{aligned}$$

The types of Reduced ML will be interpreted by arenas in the following way: $[\![\mathsf{unit}]\!] = 1$, $[\![\mathsf{int}]\!] = \mathbb{Z}$, $[\![\mathsf{int\,ref}]\!] = \mathbb{A}$ and $[\![\theta_1 \to \theta_2]\!] = [\![\theta_1]\!] \Rightarrow [\![\theta_2]\!]$. Although types are interpreted by arenas, the actual games will be played in ***prearenas***, which are defined in the same way as arenas with the exception that initial moves are O-questions. For given arenas A, B we can construct a prearena $A \to B$ by

$$\begin{aligned}
M_{A\to B} &= M_A \uplus M_B & \lambda_{A\to B} &= [(i_A, OQ) \cup (\bar{\lambda}_A \upharpoonright \bar{I}_A)\ ,\ \lambda_B] \\
I_{A\to B} &= I_A & \vdash_{A\to B} &= \{(i_A, i_B)\} \cup \vdash_A \cup \vdash_B\ .
\end{aligned}$$

Typing judgements $\Gamma \vdash \theta$, where $\Gamma = \{x_1 : \theta_1, \cdots, x_n : \theta_n\}$, will eventually be interpreted by strategies for the prearena $[\![\theta_1]\!] \otimes \cdots \otimes [\![\theta_n]\!] \to [\![\theta]\!]$ (if $n = 0$ we take the left-hand side to be 1), which we shall denote by $[\![\Gamma \vdash \theta]\!]$.

3.2 Plays

Analogously to the definition of store in Section 2, in this section a store will be a partial function $S : \mathbb{A} \rightharpoonup \mathbb{Z}$ such that $\mathsf{dom}\, S$ is finite.

A ***basic justified sequence*** in a prearena A is a finite sequence s of moves of A satisfying the following conditions: the first move must be initial, but all other moves m must be equipped with a pointer to an earlier occurrence of another move m' such that $m' \vdash_A m$ (we then say that m' justifies m; if m is an answer, we might also say that m *answers* m'). A ***justified sequence*** in A is a basic justified sequence s in which each move is, in addition, decorated with a store to yield a *move-with-store*, typically denoted by m^S. Given a justified sequence s, we write $\underline{s}$ for the underlying basic justified sequence. It should be clear that, similarly to the set of finite sequences of moves, the set of justified sequences can be viewed as a (not necessarily linear) nominal set with permutations preserving the pointer structure, but acting on moves as in A and on stores by permuting the domain.

Below we define the notions of O-view $\llcorner s \lrcorner$ and P-view $\ulcorner s \urcorner$ of a justified sequence, using o and p to range over O-moves and P-moves respectively. We write $s' \sqsubseteq s$ if s' is a prefix of s and use $\sqsubseteq^{\text{even}}$ if s' is of even length.

$$\begin{array}{rclcrcl}
\llcorner \epsilon \lrcorner &=& \epsilon & \qquad & \ulcorner \epsilon \urcorner &=& \epsilon \\
\llcorner s\, o^S \lrcorner &=& \llcorner s \lrcorner\, o^S & & \ulcorner s\, p^S \urcorner &=& \ulcorner s \urcorner\, p^S \\
\llcorner s\, o^S \overparen{t}\, p^{S'} \lrcorner &=& \llcorner s \lrcorner\, o^S p^{S'} & & \llcorner s\, p^S \overparen{t}\, o^{S'} \lrcorner &=& \llcorner s \lrcorner\, p^S o^{S'}
\end{array}$$

A name in s is said to be ***introduced by player*** X ($X \in \{O, P\}$) iff its first occurrence in $\underline{s}$ is in (the support of) an X-move. Names introduced by X in s will be referred to as ***X-names*** in s and denoted with $\mathsf{X}(s)$. We define the set $\mathsf{Av}_X(s)$ of ***X-available names after*** s by:

$$\mathsf{Av}_{\mathrm{O}}(s) = \mathsf{O}(s) \cup \nu(\llcorner \underline{s} \lrcorner) \qquad \mathsf{Av}_{\mathrm{P}}(s) = \mathsf{P}(s) \cup \nu(\ulcorner \underline{s} \urcorner)\,.$$

Definition 4. *A justified sequence is* legal *iff it satisfies the following conditions.*

Alternation. *No two adjacent moves belong to the same player.*

Bracketing. *The justifier of each answer is the most recent unanswered question.*

Visibility. *For any $tm^S \sqsubseteq s$, the justifier of m is in $\llcorner tm^S \lrcorner$ if m is an O-move and in $\ulcorner tm^S \urcorner$ otherwise.*

Frugality. *For any $tm^S \sqsubseteq s$, $\mathsf{dom}\, S \subseteq \nu(\underline{tm^S})$.*

The set of legal justified sequences will be denoted by L_A.

Note that legal sequences contain those of [11]. Because of frugality, the support of a legal sequence is that of its underlying basic sequence, and therefore L_A is a linear nominal set. Our model will be based on still more restrictive plays.

Definition 5. *A legal sequence s is a* **play** *iff it satisfies the following two conditions.*

P-availability. *For each $s'p^S \sqsubseteq^{even} s$ and any $a \in \mathbb{A}$, if $a \in \nu(p) \cap \nu(s')$ then $a \in \mathsf{Av}_{\mathrm{P}}(s')$.*

P-storage. *For any $s'm^S \sqsubseteq s$, $\mathsf{dom}\, S = \mathsf{Av}_{\mathrm{P}}(s'm^S)$.*

The set of plays over prearena A will be denoted by P_A.

Note that the two conditions are biased towards P. Equivalently, P-availability can be restated as: for any $s'p^S \sqsubseteq^{\text{even}} s$ and any $a \in \nu(p)$, if $a \in \mathsf{O}(s')$ then $a \in \nu(\underline{\ulcorner s' \urcorner})$. It is worth observing that, given $s \in P_A$, we have $\mathsf{Av}_{\mathrm{P}}(s) = \nu(\ulcorner s \urcorner)$.

3.3 Strategies

Definition 6. *A* **strategy** σ *on a prearena* A*, written* $\sigma : A$*, is a set of equivalence classes* $[s]$ *of even-length* plays *of* A *satisfying*

Even-prefix closure. *If* $[so^S p^{S'}] \in \sigma$ *then* $[s] \in \sigma$ *,*
Determinacy. *If* $[sp_1^{S_1}], [s'p_2^{S_2}] \in \sigma$ *and* $[s] = [s']$ *then* $[sp_1^{S_1}] = [s'p_2^{S_2}]$ *.*

Next we show how strategies can be composed. First, following [11], let us define an endofunction γ on justified sequences that restricts a given justified sequence to a frugal one by removing from the stores the atoms violating it. Now, let γ' be an analogous *partial* function enforcing P-storage, i.e. γ' will remove O-names violating P-storage (it is undefined when the domain of any of the stores involved contains an insufficient supply of atoms, i.e. some of the P-available names required are missing).

Now we turn to defining a suitable notion of interaction between plays. Given arenas A, B, C, let u be a sequence $m_1^{S_1} \cdots m_k^{S_k}$ of moves from $M_A + M_B + M_C$ with store, equipped with pointers such that no C-move has a pointer to an A-move and vice versa. We define $u \upharpoonright A, B$ to be u in which all C-moves are suppressed along with associated pointers. $u \upharpoonright B, C$ is defined analogously. $u \upharpoonright A, C$ is defined similarly with the caveat that, if there was a pointer from a C-move to a B-move which in turn had a pointer to an A-move, we add a pointer from the C-move to the A-move. By $u_{\leq m_i}$ we mean the initial segment of u ending in $m_i^{S_i}$.

Definition 7. *u is an interaction sequence of A, B, C iff* $\gamma'(u \upharpoonright A, B) \in P_{A \to B}$*,* $\gamma'(u \upharpoonright B, C) \in P_{B \to C}$ *and the following conditions hold:*

- $\mathsf{P}(u \upharpoonright A, B) \cap \mathsf{P}(u \upharpoonright B, C) = \emptyset$*;*
- $\mathsf{O}(u \upharpoonright A, C) \cap (\mathsf{P}(u \upharpoonright A, B) \cup \mathsf{P}(u \upharpoonright B, C)) = \emptyset$*;*
- *for each* $u' \sqsubseteq u$ *ending in a move-with-store* m^S*,*
  $\mathsf{dom}\, S = (\mathsf{O}(u' \upharpoonright A, C) \cap \nu(\ulcorner \underline{u'} \upharpoonright A, C \urcorner)) \cup \mathsf{P}(u' \upharpoonright A, B) \cup \mathsf{P}(u' \upharpoonright B, C)$*;*
- *for each* $u' \sqsubseteq u$ *ending in* $m^S m'^{S'}$*, if* m' *is:*
  - *a P-move in* $A \to B$ *then* $S'(a) = S(a)$ *for all* $a \in \mathsf{dom}\, S' \setminus \mathsf{Av}_{\mathrm{P}}(u' \upharpoonright A, B)$*;*
  - *a P-move in* $B \to C$ *then* $S'(a) = S(a)$ *for all* $a \in \mathsf{dom}\, S' \setminus \mathsf{Av}_{\mathrm{P}}(u' \upharpoonright B, C)$*;*
  - *an O-move in* $A \to C$ *then* $S'(a) = S(a)$ *for all* $a \in \mathsf{dom}\, S' \setminus \mathsf{Av}_{\mathrm{P}}(u' \upharpoonright A, C)$*.*

The set of all interaction sequences of A, B, C *will be denoted by* $Int(A, B, C)$*.*

The first two conditions ensure that name-privacy is not broken under composition; the third one imposes an extended notion of P-availability for sequences; and the fourth set of conditions ensures that participants do not change parts of the store inaccessible to them. It can be shown that, if $u \in Int(A, B, C)$ then $\gamma(u \upharpoonright A, C) \in P_{A \to C}$. Two strategies $\sigma : A \to B$ and $\tau : B \to C$ can now be composed as follows

$$\sigma;\tau = \{[\gamma(u \upharpoonright A, C)] \mid u \in Int(A, B, C),\ [\gamma'(u \upharpoonright A, B)] \in \sigma,\ [\gamma'(u \upharpoonright B, C)] \in \tau\}.$$

Associativity of composition can be established using similar arguments to those in [11][6]. Using the standard definition of the identity strategy one can then obtain a category of arenas and games. Next we shall define its lluf subcategory that will be used to prove the full abstraction result.

Given a non-empty justified sequence s, let us write s^- for s without its last element. The following definition aims to capture plays that differ by renamings of names that O introduces in the P-view.

Definition 8

- *Given a prearena A, $s \in P_A$, $a \in \mathsf{O}(s)$ and an O-move o in s, we say that a is* **P-new at** *o* **in** *s iff $a \in \nu(o)$ and $a \notin \nu(\ulcorner s_{\leq o} \urcorner^-)$.*
- *Given A, s, a, o as above and $b \in \mathbb{A}$, we say that a is* **renameable for** *b* **at** *o* **in** *s provided $b \notin \mathsf{P}(s)$ and, for any $s' \sqsubseteq s$, if o occurs in $\ulcorner s' \urcorner$ then $b \notin \nu(\ulcorner s' \urcorner)$.*
- *Under the assumptions above, we define the* **renaming** *$(a\ b)_o \cdot s$ of s by induction on the subsequences of s[7]:*

$$(a\ b)_o \cdot \epsilon = \epsilon \qquad (a\ b)_o \cdot (tm^S) = \begin{cases} ((a\ b)_o \cdot t)\,((a\ b) \cdot m^S) & o \in \ulcorner tm^S \urcorner \\ ((a\ b)_o \cdot t)\, m^S & o \notin \ulcorner tm^S \urcorner \end{cases}$$

where $(a\ b)$ is the permutation swapping a with b. We write $s \overset{r}{\sim} s'$ iff s can be obtained from s' through a sequence of renamings.

Observe that, if a is P-new at m in s, a need not be fresh for $s_{<m}$ (the converse holds, though, as long as $a \in \nu(m)$). A play s in which every a that is P-new at m in s is also fresh at $s_{<m}$ will be called ***strict***.

Example 9. Let $A = \mathbb{A} \to (\mathbb{A} \Rightarrow 1)$,

$$s_1 = n_1^{(n_1,0)}\ *^{(n_1,1)}\ n_2^{(n_1,2),(n_2,3)}\ *^{(n_1,4),(n_2,5)}\ n_3^{(n_1,6),(n_3,7)}\ *^{(n_1,8),(n_3,9)}$$

and

$$s_2 = n_1^{(n_1,0)}\ *^{(n_1,1)}\ n_3^{(n_1,2),(n_3,3)}\ *^{(n_1,4),(n_3,5)}\ n_3^{(n_1,6),(n_3,7)}\ *^{(n_1,8),(n_3,9)}\,.$$

Then n_2 is P-new at the third move (also n_2) in s_1, n_2 is renameable for n_3 at that move and $(n_2\ n_3)_{n_2} \cdot s_1 = s_2$. Note also that $(n_3\ n_2)_{n_3} \cdot s_2 = s_1$, where the subscript n_3 stands for the third move of s_2, and that s_1 is strict, whereas s_2 is not.

In general it can be shown that renamings are reversible, so $\overset{r}{\sim}$ is an equivalence relation. Observe that, for any play s, there exists a *strict* play s' (determined uniquely up to atom-abstraction) such that $s \overset{r}{\sim} s'$. We write $\widetilde{s}$ for $[s']$.

Definition 10. *A strategy $\sigma : A$ is* **blind** *iff $[s] \in \sigma$ and $s \overset{r}{\sim} s'$ imply $[s'] \in \sigma$.*

Since the identity strategy is blind and blind strategies compose we obtain a category $\mathcal{G}$ of arenas and blind strategies. Observe that blind strategies are uniquely determined by the underlying strict positions (via renamings).

[6] In fact, strategies in our framework can be regarded as compact representations of a class of strategies from [11], namely those independent of P-unavailable names.

[7] We write $o \in s$ to mean that the distinguished occurrence of o is present in s.

4 Properties of $\mathcal{G}$

Henceforth, when writing $\sigma : A$ we shall mean a blind strategy on A. Following [2] and [11], $\mathcal{G}$ can be shown equivalent to the Klesli category of another category $\mathcal{G}'$ equipped with a strong monad T. More precisely, $\mathcal{G}'$ is a lluf subcategory of $\mathcal{G}$ consisting of total single-threaded strategies [11] such that store values introduced in the first move cannot be modified in the next two moves. The strong monad T takes an arena A to $A_\perp$ given by

$$\begin{aligned} M_{A_\perp} &= \{*_1, *_2\} + M_A & I_{A_\perp} &= \{*_1\} \\ \lambda_{A_\perp} &= [\{(*_1, PA), (*_2, OQ)\}, \lambda_A] & \vdash_{A_\perp} &= \{(*_1, *_2), (*_2, i_A)\} \cup \vdash_A \ . \end{aligned}$$

Moreover, one can show that $\mathcal{G}'$ has finite products and T-exponentials, i.e., for any arena A, there is a natural bijection between $\mathcal{G}'(A \otimes B, TC)$ and $\mathcal{G}'(A, B \Rightarrow C)$. That is to say, $\mathcal{G}'$ is λ_c-model [16], which gives a canonical interpretation of the call-by-value λ-calculus in the associated Kleisli category, i.e., equivalently, the category $\mathcal{G}$. To interpret the remaining constructs of Reduced ML in $\mathcal{G}$, we follow Stark by showing the existence of special morphisms, as described in Chapter 5 of [22]. We list those related to reference manipulation below (as morphisms in $\mathcal{G}$ rather than in $\mathcal{G}'_T$).

$$get = \{[\epsilon], [n^{(n,i)} i^{(n,i)}]\} : \mathbb{A} \to \mathbb{Z} \qquad set = \{[\epsilon], [(n,i)^{(n,i')} *^{(n,i)}]\} : \mathbb{A} \otimes \mathbb{Z} \to 1$$

Memory allocation is interpreted using the strategies $new_i = \{[\epsilon], [* n^{(n,i)}]\} : 1 \to \mathbb{A}$. As a consequence, we conclude that $\mathcal{G}$ is a model of Reduced ML in the sense of Stark[8]. This lets us interpret any term-in-context $\Gamma \vdash M : \theta$ with a strategy $[\![\Gamma \vdash M : \theta]\!] : [\![\Gamma \vdash \theta]\!]$.

Example 11. The two terms from the introduction are interpreted (in $\mathcal{G}$) by the strategies given respectively (through even-prefix closure) by the plays below.

$$* \ n_1^{(n_1,0)} - *^{(n_1,k)} \ \ n_2^{(n_1,k),(n_2,k)} \qquad\qquad * \ n^{(n,0)} - *^{(n,k)} \ \ n^{(n,k)}$$

Conformance to Stark's framework guarantees Computational Soundness and Adequacy [22].

Definition 12

- $\sigma : A$ *is* **finitary** *iff it is finite.*
- $\sigma : A$ *is* **strongly deterministic** *iff, for any* $s \in P_A$ *such that* $[s] \in \sigma$, *we have* $\mathsf{P}(s) = \emptyset$.
- *A strategy* $\sigma : A$ *is* **innocent** *iff,* $[sp], [t] \in \sigma$, $to \in P_A$, $\ulcorner s \urcorner = \ulcorner to \urcorner$ *implies the existence of* $[top'] \in \sigma$ *such that* $[\ulcorner sp \urcorner] = [\ulcorner top' \urcorner]$ *(note that innocence implies blindness). An innocent strategy* $\sigma : A$ *is* finitarily innocent *iff* $\mathsf{vf}(\sigma) = \{[\ulcorner s \urcorner] \mid s \in P_A, [s] \in \sigma\} : A$ *is finite.*

[8] Strictly speaking, the cartesian closure requirement from [22] is not satisfied, but it turns out too strong: T-exponentials suffice for the author's subsequent results [2].

Using two factorizations we can show that any finitary blind strategy in a denotable[9] prearena is definable. The first one eliminates violations of strong determinism with the help of new_0 (corresponding to $\mathsf{ref}\,0$). The second one factors out non-innocence (also using new_0). Finally, we prove a direct definability result for finitarily innocent strongly deterministic strategies. We discuss the innocent factorization in more detail below, as it involves the key novelties of our framework.

Lemma 13. *Let $\sigma : 1 \to A$ be a finitary strongly deterministic blind strategy. There exists a finitarily innocent strongly deterministic strategy $\dot{\sigma} : \mathbb{A} \to A$ such that $new_0; \dot{\sigma} = \sigma$.*

The standard way [5] of proving such results is to store the history of play using the additional $\mathbb{A}$ component. This is impossible in our case, because atoms cannot be stored. However, given a play s, we can try to store a numerical representation of $[s]$ instead. Recall that the set of moves of an arena is a linear nominal set, i.e. there is a *canonical* ordering of atoms in any move. Hence, in any legal sequence, atoms can also be ordered in a *canonical* way according to their order of appearance and, if they were introduced in the same move, using the canonical ordering associated with that move. Consequently, we can represent $[s]$ as an integer by representing atoms in s with integers that correspond to their position in the associated canonical order and by encoding such a sequence as an integer. Let us write $\#(s)$ for such an encoding. In particular we have $\#(s_1) = \#(s_2)$ iff $[s_1] = [s_2]$.

Unfortunately, this is not yet sufficient for a successful factorization through an innocent strategy because, given $\ulcorner so^S \urcorner$ and $\#(s)$, it will in general be impossible to extract so^S (or $[so^S]$) due to the fact that o might contain O-names occurring in s, but not in $\ulcorner so^S \urcorner$. As a result, given $\#(s)$ and $\ulcorner so^S \urcorner$, we may then be unable to relate some names in o with those of s, which will prevent us from reconstructing $[so^S]$. Note, however, that given $\#(s)$ and $\ulcorner so^S \urcorner$ we can still determine $\widetilde{so^S}$ in absence of P-names. Furthermore, since σ is blind and strongly deterministic (in particular plays satisfy P-availability), we can uniquely identify $p^{S'}$ such that $so^S p^{S'} \in \sigma$ (though not necessarily the whole of $so^S p^{S'}$), by referring[10] to $\widetilde{so^S}$ and σ. Analogously, we can also deduce $\widetilde{so^S p^{S'}}$. Thus, the familiar factorization technique can be employed provided that, instead of $\#(s)$, the argument will rely on $\#(\widetilde{s})$, where $\#(\widetilde{s})$ stands for $\#(s')$ and s' is a strict play such that $s' \overset{r}{\sim} s$ (by previous remarks the code is independent of the exact choice of s').

Thanks to the definability result for finitary blind strategies, we can define a fully abstract model of Reduced ML in the usual way by quotienting $\mathcal{G}$ by the induced *intrinsic preorder* defined below.

Definition 14. *Suppose $\sigma_1, \sigma_2 : 1 \to A$. We define $\sigma_1 \le \sigma_2$ to hold iff, for any $\rho : A \to 1$, $\sigma_1; \rho \ne \{[\epsilon]\}$ implies $\sigma_2; \rho \ne \{[\epsilon]\}$.*

[9] $1 \to [\![\theta]\!]$, where θ is a Reduced ML type.

[10] Recall that blind strategies are generated by their strict plays.

It is common to refer to the above preorder as *testing σ_i with ρ*, where $\sigma_i;\rho \neq \{[\epsilon]\}$ is regarded as a successful outcome.

Theorem 15. *Given Reduced ML terms $\vdash M_1 : \theta$, $\vdash M_2 : \theta$, we have $\vdash M_1 \sqsubsetsim M_2$ iff $[\![\vdash M_1]\!] \leq [\![\vdash M_2]\!]$.*

5 Program Equivalence Explicitly

Let σ_1, σ_2, ρ be as in the definition of $\leq$. Note that during composition of σ_i with ρ there is a full symmetry between O-names and P-names, i.e. names which are O-names in σ_i are viewed as P-names in ρ, and vice versa. This can be contrasted with the general case of composition, where both strategies may regard a name as an O-name during composition, though not a P-name. This symmetry of roles means that, because plays of ρ satisfy P-availability, a successful outcome can only be reached by interaction with a play of σ_i that satisfies the dual condition of ***O-availability***: for each $s'o^S \sqsubseteq^{\text{odd}} s$ and any $a \in \mathbb{A}$, if $a \in \nu(o) \cap \nu(s')$ then $a \in \mathsf{Av}_{\mathrm{O}}(s')$.

Similarly, whenever the play s engages with ρ successfully, the ***O-passivity*** condition holds: for each $s'p^S o^{S'} \sqsubseteq s$ and any $a \in \mathbb{A}$, if $a \in \mathsf{P}(s'p^S) \setminus \mathsf{Av}_{\mathrm{O}}(s'p^S)$ then $S(a) = S'(a)$. This time this is due to the definition of composition, which stipulates that the part of store irrelevant to one of the strategies must be copied. This means that the plays of σ_i that "matter" must necessarily meet the above condition. Finally, whenever $\sigma_i;\rho \neq \{[\epsilon]\}$, the play witnessing this is ***complete***, i.e. all of its questions are answered.

Definition 16. *A play is* **relevant** *iff it is complete, satisfies O-availability and O-passivity. We write $\mathsf{rel}(\sigma)$ for the set of relevant plays of σ.*

We can represent *relevant* plays more succinctly by restricting the associated stores to mutually available names (both O- and P-available). The outcome is not a play any more, though it remains a legal justified sequence. We call such sequences ***protoplays*** and let γ'' be the obvious operation on justified sequences that simply erases the O-unavailable names in stores. Although some information about σ is seemingly lost by applying γ'' to $\mathsf{rel}(\sigma)$, the missing values turn out inessential for testing. By O-passivity, the lost values of O-unavailable names can be uniquely retrieved in O-moves, by copying values from the preceding P-moves. However, more surprisingly, it does not matter what values such names have in P-moves either. This is because the names are then P-unavailable for ρ and, during composition, are dealt with *uniformly* by propagation as long as they remain unavailable.

Finally, we take advantage of the fact that the test ρ is a blind strategy. Recall that blind strategies are uniquely determined by their strict plays, i.e. plays in which O-names fresh in the P-view must be genuinely fresh at the point of introduction. Consequently, if one wants to check if σ_i passes the ρ test, we can take advantage of the fact that any contribution from ρ will originate from a strict play. Let $s'' = \gamma''(s')$ ($s' \in \mathsf{rel}(\sigma)$) be a protoplay generated by σ_i. To test whether s'' represents a renaming of a strict play from ρ, it suffices to "refresh" P-names in s'' and try to match it with that the strict play. The desired refreshing operation (for P-names using O-views) is entirely dual to renamings introduced in Definition 8, though it needs to be defined on protoplays to be correct. For two protoplays s, s', we write $s \underset{r}{\sim} s'$ iff s can be obtained from s' by a series of dual renamings. A protoplay is ***dually strict*** iff any P-name fresh in the O-view is fresh at the point of introduction. Given $\sigma : A$, let $\widehat{\sigma}$ be the following set of equivalence classes of protoplays $\widehat{\sigma} = \{[s] \mid s \text{ is dually strict}, s' \in \mathsf{rel}(\sigma), s \underset{r}{\sim} \gamma''(s')\}$. We can then show the following result.

Lemma 17. *Given* $\sigma_1, \sigma_2 : 1 \to A$, $\sigma_1 \leq \sigma_2$ *iff* $\widehat{\sigma_1} \subseteq \widehat{\sigma_2}$.

Observe that $\widehat{\sigma}$, like σ, is saturated under renamings (extended to act on protoplays). This makes it possible to simplify the above result along the following lines. We call a protoplay ***mutually strict*** iff it is both strict and dually strict. Note that by using $\overset{r}{\sim}$ and $\underset{r}{\sim}$ (in any order) we can convert a protoplay to a mutually strict protoplay, unique up to atom-abstraction. Given $\sigma : A$, let $\widehat{\widehat{\sigma}}$ be $\{[s] \mid s \text{ is mutually strict}, s' \in \mathsf{rel}(\sigma), s\, (\overset{r}{\sim}; \underset{r}{\sim})\, \gamma''(s')\}$.

Theorem 18. *Given* $\sigma_1, \sigma_2 : 1 \to A$, $\sigma_1 \leq \sigma_2$ *iff* $\widehat{\widehat{\sigma_1}} \subseteq \widehat{\widehat{\sigma_2}}$.

It follows that terms of Reduced ML are equivalent iff they induce the same mutually strict protoplays.

Thus we have shown that program equivalence and approximation in Reduced ML can be captured explicitly, which is the first result of this kind for Reduced ML. The characterization immediately implies that equivalence is decidable for finitary strategies and that the fully abstract model of Reduced ML is effectively presentable.

Our results identify mutually strict protoplays as an appealing object for future study and, indeed, our fully abstract model can be presented in a more direct way by founding the games on them. It has to be said, though, that composition of such plays is quite intricate, because they cannot be combined by parallel composition with hiding: although this kind of interaction is sufficient to test plays, composition in general lacks the convenient duality between O-names and P-names. Consequently, in order to compose mutually strict protoplays, one has to allow for renamings before synchronization and follow with dual renamings afterwards. We intend to present an account of this procedure in the full version of the paper.

In this submission however we have chosen to present the model gradually: starting from the intuitive framework in which full information about the store is available we successively imposed a series of restrictions. We believe this leads to a more informative presentation and decomposes the difficulties involved in dealing with mutually strict protoplays into smaller arguments. For instance, the correctness proof of compositionality of mutually strict protoplays (for the algorithm sketched above) draws on insights obtained from all of our compositionality proofs (P-availability, P-storage, blindness) as well as the argument behind the explicit characterization.

We hope to bring our results to bear on the on-going research into algorithmic aspects of game models [1] and to contribute new methods of reasoning about program equivalence in Reduced ML. This direction has been pursued using logical relations in [20]. However, there are limits to what can be achieved, as program equivalence of finitary Reduced ML (finite types) is already undecidable at second order, due to a similar result for Reduced ML with mkvar in [17].

References

1. Abramsky, S., Ghica, D.R., Murawski, A.S., Ong, C.-H.L.: Applying game semantics to compositional software modeling and verification. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 421–435. Springer, Heidelberg (2004)
2. Abramsky, S., Ghica, D.R., Murawski, A.S., Ong, C.-H.L., Stark, I.D.B.: Nominal games and full abstraction for the nu-calculus. In: Proc. of LICS, pp. 150–159 (2004)
3. Abramsky, S., Honda, K., McCusker, G.: Fully abstract game semantics for general references. In: Proc. of LICS, pp. 334–344 (1998)
4. Abramsky, S., Jagadeesan, R., Malacaria, P.: Full abstraction for PCF. Information and Computation 163, 409–470 (2000)
5. Abramsky, S., McCusker, G.: Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In: Algol-like languages, Birkhaüser, pp. 297–329 (1997)
6. Abramsky, S., McCusker, G.: Call-by-value games. In: Nielsen, M. (ed.) CSL 1997. LNCS, vol. 1414, pp. 1–17. Springer, Heidelberg (1998)
7. Gabbay, M.J., Pitts, A.M.: A new approach to abstract syntax with variable binding. Formal Aspects of Computing 13, 341–363 (2002)
8. Honda, K., Yoshida, N.: Game-theoretic analysis of call-by-value computation. In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) ICALP 1997. LNCS, vol. 1256, pp. 225–236. Springer, Heidelberg (1997)
9. Hyland, J.M.E., Ong, C.-H.L.: On Full Abstraction for PCF. Information and Computation 163(2), 285–408 (2000)
10. Laird, J.: A game semantics of local names and good variables. In: Walukiewicz, I. (ed.) FOSSACS 2004. LNCS, vol. 2987, pp. 289–303. Springer, Heidelberg (2004)
11. Laird, J.: A game semantics of names and pointers. Annals of Pure and Applied Logic 151, 151–169 (2008)
12. McCusker, G.: Games for recursive types. BCS Distinguished Dissertation. Cambridge University Press, Cambridge (1998)
13. McCusker, G.: Games and full abstraction for FPC. Information and Computation 160(1-2), 1–61 (2000)
14. McCusker, G.: On the semantics of Idealized Algol without the bad-variable constructor. In: Proc. of MFPS. ENTCS, vol. 83 (2003)
15. Milner, R., Tofte, M., Harper, R.: The Definition of Standard ML. MIT Press, Cambridge (1990)
16. Moggi, E.: Notions of computation and monads. Information and Computation 93, 55–92 (1991)
17. Murawski, A.S.: Functions with local state: regularity and undecidability. Theoretical Computer Science 338(1/3), 315–349 (2005)

18. Murawski, A.S.: Bad variables under control. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 558–572. Springer, Heidelberg (2007)
19. Nickau, H.: Hereditarily sequential functionals. In: Matiyasevich, Y.V., Nerode, A. (eds.) LFCS 1994. LNCS, vol. 813, pp. 253–264. Springer, Heidelberg (1994)
20. Pitts, A.M., Stark, I.D.B.: Operational reasoning for functions with local state. In: Higher-Order Operational Techniques in Semantics, pp. 227–273. CUP (1998)
21. Reynolds, J.C.: The essence of Algol. In: de Bakker, J.W., van Vliet, J. (eds.) Algorithmic Languages, pp. 345–372. North Holland, Amsterdam (1978)
22. Stark, I.D.B.: Names and Higher-Order Functions. PhD thesis, University of Cambridge (1995)
23. Tzevelekos, N.: Full abstraction for nominal general references. In: Proc. of LICS, pp. 399–410 (2007)
24. Tzevelekos, N.: Nominal game semantics. D.Phil. thesis, University of Oxford (2008)

Logics and Bisimulation Games for Concurrency, Causality and Conflict

Julian Gutierrez

LFCS. School of Informatics. University of Edinburgh
Informatics Forum, 10 Crichton Street, Edinburgh, EH8 9AB, UK
J.E.Gutierrez@ed.ac.uk

Abstract. Based on a simple axiomatization of concurrent behaviour we define two ways of observing parallel computations and show that in each case they are dual to conflict and causality, respectively. We give a logical characterization to those dualities and show that natural fixpoint modal logics can be extracted from such a characterization. We also study the equivalences induced by such logics and prove that they are decidable and can be related with well-known bisimulations for interleaving and noninterleaving concurrency. Moreover, by giving a game-theoretical characterization to the equivalence induced by the main logic, which is called Separation Fixpoint Logic (SFL), we show that the equivalence SFL induces is strictly stronger than a history-preserving bisimulation (hpb) and strictly weaker than a hereditary history-preserving bisimulation (hhpb). Our study considers branching-time models of concurrency based on transition systems and petri net structures.

Keywords: Modal and temporal logics, Bisimulation games, Behavioural equivalences, Concurrent and reactive systems, Petri nets.

1 Introduction

In [12] Milner and Hennessy studied an algebraic characterization of concurrent systems and defined a formal way of comparing different processes through the equivalence induced by such an axiomatization. They called it observational equivalence. They also gave a logical characterization to the same concepts and showed their correspondence with the axiomatization when interpreted using a simple modal logic with an interleaving model of concurrency. This work led to the definition of a well-known bisimulation equivalence for interleaving concurrency, namely the one induced by the Hennessy-Milner Logic (HML).

However, when studying concurrency at a more fundamental semantic level, partial order models should be considered, different axiomatizations must be defined, and finer bisimulation equivalences have to be taken into account. A natural question immediately arises, that is, what exactly is the new relationship between concurrency in the models (a model independence) and concurrency in the logical languages (a logical independence). The answer to this question is not unique since it depends on the models and axiomatizations that are being considered. Therefore, one would like them to be as general as possible.

In this paper an alternative answer for this question is put forward. Our results are relative to a surprisingly simple axiomatization of concurrent behaviour for a category of transition systems with independence introduced by Winskel and Nielsen (see [16] or Section 2). This axiomatization was pioneered by Mazurkiewicz for trace languages [14] and has been used ever since to understand the properties of several models of concurrency with partial order semantics, also called independence models [16]. Such axioms define local properties of the behaviour of a concurrent system and can be used to generate what is called the "concurrency diamonds" in some noninterleaving models of concurrency. Our results are also extended to a class of safe Petri net structures.

In particular, we study the relationships between model independence and logical independence purely based on observable *dualities* between concurrency and conflict, on the one hand, and concurrency and causality on the other. The logical characterization of these dualities relies on geometrical properties enforced by an algebraic axiomatization of concurrent events. At a semantic level, we develop a notion of locality called *support sets* which allows the recognition of independent events (Section 3). At a logical level we define both modal operators sensitive to causal information and a parallel conjunction that allows one to separate off concurrent events while avoiding those in conflict. This basic modal logic is then extended with fixpoint operators so as to express temporal properties of concurrent systems (Section 4). The final outcome is a fixpoint modal logic, which we call Separation Fixpoint Logic (SFL). A set of application examples for SFL are given at the end of this section.

We also study the equivalence induced by SFL and some sublogics of it (Section 5). These sublogics can be obtained from SFL by restricting *syntactically* the interplay between the dualities we have defined. This strongly eases the analysis of the relationships between concurrency, causality and conflict. Four SFL fragments are considered. The first one is Kozen's modal mu-calculus, $L\mu$. This SFL sublogic help us show that SFL can deal equally well with both interleaving and noninterleaving models of concurrency. Since the mu-calculus is a natural logical language for interleaving concurrent systems (and SFL includes it), nothing is lost with respect to the main interleaving approaches to concurrency.

However, the problem of observing concurrency and nondeterminism, as studied by Milner and Hennessy in [12] for interleaving systems, can be refined to a problem of observing concurrency, causality and conflict in a noninterleaving context. As a consequence, more specialized bisimulation equivalences have been introduced so as to analyse independence models. We study some of those equivalences and focus our attention on the two strongest equivalences presented in [7,10], namely the hereditary history-preserving bisimulation (hhpb) [13] and the plain history-preserving bisimulation (hpb) [10].

The second sublogic we consider is a Separation modal mu-calculus, $SL\mu$. It extends the previous fragment by allowing the distinction between concurrency and conflict. We show that the equivalence induced by $SL\mu$ cannot be compared with any of the bisimulations we consider here. The third case is a Causal modal mu-calculus, $CL\mu$. It takes full account of causality and concurrency and induces an hp bisimulation equivalence. Although hpb captures some features of concurrent computation, it has been argued that it is actually an equivalence of only causality [9]. As a consequence, stronger equivalences such as hhpb have been defined in order to capture aspects of true-concurrency rather than only causality. Therefore, in fourth place we consider the full SFL and compare the equivalence it induces with the stronger hhpb. Using game-theoretical arguments, we prove that the bisimulation equivalence for SFL is strictly stronger than hpb and strictly weaker than hhpb (Section 6). Since hhpb is undecidable, even on finite systems, the equivalence induced by SFL ranks at the top of the decidable equivalences for true-concurrency according to discriminating power (see [10] or completed hierarchy in [7]). This feature makes the equivalence induced by SFL an interesting candidate for an equivalence of true concurrency. Finally, some concluding remarks and related work are given in Section 7.

2 An Independence Model and Axioms of Concurrency

A *Transition System with Independence* (TSI) [16] is a simple extension of a Labelled Transition System (LTS), where independent transitions can be explicitly recognised. A TSI is a structural, branching-time and noninterleaving model of concurrency. Formally, a TSI $\mathfrak{T}$ is a structure (S, T, Σ, I), where S is a finite set of states, $T \subseteq S \times \Sigma \times S$ is a transition relation, Σ is a set of action labels, and $I \subseteq T \times T$ is an irreflexive and symmetric relation on independent transitions, i.e., on concurrent transitions. The binary relation $\prec$ on transitions defined by

$$(s, a, s_1) \prec (s_2, a, q) \Leftrightarrow$$
$$\exists b.(s, a, s_1) I (s, b, s_2) \wedge (s, a, s_1) I (s_1, b, q) \wedge (s, b, s_2) I (s_2, a, q)$$

expresses when two transitions represent two occurrences of the same *event* (the same action in different interleavings). Such a relation can be used to generate a "concurrency diamond", as shown in Fig. 1.

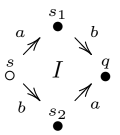


Fig. 1. A concurrency diamond for $a\ I\ b$. Concurrent transitions are recognized by the I symbol inside the square. The initial state of the TSI is marked by the circle $\circ$.

Moreover, $\sim$ is the least equivalence relation that includes $\prec$, i.e., the reflexive, symmetric and transitive closure of $\prec$. The equivalence relation $\sim$ is used to group all events that represent the same action in the TSI, and therefore, considers all occurrences of events of an action in all its possible interleavings. Additionally, the I relation is subject to the following axioms:

- **A1**. $(s,a,s_1) \sim (s,a,s_2) \Rightarrow s_1 = s_2$
- **A2**. $(s,a,s_1)I(s,b,s_2) \Rightarrow \exists q.(s,a,s_1)I(s_1,b,q) \wedge (s,b,s_2)I(s_2,a,q)$
- **A3**. $(s,a,s_1)I(s_1,b,q) \Rightarrow \exists s_2.(s,a,s_1)I(s,b,s_2) \wedge (s,b,s_2)I(s_2,a,q)$
- **A4**. $(s,a,s_1) \prec \cup \succ (s_2,a,q)I(w,b,w') \Rightarrow (s,a,s_1)I(w,b,w')$

Intuitively, **A1** states that from any state of the system, the execution of an action leads always to a unique future state. **A2** (resp. **A3**) says that if two independent actions can be executed in parallel (resp. one after the other), they can also be executed one after the other (resp. in parallel). Finally, **A4** ensures that the relation I between transitions is well defined. Roughly speaking **A4** says that if actions a and b are independent, then all the transitions that are occurrences of the action a are independent of all the transitions that are occurrences of the action b. Having said that, we can give an alternative, more intuitive, definition for axiom **A4**. Let $\mathcal{I}(t)$ be the set $\{t' \mid tIt'\}$. Then, axiom **A4** is equivalent to the following expression: **A4**. $t_1 \sim t_2 \Rightarrow \mathcal{I}(t_1) = \mathcal{I}(t_2)$.

Notation 2.1. *Given a transition $t = (s_1, a, s_2)$, also written as $s_1 \xrightarrow{a} s_2$, s_1 is the source, $src(t) = s_1$; s_2 the target, $trg(t) = s_2$; and a the label of t, $lbl(t) = a$.*

3 Local Dualities in Independence Models

We present two ways in which concurrency can be regarded as a dual concept to *conflict* and *causality*, respectively. These two ways of observing concurrency will be called *immediate concurrency* and *linearized concurrency*. Whereas immediate concurrency is dual to conflict, linearized concurrency is dual to causality.

The intuitions behind these two observations are the following. Consider a concurrent system, say, a TSI, and any two different transitions t_i and t_j with the same source node, i.e., $src(t_i) = src(t_j)$. These two transitions are either immediately concurrent, and therefore belong to I, or dependent, in which case they must be in conflict. Similarly, consider any two transitions t_i and t_j where $trg(t_i) = src(t_j)$. Again, these two transitions can either belong to I, in which case they are concurrent, yet have been linearized, or they do not belong to I, and therefore are causally dependent. In both cases, the two conditions are exclusive and *there are no other possibilities.*

Definition 3.1. Two transitions t_i and t_j, such that $trg(t_i) = src(t_j)$, are *linearly concurrent* iff $t_i I t_j$. We write $t_i \ominus t_j$ for such a relation.

Dually, causally dependent transitions can be defined.

Definition 3.2. Two transitions t_i and t_j, such that $trg(t_i) = src(t_j)$, are *causally dependent* iff $\neg(t_i I t_j)$. We write $t_i \leq t_j$ for such a relation.

This duality is defined locally with respect to a state s, such that $trg(t_i) = s = src(t_j)$, of the underlying independence model. The previous definitions will be used to give the semantics of the modal operators of SFL. Now, we turn our attention to the duality between immediate concurrency and conflict.

Definition 3.3. Two transitions t_i and t_j, such that $src(t_i) = src(t_j)$, are *immediately concurrent* iff $t_i I t_j$. We write $t_i \otimes t_j$ for such a relation.

Dually, transitions in conflict can be defined as follows.

Definition 3.4. Two transitions t_i and t_j, such that $src(t_i) = src(t_j)$, are *in conflict* iff $\neg(t_i I t_j)$. We write $t_i \# t_j$ for such a relation.

3.1 Separation and Support Sets for Local Reasoning

The definitions of immediate concurrency and conflict can be used to recognize sets where every transition is concurrent with each other and therefore can all be executed in parallel. We call these sets as *conflict-free sets of transitions*.

Definition 3.5. A *conflict-free set of transitions*, denoted by $\mathrm{cf}(E)$, is a set of transitions E such that $\forall t_i, t_j \in E.\ src(t_i) = src(t_j) \wedge (t_i \neq t_j \Rightarrow t_i \otimes t_j)$.

As we want to specify the existence of actual concurrent behaviour, we strengthen the definition of conflict-free sets of transitions given above. Notice that Definition 3.5 allows the existence of empty and singleton sets, which do not show any actual concurrent behaviour (or even any behaviour at all).

Definition 3.6. An *effective conflict-free set of transitions* is a conflict-free set of transitions E such that $|E| \geq 2$.

So, in order to do local reasoning on concurrent processes of an independence model, we want to recognize effective conflict-free sets given an arbitrary set of transitions at some state s of the independence model. In particular, the set of all transitions t such that $src(t) = s$, will be called the *maximal set* of transitions at s. Now, we introduce a notion of locality called *support sets* that is used to define the semantics of SFL formulae in the following section.

Definition 3.7. A *support set* is either a maximal set of transitions or a non-empty conflict-free set of transitions.

Given an independence model, the set of all its support sets is denoted by $\mathfrak{P}$. Notice that once one has non-empty conflict-free sets of transitions that are not singleton sets, i.e. effective conflict-free sets of transitions, to do local reasoning on sets of concurrent actions becomes easier since they can be *separated* or decomposed into smaller sets, where every transition is, as well, concurrent with each other. Finally, the following definitions are useful when defining the semantics of some SFL operators in the next section:

$$E \sqsubseteq R \overset{\text{def}}{=} E \subseteq R, \text{ provided that, } \mathrm{cf}(E) \text{ and } \neg\exists t \in (R \setminus E).\ \forall t_e \in E.\ t \otimes t_e$$
$$P_1 \uplus P_2 \overset{\text{def}}{=} P_1 \cup P_2, \text{ provided that, } P_1 \cap P_2 = \emptyset \wedge P_1 \neq \emptyset \wedge P_2 \neq \emptyset$$

The first definition, $E \sqsubseteq R$, characterizes support sets E that contain only concurrent transitions and cannot be made any bigger with respect to R; the second definition, $P_1 \uplus P_2$, formalizes the notion of *separation* for local reasoning used here. Such a definition resembles the notion of separation as used in Separation Logic [19], i.e., as disjointness of sets of independent resources.

4 Separation Fixpoint Logic

Definition 4.1. *Separation Fixpoint Logic* (SFL) has formulae ϕ built from a set Var of variables Y, Z, ... and a set $\mathcal{L}$ of labels $a, b, \ldots$ by the following grammar:

$$\phi ::= Z \mid \neg\phi_1 \mid \phi_1 \wedge \phi_2 \mid \langle K\rangle_c\phi_1 \mid \langle K\rangle_{nc}\phi_1 \mid \phi_1 * \phi_2 \mid \mu Z.\phi_1$$

where $Z \in \mathrm{Var}$, $K \subseteq \mathcal{L}$, and $\mu Z.\phi_1$ has the restriction that any free occurrence of Z in ϕ_1 must be within the scope of an even number of negations.

Informally, the meaning of these operators is the following. "$\wedge$" and "$\neg$" are the usual boolean operators, "$\langle K\rangle_c$" (resp. "$\langle K\rangle_{nc}$") asserts at the fact that there is in the set of actions K a causally dependent (resp. a non-causally dependent or linearly concurrent) transition that can be performed; as defined in Section 3, such a transition is always either causally dependent or linearly concurrent w.r.t. the last transition that has been executed. $\phi_1 * \phi_2$ specifies that there exists a partition in the support set (i.e., a partition of the actions in the set to be considered) with which both formulae ϕ_1 and ϕ_2 can hold in parallel. This does not necessarily mean that both formulae hold in parallel everywhere because the operator "$*$" has a local meaning. Finally, "μ" is simply a least fixpoint operator.

Derived operators are defined in the familiar way: $\phi_1 \vee \phi_2 \overset{\text{def}}{=} \neg(\neg\phi_1 \wedge \neg\phi_2)$, $\phi_1 \bowtie \phi_2 \overset{\text{def}}{=} \neg(\neg\phi_1 * \neg\phi_2)$, $[K]_c\,\phi_1 \overset{\text{def}}{=} \neg\langle K\rangle_c\neg\phi_1$, $[K]_{nc}\,\phi_1 \overset{\text{def}}{=} \neg\langle K\rangle_{nc}\neg\phi_1$ and $\nu Z.\phi_1 \overset{\text{def}}{=} \neg\mu Z.\neg\phi_1\,[\neg Z/Z]$. The following abbreviations are also useful: $\mathrm{ff} \overset{\text{def}}{=} \mu Z.Z$, $\mathrm{tt} \overset{\text{def}}{=} \neg\mathrm{ff}$, $\langle K\rangle\phi_1 \overset{\text{def}}{=} \langle K\rangle_c\phi_1 \vee \langle K\rangle_{nc}\phi_1$, $[K]\,\phi_1 \overset{\text{def}}{=} [K]_c\,\phi_1 \wedge [K]_{nc}\,\phi_1$. Also, sometimes we write $[-]\,\phi$ for $[\mathcal{L}]\,\phi$ and $[-K]\,\phi$ for $[\mathcal{L} - K]\,\phi$, and similarly for the other box ("$[\;]$") and diamond ("$\langle\;\rangle$") operators.

4.1 Denotation of SFL Formulae

Definition 4.2. A *TSI-based SFL model* $\mathfrak{M}$ is a Transition system with independence $\mathfrak{T} = (S, T, \Sigma, I)$ together with a valuation $\mathcal{V} : \mathrm{Var} \to 2^{\mathfrak{S}}$, where $\mathfrak{S} = S \times \mathfrak{P} \times \mathfrak{A}$ is the set of tuples (s, P, t_a) of states $s \in S$, support sets $P \in \mathfrak{P}$ in the TSI $\mathfrak{T}$, and transitions $t_a \in \mathfrak{A} = T \cup \{t_\epsilon\}$, where a is an action label in $\Sigma \cup \{\epsilon\}$. The denotation $\|\phi\|^{\mathfrak{T}}_{\mathcal{V}}$ of an SFL formula ϕ in the model $\mathfrak{M} = (\mathfrak{T}, \mathcal{V})$ is a subset of $\mathfrak{S}$, given by the following rules (omitting the superscript $\mathfrak{T}$):

$$\begin{aligned}
\|Z\|_{\mathcal{V}} &= \mathcal{V}(Z)\\
\|\neg\phi\|_{\mathcal{V}} &= \mathfrak{S} - \|\phi\|_{\mathcal{V}}\\
\|\phi_1 \wedge \phi_2\|_{\mathcal{V}} &= \|\phi_1\|_{\mathcal{V}} \cap \|\phi_2\|_{\mathcal{V}}\\
\|\langle K\rangle_c\phi\|_{\mathcal{V}} &= \{(s, P, t_a) \mid \exists b \in K.\ \exists s' \in S.\ t_a \leq s \xrightarrow{b} s' \in P \wedge (s', P', t_b) \in \|\phi\|_{\mathcal{V}}\}\\
\|\langle K\rangle_{nc}\phi\|_{\mathcal{V}} &= \{(s, P, t_a) \mid \exists b \in K.\ \exists s' \in S.\ t_a \ominus s \xrightarrow{b} s' \in P \wedge (s', P', t_b) \in \|\phi\|_{\mathcal{V}}\}\\
\|\phi_1 * \phi_2\|_{\mathcal{V}} &= \{(s, P, t_a) \mid \exists P_1, P_2.\ P_1 \uplus P_2 \sqsubseteq P \wedge (s, P_1, t_a) \in \|\phi_1\|_{\mathcal{V}} \wedge (s, P_2, t_a) \in \|\phi_2\|_{\mathcal{V}}\}
\end{aligned}$$

where P' is the maximal set at s'. A tuple (s, P, t_a) of a model $\mathfrak{M}$ is called a *process*. An initial process is a tuple (s, P, t_ϵ), where s is the initial state of the TSI $\mathfrak{T}$, P is the maximal set at s, and t_ϵ is the empty transition, such that for all t whose source node is the initial state, $t_\epsilon \leq t$. Notice that since the third component of every process denotes the last action that has been made, an initial process requires the definition of the empty transition t_ϵ. Also, since an initial state defines a unique initial process, and vice-versa, the two terms and notations are interchangeable.

Given the usual restriction on free occurrences of variables, imposed in order to obtain monotone operators in the complete lattice $\mathcal{P}(\mathfrak{S}) = 2^{\mathfrak{S}}$, it is possible to define the denotation of the fixpoint operator $\mu Z.\phi(Z)$ in the standard way, according to the Knaster-Tarski fixpoint theorem:

$$\|\mu Z.\phi(Z)\|_{\mathcal{V}} = \bigcap\{Q \in \mathcal{P}(\mathfrak{S}) \mid \|\phi\|_{\mathcal{V}[Z:=Q]} \subseteq Q\}$$

where $\mathcal{V}[Z := Q]$ is the valuation $\mathcal{V}'$ which agrees with $\mathcal{V}$ save that $\mathcal{V}'(Z) = Q$. Since positive normal form is assumed henceforth, the semantics of the dual boolean, modal, structural and fixpoint operators can be given in the usual way.

A Petri Net Semantics. The logical characterization defined previously using a TSI model can also be given using other independence models. As an example of this, it is shown how to give a Petri net semantics for SFL, but analogous procedures can be followed with other independence models.

A labelled net $\mathfrak{N}$ is a tuple $(P, T, A, \mathcal{F}, \Sigma)$, where P is a set of places, T is a set of transitions and A is a relation between places and transitions such that $A \subseteq (P \times T) \cup (T \times P)$, and $\mathcal{F}$ is a labeling function, $\mathcal{F} : T \to \Sigma$, from transitions to a finite set of action labels Σ. Places and transitions are called nodes. Given a node n, ${}^\bullet n = \{x \mid (x, n) \in A\}$ is the preset of n and $n^\bullet = \{y \mid (n, y) \in A\}$ is the postset of n. A marking M of $\mathfrak{N}$ is a mapping such that $M \subseteq P$. A marking M enables a transition t iff ${}^\bullet t \subseteq M$. If t is enabled at M, then t can occur, and its occurrence leads to a successor marking M', where $M' = (M \setminus {}^\bullet t) \cup t^\bullet$. The nets considered here are safe Petri nets [16]. All safe nets are finite systems. Given a 1-safe labelled net with an initial marking, the set of all reachable markings is fixed and can be constructed with the occurrence net.

We call a reachable marking M as a state s of the system $\mathfrak{N}$, and S as the set of all possible reachable markings or states of the system. Since in this model transitions represent actions rather than events (as in the TSI case), every transition is annotated with a particular marking M to recognize which event they refer to. Therefore we can write t_a^M for the event of the action or transition t with label a at state $s = M$, i.e., whenever ${}^\bullet t \subseteq M$ and $\mathcal{F}(t) = a$. Support sets for nets are defined as for TSIs considering that $src(t_a^M) = M$ (to define conflict-free sets of transitions) and the maximal set at a state $s = M$ is the set $\{t \mid {}^\bullet t \subseteq M\}$. Finally, the dualities used to give the denotation to SFL formulae in a model $\mathfrak{M} = (\mathfrak{N}, \mathcal{V})$ are defined as for TSIs provided that the I relation on events in the net model is defined for any two events t_a^M and t_b^N of the transitions t_a and t_b, respectively, whenever there is a marking such that t_a and t_b are enabled at such marking and ${}^\bullet t_a \cap {}^\bullet t_b = \emptyset$. With this information the set of processes $\mathfrak{S}$ and the valuation $\mathcal{V}$ can be defined, and the net model $\mathfrak{M}$ for SFL is straightforward.

4.2 Applications

A Temporal Logic. SFL can express all usual temporal properties, such as, liveness, safety, fairness, and so on. These properties are equally handled in both interleaving and noninterleaving models of concurrency.

A Process Logic. SFL can differentiate concurrency from nondeterminism using two different local dualities. Consider these concurrent systems: $P = a.0 \parallel b.0$ and $Q = a.b.0 + b.a.0$. P and Q are behaviourally equivalent in an interleaving context (since they are strongly bisimilar [12]), but different from a true-concurrency point of view (since they are not history-preserving bisimilar [10]). Such a difference can be captured in these two ways: with the formulae $\phi = \langle a \rangle \mathrm{tt} * \langle b \rangle \mathrm{tt}$ or $\psi = \langle a \rangle_c \langle b \rangle_{nc} \mathrm{tt}$, which are both true in P but not in Q.

A Petri Net Logic. SFL can express properties of net systems by defining properties of the localities and synchronization mechanisms. In order to do so, it is useful to define a derived operator that allows one to perform a causal step that may require resource from *separate localities* so as to proceed (a strongly causal step). Consider the derived operator: $\langle a \rangle_{sc} \phi \overset{\text{def}}{=} \langle a \rangle_c \mathrm{tt} \wedge \mu Z. \langle a \rangle \phi \vee (\langle a \rangle \mathrm{tt} * \langle -a \rangle Z)$. Expressing that a property p eventually holds in a causal line can be easily defined as $\phi = \mu Z.p \vee \langle - \rangle_{sc} Z$. The result of evaluating this formula corresponds to finding a "line" in the net to some state where p holds, even though separate resource may be needed. These kinds of properties can be further studied with the structural operators of SFL. For instance, the formula $\phi = \mu Z.p \vee \langle - \rangle_{sc} Z * \mu Y.q \vee \langle - \rangle_{sc} Y$ not only states that properties p and q hold eventually in two causal lines, but also that such causal lines have an independent source.

A Resource-Sensitive Logic. The parallel conjunction of SFL can also be used to define data structures, such as, lists and trees, *à la* Reynolds [19], but using a temporal specification approach. In SFL independent transitions can be treated as resources, allowing indirectly the specification of independent localities in the model. For instance, let $\phi = \mu Z. \langle \mathrm{nil} \rangle_c \mathrm{tt} \vee (\langle \mathrm{data} \rangle_c \mathrm{tt} * \langle - \rangle_c Z)$. Formula ϕ represents an abstraction of a finite list structure where all nodes in the list causally depend on the previous one (a sequential structure), whereas its data and pointers are spatially separated and thus can be regarded as independent. Since our approach is not proof-theoretic, there is not a "frame rule" similar to the one defined for Separation Logic and concurrent extensions of it [6,11,19].

A Logic for Multi-Agent Systems (MAS). MAS, such as those specified with logics like ATL [1] can be studied using SFL. In order to express properties of MAS, define a set Γ of agents, a labeling function $\mathcal{B}$ on sets of transitions, and a mapping $\mathcal{A}$ that assigns transitions to agents, subject to the restriction that if $t_1 \sim t_2$ then $\mathcal{A}(t_1) = \mathcal{A}(t_2)$. For simplicity, we write $\langle K \rangle^{\gamma}$ for $\langle \mathcal{B}(\mathcal{A}^{-1}(\gamma)) \rangle$, and similarly for the other modal operators. The formula $\psi = [-]^{\beta} \langle - \rangle^{\alpha}_{nc} \mu Z. \phi \vee \langle - \rangle^{\alpha}_{c} Z$ expresses that there is an agent α (the system) that can satisfy ϕ regardless the behaviour of an adversarial agent β (the environment).[1] Informally, ψ says

[1] I thank Bradfield for the IFML version of this example. $\mathcal{A}^{-1}$ is the inv. func. of $\mathcal{A}$.

"whatever you (the environment) do, I (the system) can get to ϕ, though I may first have to do some things that do not depend on what you did."

5 Logical and Concurrent Equivalences

We now turn our attention to the study of the relationships between model independence and logical independence. We do so by relating well-known equivalences for concurrency, namely hpb [10] and hhpb [13], with the equivalences induced by different SFL sublogics where the interplay between concurrency and conflict, and concurrency and causality is restricted *syntactically*. The reader is referred to [10,13] or [5] for a detailed description of the history-preserving bisimulations not proposed in this paper (hpb and hhpb), but studied in the following sections. Due to lack of space, in most cases, proofs are omitted or simply sketched.

Definition 5.1. ***(SFL equivalence*** $\sim_{SFL}$***).*** Two processes P and Q of two independence models $\mathfrak{T}_1$ and $\mathfrak{T}_2$, respectively, are SFL-equivalent, $P \sim_{SFL} Q$, iff for every SFL formula ϕ in $\mathcal{F}_{SFL}$, $P \models^{\mathfrak{T}_1} \phi \Leftrightarrow Q \models^{\mathfrak{T}_2} \phi$, where $\mathcal{F}_{SFL}$ is the set of all fixpoint-free closed formulae of SFL.

Remark 5.1. Similar definitions can be made for the equivalences of the SFL sublogics we present here.

Remark 5.2. In order to obtain an exact match between finitary modal logic and bisimulation, all models considered in this paper are image-finite [12].

5.1 SFL, HML and the Modal Mu-Calculus

The first SFL sublogic is obtained from SFL by disabling its sensitivity to both dualities. On the one hand, insensitivity to the duality between concurrency and causality can be captured by considering only modalities without subscript, using the abbreviations for modalities given previously in Section 4. On the other hand, insensitivity to the duality between concurrency and conflict can be captured by considering the $*$-free SFL sublanguage. The resulting logic has the following syntax: $\phi ::= Z \mid \neg\phi_1 \mid \phi_1 \wedge \phi_2 \mid \langle K \rangle \phi_1 \mid \mu Z.\phi_1$. This SFL syntactic fragment is the modal $*$-free fragment of SFL.

Proposition 1. *The modal $*$-free fragment of SFL is Kozen's mu-calculus, $L\mu$.*

Proof. By computing the semantics of the derived operators of this sublogic. $\square$

Remark 5.3. This SFL sublogic cannot recognize elements in I and therefore sees TSIs as plain LTSs, or what is equivalent, TSIs with an empty I relation. As a consequence, although using an independence model, it is possible to retain in SFL all the joys of a logic with an interleaving model, and so, *nothing is lost with respect to the main interleaving approaches to concurrency.*

Regarding logical and concurrent equivalences, it is now easy to see that Milner's *strong bisimulation* [12], $\sim_b$, the equivalence induced by modal logic is captured by the fixpoint-free fragment of this SFL sublogic, which we can denote by $\sim_{L\mu}$. Hence, the relation $\sim_{L\mu} \equiv \sim_b$ follows from Proposition 1 and the fact that modal logic characterises bisimulation on finite models.

5.2 A Separation Modal Mu-Calculus

The second sublogic is the Separation modal mu-calculus, $SL\mu$. This logic is obtained from SFL by allowing only the recognition of the duality between concurrency and conflict by using its structural operator. The syntax of $SL\mu$ is $\phi ::= Z \mid \neg\phi_1 \mid \phi_1 \wedge \phi_2 \mid \langle K \rangle \phi_1 \mid \phi_1 * \phi_2 \mid \mu Z.\phi_1$. We write $\sim_{SL\mu}$ for the equivalence induced by this SFL sublogic. It is easy to see that $SL\mu$ is more expressive than $L\mu$ in independence models simply because $SL\mu$ includes $L\mu$ and can differentiate concurrency from nondeterminism. However, the following counter-examples show that $\sim_{SL\mu}$ and $\sim_{hpb}$, in general, do not coincide.

Proposition 2. *Neither* $\sim_{hpb} \subseteq \sim_{SL\mu}$ *nor* $\sim_{SL\mu} \subseteq \sim_{hpb}$.

Proof. The two systems on the right (in Fig. 2) are hpb and yet can be distinguished by the formula $\phi = \langle a \rangle \langle c \rangle \mathrm{tt} * \langle b \rangle \langle d \rangle \mathrm{tt}$. On the other hand, the systems on the left are not hpb and cannot be differentiated by any $SL\mu$ formula. □

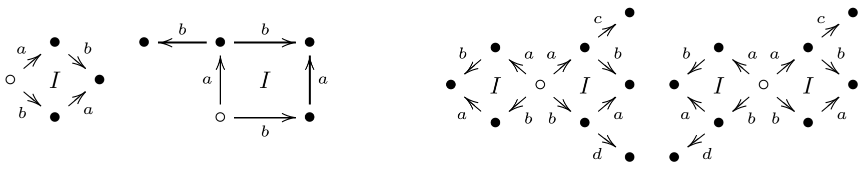


Fig. 2. Systems used for defining the equivalence induced by Separation $L\mu$

There are two reasons for the mismatch between $\sim_{hpb}$ and $\sim_{SL\mu}$. The first one (related with the two systems on the left in Fig. 2) is that $\sim_{hpb}$, unlike $\sim_{SL\mu}$, recognizes the pattern "an action a followed by both a concurrent action b and a causally dependent action b". The second reason, which is related with the systems on the right, is that $\sim_{hpb}$ cannot recognize some forms of conflict that $SL\mu$ can capture. For instance, the fact that even though two actions can be performed in parallel, it is also possible that the execution of one of them affects (prevents) the execution of transitions that depends on the other.

5.3 A Causal Modal Mu-Calculus

The third fragment to be considered is the Causal modal mu-calculus, $CL\mu$. This sublogic is obtained from SFL by allowing only the recognition of the duality between concurrency and causality throughout the modal operators of the logic. Its syntax is $\phi ::= Z \mid \neg\phi_1 \mid \phi_1 \wedge \phi_2 \mid \langle K \rangle_c \phi_1 \mid \langle K \rangle_{nc} \phi_1 \mid \mu Z.\phi_1$. Clearly, $CL\mu$ is also more expressive than $L\mu$ in independence models because of the same reasons given for $SL\mu$. The naturality of $CL\mu$ for expressing causal properties is demonstrated by the equivalence it induces in any model, written as $\sim_{CL\mu}$, which coincides with a *history-preserving bisimulation*, $\sim_{hpb}$, [10].

Theorem 1. $\sim_{CL\mu} \equiv \sim_{hpb}$

Proof. The proof goes by showing the two inclusions separately, $\sim_{hpb} \subseteq \sim_{CL\mu}$ and $\sim_{CL\mu} \subseteq \sim_{hpb}$. The first inclusion, $\sim_{hpb} \subseteq \sim_{CL\mu}$, can be proved by induction on $CL\mu$ formulae, called $\mathcal{F}_{CL\mu}$. For any two processes P and Q that belong to TSIs $\mathfrak{T}_1$ and $\mathfrak{T}_0$, respectively, if $P \sim_{hpb} Q$ then for all ϕ in $\mathcal{F}_{CL\mu}$, $P \models^{\mathfrak{T}_1} \phi \Leftrightarrow Q \models^{\mathfrak{T}_0} \phi$. The induction only requires (simplified) processes $P = (p, t_a)$ and $Q = (q, t_a)$, which are binary tuples in $S \times \mathfrak{A}$ of a TSI-model $\mathfrak{M}$, because $CL\mu$ only considers maximal sets and therefore support sets can be disregarded.

The second inclusion, $\sim_{CL\mu} \subseteq \sim_{hpb}$, is shown by contradiction. Suppose that for all ϕ in $\mathcal{F}_{CL\mu}$ we have that $P \models^{\mathfrak{T}_1} \phi \Leftrightarrow Q \models^{\mathfrak{T}_0} \phi$ and $P \not\sim_{hpb} Q$. The contradiction comes from the fact that even though processes P and Q satisfy the same $CL\mu$ formulae, there would be a transition in one of the processes that cannot be simulated by the other process in an hpb way, i.e., concurrent actions matched only with concurrent ones so as to keep the bisimulation synchronous, which is impossible. More precisely, synchrony in an hp bisimulation means that the last transition chosen in $\mathfrak{T}_1$ (resp. in $\mathfrak{T}_2$) is concurrent with the former transition also chosen in $\mathfrak{T}_1$ (resp. in $\mathfrak{T}_2$) iff the same pattern holds in the last two transitions chosen in $\mathfrak{T}_2$ (resp. in $\mathfrak{T}_1$). □

Corollary 1. $\sim_{CL\mu}$ *is decidable.*

Proof. Follows from Theorem 1 and the fact that $\sim_{hpb}$ is decidable. □

5.4 The Full Separation Fixpoint Logic

Although the equivalence induced by SFL is analysed in the following section using game-theoretical arguments, we first present a simple result that relates $\sim_{SFL}$ with $\sim_{hhpb}$, without using any game-theoretical machinery. Consider the counter-example presented in [8] (or the TSI representation of it), which is used there to disprove the coincidence between hpb and hhpb in free-choice systems. Due to lack of space such a counter-example is not reported here. Although the systems presented in [8] (in Figure 1) are not hhp bisimilar, they cannot be distinguished by any SFL formula. This result shows that in general $\sim_{hhpb}$ does not coincide with $\sim_{SFL}$. However, the precise relation between $\sim_{hhpb}$ and $\sim_{SFL}$ is to be defined in the following section. For now, we have the following result:

Proposition 3. $\sim_{SFL} \nsubseteq \sim_{hhpb}$

6 Bisimulation Games

Based on some of the results of the previous section, we give a game-theoretical characterization of the equivalence SFL induces by defining a Bisimulation Game for it. A knowledge on basic concepts about bisimulation games is assumed. An introduction to Bisimulation games can be found in [21].

The games presented here conservatively extend the hp bisimulation game, and therefore usual games for modal logics, i.e., classical bisimulation. We prove that these bisimulation games, which we call independence hp bisimulation (ihpb) games, characterize the equivalence induced by SFL. Most importantly, it is shown that the ihp bisimulation games induce an equivalence relation for concurrent systems that is strictly stronger than hpb and strictly weaker than hhpb. Remarkably, whereas ihpb games are decidable, hhpb games are undecidable in finite models. These features make ihpb games, and consequently the equivalence induced by SFL, an interesting candidate for an equivalence of true-concurrency. A hierarchy of true concurrent equivalences can be found in [7].

Definition 6.1. A *linearized concurrent run* is any sequence of transitions $r = t_0, ..., t_n$ of an independence model $\mathfrak{T}$ such that $s \xrightarrow{r} s'$ for some states s and s' in $\mathfrak{T}$. The set of linearized concurrent runs of a structure $\mathfrak{T}$ with respect to an initial state s can be written as $cRuns_s(\mathfrak{T})$. An empty run is written as $r = \epsilon$. The last transition of the sequence $r = t_0, ..., t_n$ is denoted by $last(r) = t_n$.

Remark 6.1. Any play of an (h)hp bisimulation game corresponds exactly to a pair of linearized concurrent runs in the structures at hand.

Before presenting the games for SFL, let us introduce a final definition that is related with the role of support sets as locally identifiable sets of concurrent transitions (conflict-free sets of transitions).

Definition 6.2. Two sets P_1 and P_2 are said to be *history-preserving isomorphic* with respect to a transition t iff there exists a bijection $\mathcal{B}$ between them such that for every $(t_i, t_j) \in \mathcal{B}$, if $t \leq t_i$ (resp. $t \ominus t_i$) then $t \leq t_j$ (resp. $t \ominus t_j$).

Definition 6.3. *(Independence history-preserving bisimulation Games).* Consider the standard bisimulation game for modal logics. There are two players, Spoiler and Duplicator, and a pair of structures $\mathfrak{T}_1$ and $\mathfrak{T}_2$ with initial states/processes P and Q, respectively. A configuration of a play in the game is a pair (r_1, r_2), where $r_1 \in cRuns_P(\mathfrak{T}_1)$ and $r_2 \in cRuns_Q(\mathfrak{T}_2)$. R is an independence history-preserving bisimulation between $\mathfrak{T}_1$ and $\mathfrak{T}_2$ if:

1. The initial configuration is (ϵ, ϵ). Therefore $(\epsilon, \epsilon) \in R$
2. (Rule for hp bisimulation). Let (r_1, r_2) be the current configuration, thus $(r_1, r_2) \in R$. Spoiler chooses one of the two systems, say $\mathfrak{T}_1$ ($\mathfrak{T}_2$), and picks a transition t_1 (t_2) that is enabled at r_1 (r_2). Duplicator has to respond by executing a transition t_2 (t_1) in the opposite structure $\mathfrak{T}_2$ ($\mathfrak{T}_1$) such that the two extended linearized concurrent runs stay synchronous. Synchrony means that whenever $last(r_1) \leq t_1$ (resp. $last(r_1) \ominus t_1$) then $last(r_2) \leq t_2$ (resp. $last(r_2) \ominus t_2$). The new configuration of the play is $(r_1.t_1, r_2.t_2)$, and hence $(r_1.t_1, r_2.t_2) \in R$.
3. (Additional rule for ihp bisimulation). Before Spoiler chooses a transition t_1 (t_2) from the enabled ones at r_1 (r_2), he can also choose a non-empty conflict-free subset of them to be the new set of enabled transitions P_1 (P_2) of size n. Duplicator must choose a history-preserving isomorphic set P_2 (P_1) with respect to $last(r_2)$ ($last(r_1)$) in the opposite structure $\mathfrak{T}_2$ ($\mathfrak{T}_1$).

If the play continues forever or Spoiler cannot make a move, then Duplicator wins. Otherwise Spoiler wins. The two structures are ihp bisimilar iff Duplicator has a winning strategy for every play in the game.

Lemma 1. *If $P \sim_{SFL} Q$, then Duplicator has a winning strategy for every play in the independence history-preserving bisimulation game $\mathcal{G}(\mathfrak{T}_1, \mathfrak{T}_2, P, Q)$.*

Proof. By constructing a winning strategy for Duplicator based on the fact that $P \sim_{SFL} Q$. Since $CL\mu$ induces an hp bisimulation and the ihpb game conservatively extends the hpb game, w.l.o.g. we can consider only the case when Spoiler plays Rule 3 of the ihpb game. □

Lemma 2. *If Duplicator has a winning strategy for every play in the independence history-preserving bisimulation game $\mathcal{G}(\mathfrak{T}_1, \mathfrak{T}_2, P, Q)$, then $P \sim_{SFL} Q$.*

Proof. By contradiction suppose that Duplicator has a winning strategy and $P \not\sim_{SFL} Q$. There are two cases. Suppose that Spoiler cannot make a move. This means that both $P \models^{\mathfrak{T}_1} [-]\,\mathrm{ff}$ and $Q \models^{\mathfrak{T}_2} [-]\,\mathrm{ff}$ only, which is a contradiction. The other case is when Duplicator wins in an infinite play. For the same reasons given previously, w.l.o.g., it is possible to consider only the case when Spoiler uses the Rule 3 of the ihpb game. But, this case also leads to a contradiction. □

The previous two Lemmas give a full game-theoretical characterization to the equivalence induced by SFL.

Theorem 2. *$P \sim_{SFL} Q$ iff Duplicator has a winning strategy for every play in the independence history-preserving bisimulation game $\mathcal{G}(\mathfrak{T}_1, \mathfrak{T}_2, P, Q)$.*

Theorem 3. *$\sim_{ihpb} \equiv \sim_{SFL}$ is decidable on finite systems.*

Proof. An independence history-preserving bisimulation game is a two-player zero-sum perfect-information (infinite) game, thus it is determined. Moreover, the length of the game is bounded. Duplicator wins when Spoiler cannot make a move or when a finite set of repeated configurations is visited infinitely often. In either case all possible winning strategies can be computed and therefore decidability follows. □

The previous results let us relate $\sim_{hhpb}$ with $\sim_{SFL}$ using game-theoretical arguments. Since both bisimulation games, namely the one for hhpb one and the one for ihpb, are conservative extensions of the hp bisimulation game, they can be compared just by looking at their additional rules with respect to the hpb game. So, consider the game-theoretical definition of hhpb as presented in [15]. We will describe the hhpb rule that extends the hpb game in the style used here.

Definition 6.4. *(Hereditary history-preserving bisimulation Games).* An hhp bisimulation game is just as an hp bisimulation game, as presented in Definition 6.3 (only Rules one and two), adding the following rule:

– (Additional rule for hhp bisimulation). Alternatively to a forwards move, having chosen one of the two systems, say $\mathfrak{T}_1$, Spoiler can choose a transition t_i that is backwards enabled at r_1 with respect to a previous configuration (w_1, w_2), i.e., a transition t_i that is concurrent with every transition t after w_1, where w_1 is obtained from r_1 by using the "diamond axioms" to push

t_i to the end, and then deleting t_i, and symmetrically for w_2. Duplicator must respond by choosing (deleting) the corresponding t_j in w_2. The new configuration of the game is that obtained by deleting both transitions t_i and t_j from the history of the game.

Now, only by showing that the additional rule for the hhpb game is at least as powerful as the additional rule for the ihpb game, and taking into account that $\sim_{hhpb}$ and $\sim_{SFL}$ do not coincide, by Proposition 3, we have:

Theorem 4. $\sim_{hhpb} \subset \sim_{SFL}$

7 Concluding Remarks and Related Work

We have given a logical characterization to the *dualities* that can be found when analysing *locally* the relationships between concurrency and conflict as well as concurrency and causality. This characterization aims at defining relationships between *model independence* and *logical independence*. Our study led to several positive results with respect to the equivalences induced by a number of modal logics whose denotations are based on these dualities.

Related Work: At a philosophical level, this study is similar to that in [3,5], a work primarily on mathematical logic using game logics for concurrency where model independence is captured explicitly with the use of Henkin quantifiers. At a more practical level, this work can be related to logics with partial order semantics at large. These logics, in the simplest cases, are given denotations that consider the one-step interleaving semantics of a particular independence model. Therefore no new logical constructions have to be introduced. The problem is that the explicit notion of independence in the models is completely lost. Thus, the usual approach is to introduce logical operators that somehow capture the independence in the models. In most cases that kind of logical independence is actually a *sequential* interpretation of concurrency based on the introduction of past operators sensitive to concurrent actions and a mixture of forwards and backwards reasoning (which usually leads to several undecidable results). A survey of logics of this kind can be found in [17]. Other logics with partial order semantics can be found in [2,15], but the literature has many more references.

Also, the work here presented can be related to logics for local reasoning. In particular, the use of separation properties to do local reasoning has been investigated elsewhere and applied in other settings [4,6,11,18,19,20]. However, as said before, apart from [3,5] the main motivation for this paper is different from all the examples given above, since we actually want to distill the relationships between model and logical independence so as to understand the semantic foundations of concurrent computations.

Acknowledgements. I am grateful to Julian Bradfield for helpful discussions. I thank Ian Stark and Colin Stirling for comments on a preliminary version of this paper. I also thank the referees for their comments.

References

1. Alur, R., Henzinger, T., Kupferman, O.: Alternating-time temporal logic. J. ACM 49(5), 672–713 (2002)
2. Alur, R., Peled, D., Penczek, W.: Model-Checking of Causality Properties. In: LICS, pp. 90–100. IEEE Computer Society, Los Alamitos (1995)
3. Bradfield, J.: Truth and Games: Essays in Honour of Gabriel Sandu. In: Aho, T., Pietarinen, A. (eds.) Acta Philosophica Fennica. Independence: Logics and Concurrency, vol. 78, pp. 47–70. Phil. Soc. of Finland (2006)
4. Bradfield, J., Esparza, J., Mader, A.: A Causal Fixpoint Logic. Unpub. (1997)
5. Bradfield, J., Fröschle, S.: Independence-Friendly Modal Logic and True Concurrency. Nord. J. Comput. 9(1), 102–117 (2002)
6. Brookes, S.: A semantics for concurrent separation logic. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 16–34. Springer, Heidelberg (2004)
7. Fecher, H.: A Completed Hierarchy of True Concurrent Equivalences. Inf. Process. Lett. 89(5), 261–265 (2004)
8. Fröschle, S.: The Decidability Border of Hereditary History Preserving Bisimilarity. Inf. Process. Lett. 93(6), 289–293 (2005)
9. Fröschle, S., Hildebrandt, T.: On Plain and Hereditary History-Preserving Bisimulation. In: Kutyłowski, M., Wierzbicki, T., Pacholski, L. (eds.) MFCS 1999. LNCS, vol. 1672, pp. 354–365. Springer, Heidelberg (1999)
10. Glabbeek, R., Goltz, U.: Refinement of Actions and Equivalence Notions for Concurrent Systems. Acta Inf. 37(4/5), 229–327 (2001)
11. Hayman, J., Winskel, G.: Independence and Concurrent Separation Logic. In: LICS, pp. 147–156. IEEE Computer Society, Los Alamitos (2006)
12. Hennessy, M., Milner, R.: Algebraic Laws for Nondeterminism and Concurrency. J. ACM 32(1), 137–161 (1985)
13. Joyal, A., Nielsen, M., Winskel, G.: Bisimulation from Open Maps. Inf. Comput. 127(2), 164–185 (1996)
14. Mazurkiewicz, A.: Trace Theory. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) APN 1986. LNCS, vol. 255, pp. 279–324. Springer, Heidelberg (1987)
15. Nielsen, M., Clausen, C.: Games and Logics for a Noninterleaving Bisimulation. Nord. J. Comput. 2(2), 221–249 (1995)
16. Nielsen, M., Winskel, G.: Models for Concurrency. In: Handbook of Logic in Computer Science, vol. 4, pp. 1–148. Oxford University Press, Oxford (1995)
17. Penczek, W.: Branching Time and Partial Order in Temporal Logics. In: Time and Logic: A Computational Approach, pp. 179–228. UCL Press (1995)
18. Pym, D., Tofts, C.: A Calculus and Logic of Resources and Processes. Formal Asp. Comput. 18(4), 495–517 (2006)
19. Reynolds, J.: Separation Logic: A Logic for Shared Mutable Data Structures. In: LICS, pp. 55–74. IEEE Computer Society, Los Alamitos (2002)
20. Sims, É.-J.: Extending Separation Logic with Fixpoints and Postponed Substitution. Theor. Comput. Sci. 351(2), 258–275 (2006)
21. Stirling, C.: Modal and Temporal Properties of Processes. Springer, Heidelberg (2001)

Separating Graph Logic from MSO

Timos Antonopoulos and Anuj Dawar

University of Cambridge Computer Laboratory, Cambridge CB3 0FD, UK
{timos.antonopoulos,anuj.dawar}@cl.cam.ac.uk

Abstract. Graph logic (GL) is a spatial logic for querying graphs introduced by Cardelli et al. It has been observed that in terms of expressive power, this logic is a fragment of Monadic Second Order Logic (MSO), with quantification over sets of edges. We show that the containment is proper by exhibiting a property that is not GL definable but is definable in MSO, even in the absence of quantification over labels. Moreover, this holds when the graphs are restricted to be forests and thus strengthens in several ways a result of Marcinkowski. As a consequence we also obtain that Separation Logic, with a separating conjunction but without the magic wand, is strictly weaker than MSO over memory heaps, settling an open question of Brochenin et al.

1 Introduction

Graph Logic (GL) was introduced by Cardelli et al. [2] as a query language on labelled directed graphs, modelled on spatial logic. It extends the first-order logic of such graphs (with quantification over vertices, edges and labels) with a spatial connective: thus, a formula $(\varphi|\psi)$ is true in a graph G if, and only if, G can be decomposed into two subgraphs G_1 and G_2 such that $G_1 \models \varphi$ and $G_2 \models \psi$. Here, when we say that G is decomposed into G_1 and G_2, we mean that these two graphs may share vertices but not edges.

It is easy to see that any formula of GL can be translated into an equivalent formula of second-order logic in which the second-order quantifiers are restricted to sets of edges. This is a version of monadic second-order logic (MSO) known as MS_2 in the works of Courcelle (see [4]) and also as *guarded second order logic* or GSO (see [8]). The expressive power and complexity of GL were systematically investigated in [6], where it was shown that, like MSO, GL can express complete problems at every level of the polynomial hierarchy. Moreover, when we restrict ourselves to labelled graphs that code *words* in the natural way, then GL can (just like MSO) define exactly the regular languages. A conjecture that was left open in [6] was that the containment of GL in MS_2 is strict, i.e. that there is a property of graphs definable in MS_2 that is not definable in GL.

Marcinkowski [9] settled this conjecture positively. The property he constructs crucially relies on the presence of an unbounded set of labels in the graphs considered, and on the ability of formulas of GL and (an enhanced) monadic second-order logic, which he calls MSO+ to quantify over labels. This reliance on label-quantification is something Marcinkowski calls a "win on technicalities" and he explicitly leaves open the more fundamental question of whether GL *without quantification over labels*, which he calls GL−, is strictly weaker than MSO. We settle this question in this paper. To be precise, we show that there is a property of unlabelled, directed *forests* that is expressible in MSO but not in GL. Since, on forests, one can replace quantification over sets of edges with quantification overs sets of vertices, this yields the stronger result that GL does not even contain MS_1.

As a corollary, we obtain a result concerning separation logic (SL), a logic of assertions used in Hoare-style proof systems [10]. Brochenin et al. [1] consider the expressive power of SL($*$), which is separation logic with a separating conjunction ($*$) but without the magic wand ($-\!\!*$) and conjecture that it is properly contained in MSO. Since SL is essentially the same as GL over structures known as memory heaps, and since the forests we use to separate GL from MSO can be coded as such heaps, a direct consequence of our proof is a positive resolution of this conjecture.

The MSO-definable property of forests that we demonstrate is not definable in GL is the following: a forest contains a tree in which the number of leaves is a multiple of 3. The use of forests rather than trees is crucial to the proof and thus the question of whether there is a regular tree language that is not definable in GL remains open. We show that a natural candidate, the class of binary trees with an even number of leaves, is definable in GL. Indeed, any regular binary tree language accepted by a bottom-up automaton that is the product of two-state automata can be defined in GL.

In the rest of the paper, we first introduce, in Section 2, the logics we deal with. In Section 4 we investigate the power of GL on trees by showing that a certain class of regular tree languages is included in those definable in GL. Section 3 presents the main result, while Section 5 explores the consequences for separation logic.

2 Background and Preliminaries

Graph Logic was introduced in [2] as a logic for querying graphs. It is based on a view of graphs as terms in a suitable algebra, involving a composition operator. Thus, the graphs are built up from single edges by repeated compositions. In the present paper, we treat the logic instead as an extension of the ordinary first-order logic of graphs, with the composition operator appearing in the formulas. The two views are equivalent, as discussed in [6]. We assume familiarity with the syntax and semantics of first-order logic (FO).

2.1 Graph Logic

Fix a set X of vertex names. A graph G consists of a finite set E of edges, and an incidence map $I_G : E \to X \times X$ that associates with each edge a pair of vertices that we call its endpoints. It is easy to see that any such graph can be seen as a finite, directed, unlabelled graph (with no isolated nodes) in the usual sense. We will briefly consider the case of edge labels below. The syntax of Graph Logic can then be specified as follows.

Definition 2.1 (GL Syntax). *Let V be a countable set of vertex variables. A GL formula is defined to be one of the following for $t_i \in X \cup V$, and $x \in V$ and for GL formulas φ_i:*

$$\varphi := \mathbf{0} \mid \top \mid E(t_1, t_2) \mid t_1 = t_2 \mid \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1|\varphi_2 \mid \exists x\ \varphi_1.$$

This appears at first sight to be an extension of first-order logic with one extra formula formation rule: $\varphi_1|\varphi_2$. However, the interpretation of the atomic formulas is different from the usual rules for FO. To be precise, $E(t_1, t_2)$ is true in a graph G just in case G consists of a single edge e whose end points are named by the terms t_1 and t_2. To define the semantics of the composition operator, we need to first define graph composition.

Let G_1 and G_2 be two graphs with edge sets E_1 and E_2 and incidence maps I_1 and I_2 respectively. The graph $G = G_1 \mid G_2$ is defined to be the graph whose edge set and incidence map are the disjoint unions $E_1 \uplus E_2$ and $I_1 \uplus I_2$. Note that G_1 and G_2 may share vertices. The semantics of the formula $\varphi_1|\varphi_2$ is now defined by saying $G \models (\varphi_1|\varphi_2)$ just in case $G = G_1 \mid G_2$ for some G_1 and G_2 such that $G_1 \models \varphi_1$ and $G_2 \models \varphi_2$. Formally, the semantics is as given below.

Definition 2.2 (GL Semantics). *Let G be a graph with edge set E and incidence map I. Also let $\alpha : V \to X$ be an assignment of vertex names to variables. We write $\hat{\alpha}$ for the extension of α to the domain $X \cup V$ by letting $\hat{\alpha}(x) = x$ for all $x \in X$. The semantics for the Boolean connectives are defined as usual. The following holds for the rest of the GL constructs.*

$$\begin{array}{ll}
(G,\alpha) \models \mathbf{0} & \textit{iff } E = \emptyset, \\
(G,\alpha) \models \top & \quad \textit{for any graph } G, \\
(G,\alpha) \models E(t_1,t_2) & \textit{iff } E = \{e\} \textit{ and } I(e) = (\hat{\alpha}(t_1), \hat{\alpha}(t_2)), \\
(G,\alpha) \models t_1 = t_2 & \textit{iff } \hat{\alpha}(t_1) = \hat{\alpha}(t_2), \\
(G,\alpha) \models \varphi_1|\varphi_2 & \textit{iff there exist } G_1, G_2 \textit{ s.t. } G = G_1|G_2 \textit{ and } (G_i,\alpha) \models \varphi_i, \\
(G,\alpha) \models \exists x\ \varphi_1 & \textit{iff there is an } a \in X \textit{ s.t. } (G, \hat{\alpha}[x \mapsto a]) \models \varphi_1,
\end{array}$$

where $\hat{\alpha}[x \mapsto a]$ denotes the map that agrees with $\hat{\alpha}$ at all values other than x and maps x to a.

In spite of the variance in the interpretation of atomic formulas, it is not difficult to prove that every FO formula on finite graphs can be translated into a formula of GL. In particular, an atomic FO formula of the form $E(a, b)$ asserting the existence of an edge between vertices a and b is equivalent to the GL formula $(E(a, b)|\top)$. In turn, every GL formula can be translated into one of second-order logic (see [6] for details).

The *monadic second-order logic* (MSO) of graphs comes in two variants, called MS_1 and MS_2 by Courcelle [4]. Both extend FO with second-order quantifiers that can quantify over sets. In the case of MS_1, this second-order quantification is limited to sets of *vertices*, while MS_2 allows quantification over sets of *edges*.

To be precise MS_1 extends FO with a countable set $\mathcal{Z}$ of second-order variables. For every $Z \in \mathcal{Z}$ and term t, $Z(t)$ is an atomic formula and for every formula φ, $\exists Z\varphi$ is also a formula. The latter is satisfied by a graph G just in case there is a set $A \subseteq X$ of vertices such that φ is true in G when all free occurrences of Z are interpreted by the set A. Similarly, MS_2 allows atomic formulas $Z(t_1, t_2)$ and quantifiers $\exists Z\varphi$. The latter is true in a graph G just in case there is a set $A \subseteq E$ of edges such that φ is true in G when all free occurrences of Z are interpreted by the set A.

It is not difficult to see that each formula of GL can be translated into an equivalent formula of MS_2. It is not the case that GL can be translated into MS_1 and in Section 6 we construct an example exhibiting this. The question left open from [6] and [9] is whether there is a sentence of MS_2 that is not equivalent to any sentence of GL. This is the question we answer in the present paper.

Apart from Section 6, we are concerned in this paper with graphs that are trees or forests. On such graphs, MS_2 is no more expressive than MS_1. Indeed, since each vertex has at most one incoming edge, there is an easily definable one-to-one map from edges to vertices that allows us to replace quantification over edges with quantification over vertices. Thus, for our purposes, it makes sense to speak just of MSO, without distinguishing the two varieties. We will simply speak about comparing GL and MSO.

In the intended applications of GL, graphs do not just consist of sets of edges, but also come with labels. There are two ways in which labels can be introduced into the language of GL. If the set of possible labels is a fixed finite set Σ, we may regard each $\sigma \in \Sigma$ as defining a set of edges E_σ with associated incidence relation $I : E_\sigma \to X \times X$. The logic then allows atomic formulas $E_\sigma(t_1, t_2)$ for each $\sigma \in \Sigma$. On the other hand, if the set of labels is unbounded, we include in the language the set Σ and a countable set of label variables L. A graph now is a set of edges E together with an incidence relation $I : X \times \Sigma \times X$. We allow atomic formulas $E_\sigma(t_1, t_2)$ for each $\sigma \in \Sigma \cup L$ and also allow equality testing and quantification over labels. Thus, for $\sigma, \tau \in \Sigma \cup L$, we have atomic formulas $\sigma = \tau$ and for any formula φ, we can form the formula $\exists l\varphi$ for $l \in L$. Marcinkowski [9] termed this extended logic GL+ and proved that it was strictly weaker than MSO+, the corresponding extension of MS_2. Our result strengthens his. In the rest of this paper, we will not be concerned with GL+ or MSO+.

Regular Languages. The case of bounded alphabets Σ includes many interesting applications of MSO. For instance, Σ-labelled graphs which consist of a single path from a source s to a terminal t can be identified with words over the alphabet Σ. It is well known, by results of Büchi, Elgot and Trakhtenbrot (see [7]) that in this case MSO can define exactly the regular languages. It was shown in [6] that the same is true of GL. The correspondence between MSO and finite automata extends further to Σ-labelled trees, and it remains an open question (see [9]) whether GL can express all regular tree languages. In Section 6, we show that one suggested candidate for separation—the class of binary trees with an even number of leaves—is definable in GL. Indeed, any binary tree language accepted by a bottom-up deterministic two-state automaton is shown to be definable. However, if the graphs are *forests* rather than trees, our main result in Section 3 proves that GL is strictly weaker than MSO.

Recall the following. A deterministic bottom-up tree automaton is a tuple $A = (\Sigma, \delta_2, \delta_1, q_1, Q, Q_f)$, where Σ is the alphabet, Q the set of states, $q_1 \in Q$ the initial state, $Q_f \subseteq Q$ the set of accepting states and $\delta_1 : Q \times \Sigma \to Q$ and $\delta_2 : Q \times \Sigma \times Q \times \Sigma \to Q$ are two transition functions. Given a Σ-labelled binary tree, the automaton A assigns a state in Q to each node of the tree. The leaves are assigned the initial state q_1. Each node a with one child is assigned $\delta_1(q, \sigma)$ where σ is the label on the edge leaving a and q is the state assigned to its child. Similarly, each node a with two children is assigned $\delta_2(q, \sigma, q', \sigma')$ where σ and σ' are the labels on the two edges leaving a and q and q' are the states assigned to its two children. A accepts a tree T if the state assigned to the root of T is in Q_f.

2.2 GL Games

The main tool for proving non-expressibility of a property in FO or MSO is Ehrenfeucht-Fraïssé Games (see [7]). Such games have also been adapted to spatial logics (see [5]). Here we present an adaptation of the game for GL.

A GL Game is played by two players, Spoiler and Duplicator, and consists of k rounds, for some $k \in \mathbb{N}$. The game is played on two graphs F and G with the initial position being $\langle (F, \bar{c}), (G, \bar{c}) \rangle$ for some tuple $\bar{c}$ of vertex names. The position at a round $i \leq k$ is defined to be a pair of structures with distinguished vertices $\langle (F^i, \bar{a}), (G^i, \bar{b}) \rangle$ where $\bar{a} = a_1, \ldots, a_p$ and $\bar{b} = b_1, \ldots, b_p$ are tuples of vertex names extending $\bar{c}$, and F^i and G^i are subgraphs of F and G respectively.

At the beginning of the ith round, Spoiler chooses one of the two structures $(F^i, \bar{a})$ or $(G^i, \bar{b})$ and makes either a *colouring* move or a *first order* move. Suppose, without loss of generality, that Spoiler chooses $(F^i, \bar{a})$. For a first order move, he chooses a vertex $a_{p'}$ in F^i and Duplicator must respond with a vertex $b_{p'}$ in G^i. The position at the next round is $\langle (F^i, \bar{a}, a_{p'}), (G^i, \bar{b}, b_{p'}) \rangle$.

If he chooses to play a colouring move, Spoiler finds two graphs F_1^i and F_2^i such that $F^i = F_1^i | F_2^i$. Duplicator must respond with two graphs G_1^i and G_2^i such that $G^i = G_1^i | G_2^i$. Spoiler then decides whether the position at the next round is $\langle (F_1^i, \bar{a}), (G_1^i, \bar{b}) \rangle$ or $\langle (F_2^i, \bar{a}), (G_2^i, \bar{b}) \rangle$. In general, when describing a colouring move, we will say that Spoiler has coloured the edges in F_1^i *white* and those in F_2^i *black* and Duplicator has responded by colouring G_1^i white and G_2^i black.

The game ends after k rounds, or if one of the two graphs in some position is empty or consists of a single edge. Let $h : X \rightharpoonup X$ be the partial map defined by $a_j \mapsto b_j$. Spoiler wins if one of the following three conditions holds, and Duplicator wins in all the other cases.

1. Exactly one of the graphs is empty.
2. One of the graphs is a single edge, with both endpoints in the domain of h and h is not an isomorphism between the two graphs.
3. The mapping h is not one-to-one.

We define the quantifier rank of a GL formula, by counting first-order quantifiers and composition operators equally. To be precise, the quantifier rank of a

GL formula φ is rank(φ) which is defined as usual for the Boolean connectives and as follows for the rest:

If $\varphi = \mathbf{0}, \top, t_1 = t_2, E(t_1, t_2)$	then $\text{rank}(\varphi) = 0$.
If $\varphi = \varphi_1 \vert \varphi_2$	then $\text{rank}(\varphi) = \max(\text{rank}(\varphi_1), \text{rank}(\varphi_2)) + 1$.
If $\varphi = \exists x\ \varphi_1$	then $\text{rank}(\varphi) = \text{rank}(\varphi_1) + 1$.

The proof of the following lemma then follows the standard methods for Ehrenfeucht-Fraïssé games.

Lemma 2.3 ([6]). *If Duplicator has a winning strategy for the k-round game on graphs F and G with initial position $\langle (F, \bar{c}), (G, \bar{c}) \rangle$, then for any GL formula φ with rank(φ) $\leq k$ and using only names from $\bar{c}$, it holds that*

$$F \models \varphi \textit{ if, and only if, } G \models \varphi.$$

The main use of this lemma is to show that some property P is not expressible in GL. This can be formulated as in the following corollary. Since we are interested in properties that are invariant under the choice of vertex names, we can restrict ourselves to games in which $\bar{c}$ is empty in the initial position.

Corollary 2.4. *A property P is inexpressible in GL, if and only if for each $k \in \mathbb{N}$, there exist structures F_k and G_k, such that $F_k \in P$ and $G_k \notin P$, and Duplicator has a winning strategy for the k-round GL played game on $\langle F_k, G_k \rangle$.*

We write $F \equiv_k^{\text{GL}} G$ to denote that F and G cannot be distinguished by any sentence φ of GL with rank(φ) $\leq k$. Similarly, $F \equiv_k^{\text{MSO}} G$ denotes that F and G agree on all MSO sentences with quantifier rank at most k. A translation of GL formulas to MSO is given in [6] that takes a GL formula of quantifier rank k to an MSO formula of rank at most $2k + 1$. From this, Lemma 2.5 below follows.

Lemma 2.5. *For any $k \in \mathbb{N}$, and graphs G, H, if $G \equiv_{2k+1}^{\text{MSO}} H$ then $G \equiv_k^{\text{GL}} H$.*

Disjoint Unions. We will often make use of constructions involving disjoint unions of graphs as well as unions disjoint apart from a fixed number of named vertices. We make these notions precise and fix notation here. Recall that we are primarily interested in properties of graphs that are invariant under isomorphisms or, equivalently, invariant under renaming of vertices. Given graphs G and H, we write $G \oplus H$ for the *disjoint union* of G and H. This is a graph $G'|H$ for a graph G' that is isomorphic to G but shares no vertices with H. For a fixed tuple $\bar{a}$ of vertex names, we write $G \oplus_{\bar{a}} H$ for the graph $G'|H$ where G' is obtained from G by renaming all vertices apart from those in $\bar{a}$ to be distinct from any vertex in H. In other words, $G \oplus_{\bar{a}} H$ is a graph obtained from the disjoint union of G and H while identifying vertices in $\bar{a}$. For an indexed family $\{G_i \mid i \in I\}$ of graphs, we write $\bigoplus_{i \in I} G_i$ to denote the disjoint union of all the graphs in the family. For a natural number n, we also write $n \cdot G$ for the graph $\bigoplus_{1 \leq i \leq n} G_i$ where each G_i is isomorphic to G.

3 Separating GL from MSO

In this section we present the main result, namely that there are properties of forests that are expressible in MSO but not in GL. The property in question is that one of the trees in the forest has a number of leaves that is a multiple of three. To see that this property is MSO definable, note that there is a simple 3-state deterministic bottom-up tree automaton that checks whether a given tree T has a number of leaves that is a multiple of three. Thus, there is an MSO sentence θ that defines this class of trees. Since we can also construct an MSO formula $\mathsf{path}(x, y)$ that asserts that there is a path from x to y, we can use this to obtain a formula that asserts the existence of a root x such that the tree of nodes reachable from x satisfies θ.

We begin with a simple intuitive example to illustrate why the colouring move in a GL game involves a loss of information for Spoiler, which Duplicator can exploit in a way that she cannot in the corresponding MSO game. Consider two graphs, G_1 and G_2. Let $G = G_1 \oplus G_2$ and let G' be $G_1 \oplus_v G_2$ for some vertex v. On a game played on $\langle G, G' \rangle$, if Spoiler plays a colouring move that splits either graph into G_1 and G_2, it is clear that Duplicator has a winning response unless the vertex v was previously chosen. Thus, information on how the original graph (G or G') was connected has been lost.

This simple example illustrates the idea behind the construction of the two forests on which the game will be played. The forests we consider consist of a single *comb*—i.e. a binary tree consisting of a simple path with other simple paths branching off from it—and a large number of disjoint simple paths. These simple paths act as *noise* which allow Duplicator, in response to a suitable colouring move by Spoiler, to remove some of the branches of the comb (thereby changing the number of leaves) and hide them among the noise. We now proceed to a more detailed description of the construction.

Definition 3.1. *A* fork *is a node in a binary tree that has two distinct children.*
A comb *is a binary tree where for any two distinct forks v_1 and v_2, either v_1 is an ancestor of v_2 or v_2 is an ancestor of v_1.*

Let C be a comb, and r its root. As long as there is at least one fork in C, there exist two leaves t, t' such that the path from r to either of them contains all the forks of C. Fix t to be one of those leaves and call the path from r to t, the *spine* of the comb C. Each fork a of C, has one child on the spine of C while the other is a vertex that is the root of a subtree consisting of a simple path to a leaf b. For each fork a, we call the path from a to b, a *tooth* of the comb C. Note that the number of leaves of a comb is one more than the number of teeth.

Let $n, s \in \mathbb{N}$. We write $C_{n,s}$ for the comb with n teeth where the length of each tooth, as well as the distance between any two successive forks is s. We also define $C_{n,s}^{-i}$, for $1 \leq i \leq n$, to be the comb $C_{n,s}$ with the i-th tooth missing. That is, the distance between the $(i-1)$st fork and the next one is $2s$. Finally, let S_n denote a string (i.e. a tree consisting of a single path) of length n.

Suppose a, a' are successive forks in the spine of a comb. Then the substring of the spine with endpoints a, a' is called a *segment*. A *block* of a comb is a segment with endpoints a, a', together with the tooth attached to a'.

The following lemma states some easy consequences of the fact that MSO can only define regular languages when restricted to strings.

Lemma 3.2. *For each $k \in \mathbb{N}$, there exist $s, n, l \in \mathbb{N}$, such that:*

1. *for every $w > s$ and every m, $S_w \equiv_k^{\mathrm{MSO}} S_{w+ms}$,*
2. *for every $t > n$, and every m, $C_{t-2l,ms} \equiv_k^{\mathrm{MSO}} C_{t+2,ms}$.*

Lemma 3.2 essentially gives specific periodic properties for the sizes of strings and of combs of constant segment length, with respect to $\equiv_k^{\mathrm{MSO}}$-equivalence, for any $k \in \mathbb{N}$. One consequence we can derive is that $C_{n,s} \equiv_k^{\mathrm{MSO}} C_{n+1,s}^{-i}$, for any n and the s given by the lemma, since $S_s \equiv_k^{\mathrm{MSO}} S_{2s}$ and the comb $C_{n+1,s}^{-i}$ can be seen as $C_{n,s}$ with its ith segment of length s replaced by a segment of length $2s$.

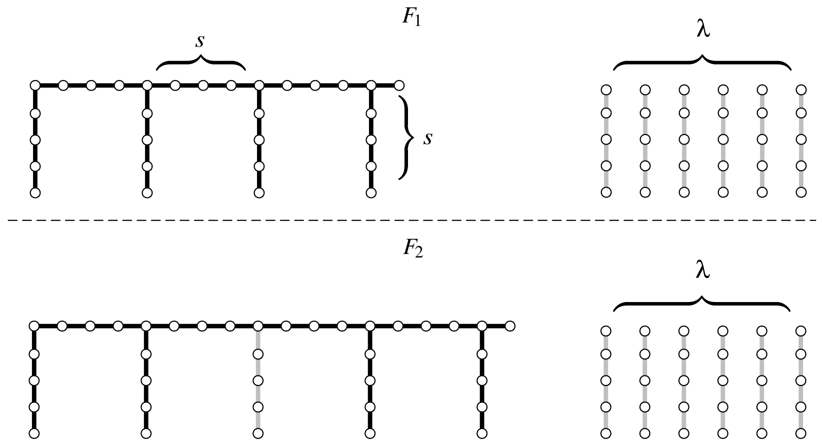
The next lemma can be proved by a standard application of Ehrenfeucht-Fraïssé games and appears in [3] for general structures $\mathbb{A}$ and $\mathbb{B}$.

Lemma 3.3. *For any $k \in \mathbb{N}$, there is $\lambda \in \mathbb{N}$, such that if $\mathbb{A}$ is the disjoint union of λ pairwise $\equiv_k^{\mathrm{MSO}}$-equivalent structures, and $\mathbb{B}$ is the disjoint union of $\lambda + 1$ such structures, each $\equiv_k^{\mathrm{MSO}}$-equivalent to the ones in $\mathbb{A}$, then $\mathbb{A} \equiv_k^{\mathrm{MSO}} \mathbb{B}$.*

Before giving the details of the construction, we consider an example. Consider two forests $F_1 \cong C_{n,s} \oplus \lambda \cdot S_s$, and $F_2 \cong C_{n+1,s} \oplus \lambda \cdot S_s$. That is, each one consists of the disjoint union of a comb with a large number of strings. Suppose that, in the GL game played on this pair of structures for $\lfloor (k-1)/2 \rfloor$ moves, Spoiler plays a colouring move on F_1, by colouring the comb $C_{n,s}$ in black, and the set of disjoint strings $\lambda \cdot S_s$ in white. Then Duplicator can respond by colouring for some $1 \leq i \leq n$, the subgraph $C_{n+1,s}^{-i}$ in black and the λ copies of S_s together with the remaining tooth of the comb in white, as shown in the figure below. By Lemmas 3.2 and 3.3, whichever the pair of graphs on which Spoiler chooses to continue the game, they are $\equiv_k^{\mathrm{MSO}}$-equivalent. Since $\equiv_k^{\mathrm{MSO}}$-equivalence implies $\equiv_{\lfloor (k-1)/2 \rfloor}^{\mathrm{GL}}$, this means that Duplicator has a winning strategy for the rest of the game. This shows that using the colouring move to distinguish the comb from the noise, which would have been the right move for Spoiler in the MSO game on this pair of structures, is a losing move in the GL game. Extending this idea, we show that by taking the comb to be big enough and by adding enough noise, we can ensure that Spoiler has no good moves.

Let $\mathcal{F}$ be the class of forests in which some tree has 0 (mod 3) leaves. In what follows, we show that for any $k \in \mathbb{N}$, there exist two forests F and G, such that $F \in \mathcal{F}$, $G \notin \mathcal{F}$, and Duplicator wins the k-round game on F and G. We proceed with the details of the construction of these two forests.

Fix $k \in \mathbb{N}$ and let s and n be as given by Lemma 3.2 for k, and λ as given by Lemma 3.3 for k. By Lemma 3.2, for every $w \in \mathbb{N}$ there is a $w' \in \{1, \ldots, 2s\}$ with $S_w \equiv_k^{\mathrm{MSO}} S_{w'}$ (indeed, either $w \leq 2s$ and we can take $w = w'$ or we can find a suitable w' and m such that $w = w' + ms$). We define $N = \prod_{1 \leq i,j \leq 2s}(i + j)$.



Note that this has the property that $(i+j) \mid N$ (i.e. $i+j$ divides N) for all $i, j \leq 2s$.

Let $f_k = (2^5 \cdot \lambda s N)^k \cdot 6n \cdot \lambda \cdot 2^{6N} - 1$. We define the forests F and G as follows.

$$F = C_{f_k,N} \oplus \bigoplus_{2s+1 \leq i \leq 3s} (\lambda \cdot 2^{3sk} \cdot S_i),$$
$$G = C_{f_k+2,N} \oplus \bigoplus_{2s+1 \leq i \leq 3s} (\lambda \cdot 2^{3sk} \cdot S_i).$$

The collection of strings $\bigoplus_{2s+1 \leq i \leq 3s} (\lambda \cdot 2^{3sk} \cdot S_i)$ in both graphs, is called *noise*, and each individual string from that collection is called a *noise-string*.

By Lemma 2.5, for each $\ell \geq 3$, and for any two graphs H_1 and H_2, $H_1 \equiv_\ell^{\mathrm{MSO}} H_2$ implies that $H_1 \equiv_{\lfloor(\ell-1)/2\rfloor}^{\mathrm{GL}} H_2$. In the following, to simplify notation we show that for any $k \geq 1$, Duplicator can win the $\lfloor(k-1)/2\rfloor$-round GL game on F and G. To show that $F \equiv_{\lfloor(k-1)/2\rfloor}^{\mathrm{GL}} G$, we establish that Duplicator can maintain the following condition in the $\lfloor(k-1)/2\rfloor$-round game on F and G.

If the game position after i rounds is $(F_i, \bar{a})$ and $(G_i, \bar{b})$, then one of the two conditions below holds, for $k' = k - i$:

1. $(F_i, \bar{a}) \equiv_{k'}^{\mathrm{MSO}} (G_i, \bar{b})$, or
2. $F_i = F' \oplus_{c_1,c_2} \overline{F'}$ and $G_i = G' \oplus_{d_1,d_2} \overline{G'}$, where:
   (a) $(F', c_1, c_2) \cong (C_{f_{k'},N}, r, t) \oplus \bigoplus_{2s+1 \leq i \leq 3s} (\lambda \cdot 2^{3sk'} \cdot S_i)$,
   (b) $(G', d_1, d_2) \cong (C_{f_{k'}+2,N}, r, t) \oplus \bigoplus_{2s+1 \leq i \leq 3s} (\lambda \cdot 2^{3sk'} \cdot S_i)$,
   (c) no element of $\bar{a}$ is in F' and no element of $\bar{b}$ is in G',
   (d) $(\overline{F'}, c_1, c_2) \equiv_{k'}^{\mathrm{MSO}} (\overline{G'}, d_1, d_2)$.

Notice that for any $i \leq \lfloor(k-1)/2\rfloor$, $(F_i, \bar{a}) \equiv_{k-i}^{\mathrm{MSO}} (G_i, \bar{b})$ implies $(F_i, \bar{a}) \equiv_{\lfloor(k-1)/2\rfloor - i}^{\mathrm{GL}} (G_i, \bar{b})$. The condition above states that at each round i of the game, either both graphs are $\equiv_{k'}^{\mathrm{MSO}}$-equivalent, and thus Duplicator wins the game, or the following holds. In both graphs F_i and G_i after i rounds, there exist subgraphs F' and G' respectively, each composed of a large enough comb and enough noise-strings. Furthermore the complements of these graphs F' and G' inside F_i and G_i, are $\equiv_{k'}^{\mathrm{MSO}}$-equivalent.

Duplicator has a reply to any move Spoiler makes whenever condition (1) holds, namely $(F_i, \bar{a}) \equiv_{k'}^{\mathrm{MSO}} (G_i, \bar{b})$. We need to show, that if (2) holds, Duplicator has a response that maintains the condition. We proceed by induction on the number of rounds. In the beginning of the game, (2) holds by construction.

Suppose then that at some round i, (2) holds. Suppose furthermore that Spoiler makes a first order move, and chooses a vertex in one of the two graphs. If he chooses a vertex in $\overline{F'}$ or $\overline{G'}$, Duplicator can reply since $\overline{F'} \equiv_{k'}^{\mathrm{MSO}} \overline{G'}$. The subgraphs guaranteeing the condition (2) can then be found. Assume then that Spoiler picks a vertex v in F' or G', and by symmetry, assume that he does so in F'. Then two subgraphs ensuring the condition (2) holds, can also be found, for $k'-1$. The details are omitted.

Consider the case where Spoiler makes a colouring move. As the arguments are similar for both cases, we assume that Spoiler colours the graph G_i. Each noise-string in G_i of length h receives one of 2^h possible colourings c, i.e. a labelling of each of the edges as either black or white. Since there are, in general, many more noise-strings of length h than this, some colouring may be repeated many times. We will call the most frequently occurring colouring c (or any one of them if there is a tie), the *primary colouring* of the noise-strings of length h.

A colouring move of Spoiler is considered in cases, depending on how he colours the subgraph F' or G'. We give an outline of the main procedure that Duplicator applies as a reply to many of Spoiler's moves. We denote with C the subgraph inside F' that is isomorphic to the comb $C_{f_{k'},N}$, and similarly we denote with D the respective subgraph isomorphic to $C_{f_{k'}+2,N}$ of G'. The remaining subgraphs in F' and G' comprising the set of noise-strings are denoted by F_S and G_S respectively.

The graph F_i (resp. G_i) comprises the subgraphs C, F_S and $\overline{F'}$ (resp. D, G_S and $\overline{G'}$). Since $F_S \cong G_S$ and $\overline{F'} \equiv_{k'}^{\mathrm{MSO}} \overline{G'}$, Duplicator has a reply to the way Spoiler colours G_S and $\overline{G'}$. For the response to the colouring of D, Duplicator proceeds as follows. She defines vertices c_3, c_4 and d_3, d_4 in (C, c_1, c_2) and (D, d_1, d_2) respectively, so that:

$$(C, c_1, c_2) = (C_1, c_1, c_3) \oplus_{c_3} (C_2, c_3, c_4) \oplus_{c_4} (C_3, c_4, c_2),$$
$$(D, d_1, d_2) = (D_1, d_1, d_3) \oplus_{d_3} (D_2, d_3, d_4) \oplus_{d_4} (D_3, d_4, d_2),$$

for some C_1, C_2, C_3 and D_1, D_2, D_3, such that $C_1 \equiv_k^{\mathrm{MSO}} D_1$ and $C_3 \equiv_k^{\mathrm{MSO}} D_3$. Furthermore, she ensures the following for C_2 and D_2. Given a choice of vertices d_3 and d_4, either the black and the white components of D_1 and D_3 are disconnected from the black and the white ones in D_2 respectively or not. In the first case, Duplicator ensures the same for c_3, c_4 and also makes sure that all the black and white components in D_2 appear in C_2 in equal numbers, and C_2 contains additional components that appear more than λ times in the graphs C_1, C_3 and F_S, and thus also more than λ times in D_1,D_3 and G_S, by definition. The resulting white and black subgraphs of F' and G' are therefore respectively $\equiv_k^{\mathrm{MSO}}$-equivalent (and therefore $\equiv_{k'}^{\mathrm{MSO}}$-equivalent).

In the second case, Duplicator ensures that the spine of C_2 is $\equiv_k^{\mathrm{MSO}}$-equivalent to the whole of D_2, and the teeth of C_2 are split into white and black components that appear more than λ times in C_1,C_3 and F_S, and the splitting is such that the resulting components from the teeth of C_2 are disconnected from the spine of C_2. Again, the resulting white (respectively black) subgraphs of F' and G' are $\equiv_k^{\mathrm{MSO}}$-equivalent (and therefore $\equiv_{k'}^{\mathrm{MSO}}$-equivalent).

As was stated above, the argument for the colouring moves of Spoiler, is considered in cases, depending on how Spoiler chooses to colour the subgraphs D and G_S. In all cases considered except one, Duplicator can guarantee that condition (1) holds in the next stage, that is $(F_{i+1}, \bar{a}) \equiv_{k'-1}^{\mathrm{MSO}} (G_{i+1}, \bar{b})$. The one exception is the case where Spoiler colours in black some part of a segment in D, that is longer than s edges and furthermore: for all $h \in [2n+1, 3n]$, the noise-strings of length h are primarily coloured black; no substring of a segment, larger than s edges is coloured white in D by Spoiler; and Spoiler colours at most $8 \cdot \lambda \cdot s \cdot N$ blocks in D using both colours.

We omit the details of the argument, which lead us to the following theorem.

Theorem 3.4. *The class of forests that contain a tree with* 0 (*mod* 3) *number of leaves, is not definable in GL.*

4 GL on Binary Trees

It is known that GL is as expressive as MSO on words, and we have shown in the previous section that it is strictly weaker on some classes of graphs, in particular forests. A natural question that arises is whether GL is as expressive as MSO on trees, especially as the latter is a widely studied logic of trees. We are not able to settle this question, but we do show that GL can be more expressive than expected. In particular, the property of a tree having an even number of leaves is expressible in GL. Note that, we do not know how to extend this to forests—i.e. to show that use of the modulus 3 in Theorem 3.4 is essential. Nor do we know how to express in GL that a tree has a number of leaves that is a multiple of 3. Thus, a gap remains between Theorem 3.4 and Corollary 4.2 below.

We consider binary trees, where each vertex has at most two children.

Theorem 4.1. *Any binary tree language accepted by a bottom-up deterministic automaton with 2 states, is definable in GL.*

Proof. Suppose that $A = (\Sigma, \delta_2, \delta_1, q_1, Q, Q_f)$ is a deterministic bottom-up binary tree automaton with 2 states, i.e. $Q = \{q_1, q_2\}$. We show how to construct a formula that defines the class of binary trees in which A assigns q_1 to the root. We do this just for binary trees where the root is a fork. The result then easily extends to general binary trees, since such a tree consists of the composition of a tree with a fork root and a simple word, and the behaviour of the automaton on words is known to be expressible in GL.

Let $\mathcal{T}$ be the class of binary trees T such that A assigns the state q_1 to all the leaves and the root of T, and assigns the state q_2 to all the forks of T other than the root. Suppose there is a GL formula ψ defining the class of forests where each tree in the forest is in $\mathcal{T}$. Before defining this formula ψ explicitly, we show how it can be used to define the class of trees accepted by A.

In particular, we show that for any tree T, $T \models (\psi \mid \psi)$ if, and only if, the state assigned to the root of T by A is q_1. For the *only if* direction assume that $T \models (\psi \mid \psi)$. Then it can be split into two forests F_1 and F_2, where each one satisfies ψ. Then, the root and the leaves of each tree in F_1 and F_2, is assigned the state q_1, and since all roots have two children, if a tree T_1 in F_1 is connected to a tree T_2 in F_2 within the tree T, then the root of one is the same vertex as the leaf of the other. Therefore, A assigns q_1 to the root of T.

For the *if* direction, suppose that the root of a tree T is assigned the state q_1 by A. For each fork x in T to which A assigns the state q_1, define the subtree rooted at x and whose leaves are the closest descendants to x that are either leaves or forks to which A assigns the state q_1. By definition, all forks other than the root in such a tree, are assigned the state q_2 by A. We define F_1 and F_2 to be two forests, each containing the subtrees defined above, such that no two such subtrees that are connected in T, are both in the same forest. According to the above, $F_i \models \psi$, and therefore $T \models (\psi \mid \psi)$.

We now give an explicit definition of the formula ψ that is used in the argument above. Recall that $\mathcal{T}$ is the class of binary trees T such that A assigns the state q_1 to all the leaves and the root of T, and assigns the state q_2 to all the forks of T other than the root. The formulas $\mathsf{root}(x)$ and $\mathsf{leaf}(x)$ are used to identify roots and leaf vertices respectively, in first order logic. Finally, the formula $\mathsf{fork}(x)$ expresses in FO that the vertex x is a fork, and $\mathsf{one\text{-}child}(x)$ expresses in FO that the vertex x has a single child.

Any path between two vertices x and y, where x and all vertices in that path apart from y, are non-forks, is called a *unary branch*. On a unary branch, a tree automaton works as a word automaton, and uses only the transition function δ_1. On words we know that GL and MSO are equi-expressive, so let the formula φ_{q_i,q_j} for $q_i, q_j \in \{q_1, q_2\}$ be the GL formula that defines the class of words on which the automaton A, starting with state q_i at the first vertex, assigns the state q_j at the last one.

The vertices x and y in some tree T, with x an ancestor of y, are the endpoints of a unary branch if and only if $(T, x, y) \models \mathsf{unaryBranch}(x, y)$, where:

$$\begin{aligned}
\mathsf{Path}(x,y) &= \mathsf{root}(x) \land \mathsf{one\text{-}child}(x) \land \neg\mathsf{root}(y) \land \mathsf{leaf}(y) \land \\
&\quad \land \forall z\ (z \neq x \land z \neq y \to \neg\mathsf{root}(z) \land \mathsf{one\text{-}child}(z)), \\
\mathsf{unaryBranch}(x,y) &= \mathsf{one\text{-}child}(x) \land (\mathsf{leaf}(y) \lor \mathsf{fork}(y)) \land (\mathsf{Path}(x,y) \mid \top) \land \\
&\quad \land \forall z\ (z \neq y \land (\mathsf{Path}(x,z) | \mathsf{Path}(z,y) | \top) \to \mathsf{one\text{-}child}(z)).
\end{aligned}$$

We present the following formulas that assist with defining the GL formula ψ.

$$\begin{aligned}
\mathsf{unary}\text{-}q_2q_j(x) &= \exists y\ (\mathsf{fork}(y) \land \mathsf{unaryBranch}(x,y) \land (\mathsf{Path}(x,y) \land \varphi_{q_2,q_j} \mid \top)), \\
\mathsf{unary}\text{-}q_1q_j(x) &= \exists y\ (\mathsf{leaf}(y) \land \mathsf{unaryBranch}(x,y) \land (\mathsf{Path}(x,y) \land \varphi_{q_1,q_j} \mid \top)), \\
\mathsf{unary}\text{-}q_i(x) &= \mathsf{one\text{-}child}(x) \land (\mathsf{unary}\text{-}q_1q_i(x) \lor \mathsf{unary}\text{-}q_2q_i(x)), \\
\mathsf{state}\text{-}q_2(x) &= (\mathsf{fork}(x) \land \neg\mathsf{root}(x)) \lor \mathsf{unary}\text{-}q_2(x), \\
\mathsf{state}\text{-}q_1(x) &= (\mathsf{leaf}(x) \lor \mathsf{root}(x)) \lor \mathsf{unary}\text{-}q_1(x).
\end{aligned}$$

The formula $\mathsf{unary}\text{-}q_2q_j(x)$ expresses that x is the top vertex of a unary branch to some y, a descendant of x, that is a fork, and furthermore that on this unary branch, if the automaton A starts at state q_2 at y, it will reach the state q_j at x. The case is similar for the formula $\mathsf{unary}\text{-}q_1q_j(x)$, but in this case y is a leaf and the automaton reaches q_j at x if it starts with state q_1 at y. The formula $\mathsf{unary}\text{-}q_j(x)$ is simply the disjunction of the two formulas above. Finally, the formula $\mathsf{state}\text{-}q_1(x)$ holds at some vertex if it is a leaf, a root or if it is the top vertex of a unary branch, where A reaches q_1 according to the assumptions stated above. Similarly for $\mathsf{state}\text{-}q_2(x)$ applying to forks that are not roots.

We show that for any vertex x in any tree T in the class $\mathcal{T}$, $(T, x) \models \mathsf{state}\text{-}q_i(x)$ if, and only if, the state q_i is assigned to the vertex x by A. Now define

$$\begin{aligned}\varphi_{q_1}(x,y,z) &= \bigvee\nolimits_{\delta(q_i,\sigma,q_j,\sigma')=q_1}(\mathsf{state}\text{-}q_i(y) \wedge E_\sigma(x,y) \wedge \mathsf{state}\text{-}q_j(z) \wedge E_{\sigma'}(x,z)),\\ \mathsf{fork\text{-}roots} &= \forall x\ (\mathsf{root}(x) \rightarrow \mathsf{fork}(x)),\\ \psi &= \mathbf{0} \vee \big(\mathsf{fork\text{-}roots} \wedge \forall x\ \big((\exists y, z\ (y \neq z) \wedge \varphi_{q_1}(x,y,z)) \leftrightarrow \mathsf{root}(x)\big)\big).\end{aligned}$$

Notice that for any forest F, $F \models \psi$ if and only if for every tree T in F, $T \models \psi$. This is because ψ expresses that a combination of states and symbols that lead to the state q_1, occurs at a fork if and only if this fork is the root of the tree it belongs to. Furthermore, whether a vertex satisfies any of the formulas given above, depends only on the tree the vertex belongs to.

Thus, we are left with the following claim to prove, the proof of which is omitted due to lack of space.

Claim. For any tree T, $T \models \psi$ if, and only if, $T \in \mathcal{T}$.

Corollary 4.2. *The class of binary trees with an even number of leaves is GL definable.*

5 Separation Logic

Separation Logic (SL) is a logic for analyzing programs that involve pointer variables for memory management, introduced by Reynolds and widely studied since then (see [10]). We give a brief account here, and refer to [1] for details.

The structures on which Separation Logic works, consist of a partial function representing the memory heap of a program. Let Loc be a countable set of *locations*, namely memory addresses and let Var be a set of variables.

Definition 5.1 ([1]). *A memory state is a pair (s, h) such that $s : Var \rightarrow Loc$ and h is a partial function of type $h : Loc \rightharpoonup Loc$. The function s is the store and the function h is the heap of the memory state.*

When the domains of two partial functions h_1 and h_2 are disjoint, this fact is denoted by $h_1 \perp h_2$, and their disjoint union is denoted by $h_1 * h_2$. The syntax of a Separation Logic formula is inductively defined as:

$$\psi := x = y \mid x \hookrightarrow y \mid \psi_1 \wedge \psi_2 \mid \neg\psi_1 \mid \exists x.\psi_1(x) \mid \psi_1 * \psi_2 \mid \psi_1 \mathrel{-\!\!*} \psi_2,$$

where $x, y \in \mathrm{Var}$ and ψ_1, ψ_2 are SL formulas. The semantics of the non-obvious connectives is given below.

$$\begin{aligned}
(s,h) \models x \hookrightarrow y &\Leftrightarrow h(s(x)) = s(y),\\
(s,h) \models \psi_1 * \psi_2 &\Leftrightarrow \text{there are } h_1, h_2 \text{ such that } h = h_1 * h_2\\
&\qquad \text{and } (s,h_i) \models \psi_i, i \in \{1,2\},\\
(s,h) \models \psi_1 \mathrel{-\!\!*} \psi_2 &\Leftrightarrow \text{for any } h' \text{ disjoint from } h \text{ such that}\\
&\qquad (s,h') \models \psi_1, (s, h * h') \models \psi_2.
\end{aligned}$$

Note that if a formula φ does not have any free variables then $(s,h) \models \varphi$ if, and only if, $(s',h) \models \varphi$ for all s'. In this case we simply write $h \models \varphi$.

In [1], the syntactic fragment SL($*$) is considered, in which the conjunction operation, $*$, is present but its adjoint the magic wand, $\mathrel{-\!\!*}$, is absent. They show that this logic is decidable by a translation to MSO. They conjecture that the inclusion of SL($*$) in MSO in terms of expressive power is strict.

We can associate with a heap h the graph G_h consisting of the set of edges (x,y) such that $h(y) = x$. It is straightforward to see that for every sentence φ of SL($*$) there is a sentence φ^* of GL such that $G_h \models \varphi^*$ if, and only if, $h \models \varphi$. Note, in particular, that for every forest G there is an h such that $G \cong G_h$.

Say that a location l is a *leaf* of h if $h(l)$ is defined and there is no l' such that $h(l') = l$. Define a *component* C of h to be a connected component of the graph G_h. Then the following is a consequence of Theorem 3.4.

Theorem 5.2. *There is no sentence θ of SL such that $h \models \theta$ if, and only if, some component of h has 0 (mod 3) leaves.*

As a consequence, we resolve the conjecture of Brochenin et al. [1].

Corollary 5.3. *SL($*$) is strictly less expressive than MSO on memory states.*

6 GL Is Not Included in MS_1

As we noted in Section 2, the expressive power of GL is included in MS_2, that is monadic second-order logic of graphs with quantification over sets of edges. Theorem 3.4 shows that this inclusion is proper. Furthermore, since the separation is shown on a class of graphs where the expressive power of MS_1 and MS_2 coincide, this shows that $\mathrm{MS}_1 \not\subseteq \mathrm{GL}$. We now note that, in general, $\mathrm{GL} \not\subseteq \mathrm{MS}_1$. In particular, we show this over the class of labelled graphs with two edge labels.

Recall that in an ordinary undirected graph $G = (V,E)$ a *Hamiltonian cycle* is a cycle that visits every vertex in V exactly once. It is not difficult to write a sentence μ of MS_2 that defines those graphs that contain a Hamiltonian cycle. However it is known that this property is not definable in MS_1, even on ordered graphs (see [7, Cor. 6.3.5]). It is not known whether or not Hamiltonicity of unordered graphs is definable in GL (indeed, this was an open quesion posed in [6]), but we are able to show that in the presence of a second edge label, which acts as a *successor relation*, it is definable. To explain the construction, note that in GL, once we select a set of edges using a composition operator, we are able to say that they form a cycle, but we cannot say in the subformula that the cycle visits all vertices, since some may have been lost in the decomposition. The presence of the successor relation allows us to assert that all vertices are still present.

Theorem 6.1. *GL is not included in* MS_1.

Proof. We consider a vocabulary with two edge labels S and E and two constants s and t. We restrict ourselves to graphs in which the S edges form a simple path from s to t. This condition is easily expressed by a GL sentence succ. Now, let cycle be the GL sentence that defines the graphs in which the E-edges form a simple cycle. Then the following sentence

$$(\mathsf{succ} \land \mathsf{cycle} \land \forall x[\exists y(S(y,x) \lor S(x,y)) \to \exists y(E(y,x) \lor E(x,y))] \mid \forall x, y \neg S(x,y))$$

defines the class of such graphs that contain a Hamiltonian cycle.

References

1. Brochenin, R., Demri, S., Lozes, É.: On the almighty wand. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 323–338. Springer, Heidelberg (2008)
2. Cardelli, L., Gardner, P., Ghelli, G.: A spatial logic for querying graphs. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 597–610. Springer, Heidelberg (2002)
3. Compton, K.J.: A logical approach to asymptotic combinatorics II: Monadic second-order properties. J. Comb. Theory, Ser. A 50(1), 110–131 (1989)
4. Courcelle, B.: The expression of graph properties and graph transformations in monadic second-order logic. In: Rozenberg, G. (ed.) Handbook of Graph Grammars, pp. 313–400. World Scientific, Singapore (1997)
5. Dawar, A., Gardner, P., Ghelli, G.: Adjunct elimination through games in static ambient logic*(Extended abstract)*. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004. LNCS, vol. 3328, pp. 211–223. Springer, Heidelberg (2004)
6. Dawar, A., Gardner, P., Ghelli, G.: Expressiveness and complexity of graph logic. Inf. Comput. 205(3), 263–310 (2007)
7. Ebbinghaus, H.-D., Flum, J.: Finite Model Theory, 2nd edn. Springer, Heidelberg (1999)
8. Grädel, E., Hirsch, C., Otto, M.: Back and forth between guarded and modal logics. ACM Trans. Comput. Log. 3(3), 418–463 (2002)
9. Marcinkowski, J.: On the expressive power of graph logic. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 486–500. Springer, Heidelberg (2006)
10. Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: LICS, pp. 55–74. IEEE Computer Society, Los Alamitos (2002)

On the Completeness of Dynamic Logic

Daniel Leivant

Computer Science Department,
Indiana University,
Bloomington, IN 47405, USA
leivant@cs.indiana.edu

Abstract. The impossibility of semantically complete deductive calculi for logics for imperative programs has led to the study of two alternative approaches to completeness: "local" semantic completeness on the one hand (Cook's relative completeness, Harel's Arithmetical completeness), and completeness with respect to other forms of reasoning about programs, on the other. However, local semantic completeness is problematic on several counts, whereas proof theoretic completeness results often involve *ad hoc* ingredients, such as formal theories for the natural numbers.

The notion of inductive completeness, introduced in [18], provides a generic proof theoretic framework which dispenses with extraneous ingredients, and yields local semantic completeness as a corollary.

Here we prove that (first-order) Dynamic Logic for regular programs (DL) is inductively complete: a DL-formula φ is provable in (the first-order variant of) Pratt-Segerberg deductive calculus **DL** iff φ, is provable in first-order logic from the inductive theory for program semantics. The method can be adapted to yield the *schematic* relative completeness of **DL**: if $\mathcal{S}$ is an expressive structure, then every formula true in $\mathcal{S}$ is provable from the axiom-*schemas* that are valid in $\mathcal{S}$. Harel's Completeness Theorem falls out then as a special case.

Keywords: Dynamic logic, inductive completeness, relative completeness, arithmetical completeness.

1 Introduction

1.1 Background

Logics of programs inherently exceed first-order logic, because program semantics is defined in terms of iterative processes, which can be formalized using second-order formulas [15], existential fixpoints [3], or explicit reference to the natural numbers (e.g. [6]), but not by a first-order theory. Consequently, the delineation of logics of programs cannot parallel the characterization of first-order logic by soundness and completeness for general ("uninterpreted") validity.

The early attempts to refer instead to *local completeness,* i.e. completeness with respect to one structure at a time, led to Cook's Relative Completeness Theorem. However, notwithstanding the persistent centrality of Cook's relative completeness [5] in research on logics of programs, that notion has foundational and practical drawbacks

[17]. For one, relative completeness fails to demarcate the boundaries of logics of programs: there are, for example, proper extensions of Hoare's Logic that are sound and relatively complete!

An alternative approach is to match logics of programs with formalisms for *explicit* reasoning about programs, such as second-order logic. A common objection to second-order logic, that it is non-axiomatizable for its intended semantics, is off the mark here: On the one hand there are natural proof calculi for second-order logic that are sound and complete for a natural non-standard semantics (Henkin's structures); on the other hand, the same objection applies to arithmetic! Indeed, it turns out that Hoare's Logic matches second-order logic with first-order set-existence, and Dynamic Logic matches second-order logic with "computational" (i.e. strict-Π^1_1) set-existence [15,17].

Here we consider an approach dual to explicit second-order rendition of program semantics, namely an *implicit* but *first-order* rendition. We consider the inductive definition of program semantics, and invoke the well-known framework of first-order theories for such definitions [13,14,7,19,8]. We show that Dynamic Logic matches reasoning about programs in the inductive theory for regular programs.

Since this approach is strictly first-order, it is particularly accessible conceptually and expositorily, and directly amenable to automated theorem proving tools. It also meshes generically with the long-standing tradition of defining program semantics inductively. Whenever an inductive definition is given for a programming language, one obtains the corresponding first-order inductive-definition theory, and can tackle the question of completeness of a proposed logic of programs for that theory.

A match between Dynamic Logic and inductive theories was observed already by the Hungarian school of "nonstandard dynamic logic" (see e.g. [6,23]). However, the first-order theories they considered invoke the natural numbers as an auxiliary data-type. Our present approach calls for no such extraneous data-types, and is more generic, in that it applies directly to all programming constructs with inductively defined semantics, even when they lack a simple iterative definition in terms of the natural numbers.

1.2 Regular Programs

Regular programs distill and separate the essential components of imperative programs, and are therefore particularly amenable to logical analysis. Given a vocabulary V (i.e. "similarity type", "signature"), the *atomic* V*-programs* are *assignments* $x := \mathbf{t}$ of V-terms to variables, and *tests* $?\varphi$, where φ is a quantifier-free V-formula.[1] Compound V*-programs* are generated from atomic V-programs by composition, union, and Kleene's $*$ (nondeterministic iteration). Deterministic **while** programs are definable in terms of regular programs: $(\texttt{if}\ \varphi\ \texttt{then}\ \alpha\ \texttt{else}\ \beta)$ is an abbreviation for $(?\varphi);\ \alpha \cup (?\neg\varphi);\ \beta$, and $(\texttt{while}\ \varphi\ \texttt{do}\ \alpha)$ for $(?\varphi;\ \alpha)^*;\ ?(\neg\varphi)$. We refer to [11] for background and detail.

Given a V-structure $\mathcal{S}$, the *states* are the $\mathcal{S}$-environments, i.e. partial functions from the set of variables to elements of $\mathcal{S}$. Each V-program α is interpreted semantically as a binary relation over states, denoted $\xrightarrow{\alpha}$, or $\xrightarrow[\mathcal{S}]{\alpha}$ when $\mathcal{S}$ is not obvious from the context.

[1] Our discussion below remains unchanged if tests are generalized to arbitrary first-order formulas. Tests for arbitrary DL formulas, known as "rich tests", can also be accommodated with minor changes.

These relations are defined by recurrence on α: $\eta\xrightarrow{x:=\mathbf{t}}\eta'$ iff $\eta' = \eta[x \leftarrow [\![\mathbf{t}]\!]_\eta]$; $\eta\xrightarrow{?\varphi}\eta'$ iff $\eta' = \eta$ and $\mathcal{S}, \eta \models \varphi$; $\xrightarrow{\alpha;\beta}$ is the composition of $\xrightarrow{\alpha}$ and $\xrightarrow{\beta}$; $\xrightarrow{\alpha\cup\beta}$ is the union of $\xrightarrow{\alpha}$ and $\xrightarrow{\beta}$; and $\xrightarrow{\alpha^*}$ is the $(\xrightarrow{\alpha})^*$, i.e. the reflexive-transitive closure of $\xrightarrow{\alpha}$.

1.3 Dynamic Logic Formulas

Given a vocabulary V, the DL V-formulas are generated inductively, just like first-order V-formulas, but with the added clause: If φ is a formula, and α a program, then $[\alpha]\varphi$ is a formula. The operator $\langle\alpha\rangle$, dual to $[\alpha]$, can be defined by $\langle\alpha\rangle\,\varphi \;\equiv_{\mathrm{df}}\; \neg[\alpha]\neg\varphi$.

The common convention, of not distinguishing between (assignable) program variables on the one hand and logical variables on the other, lightly complicates the interaction of assignments and quantifiers: We must posit that a quantifier cannot bind a variable that is assigned-to in its scope. For example, $\forall x\ [x := 1](x = 1)$ is not a legal DL formula, whereas $\forall x\ [y := x](y = x)$ is. Also, the definition of "free occurrence of variable x in formula φ" is amended to exclude x occurring in the scope of the modal operators $[\alpha]$ and $\langle\alpha\rangle$, when x is assigned-to in α.[2]

The definition of the satisfaction relation $\mathcal{S}, \eta \models \varphi$ is as usual by recurrence on φ. In particular, $\mathcal{S}, \eta \models [\alpha]\varphi$ iff $\mathcal{S}, \eta' \models \varphi$ whenever $\eta\xrightarrow{\alpha}\eta'$. Consequently, $\mathcal{S}, \eta \models \langle\alpha\rangle\,\varphi$ iff $\mathcal{S}, \eta' \models \varphi$ for some environment η' where $\eta\xrightarrow{\alpha}\eta'$.

1.4 Complexity

The set of valid DL formulas is highly complex. In fact, even DL formulas of a seemingly modest appearance have a decision problem more complex than that of first-order arithmetic:

Proposition 1
1. The validity problem for DL formulas is Π^1_1.

2. Even the validity problem for formulas of the form $\exists \boldsymbol{x}.[\alpha]\,\varphi$, where α is a deterministic program and φ is quantifier-free, is Π^1_1-hard.

Proof. (1) is easy, and proved in [11, Theorem 13.1], where (2) is also stated and proved, but for φ first-order and α non-deterministic. On closer inspection the proof there, which uses a Π^1_1-hard tiling problem, yields the result with φ quantifier-free.

However, already basic properties of Π^1_1 easily imply the refinement stated here (with α deterministic). Recall that every Π^1_1 formula φ is equivalent over $\mathbb{N}$ to a formula of the form

$$\forall f\,\exists x.\ g(x) = 0 \tag{1}$$

where f ranges over unary functions, and g is defined uniformly from f by primitive-recursion. More precisely, if D_g is the conjunction of the recurrence equations defining g from f, and $\boldsymbol{u}$ the variables free in D_g, then (1) is expressed by

$$\forall f, g\,(\,\bar{D}_g\ \rightarrow\ \exists x.\ g(x) = 0\,) \tag{2}$$

where $\bar{D}_g$ is the universal closure of D_g.

[2] This issue is central to [12], but is inconsequential here.

The truth of (2) in $\mathbb{N}$ is equivalent to the validity in all structures (over the vocabulary in hand) of the informal statement

$$\bar{D}_g \quad \wedge \quad (\forall v.\text{“}f(v) \text{ is the denotation of a numeral”}) \qquad \rightarrow \qquad \exists x.\ g(x) \ = \ 0 \quad (3)$$

where the *numerals* are the terms 0, $s(0)$, $s(s(0))$ (with s intended to denote the successor function).[3]

Now, with p intended to denote the predecessor function, let ψ be the conjunction of the three formulas

$$p(0) = 0$$
$$\forall y\ p(s(y)) = y$$
$$\text{and} \qquad \forall y, w\ (\ p(y) = p(w)\ \wedge\ y \neq 0\ \wedge w \neq 0\ \ \rightarrow\ \ y = w\ )$$

Thus, if ψ holds in a structure, and N is the program

$$z := f(v);\ \texttt{while}\ \ z \neq 0\ \ \texttt{do}\ \ z := p(z)\ \ \texttt{end}$$

then N terminates in k steps iff $f(v)$ is the denotation of the numeral $s^k(0)$. Therefore, (3) can be expressed in DL as

$$\bar{D}_g\ \wedge\ \ \psi\ \ \wedge\ \ (\forall v.\, \langle N\rangle \texttt{true})\ \ \rightarrow\ \ \exists x\, g(x) = 0 \qquad (4)$$

Since v and z are the only variables in N, quantifiers over other variables commute with $[N]$, and so (4) can be converted into an equivalent formula of the form stated in the Proposition. ■

1.5 Axiomatics

One way of addressing the challenge presented by of Proposition 1 is to consider an infinitary deductive system. So-called omega-rules go back to the 1930's and their adaptation to logics of programs is due to Mirkowska [20]. In [11, Theorem 14.7] that proof is adapted to Dynamic Logic.

Since no effective axiomatization of DL can be complete for validity, the dual task is all the more interesting: articulate a *natural* axiomatization of DL, and delineate it in terms of a familiar deductive system. This is analogous to the situation with Hoare's Logic: the validity problem for Partial-correctness assertions is Π^0_2-complete [11, Theorem 13.5], implying that Hoare's Logic is not complete for validity, and that alternative completeness properties are needed to delineate the logic and explicate its significance.

A natural deductive calculus **DL** for Dynamic Logic is obtained by augmenting first-order logic with the rules of Table 1. This formalization is due primarily to Pratt [22,9]. The assignment rule is Hoare's, and the others are related to Segerberg's Propositional Dynamic Logic for regular programs [24]. This is closely related to the formalism 14.12 of [11], with the Convergence Rule omitted. Note that the quantifier rules of first-order logic can never be used to instantiate a variable x in an assignment $x := \mathtt{t}$, due to our syntactic restriction above on the formation of DL formulas.

[3] The denotations of the numerals form a copy of $\mathbb{N}$ in a structure satisfying Peano's Third and Fourth Axioms, which we could include here trivially. However, (3) remains true in structures that identify the values of some distinct numerals!

Table 1. Pratt-Segerberg's Dynamic Logic

First-order logic		
Modality:	Generalization:	$\dfrac{\vdash \varphi}{\vdash [\alpha]\varphi}$
	Distribution:	$[\alpha](\varphi \to \psi) \to ([\alpha]\varphi \to [\alpha]\psi)$
Atomic Programs:	Assignment:	$[x := \mathbf{t}]\,\varphi \leftrightarrow \{\mathbf{t}/x\}\varphi$ **t** free for x in φ
Basic constructs:	Test:	$[?\chi]\varphi \leftrightarrow (\chi \to \varphi)$
	Composition:	$[\alpha;\beta]\varphi \leftrightarrow [\alpha][\beta]\varphi$
	Union:	$[\alpha \cup \beta]\varphi \leftrightarrow [\alpha]\varphi \wedge [\beta]\varphi$
Iteration:	Invariance (Folding):	$\dfrac{\vdash \varphi \to [\alpha]\,\varphi}{\vdash \varphi \to [\alpha^*]\,\varphi}$
	Unfolding:	$[\alpha^*]\varphi \to \varphi \wedge [\alpha][\alpha^*]\varphi$

From the Invariance Rule we obtain the Schema of Induction:

$$\psi \wedge [\alpha^*](\psi \to [\alpha]\,\psi) \to [\alpha^*]\,\psi)$$

Indeed, taking $\varphi \equiv \psi \wedge [\alpha^*](\psi \to [\alpha]\psi)$, we have $\vdash \varphi \to [\alpha]\varphi$ using the remaining axioms and rules, and so $\varphi \to [\alpha^*]\varphi$ by Iteration. The Induction template above readily follows.

Also, the converse of the Unfolding Schema is easily provable: Taking $\psi \equiv \varphi \wedge [\alpha;\alpha^*]\varphi$, we have $\vdash \psi \to [\alpha]\psi$, by Unfolding and the modality rules, yielding $\psi \to [\alpha^*]\psi$ by Invariance, while $[\alpha^*]\psi \to [\alpha^*]\varphi$ by Iteration and modality rules.[4]

A *V-theory* is a set of closed first-order *V*-formulas. Given a *V*-theory **T** we write $\mathbf{DL}(\mathbf{T})$ for the deductive formalism **DL** augmented with the formulas in **T** as axioms. We refer to **T** as the *background theory.*

A straightforward induction on proofs establishes the soundness of **DL**:

Theorem 1. (Soundness of DL) *Let* **T** *be a V-theory,* φ *a DL V-formula. Suppose* $\mathbf{DL}(\mathbf{T}) \vdash \varphi$. *Then* φ *is true in every model of* **T**.

2 Inductive Completeness

2.1 Inductive Definition of Program Semantics

Generic methods for associating to a given collection of inductive (i.e. generative) definitions first-order inductive theories are well-known. The inductive definition of the semantics of regular programs has a particularly simple form, using atomic production rules, i.e. natural-deduction inferences with atomic premises and conclusion, as follows.

[4] Note that this argument uses Invariance for modal formulas, and is not available when the Invariance Rule is restricted to first-order formulas.

For a list $\boldsymbol{x} = (x_1, \ldots, x_n)$ of variables, let $\mathbf{P}[\boldsymbol{x}]$ consist of the regular V-programs with all assigned variables among $x_1 \ldots x_n$. Note that if α is such a program, then so are all its subprograms. Given a V-structure $\mathcal{S}$, each program $\alpha \in \mathbf{P}[\boldsymbol{x}]$ defines a $2n$-ary relation $[\![\alpha]\!]_{\mathcal{S}}$ on the universe $|\mathcal{S}|$ of $\mathcal{S}$, that holds between $\boldsymbol{a}, \boldsymbol{b} \in |\mathcal{S}|^n$ iff α has a complete execution that starts with $\boldsymbol{x}$ bound to $\boldsymbol{a}$, and terminates with $\boldsymbol{x}$ bound to $\boldsymbol{b}$.

For $n \geqslant 1$, let $\hat{V}^n$ be the expansion of the underlying vocabulary V with $2n$-ary relational identifiers $\mathtt{M}^n_\alpha$ for each $\alpha \in \mathbf{P}[\boldsymbol{x}]$. The intent is that $\mathtt{M}^n_\alpha$ denotes, in each V-structure $\mathcal{S}$, the relation $[\![\alpha]\!]_{\mathcal{S}}$ above. We omit the superscript n throughout when in no danger of confusion.

An inductive definition of $[\![\alpha]\!]$, uniform for all V-structures, is given by generative clauses that can be rendered by the following atomic rule-templates, which can be construed as natural-deduction rules.

ASSIGNMENT
$$\frac{}{\mathtt{M}_{x_i:=\mathbf{t}[\boldsymbol{x}]}(\boldsymbol{u}, \boldsymbol{u}_{i\leftarrow\mathbf{t}})}$$
where $\quad \boldsymbol{u}_{i\leftarrow\mathbf{t}}$ is $u_1 \ldots u_{i-1}, \mathbf{t}[\boldsymbol{u}], u_{i+1} \ldots u_n$

TEST
$$\frac{\varphi[\boldsymbol{u}]}{\mathtt{M}_{?\varphi}(\boldsymbol{u}, \boldsymbol{u})}$$

COMPOSITION
$$\frac{\mathtt{M}_\beta(\boldsymbol{u}, \boldsymbol{w}) \quad \mathtt{M}_\gamma(\boldsymbol{w}, \boldsymbol{v})}{\mathtt{M}_{\beta;\gamma}(\boldsymbol{u}, \boldsymbol{v})}$$

BRANCHING
$$\frac{\mathtt{M}_\beta(\boldsymbol{u}, \boldsymbol{v})}{\mathtt{M}_{\beta\cup\gamma}(\boldsymbol{u}, \boldsymbol{v})} \qquad \frac{\mathtt{M}_\gamma(\boldsymbol{u}, \boldsymbol{v})}{\mathtt{M}_{\beta\cup\gamma}(\boldsymbol{u}, \boldsymbol{v})}$$

ITERATION
$$\frac{}{\mathtt{M}_{\beta^*}(\boldsymbol{u}, \boldsymbol{u})} \qquad \frac{\mathtt{M}_\beta(\boldsymbol{u}, \boldsymbol{w}) \quad \mathtt{M}_{\beta^*}(\boldsymbol{w}, \boldsymbol{v})}{\mathtt{M}_{\beta^*}(\boldsymbol{u}, \boldsymbol{v})}$$

2.2 Expressing Program Properties

It is easy to see that, modulo the intended reading of the identifiers $\mathtt{M}_\alpha$, the expressive power of the $\hat{V}$-formulas is identical to the expressive power of DL formulas over the base vocabulary V. To avoid clutter we posit that all programs are in $\mathbf{P}[\boldsymbol{x}]$, i.e. all assigned-variables are among $x_1 \ldots x_n$.

Each DL V-formula φ can be expressed as a $\hat{V}$-formula $\varphi^\sharp$, defined by structural recurrence on φ. For φ modal-free we take of course $\varphi^\sharp$ to be φ itself. If φ is $[\alpha]\varphi_0$, then $\varphi^\sharp$ is $\forall v_1 \ldots v_n \; \mathtt{M}_\alpha(\boldsymbol{x}, \boldsymbol{v}) \rightarrow \{\boldsymbol{v}/\boldsymbol{x}\}\varphi_0^\sharp$. Finally, we let $\sharp$ commute with the first-order logical operations; for instance, $(\varphi_0 \wedge \varphi_1)^\sharp$ is $\varphi_0^\sharp \wedge \varphi_1^\sharp$, and $(\forall u\varphi)^\sharp$ is $\forall u(\varphi^\sharp)$ (recall that assigned-to variables are not quantified.)

Conversely, each $\hat{V}$-formula ψ is expressible as a DL V-formula $\psi^\natural$, defined by structural recurrence on ψ. If ψ is a V-formula, we defined $\psi^\natural$ to be ψ. If ψ is $\mathtt{M}_\alpha(\mathbf{t}, \mathbf{s})$ then $\psi^\natural$ is $\boldsymbol{x} = \mathbf{t} \rightarrow \langle\alpha\rangle(\boldsymbol{x} = \mathbf{s})$. Again, we let $\natural$ commute with connectives and quantifiers.

Observe that these interpretations are *sound* in the following sense.

Theorem 2. *For every V-structure $\mathcal{S}$, if $\hat{\mathcal{S}}$ is the $\hat{V}$-expansion of $\mathcal{S}$ in which each $\mathtt{M}_\alpha$ is interpreted as the denotational semantics of α, then for every DL V-formula φ, $\hat{\mathcal{S}} \models \varphi \leftrightarrow \varphi^\sharp$, and for every $\hat{V}$-formula ψ, $\hat{\mathcal{S}} \models \psi \leftrightarrow \psi^\natural$.*

Since $\hat{\mathcal{S}}$ is trivially conservative over $\mathcal{S}$ for DL V-formulas, we conclude

Corollary 1. *For every DL formula φ, $\mathcal{S} \models \varphi$ iff $\hat{\mathcal{S}} \models \varphi^\sharp$.*

We prove below (Proposition 4) that the equivalence $\varphi \leftrightarrow \varphi^{\sharp\natural}$ is in fact provable in **DL**.

2.3 The Inductive Theory of Regular Programs

The generative rules above, for inductively defining program semantics, bound the interpretation of the relation-identifiers $\mathtt{M}_\alpha$ from below. Bounding inductively generated sets from above, namely as the *minimal* relations closed under the generative clauses, is a second-order condition which has no first-order axiomatization (except for degenerated cases). However, we can approximate that delineation as the minimal one among a given collection of *definable* relations. Namely, the *deconstruction* template for $\mathtt{M}_\alpha$ states that $\mathtt{M}_\alpha$ is contained in every definable relation closed under the generative rules for $\mathtt{M}_\alpha$. This is analogous to the familiar deconstruction for the set $\mathbb{N}$ of natural numbers: With N as unary relational-identifier, the generative clauses are

$$\frac{}{N(\mathtt{0})} \quad \text{and} \quad \frac{N(x)}{N(\mathtt{s}(x))}$$

yielding the Deconstruction template

$$\frac{N(x) \quad \varphi[\mathtt{0}] \quad \begin{array}{c}\varphi[z]\\ \cdots\\ \varphi[\mathtt{s}z]\end{array}}{\varphi[x]}$$

(assumption $\varphi[z]$ is discharged,
z not free in other open assumptions)

that is, the natural-deduction rule of induction on $\mathbb{N}$ [19].

Analogously, the DECONSTRUCTION Rule for the iteration construct $*$ should be

$$\frac{\mathtt{M}_{\beta*}(\boldsymbol{s},\boldsymbol{t}) \quad \varphi[\boldsymbol{u},\boldsymbol{u}] \quad \begin{array}{c}\mathtt{M}_\beta(\boldsymbol{u},\boldsymbol{w}) \quad \varphi[\boldsymbol{w},\boldsymbol{v}]\\ \cdots\\ \varphi[\boldsymbol{u},\boldsymbol{v}]\end{array}}{\varphi[\boldsymbol{s},\boldsymbol{t}]}$$

(assumptions $\mathtt{M}_\beta(\boldsymbol{u},\boldsymbol{w})$ and $\varphi[\boldsymbol{w},\boldsymbol{v}]$ are discharged,
$\boldsymbol{u},\boldsymbol{v},\boldsymbol{w}$ not free in other open assumptions)

The formula φ above is the *eigen-formula* of the inference.

A related, more practical, rule is

$$\text{INVARIANCE} \qquad \frac{\mathtt{M}_{\beta^*}(\boldsymbol{s},\boldsymbol{t}) \quad \begin{array}{c}\mathtt{M}_\beta(\boldsymbol{u},\boldsymbol{w})\\ \cdots \\ \psi[\boldsymbol{u}] \rightarrow \psi[\boldsymbol{w}]\end{array}}{\psi[\boldsymbol{s}] \rightarrow \psi[\boldsymbol{t}]}$$

(assumption $\mathtt{M}_\beta(\boldsymbol{u},\boldsymbol{w})$ is discharged
$\boldsymbol{u},\boldsymbol{w}$ not free in other open assumptions)

Put differently,

$$\frac{\forall \boldsymbol{u},\boldsymbol{w} \quad \psi[\boldsymbol{u}] \wedge \mathtt{M}_\beta(\boldsymbol{u},\boldsymbol{w}) \rightarrow \psi[\boldsymbol{w}]}{\forall \boldsymbol{y},\boldsymbol{z} \quad \psi[\boldsymbol{y}] \wedge \mathtt{M}_{\beta^*}(\boldsymbol{y},\boldsymbol{z}) \rightarrow \psi[\boldsymbol{z}]}$$

However, we have

Proposition 2. *The rules* DECONSTRUCTION *and* INVARIANCE *are equivalent.*

Proof. Posit DECONSTRUCTION, and assume the premises of INVARIANCE. Then the three premises of DECONSTRUCTION hold with $\varphi[\boldsymbol{x},\boldsymbol{y}]$ taken as $\psi[\boldsymbol{x}]\rightarrow\psi[\boldsymbol{y}]$. We thus obtain $\psi[\boldsymbol{s}]\rightarrow\psi[\boldsymbol{t}]$, as required.

Conversely, posit INVARIANCE, and assume the premises of DECONSTRUCTION. Then the premises of INVARIANCE hold with $\psi[\boldsymbol{x}]$ taken to be $\neg\varphi[\boldsymbol{x},\boldsymbol{t}]$. Thus INVARIANCE yields $\psi[\boldsymbol{s}] \rightarrow \psi[\boldsymbol{t}]$, i.e. $\varphi[\boldsymbol{t},\boldsymbol{t}] \rightarrow \varphi[\boldsymbol{s},\boldsymbol{t}]$. Since we have $\varphi[\boldsymbol{t},\boldsymbol{t}]$ by the second premise of DECONSTRUCTION (recall that $\boldsymbol{u}$ is not free in assumptions), we obtain $\varphi[\boldsymbol{s},\boldsymbol{t}]$, as required. ■

Note that deconstruction rules for the remaining program constructs are degenerate, in the sense that they are equivalent to explicit definitions. For example, the Deconstruction of composition, combined with the Composition Rule, yield an explicit definition of $\mathtt{M}_{\beta;\gamma}$. More generally, $\mathtt{M}_\alpha$ can be explicitly defined in terms of components of α, for all non-loop programs α:

- $\mathtt{M}_{x_i:=\mathbf{t}}(\boldsymbol{u},\boldsymbol{v}) \;\leftrightarrow\; (v_i = \mathbf{t}[\boldsymbol{u}] \wedge \bigwedge_{j\neq i} v_j = u_j)$.
- $\mathtt{M}_{?\varphi}(\boldsymbol{u},\boldsymbol{v}) \;\leftrightarrow\; (\varphi \;\wedge\; \boldsymbol{v} = \boldsymbol{u})$
- $\mathtt{M}_{\beta;\gamma}(\boldsymbol{u},\boldsymbol{v}) \leftrightarrow \exists \boldsymbol{w}\; \mathtt{M}_\beta(\boldsymbol{u},\boldsymbol{w}) \;\wedge\; \mathtt{M}_\gamma(\boldsymbol{w},\boldsymbol{v})$
- $\mathtt{M}_{\beta\cup\gamma}(\boldsymbol{u},\boldsymbol{v}) \;\leftrightarrow\; \mathtt{M}_\beta(\boldsymbol{u},\boldsymbol{v}) \;\vee\; \mathtt{M}_\gamma(\boldsymbol{u},\boldsymbol{v})$

We write $\mathbf{Ind}^n$ for the inductive theory given by the universal closure of the formulas above (recall that our $\mathtt{M}_\alpha$'s are for programs α with variables among $x_1,\ldots,x_n$). We omit the superscript n when in no danger of confusion.

2.4 Inductive Soundness of DL

Clearly, the deductive calculus **DL** is semantically sound (Theorem 1). Only slightly less trivial is the observation that it is sound for **Ind**:

Theorem 3. *If* $\mathbf{DL}(\mathbf{T}) \vdash \varphi$ *then* $\mathbf{T} + \mathbf{Ind} \vdash \varphi^\sharp$.

Proof. Induction on proofs in $\mathbf{DL}(\mathbf{T})$.

Consider, for example, an instance of the INVARIANCE Rule, deriving $\psi \rightarrow [\alpha^*]\ \psi$ from $\psi \rightarrow [\alpha]\ \psi$. By IH we have

$$\mathbf{T} + \mathbf{Ind} \vdash \psi^\sharp[\boldsymbol{u}] \wedge \mathtt{M}_\alpha(\boldsymbol{u},\boldsymbol{v}) \ \rightarrow\ \psi^\sharp[\boldsymbol{v}]$$

which by the INVARIANCE Rule of **Ind** yields

$$\mathbf{T} + \mathbf{Ind} \vdash \psi^\sharp[\boldsymbol{u}] \wedge \mathtt{M}_{\alpha^*}(\boldsymbol{u},\boldsymbol{v}) \ \rightarrow\ \psi^\sharp[\boldsymbol{v}]$$

For another example, consider the Generalization rule, deriving $\vdash\ [\alpha]\varphi$ from $\vdash\ \varphi$. By IH the premise implies the provability in $\mathbf{T} + \mathbf{Ind}$ of $\varphi^\sharp$, from which the provability of $([\alpha]\varphi)^\sharp$, i.e. $\mathtt{M}_\alpha(\boldsymbol{x},\boldsymbol{u}) \rightarrow \{\boldsymbol{u}/\boldsymbol{x}\}\varphi^\sharp$, follows trivially. ■

2.5 Inductive Completeness of DL

The main interest in the inductive theory **Ind** is its relation to **DL**, namely the inductive completeness of **DL**:

Theorem 4. *For all V-theories* $\mathbf{T}$*, if φ is a DL V-formula, and* $\mathbf{T} + \mathbf{Ind} \vdash \varphi^\sharp$*, then* $\mathbf{DL}(\mathbf{T}) \vdash \varphi$.

The proof of Theorem 4 will use the interpretation $\psi \mapsto \psi^\natural$ defined above.

Proposition 3. *Let* $\mathbf{T}$ *be a V-theory, and ψ a $\hat{V}$-formula. If* $\mathbf{T} + \mathbf{Ind} \vdash \psi$*, then* $\mathbf{DL}(\mathbf{T}) \vdash \psi^\natural$.

Proof. The proof is by induction on natural-deduction derivations of $\mathbf{T} + \mathbf{Ind}$. More precisely, if φ is provable in $\mathbf{T} + \mathbf{Ind}$ from assumptions Γ, then $\varphi^\natural$ is provable from $\Gamma^\natural$ in $\mathbf{DL}(\mathbf{T})$.

It is easy to verify that the rules of **Ind** translate correctly. Consider, for example, the Iteration rules. If ψ is $\mathtt{M}_{\beta^*}(\boldsymbol{u},\boldsymbol{u})$ then $\psi^\natural$ is $\boldsymbol{x} = \boldsymbol{u} \rightarrow \langle\beta^*\rangle\, \boldsymbol{x} = \boldsymbol{u}$, which is readily provable in **DL** using Unfolding. To tackle the second Iteration rule of **Ind**, suppose that ψ is $\mathtt{M}_{\beta^*}(\boldsymbol{u},\boldsymbol{u})$, and is derived from $\mathtt{M}_\beta(\boldsymbol{u},\boldsymbol{w})$ and $\mathtt{M}_{\beta^*}(\boldsymbol{w},\boldsymbol{u})$. Assume now that the premises translate correctly, i.e. the formulas

$$\boldsymbol{x} = \boldsymbol{u} \rightarrow \langle\beta\rangle\, \boldsymbol{x} = \boldsymbol{w} \tag{5}$$

and

$$\boldsymbol{x} = \boldsymbol{w} \rightarrow \langle\beta^*\rangle\, \boldsymbol{x} = \boldsymbol{v} \tag{6}$$

are given. From (6) we get

$$\langle\beta\rangle\ \boldsymbol{x} = \boldsymbol{w} \rightarrow \langle\beta\rangle\, \langle\beta^*\rangle\, \boldsymbol{x} = \boldsymbol{v}$$

which readily yields in **DL**

$$\langle\beta\rangle\ \boldsymbol{x} = \boldsymbol{w} \rightarrow \langle\beta^*\rangle\, \boldsymbol{x} = \boldsymbol{v}$$

Combined with (5) we get $\psi^\natural$.

It is also easy to verify that the inference rules of first order logic preserve the translation. ■

Lemma 1. *Let $\boldsymbol{x}$ be the assigned-to variables in a DL-formula $\varphi \equiv \varphi(\boldsymbol{x})$. The following is provable in* **DL***.*

$$[\alpha]\varphi \quad \leftrightarrow \quad \forall \boldsymbol{v} \; (\; \langle\alpha\rangle \, (\boldsymbol{x} = \boldsymbol{v}) \rightarrow \varphi(\boldsymbol{v}))$$

The proof is straightforward by induction on α. Only the iteration case $\alpha = \beta^*$ is non-trivial.

Proposition 4. *For all DL-formulas φ the following is provable in* **DL***:*

$$(\varphi^\sharp)^\natural \quad \leftrightarrow \quad \varphi$$

Proof. We use structural induction on φ. The only non-trivial case is where φ is of the form $[\alpha]\psi$. Then

$$\begin{aligned}
(\varphi^\sharp)^\natural &\equiv (\forall \boldsymbol{v} \; (\mathtt{M}_\alpha(\boldsymbol{x}, \boldsymbol{v}) \rightarrow \varphi^\sharp(v)))^\natural \\
&\leftrightarrow \forall \boldsymbol{v} \; (\langle\alpha\rangle(\boldsymbol{x} = \boldsymbol{v}) \rightarrow \varphi^{\sharp\natural}(v)) \\
&\leftrightarrow [\alpha]\varphi^{\sharp\natural} && \text{(Lemma 1)} \\
&\leftrightarrow [\alpha]\varphi(v) && \text{(IH and the Distribution Rule)}
\end{aligned}$$ ■

Proof of Theorem 4. Suppose $\mathbf{T} + \mathbf{Ind} \vdash \varphi^\sharp$. Then, by Proposition 3, $\mathbf{DL}(\mathbf{T}) \vdash (\varphi^\sharp)^\natural$, from which $\mathbf{DL}(\mathbf{T}) \vdash \varphi$ follows by Proposition 4. ■

Note that this proof is different from the proof of the corresponding result for Hoare's Logic [18, Theorem 2.4], stating the soundness and completeness of Hoare's Logic for a theory $\mathbf{Ind}_0$, obtained by restricting the Deconstruction Rule to eigen-formulas φ in the vocabulary V. However, the latter theorem can be obtained from the proof presented here, using the conservation theorem [16, Theorem 7], which states that Dynamic Logic is conservative over Hoare's Logic when Invariance is restricted to first-order formulas.

3 Relative Completeness and Arithmetical Completeness

3.1 Failure of Relative Completeness for Termination Assertions

Relative completeness in the sense of Cook fails for **DL**, since there is no way to infer that a termination assertion of the form $\langle\alpha^*\rangle\mathtt{true}$ is true in a given structure from any collection of first-order formulas true in the structure. This reflects a gap between the semantics of program convergence, which is anchored in the intended data (e.g. $\mathbb{N}$), and the semantics of data in the background theory, which might include non-standard elements. To illustrate, consider the termination assertion

$$P \;\rightarrow\; \langle \, (x := p(x))^* \rangle \; (x = 0) \tag{7}$$

where P is some finite axiomatization of arithmetic,[5] that includes the definition of p as cut-off predecessor: $p(0) = 0$, $p(s(x)) = x$. Since there are non-standard models

[5] Take, for example, Peano's Arithmetic with induction up to some fixed level Σ_n in the arithmetical hierarchy.

of P, with elements that are not denotations of numerals, the assertion (7) is not valid. It follows that (7), although trivially true in the intended structure, cannot be proved in any DL formalism which is sound for all structures, such as **DL**.

This remains true even if we augment **DL** with axioms for inductive data, such as Peano's Arithmetic, since the semantics of program convergence will remain different from the semantics of counting in the grafted first-order theory.

Of course, (7) can be proved by induction on the formula

$$\boldsymbol{\varphi}[n] \quad \equiv \qquad (x = \bar{n}) \rightarrow \langle\ (x := p(x))^* \rangle\ (x{=}0)$$

but $\boldsymbol{\varphi}$ is not a first-order formula, and so this instance of induction is not part of the background theory. As long as the background theory has no direct access to modal formulas, it cannot be used to derive termination assertions whose truth depends on the inductive data in hand.

3.2 Schematic Relative Completeness

The remarks above suggest a way to modify Cook's notion of relative completeness, so as to apply to Dynamic Logic. Define a *V-schema* to be a closed first-order formula φ in V augmented with additional identifiers (place-holders) for relations. For example, the Schema of Induction over $\mathbb{N}$ is

$$P(0)\ \wedge\ (\forall x\ P(x) \rightarrow P(s(x)))\ \rightarrow\ \forall x\ P(x)$$

Here 0 and s (denoting the successor function) are part of the given vocabulary, whereas P is a new, unary, place-holder. Note that a V-formula is trivially also a V-schema.

A *DL-instance* of a schema φ is the result of replacing in φ each such place-holder identifier, of arity k say, by a k-ary DL-predicate, i.e. $\lambda x_1 \ldots x_k. \psi$, where ψ is a DL-formula (and bound variables in ψ are renamed to avoid scoping clashes). For example, if $\psi[u, v, w]$ is a formula with variables u, v, w free, then the instance of Induction for $\lambda v.\psi$ is

$$\psi[u, 0, w]\ \wedge\ (\forall x\ \psi[u, x, w] \rightarrow \psi[u, s(x), w])\ \rightarrow\ \forall x\ \psi[u, x, w]$$

A schema φ is *true* in a V-structure $\mathcal{S}$ if it is true regardless of the interpretation of each k-ary place-holders as a k-ary relation over the universe $|\mathcal{S}|$ of $\mathcal{S}$; i.e., if it is true in every expansion of $\mathcal{S}$. For example, the schema of induction above is true in the structure $\mathbb{N}$ (with zero and successor). If $\mathcal{S}$ is a V-structure, then the *schematic theory of* $\mathcal{S}$ consists of the V-schemas true in $\mathcal{S}$. We write $\mathbf{DL}(\mathcal{S})$ for the formalism **DL** augmented with all DL-instances of *schemas* true in $\mathcal{S}$.

We continue to refer to Cook's notion of expressiveness: a V-structure $\mathcal{S}$ is expressive if for every program α there is a V-formula ξ_α equivalent in $\mathcal{S}$ to $\langle\alpha\rangle(\boldsymbol{x} = \boldsymbol{v})$.

Theorem 5. **DL** *is (schematic) relatively complete in the following sense: for every expressive structure $\mathcal{S}$ and DL formula φ, if $\mathcal{S} \models \varphi$, then φ is provable in* $\mathbf{DL}(\mathcal{S})$.

A proof of Theorem 5 will be given elsewhere. The core idea is to emulate the proof above of Theorem 4, with the formulas ξ_α replacing $\mathtt{M}_\alpha$. Each formula ξ_{β^*} satisfies the

Iteration Rule for β, read as a schema, and so the schematic theory of a structure makes it possible to use the formulas ξ_α in place of $\mathtt{M}_\alpha$.

In [18, §2.5] we showed that Cook's notion of relative completeness is the local projection (to expressive structures) of the inductive completeness theorem proved there for Hoare's Logic. Analogously, Theorem 5 provides a notion of relative completeness which is the projection of inductive completeness of **DL**.

3.3 Arithmetical Completeness

The termination of imperative programs is commonly proved by the Variance Method: one attaches to each instance of a looping construct in the program (such as a **while** loop or a recursive procedure) a parameter ranging over the field A of a well-founded relation $\succ$, and shows that each cycle reduces that parameter under $\succ$:

$$\frac{\vdash \boldsymbol{\varphi} \wedge a{=}x \rightarrow \langle\boldsymbol{\alpha}\rangle\boldsymbol{\varphi} \wedge a \succ x}{\vdash \boldsymbol{\varphi} \wedge A(x) \rightarrow \langle\boldsymbol{\alpha}^*\rangle\boldsymbol{\varphi}}$$
$$(a \text{ not assigned-to in } \boldsymbol{\alpha})$$

Taking $\succ$ to be the natural order on $\mathbb{N}$, the Variance Rule yields the Convergence Rule of [10]:

$$\frac{\vdash \boldsymbol{\varphi}(\mathbf{s}x) \rightarrow \langle\boldsymbol{\alpha}\rangle\boldsymbol{\varphi}(x)}{\vdash \boldsymbol{\varphi}(x) \wedge N(x) \rightarrow \langle\boldsymbol{\alpha}^*\rangle\boldsymbol{\varphi}(\mathbf{0})}$$
$$(x \text{ not assigned-to in } \boldsymbol{\alpha}$$
$$N \text{ interpreted as } \mathbb{N})$$

Note that this rule fuses the interpretation of counting in the background theory and in the program semantics, thus forcing the numeric variables to range precisely over the natural numbers. In particular, the rule is sound only for structures in which N is interpreted as $\mathbb{N}$, structures dubbed *arithmetical* in [10].

The rationale of [10] for the Convergence Rule was ostensibly to establish a completeness property for DL, analogous to Cook's Relative Completeness Theorem for Hoare-style logics. However, Cook's notion of relative completeness is itself problematic, and the arithmetic completeness of [10] faces additional pitfalls. One is the fact that the Convergence Rule is sounds only for a special class of structures, which is itself not first-order axiomatizable.

Also, whereas Hoare's Logic for Partial-correctness assertions is based on a formal separation between rules for programs (Hoare's rules) and rules for data (the background theory), the essential feature of the Convergence Rule is that it fuses the two. When programs and data are fused, and programs and their semantics are codable by data (as is the case in arithmetic structures), the very rationale for factoring out rules for programs from axioms for data is weakened, and one might arguably reason directly about programs in a first-order theory, as done for example in [1]. Needless to say, proving program termination by the Variance Method is of immense practical importance, and our contention is simply that it is a mathematical tool (referring to particular structures, i.e. well-orderings) rather than a logical principle. The misfit of the Convergence Rule in the rest of the axiomatics of DL is indeed manifest in the rather arbitrary choice of the natural numbers as inductive measure.

The schematic relative completeness of DL clarifies the status of the Convergence Rule, and more generally of the notion of arithmetical completeness. As observed above, our ability to use induction on $\mathbb{N}$ to prove termination (and more complex) assertions in DL is hindered by the restriction of induction to first-order formulas. By referring to data-induction as a schema, which can be instantiated to any DL formula, we recover the freedom of action that we have in reasoning about DL formulas, and which Harel's Convergence Rule is providing in an *ad hoc* fashion, and only for arithmetical structures. Indeed, the Convergence Rule is merely a syntactic variant of the schema of induction for $\mathbb{N}$. The reference to $\mathbb{N}$, however, is off target, as our result on schematic relative completeness shows: when $\mathtt{M}_{\alpha^*}$ is expressed in a structure by a first-order formula ξ_α, as it should in an expressive structure, then Invariance (or — equivalently — Induction) for ξ_α is the relevant true schema, and there is no need to invoke the natural numbers.

True, schematic relative completeness refers to valid schemas of the structure in hand, and recognizing these is complete-Π^1_1 (for the natural numbers). But the entire framework of Cook's relative completeness is highly non-effective to begin with.

4 Summary and Directions

Our inductive theories provide a generic framework for explicit reasoning about programs (i.e. without using the modal operators of logics of programs). They build directly on the inductive definition of program semantics, as opposed to frameworks proposed in the past, based on natural numbers (see e.g. [25]). This approach provides a close match between the semantic definition of programming languages, a specification languages for program behavior, and formal verification theories for them. Correspondingly, the notion of inductive completeness establishes a natural match between a proposed (modal) logic of programs and the programming language to which it refers.

Moreover, inductive theories for programming languages provide a setting in which tools of automated deduction can be applied directly, notably various methods for identifying inductive assertions in proofs.

The methods and results presented here can be applied to richer programming paradigms, such as recursive procedures, parallelism, and object-oriented programming (compare [21] and [2]). Of course, programming constructs that use dynamic memory allocation, say recursive procedures with local variables, would require more expressive logical constructs, such as relations with undetermined arities (or, equivalently, direct reference to lists or streams). However, these do not seem to raise any conceptual difficulty. In particular, we would expect to obtain inductively complete logics for programming languages for which relative completeness (even for Hoare's logic, let alone Dynamic Logic) is excluded by [4].

References

1. Andreka, H., Nemeti, I., Sain, I.: A complete logic for reasoning about programs via non-standard model theory, Parts I and II. Theoretical Computer Science 17, 193–212, 259–278 (1982)
2. Beckert, B., Platzer, A.: Dynamic logic with non-rigid functions. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS, vol. 4130, pp. 266–280. Springer, Heidelberg (2006)

3. Blass, A., Gurevich, Y.: The underlying logic of Hoare logic. Current Trends in Theoretical Computer Science, 409–436 (2001)
4. Clarke, E.: Programming language constructs for which it is impossible to obtain good Hoare-like axioms. J. ACM 26, 129–147 (1979)
5. Cook, S.A.: Soundness and completeness of an axiom system for program verification. SIAM J. Computing 7(1), 70–90 (1978)
6. Csirmaz, L.: Programs and program verification in a general setting. Theoretical Computer Science 16, 199–210 (1981)
7. Feferman, S.: Formal theories for transfinite iterations of generalized inductive definitions and some subsystems of analysis. In: Intuitionism and Proof Theory, pp. 303–326. North-Holland, Amsterdam (1970)
8. Feferman, S., Sieg, W.: Iterated inductive definitions and subsystems of analysis. In: Iterated Inductive Definitions and Subsystems of Analysis: Recent Proof-Theoreitc Studies. LNM, vol. 897, pp. 16–77. Springer, Berlin (1981)
9. Harel, D., Meyer, A., Pratt, V.: Computability and completeness in logics of programs. In: Proceedings of the ninth symposium on the Theory of Computing, Providence, pp. 261–268. ACM, New York (1977)
10. Harel, D.: First-Order Dynamic Logic. LNCS, vol. 68. Springer, Heidelberg (1979)
11. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. MIT Press, Cambridge (2000)
12. Honsell, F., Miculan, M.: A natural deduction approach to dynamic logics. In: Berardi, S., Coppo, M. (eds.) TYPES 1995. LNCS, vol. 1158, pp. 165–182. Springer, Heidelberg (1996)
13. Kreisel, G.: Generalized inductive definitions. Reports for the seminar on foundations of analysis, Stanford, vol. 1 §3 (1963)
14. Kreisel, G.: Mathematical logic. In: Saaty, T. (ed.) Lectures on Modern Mathematics, vol. III, pp. 95–195. John Wiley, New York (1965)
15. Leivant, D.: Logical and mathematical reasoning about imperative programs. In: Conference Record of the Twelfth Annual Symposium on Principles of Programming Languages, pp. 132–140. ACM, New York (1985)
16. Leivant, D.: Partial correctness assertions provable in dynamic logics. In: Walukiewicz, I. (ed.) FOSSACS 2004. LNCS, vol. 2987, pp. 304–317. Springer, Heidelberg (2004)
17. Leivant, D.: Matching explicit and modal reasoning about programs: A proof theoretic delineation of dynamic logic. In: Twenty-first Symposium on Logic in Computer Science (LiCS 2006), Washington, pp. 157–166. IEEE Computer Society Press, Los Alamitos (2006)
18. Leivant, D.: Inductive completeness of logics of programs. In: Proceedings of the Workshop on Logical Frameworks and Meta-Languages (to appear, 2008)
19. Martin-Löf, P.: Hauptsatz for the intuitionistic theory of iterated inductive definitions. In: Fenstad, J.E. (ed.) Proceedings of the Second Scandinavian Logic Symposium, pp. 63–92. North-Holland, Amsterdam (1971)
20. Mirkowska, G.: On formalized systems of algorithmic logic. Bull. Acad. Polon. Sci. 19, 421–428 (1971)
21. Nishimura, H.: Arithmetical completeness in first-order dynamic logic for concurrent programs. Publ. Res. Inst. Math. Sci. 17, 297–309 (1981)
22. Pratt, V.: Semantical considerations on Floyd-Hoare logic. In: Proceedings of the seventeenth symposium on Foundations of Computer Science, Washington, pp. 109–121. IEEE Computer Society, Los Alamitos (1976)
23. Sain, I.: An elementary proof for some semantic characterizations of nondeterministic Floyd-Hoare logic. Notre Dame Journal of Formal Logic 30, 563–573 (1989)
24. Segerberg, K.: A completeness theorem in the modal logic of programs (preliminary report). Notics of the American Mathematical Society 24(6), A–552 (1977)
25. Szalas, A.: On strictly arithmetical completeness in logics of programs. Theoretical Computer Science 79(2), 341–355 (1991)

Dependency Tree Automata

Colin Stirling

School of Informatics
Informatics Forum
University of Edinburgh
cps@inf.ed.ac.uk

Abstract. We introduce a new kind of tree automaton, a dependency tree automaton, that is suitable for deciding properties of classes of terms with binding. Two kinds of such automaton are defined, nondeterministic and alternating. We show that the nondeterministic automata have a decidable nonemptiness problem and leave as an open question whether this is true for the alternating version. The families of trees that both kinds recognise are closed under intersection and union. To illustrate the utility of the automata, we apply them to terms of simply typed lambda calculus and provide an automata-theoretic characterisation of solutions to the higher-order matching problem.

Keywords: Tree automata, binding terms, typed lambda calculus.

1 Introduction

A standard method for solving problems over families of terms is to show that the solutions are tree recognisable: that is, that there is a tree automaton that accepts a term if, and only if, it is a solution to the problem [4]. In such a case, terms are built out of a finite family of (graded) symbols, that is symbols with an arity, which are naturally represented as trees. A tree automaton involves a finite set of states and a finite set of transitions. It traverses a term bottom-up or top-down labelling it with states according to the transitions and if it succeeds then the term is accepted.

Many logical and computational notations employ binders such as $\exists x$, μX, λx, $a(x)$ from first-order logic, fixed-point logic, lambda calculus, π-calculus, and so on. Although each term of such a notation can be represented as a finite tree, to represent families of such terms may require an infinite alphabet: as illustrated by the following formulas $\forall z.\exists f_1 \ldots \exists f_n.f_n(f_{n-1}(\ldots(f_1(z))\ldots))$ for all $n \geq 0$. Although there is research in extending standard automata to infinite alphabets, see the survey [9], it does not cover the specific case caused by binding.

We introduce a new type of tree automaton, a dependency tree automaton, for recognising terms with binding. To maintain a finite alphabet, terms are represented as finite trees which also have an extra binary relation $\downarrow$ between their nodes that represents binding: an idea partly inspired by nested automata which also employ a binary relation $\downarrow$ between nodes representing nesting such as calls and returns [1,2]. Two kinds of dependency tree automaton are defined, nondeterministic and alternating. We show that the nonemptiness problem, whether a given automaton accepts at least one tree, is decidable for nondeterministic automata. However, we are unable to show this for the alternating automata and we are also unable to determine whether they are more expressive than nondeterministic automata. The families of trees that both kinds of automata recognise are closed under intersection and union. To illustrate the utility of the automata, we apply them to terms of simply typed lambda calculus and use alternating automata to provide an automata-theoretic characterisation of solutions to the higher-order matching problem.

In Section 2 we define binding trees and the two kinds of dependency tree automaton and show decidability of nonemptiness for the nondeterministic case. We also illustrate how the nondeterministic automata can be used to recognise normal form terms of simply typed lambda calculus of a fixed type. In Section 3, we apply the alternating dependency tree automata to higher-order matching. The proof of characterisation is presented in Section 4.

2 Dependency Tree Automata

In this section we introduce *binding* trees and *dependency* tree automata that operate on them.

Definition 1. *Assume Σ is a finite graded alphabet where each element $s \in \Sigma$ has an arity $ar(s) \geq 0$. Moreover, Σ consists of three disjoint sets Σ_1 that are the binders which have arity 1, Σ_2 are (the bound) variables and Σ_3 are the remaining symbols. A binding Σ-*tree *is a finite tree where each node is labelled with an element of Σ together with a binary relation $\downarrow$ (representing binding). If node n in the tree is labelled with s and $ar(s) = k$ then n has precisely k successors in the tree, the nodes $n1, \ldots, nk$. Also, if a node n is labelled with a variable in Σ_2 then there is a unique node b labelled with a binder occurring above n in the tree such that $b \downarrow n$. For ease of exposition we also assume the following restrictions on Σ-trees: if node n is labelled with a binder then $n1$ is labelled with an element of $\Sigma_2 \cup \Sigma_3$ and if n is labelled with an element of $\Sigma_2 \cup \Sigma_3$ and ni is a successor then it is labelled with a binder. Let T_Σ be the set of binding Σ-trees.*

Definition 2. *A dependency Σ-tree automaton $\mathsf{A} = (Q, \Sigma, q_0, \Delta)$ where Q is a finite set of states, Σ is the finite alphabet, $q_0 \in Q$ is the initial state and Δ is a finite set of transition rules each of which has one of the following three forms.*

1. *$qs \Rightarrow (q_1, \ldots, q_k)$ where $s \in \Sigma_2 \cup \Sigma_3$, $ar(s) = k$, $q, q_1, \ldots, q_k \in Q$*
2. *$qs \Rightarrow q's'$ where $s \in \Sigma_1$, $s' \in \Sigma_3$ and $q, q' \in Q$*
3. *$(q', q)s \Rightarrow q_1 x$ where $s \in \Sigma_1$, $x \in \Sigma_2$ and $q', q, q_1 \in Q$*

Definition 3. *A run of $\mathsf{A} = (Q, \Sigma, q_0, \Delta)$ on $t \in \mathsf{T}_\Sigma$ is a $(\Sigma \times Q)$-tree whose nodes are pairs (n, q) where n is a node of t and $q \in Q$ labelled (s, q) if n is labelled s in t which is defined top-down with root (ϵ, q_0) where ϵ is the root of t. Consider a node (n, q) labelled (s, q) of a partial run tree which does not have* *successors. If $s \in \Sigma_2 \cup \Sigma_3$ and $qs \Rightarrow (q_1, \ldots, q_k) \in \Delta$ then the successors of (n, q) are the nodes (ni, q_i), $1 \leq i \leq k$. If $s \in \Sigma_1$, $n1$ is labelled $s' \in \Sigma_3$ and $qs \Rightarrow q's' \in \Delta$ then $(n1, q')$ is the successor of (n, q). If $s \in \Sigma_1$, $n1$ is labelled $x \in \Sigma_2$, $m \downarrow n1$ in t, (m, q') occurs above or at (n, q) and $(q', q)s \Rightarrow q_1 x \in \Delta$ then $(n1, q_1)$ is the successor of (n, q). A accepts the Σ-tree t iff there is a run of A on t such that if (n, q) is a leaf then n is a leaf of t. Let $\mathsf{T}_\Sigma(\mathsf{A})$ be the set of Σ-trees accepted by A.*

A dependency tree automaton A has a finite set of states Q and transitions Δ (which can be nondeterministic). A run of A on a Σ-tree t adds an additional Q labelling to (a subtree of) t: so it is a $(\Sigma \times Q)$-tree. It starts with (ϵ, q_0) where ϵ is the root of t and q_0 is the initial state of A. Subsequent nodes are derived by percolating states down t. The state at a node that is labelled with a variable not only depends on the state of its immediate predecessor but also on the state of the node that labels its binder. This introduces non-local dependence in the automaton (hence the name). A run on t is accepting if it is complete in the sense that each node of t is labelled with an element of Q: if (n, q) is a leaf of the run tree then n is a leaf of t.

Dependency tree automata were partly inspired by nested word and tree automata [1,2] which are also an amalgam of a traditional automaton and a binary relation $\downarrow$ on nodes of the (possibly infinite) word or tree. However, in that setting $\downarrow$ represents *nesting* such as provided by bracketing and useful for modelling procedure calls and returns. Nesting involves natural restrictions on the relation $\downarrow$ such as "no-crossings": if $m_1 \downarrow m_2$ and $n_1 \downarrow n_2$ and m_1 is above n_1 then either m_2 is above n_1 or n_2 is above m_2. Such restrictions are not appropriate when modelling binding, for instance as with a formula $\forall f.\exists x.\phi(f(x))$.

A fundamental exemplar of binding is terms of the simply typed lambda calculus. Simple types are generated from a single base type $\mathbf{0}$ using the binary $\rightarrow$ operator[1]: $A \rightarrow B$ is the type of functions from A to B. Assuming $\rightarrow$ associates to the right, if type $A \neq \mathbf{0}$ then it has the form $A_1 \rightarrow \ldots \rightarrow A_n \rightarrow \mathbf{0}$, written $(A_1, \ldots, A_n, \mathbf{0})$ here. The *order* of $\mathbf{0}$ is 1 and the *order* of $(A_1, \ldots, A_n, \mathbf{0})$ is $k+1$ where k is the maximum of the orders of the A_is.

Terms of the simply typed λ-calculus (in Church style) are built from a countable set of typed variables $x, y, \ldots$ and constants $a, f, \ldots$ (each variable and constant has a unique type).

Definition 4. *The smallest set T of simply typed terms is:*

1. *if x (f) has type A then $x : A \in T$ ($f : A \in T$),*
2. *if $t : B \in T$ and $x : A \in T$ then $\lambda x.t : A \rightarrow B \in T$,*
3. *if $t : A \rightarrow B \in T$ and $u : A \in T$ then $(tu) : B \in T$.*

The order of a typed term is the order of its type. In a sequence of unparenthesized applications, we adopt the usual convention that application associates to the left, so $tu_1 \ldots u_k$ is $((\ldots(tu_1)\ldots)u_k)$. The usual definitions of free and bound

[1] For simplicity, we assume just one base type: everything that is to follow can be extended to the case of arbitrary many base types.

variable occurrences and when a typed term is closed are assumed. Moreover, we assume the standard definitions and properties of α-equivalence, β-reduction, η-reduction and $\beta\eta$-equivalence, $=_{\beta\eta}$, such as strong normalization of β-reduction: see, for instance, Barendregt [3].

Fact 1. *Every simply typed λ-calculus term is $\beta\eta$-equivalent to a unique term (up to α-equivalence) in* η-long form *as follows,*

1. *if $t : \mathbf{0}$ then it is $u : \mathbf{0}$ where u is a constant or a variable, or $u\, t_1 \dots t_k$ where $u : (B_1, \dots, B_k, \mathbf{0})$ is a constant or a variable and each $t_i : B_i$ is in η-long form,*
2. *if $t : (A_1, \dots, A_n, \mathbf{0})$ then t is $\lambda y_1 \dots y_n.t'$ where each $y_i : A_i$ and $t' : \mathbf{0}$ is in η-long form.*

Throughout, we write $\lambda z_1 \dots z_m$ for $\lambda z_1 \dots \lambda z_m$. A term is in *normal form* if it is in η-long form.

Definition 5. *For any type A and set of constants C, $T_A(C)$ is the set of closed terms in normal form of type A whose constants belong to C.*

Example 1. The monster type $M = ((((\mathbf{0},\mathbf{0}),\mathbf{0}),\mathbf{0}),\mathbf{0},\mathbf{0})$ has order 5. Assume $x_1 : (((\mathbf{0},\mathbf{0}),\mathbf{0}),\mathbf{0})$, $x_2 : \mathbf{0}$ and $z_i : (\mathbf{0},\mathbf{0})$ for $i \geq 1$. The following family of terms in normal form $\lambda x_1 x_2.x_1(\lambda z_1.x_1(\lambda z_2 \dots x_1(\lambda z_n.z_n(z_{n-1}(\dots z_1(x_2))\dots))\dots))$ for $n \geq 0$ belong to $T_M(\emptyset)$. Even to write down this subset of terms up to α-equivalence requires an alphabet of unbounded size. More technically, M is known not to be finitely generable [6]. However, there is a straightforward representation of this family of terms as binding Σ-trees (when dummy λs are added to fulfil the restrictions in Definition 1). Nodes are labelled with binders $\lambda x_1 x_2$, λz, λ, or with variables x_1, z of arity 1 and x_2 of arity 0: in linear form $\lambda x_1 x_2.x_1(\lambda z.x_1(\lambda z \dots x_1(\lambda z.z(\lambda.z(\dots \lambda.z(\lambda.x_2))\dots))\dots))$ where there is an edge $\downarrow$ from the node labelled $\lambda x_1 x_2$ to each node labelled x_1 or x_2, and an edge $\downarrow$ from the first node labelled λz to the last node labelled z, and so on. There are no edges $\downarrow$ from nodes labelled with the empty binder λ. Given such a representation of normal form terms in $T_M(\emptyset)$, dependency tree automata can be defined that recognize subsets: there is a simple deterministic two state automaton that recognizes the subset which have an even number of occurrences of x_1. □

Fact 2. *For any type A and finite C, there is a finite Σ such that every $t \in T_A(C)$ up to α-equivalence is representable as a binding Σ-tree (with dummy λs).*

The nonemptiness problem for classical (bottom-up or top-down) nondeterministic tree automata, whether a given automaton accepts at least one tree, is decidable in linear time. Also, the set of families of trees that are recognizable is regular (which implies closure under complement and intersection) [4].

Theorem 1. *Assume A, A_1 and A_2 are dependency Σ-tree automata.*

1. *The nonemptiness problem, given A is $\mathsf{T}_\Sigma(\mathsf{A}) \neq \emptyset$?, is decidable.*
2. *Given A_1 and A_2, there is an A such that $\mathsf{T}_\Sigma(\mathsf{A}) = \mathsf{T}_\Sigma(\mathsf{A}_1) \cap \mathsf{T}_\Sigma(\mathsf{A}_2)$.*
3. *Given A_1 and A_2, there is an A such that $\mathsf{T}_\Sigma(\mathsf{A}) = \mathsf{T}_\Sigma(\mathsf{A}_1) \cup \mathsf{T}_\Sigma(\mathsf{A}_2)$.*

Proof. Assume $\mathsf{A} = (Q, \Sigma, q_0, \Delta)$ is a Σ-tree automaton and $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$ where Σ_1 are the binders, Σ_2 the variables and Σ_3 the other symbols. Let $\parallel t \parallel$ be the height of the Σ-tree t and $|S|$ be the size of the finite set S. We show that if $\mathsf{T}_\Sigma(\mathsf{A}) \neq \emptyset$ then A accepts a Σ-tree t such that $\parallel t \parallel \leq (|\Sigma_1||Q|+1)(|\Sigma||Q|+1)$. If A accepts t and $\parallel t \parallel > l(|\Sigma||Q|+1)$ then in the accepting run of A on t there are l nodes of t, $n_1, \ldots, n_l$ with the same label in Σ and labelled with the same state $q \in Q$ such that each n_i occurs above n_j when $i < j$. Let $B(i, j)$, where $1 \leq i < j \leq l$, be the set of pairs binders $a \in \Sigma_1$ and states $q' \in Q$ such that there is a node n' between n_i and n_j (excluding n_j) labelled with a and q' in the successful run of A on t such that there is an edge $n' \downarrow n''$ where n'' is n_j or occurs below it in t. Also, let $U(i)$ be the set of pairs binders $a \in \Sigma_1$ and states $q \in Q$ such that there is a node n' above n_i in t labelled with a and q in the successful run of A on t. Clearly, if $B(i, j) \subseteq U(i)$ then there is a smaller Σ-tree t' which is accepted by A: the subtree at node n_i is replaced with the subtree at n_j and any edge $n' \downarrow n''$ where n'' is n_j or below it and n' is between n_i and n_j (excluding n_j) is replaced with an edge $n \downarrow n''$ where n is the node above n_i labelled with the same binder and state as n'. Clearly, if A accepts t then it accepts t'. By simple counting, there must be an i, j with $1 \leq i < j \leq (|\Sigma_1||Q| + 1)$ such that $B(i, j) \subseteq U(i)$. Therefore, nonemptiness is decidable. The other parts of the theorem follow from the usual product and disjoint union of automata (which here includes the binding relations). □

We do not know if the families of trees recognized by these automata are closed under complement.

Definition 6. *An alternating dependency Σ-tree automaton $\mathsf{A} = (Q, \Sigma, q_0, \Delta)$ is as in Definition 2 except for the first clause for transitions which now is*

1. $qs \Rightarrow (Q_1, \ldots, Q_k)$ *where* $s \in \Sigma_2 \cup \Sigma_3$, $ar(s) = k$, $q \in Q$ *and* $Q_1, \ldots, Q_k \subseteq Q$.

Definition 7. *A run of alternating dependency Σ-automaton $\mathsf{A} = (Q, \Sigma, q_0, \Delta)$ on $t \in \mathsf{T}_\Sigma$ is a $(\Sigma \times Q)$-tree whose nodes are pairs (n, α) where n is a node of t and $\alpha \in Q^*$ is a sequence of states, labelled (s, q) if n is labelled s in t and $\alpha = \alpha' q$ which is defined top-down with root (ϵ, q_0) where ϵ is the root of t. Consider a node (n, α) labelled (s, q) of a partial run tree which does not have successors. If $s \in \Sigma_2 \cup \Sigma_3$ and $qs \Rightarrow (Q_1, \ldots, Q_k) \in \Delta$ then the successors of (n, α) are $\{(ni, \alpha q') \mid 1 \leq i \leq k$ and $q' \in Q_i\}$. If $s \in \Sigma_1$, $n1$ is labelled $s' \in \Sigma_3$ and $qs \Rightarrow q's' \in \Delta$ then $(n1, \alpha q')$ is the successor of (n, α). If $s \in \Sigma_1$, $n1$ is labelled $x \in \Sigma_2$, $m \downarrow n1$ in t, $(m, \alpha' q')$ occurs above or at (n, α) and $(q', q)s \Rightarrow q_1 x \in \Delta$ then $(n1, \alpha q_1)$ is the successor of (n, α). A accepts the Σ-tree t iff there is a run of A on t such that if $(n, \alpha q)$ is a leaf labelled (s, q) of the run tree then either s has arity 0 or $qs \Rightarrow (\emptyset, \ldots, \emptyset) \in \Delta$. Let $\mathsf{T}_\Sigma(\mathsf{A})$ be the set of Σ-trees accepted by A.*

A run of an alternating automaton on a Σ-tree t is itself a tree built out of the nodes of t and sequences of states Q^+. There can be multiple copies of nodes of t within a run because a transition applied to a node n $qs \Rightarrow (Q_1, \ldots, Q_k)$ spawns individual copies at ni for each state in Q_i. These automata are alternating as the Q_is can be viewed as conjuncts $\bigwedge_{q \in Q_i} q$ and nondeterminism provides the disjuncts.

Classically, nondeterministic and alternating tree automata accept the same families of trees and the nonemptiness problem for alternating automata is decidable in exponential time [4]. We do not know whether nondeterministic dependency tree automata are as expressive as the alternating automata. Also, it is an open question whether the nonemptiness problem for alternating dependency tree automata is decidable. However, the families of trees recognized by the alternating automata are closed under intersection and union using a similar argument to Theorem 1.

Despite these open expressiveness and algorithmic questions, we shall show that alternating dependency tree automata do have an interesting application.

3 Application of Dependency Automata

We apply alternating dependency tree automata to higher-order matching.

Definition 8. *A matching problem in simply typed lambda calculus is an equation $v = u$ where $v, u : \mathbf{0}$ are in normal form and u is closed. The order of the problem is the maximum of the orders of the free variables $x_1, \ldots, x_n$ in v. A solution is a sequence of terms $t_1, \ldots, t_n$ such that $v\{t_1/x_1, \ldots, t_n/x_n\} =_{\beta\eta} u$ where $v\{t_1/x_1, \ldots, t_n/x_n\}$ is the simultaneous substitution of t_i for each free occurrence of x_i in v for $i : 1 \leq i \leq n$.*

Given a matching problem $v = u$, one question is whether it has a solution: can v be pattern matched to u? The motivation here is a different question: is there an automata-theoretic characterization of the set of solutions of a matching problem? Comon and Jurski define (almost classical) bottom-up tree automata that characterize all solutions to a 4th-order problem [5]: the slight elaboration is the use of $\square_A$ symbols standing for arbitrary typed subterms of type A. The authors describe two problems with extending their automata beyond the 4th-order case. The first is how to guarantee only finitely many states. States of their automata are constructed out of observational equivalence classes of terms due to Padovani [8]. Up to a 4th-order problem, one only needs to consider finitely many terms. With 5th and higher orders, this is no longer true and one needs to quotient the potentially infinite terms into their respective observational equivalence classes in order to define only finitely many states: however as Padovani shows this procedure is, in fact, equivalent to the matching problem itself [8]. The second problem is how to guarantee that the alphabet has finite size. As we saw with the monster type in Example 1, fifth-order terms may (essentially) contain infinitely many different variables. In [14], we overcame the first problem but not the second: relative to a fixed finite alphabet, the set of solutions over that alphabet to a matching problem is tree automata recognizable. The proof relies on a similar technology to that used here (a game-theoretic characterisation of matching). We now overcome the second problem using alternating dependency tree automata.

Definition 9. *Assume $u : \mathbf{0}$ and $w : A$ are closed terms in normal form and $x : (A, \mathbf{0})$. An interpolation problem P has the form $xw = u$. The type of problem P is that of x and the order of P is the order of x. A solution of P of type B is a closed term $t : B$ in normal form such that $tw =_\beta u$. We write $t \models P$ if t is a solution of P.*

Because terms are in η-long form, β-equality and $\beta\eta$-equality coincide (for instance, see [15] for a recent account).

Conceptually, interpolation is simpler than matching because there is a single variable x that appears at the head of the equation. If $v = u$ is a matching problem with free variables $x_1 : A_1, \ldots, x_n : A_n$ where v and u are in normal form, then its *associated* interpolation problem is $x(\lambda x_1 \ldots x_n.v) = u$ where $x : ((A_1, \ldots, A_n, \mathbf{0}), \mathbf{0})$. This appears to raise order by 2 as with the reduction of matching to pairs of interpolation equations in [10]. However, we only need to consider potential solution terms (in normal form with the right type) $\lambda z.zt_1 \ldots t_n$ where each $t_i : A_i$ is closed and so cannot contain z: we say that such terms are *canonical.*

Proposition 3. *A matching problem has a solution iff its associated interpolation problem has a canonical solution.*

Proof. Assume $v = u$ is a matching problem with $x_1 : A_1, \ldots, x_n : A_n$ as free variables and where v and u are in normal form. If it has a solution $t_1, \ldots, t_n$ where each t_i is in normal form, then $v\{t_1/x_1, \ldots, t_n/x_n\} =_\beta u$. Clearly, it therefore follows that $\lambda z.zt_1 \ldots t_n(\lambda x_1 \ldots x_n.v) =_\beta v\{t_1/x_1, \ldots, t_n/x_n\} =_\beta u$. Conversely, if $\lambda z.zt_1 \ldots t_n$ is a canonical solution to its associated interpolation problem $x(\lambda x_1 \ldots x_n.v) = u$ then $t_1, \ldots, t_n$ solves the problem $v = u$. $\square$

In the literature there are slight variant definitions of matching. Statman describes the problem as a range problem [11]: given $v : (A_1, \ldots, A_n, B)$ and $u : B$ where both u and v are closed, are there terms $t_1 : A_1, \ldots, t_n : A_n$ such that $vt_1 \ldots t_n =_{\beta\eta} u$? If $B = (A_1, \ldots, A_m, \mathbf{0})$ is of higher type then u in normal form is $\lambda x'_1 \ldots x'_m.w$. Therefore, we can consider the matching problem $(vx_1 \ldots x_n)c_1 \ldots c_m = w\{c_1/x'_1, \ldots, c_m/x'_m\}$ where the c_i's are new constants that cannot occur in a solution term. In [8] a matching problem is a family of equations $v_1 = u_1, \ldots, v_m = u_m$ to be solved uniformly: they reduce to a single equation $fv_1 \ldots v_m = fu_1 \ldots u_m$ where f is a constant of the appropriate type.

Example 2. The matching problem $x_1(\lambda z.x_1(\lambda z'.za)) = a$ from [5] is 4th-order where $z, z' : (\mathbf{0}, \mathbf{0})$ and $x_1 : (((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0})$. Its associated interpolation problem is $x(\lambda x_1.x_1(\lambda z.x_1(\lambda z'.za))) = a$ with $x : (((((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0}), \mathbf{0}), \mathbf{0})$. A canonical solution has the form $\lambda x.x(\lambda y.y(\lambda y_1^1 \ldots y(\lambda y_1^k.s) \ldots))$ where s is the constant a or one of the variables y_1^j, $1 \leq j \leq k$. $\square$

Definition 10. *If P is $xw = u$ then C_P is the set of constants that occur in u together with one fresh constant $b : \mathbf{0}$.*

Fact 4. *Let C be any set of constants and let P be an interpolation problem of type B. If $t \models P$ and $t \in \mathsf{T}_B(C)$ then there is a $t' \in \mathsf{T}_B(C_p)$ such that $t' \models P$.*

$$\begin{tikzcd}
 & @ \arrow[dl] \arrow[dr] & \\
t & & w
\end{tikzcd}$$

Fig. 1. An interpolation tree

Given a potential solution term t in normal form to the interpolation problem P, $xw = u$, there is the tree in Figure 1. If $x : (A, \mathbf{0})$ then the explicit application operator $@ : ((A, \mathbf{0}), A, \mathbf{0})$ has its expected meaning: $@tw = tw$. Our goal is to define an alternating dependency tree automaton that accepts the tree in Figure 1 when t is a solution of P. By Fact 4 we can assume that any such solution term t only contains constants in the finite set C_P. Moreover, we assume the following representation of binders and variables. A binder $\lambda\overline{y}$ is such that either $\overline{y}$ is empty and therefore is a dummy λ and can not bind a variable occurrence or $\overline{y} = y_1 \dots y_k$ and $\lambda\overline{y}$ can only then bind variable occurrences of the form y_i, $1 \leq i \leq k$. Consequently, in the binding tree representation if $n \downarrow m$ and n is labelled $\lambda y_1 \dots y_k$ then m is labelled y_i for some i.

In general the right term u of an interpolation problem may contain bound variables: for instance, $x(\lambda z.z) = f(\lambda x_1 x_2 x_3.x_1 x_3)a$ has order 3 where x has type $((\mathbf{0}, \mathbf{0}), \mathbf{0})$ and $f : (((\mathbf{0}, \mathbf{0}), \mathbf{0}, \mathbf{0}, \mathbf{0}), \mathbf{0}, \mathbf{0})$ assuming $x_2 : \mathbf{0}$. For ease of exposition, as it simplifies the presentation considerably, we restrict ourselves to the case where there are no such variables: this is discussed further in Section 5.

Definition 11. *Assume $u : \mathbf{0}$ is closed and does not contain bound variables. The set of subterms of u, $Sub(u)$, is defined inductively: if $u = a : \mathbf{0}$ then $Sub(u) = \{u\}$ and if $u = fu_1 \dots u_k$ then $Sub(u) = \bigcup_{1 \leq i \leq k} Sub(u_i) \cup \{u\}$.*

Given P, we assume a simple finite alphabet Σ (containing C_P, the constants in w, @ and suitable $\lambda\overline{y}$s and variable occurrences).

Example 3. In the case of Example 2, there is the finite syntax where $\Sigma_1 = \{\lambda x, \lambda y, \lambda y', \lambda x_1, \lambda z, \lambda z', \lambda\}$, $\Sigma_2 = \{x, y, y', x_1, z, z'\}$ and $\Sigma_3 = \{a, b, @\}$. □

The states of our dependency tree automaton are based on Ong [7] (which is a different setting, with a fixed infinite λ-term built out of a fixed finite alphabet and an alternating parity automaton). To give intuition, consider a game-theoretic understanding (such as with game-semantics) of Figure 1 where $t = \lambda z.zt_1 \dots t_n$ and $w = \lambda\overline{x}.w'$ as pictured in Figure 2. In the game, play jumps around the interpolation tree. It starts at @ and proceeds down from λz to z and then jumps to $\lambda\overline{x}$ of w (as it labels the subterm that would replace z in a β-reduction). It then proceeds down w and eventually may reach x_j, in which case it jumps to the jth successor of z in t labelled with $\lambda\overline{y}$ and then play proceeds in t and may eventually reach y_k and so jump to the kth successor of x_j in w and so on. The question is how to capture jumping within a tree automaton. This we do using variable assumptions as in [14]: Ong calls them "variable profiles" in his setting.

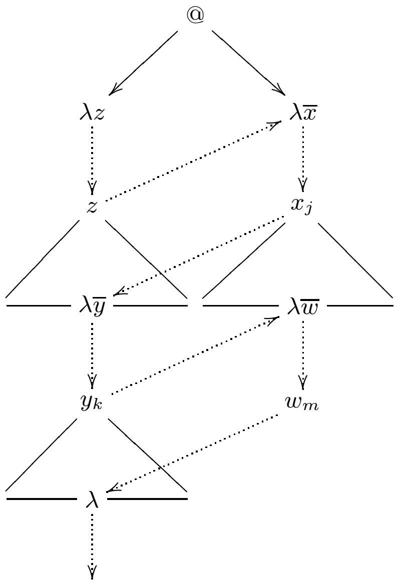


Fig. 2. Game-theoretic view

Definition 12. *Assume Σ is the alphabet for P, $xw = u$, and $\mathsf{R} = Sub(u)$. Relative to R, for each variable $z \in \Sigma_2$ the set of z assumptions, $\Gamma(z)$ is defined inductively: if $z : \mathbf{0}$ then $\Gamma(z) = \{(z, r, \emptyset) \mid r \in \mathsf{R}\}$; if $z : (A_1, \ldots, A_k, \mathbf{0})$ then $\Gamma(z) = \{(z, r, \Gamma) \mid r \in \mathsf{R}, \Gamma \subseteq \bigcup_{1 \leq i \leq k} \bigcup_{x : A_i \in \Sigma_2} \Gamma(x)\}$. A mode is a pair (r, Γ) where $r \in \mathsf{R}$ and $\Gamma \subseteq \bigcup_{z \in \Sigma_2} \Gamma(z)$.*

A variable assumption is an abstraction from a sequence of moves in a game. For instance $(z, u, \{(x_j, r_1, \{(y_k, r_2, \{(w_m, r_3, \emptyset)\})\})\})$ abstracts from the play pictured in Figure 2: the subterms r_i of u represent what is left of u that still needs to be achieved in order for $tw =_\beta u$.

A mode is a pair (r, Γ) where $r \in \mathsf{R}$ and Γ is a set of variable assumptions. Because R is finite and Σ is fixed, there can only be boundedly many different modes (r, Γ): modes are the states of our automaton.

Definition 13. *Assume P is $xw = u$, Σ is its alphabet and $\mathsf{R} = Sub(u)$. The dependency tree automaton is $\mathsf{A}_P = (Q, \Sigma, q_0, \Delta)$ where Q is the set of modes (r_i, Γ_i), $q_0 = (u, \emptyset)$ and the transition relation Δ is defined on nodes of the binding Σ-tree by cases on Σ.*

1. $(u, \emptyset)@ \Rightarrow (\{(u, \Gamma)\}, \{(u, \Gamma_1)\})$ *if* $\Gamma = \{(z, u, \Gamma_1)\}$
2. $((r, \Gamma), (r', \Gamma'))\lambda\overline{y} \Rightarrow (r', \Sigma)x_i$ *if* $(x_i, r', \Sigma) \in \Gamma$
3. $(fr_1 \ldots r_k, \Gamma)\lambda\overline{y} \Rightarrow (fr_1 \ldots r_k, \emptyset)f$
4. $(a, \emptyset)\lambda\overline{y} \Rightarrow (a, \emptyset)$
5. $(r, \Gamma)x_j \Rightarrow (Q_1, \ldots, Q_k)$ *if* $Q_i = \{(r', \Gamma') \mid (y_i, r', \Gamma') \in \Gamma\}$ *for each* $i : 1 \leq i \leq k$ *and* $ar(x_j) = k > 0$
6. $(fr_1 \ldots r_k, \emptyset)f \Rightarrow (\{(r_1, \emptyset)\}, \ldots, \{(r_k, \emptyset)\})$

The root of the interpolation tree labelled @ has two successors t of the form $\lambda z.zt_1 \dots t_n$ and w. The automaton starts with state $(u, \emptyset)$ at @ and then a single z variable assumption (z, u, Γ_1) is chosen. The state at node labelled λz is then $(u, \{(z, u, \Gamma_1)\})$ and (u, Γ_1) at the other successor of @. Assume the current state is (r', Γ') at node n of the interpolation tree labelled $\lambda\overline{y}$. If $n1$ is labelled with variable x_i and $m \downarrow n1$ then m is labelled $\lambda x_1 \dots x_k$ for some k and the state above (r', Γ') at m has the form (r, Γ) where Γ is a set of assumptions for each x_j, $1 \leq j \leq k$. One of the x_i assumptions, (x_i, r', Σ) where the right term r' is as in the state at n is chosen and state (r', Σ') labels $n1$. If $n1$ is labelled f then for the automaton to proceed from node n to $n1$, r' must have the form $fr_1 \dots r_k$. In which case $n1$ is labelled with state $(r', \emptyset)$. Similarly, if $n1$ is labelled with the constant $a : \mathbf{0}$ then r' must be a and $\Gamma' = \emptyset$. If the state is (r, Γ) at node n of the interpolation tree and n is labelled x_j with arity $k > 0$ then Γ consists of sets of y_i assumptions, $1 \leq i \leq k$ for some y (reflecting when play would return to successors of n: for instance, in Figure 2 play jumps from x_j to $\lambda\overline{y}$ and returns to x_js kth successor if it reaches y_k and there can be multiple occurrences of y_k meaning that there could be further returns jumps). For each y_i assumption (y_i, r', Γ') the automaton spawns a copy at ni with state (r', Γ'). Finally, if the state is $(fr_1 \dots r_k, \emptyset)$ at node n of the interpolation tree labelled with f then the automaton proceeds down each successor ni with state $(r_i, \emptyset)$.

Theorem 2. *Assume P is $xw = u$, Σ is the alphabet and A_P is the dependency Σ-tree automaton in Definition 13. For any canonical Σ-term t, A_P accepts the tree @tw iff $t \models P$.*

4 Proof of Theorem 2

The proof of Theorem 2 employs a game-theoretic interpretation of an interpolation tree as illustrated in Figure 2 and developed in [14]. (It avoids questions, answers and justification pointers of game-semantics [7] and uses iteratively defined look-up tables.)

Assume P is the problem $xw = u$, Σ is the alphabet, $\mathsf{R} = \mathrm{Sub}(u)$ and t is a potential solution term. We define the game $\mathsf{G}(t, P)$ played by one participant, player $\forall$, the *refuter* who attempts to show that t is not a solution of P. The game is played on the Σ-binding tree @tw of Figure 1.

Definition 14. *N is the set of nodes of the binding tree @tw labelled with elements of $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$ and S is the set $\{[\,x\,] \,|\, x \in \mathsf{R} \cup \{\forall, \exists\}\}$ of game-states. $[\,\forall\,]$ and $[\,\exists\,]$ are the final game-states. Let N_1 be the subset of nodes N whose labels belong to Σ_1 (the binders). For each $i \geq 1$, the set of look-up tables Θ_i is iteratively defined: $\Theta_1 = \{\theta_1\}$ where $\theta_1 = \emptyset$ and Θ_{i+1} is the set of partial maps from $N_1 \to (\bigcup_{s \in \Sigma_1} N^{ar(s)} \times \bigcup_{j \leq i} \Theta_j)$.*

Definition 15. *A play of $\mathsf{G}(t, P)$ is a finite sequence $n_1 q_1 \theta_1, \dots, n_l q_l \theta_l$ of positions where each $n_i \in N$, each $q_i \in S$ and q_l is final and each $\theta_i \in \Theta_i$ is a look-up table. For the initial position n_1 is the root of the interpolation tree labelled @, $q_1 = [\,u\,]$ where u is the right term of P and θ_1 is the empty look-up table. Player $\forall$ loses the play if the final state is $[\,\exists\,]$, otherwise she wins the play.*

The game $\mathsf{G}(t, P)$ appeals to a finite set of states S comprising goal states $[\,r\,]$, $r \in \mathsf{R}$, and *final* states, $[\,\forall\,]$, winning for the refuter, and $[\,\exists\,]$, losing for the refuter. The central feature of a play of $\mathsf{G}(t, P)$, as depicted in Figure 2, is that repeatedly control may jump from a node of t to a node of w and back again. Therefore, as play proceeds, one needs an account of the meaning of free variables in subtrees. A free variable in a subtree of t (a subtree of w) is associated with a subtree of w (a subtree of t). This is the role of the look-up table $\theta_k \in \Theta_k$ at position $k \geq 1$. If n is labelled $\lambda y_1 \ldots y_m$ and $\theta_k(n)$ is defined then it is of the form $((n_1, \ldots, n_m), \theta_j)$ which tells us that any node m labelled y_i such that $n \downarrow m$ is associated with the subtree rooted at node n_i: that subtree may itself contain free variables, hence, the presence of a previous look-up table θ_j.

Current position is $n[r]\theta$. Next position by cases on label at node n:

1. $@$ then $n1[r]\theta'$ where $\theta' = \theta\{((n2), \theta)/n1\}$
2. $\lambda\overline{y}$ then $n1[r]\theta$
3. $a : \mathbf{0}$ if $r = a$ then $n[\,\exists\,]\theta$ else $n[\,\forall\,]\theta$
4. $f : (B_1, \ldots, B_k, \mathbf{0})$ if $r = fr_1 \ldots r_k$ then $\forall$ chooses $j \in \{1, \ldots, k\}$ and $nj\,[r_j]\theta$ else $n[\,\forall\,]\theta$
5. $y_j : \mathbf{0}$ if $m \downarrow n$ and $\theta(m) = ((m_1, \ldots, m_l), \theta')$ then $m_j\,[r]\theta'$
6. $y_j : (B_1, \ldots, B_k, \mathbf{0})$ if $m \downarrow n$ and $\theta(m) = ((m_1, \ldots, m_l), \theta')$ then $m_j\,[r]\theta''$ where $\theta'' = \theta'\{((n1, \ldots, nk), \theta)/m_j\}$

Fig. 3. Game moves

Definition 16. *If the current position in* $\mathsf{G}(t, P)$ *is* $n[r]\theta$ *and* $[r]$ *is not final then the next position is determined by a unique move in Figure 3 according to the label at node* n*.*

At the initial node labelled @, play proceeds to its first successor labelled λz and the look-up table is updated as its other successor is associated with λz. Later, if play reaches a node labelled z (bound by initial successor of root) then it jumps to the second successor of the root node. Standard updating notation is assumed: $\gamma\{((m1, \ldots, mk), \gamma')/n\}$ is the partial function similar to γ except that $\gamma(n) = ((m1, \ldots, mk), \gamma')$ where n will be labelled $\lambda y_1 \ldots y_k$ for some y. If play is at a node labelled $\lambda\overline{y}$, where $\overline{y}$ can be empty, then it descends to its successor. At a node labelled with the constant $a : \mathbf{0}$, the refuter loses if the goal state is $[\,a\,]$ and wins otherwise. At a node labelled with a constant f with arity more than 0, $\forall$ immediately wins if the goal state is not of the form $[\,fr_1 \ldots r_k\,]$. Otherwise $\forall$ chooses a successor j and play moves to its jth successor. If play is at node n labelled with variable $y_j : \mathbf{0}$ and $\theta(m) = ((m_1, \ldots, m_l), \theta')$ when $m \downarrow n$ then play jumps to m_j and θ' becomes the look-up table. If n is labelled $y_j : (B_1, \ldots, B_k, \mathbf{0})$ and $\theta(m) = ((m_1, \ldots, m_l), \theta')$ when $m \downarrow n$ then play jumps to m_j which is labelled $\lambda x_1 \ldots x_k$ for some x and the look-up table is θ' together with the association of $((n1, \ldots, nk), \theta)$ to m_j.

Definition 17. *Player* $\forall$ *loses the game* $\mathsf{G}(t, P)$ *if she loses every play of it and otherwise she wins the game.*

Lemma 1. *Player $\forall$ loses $\mathsf{G}(t, P)$ iff $t \models P$.*

Proof. Given $P : A$, $xw = u$ and $t : A$ either $t \models P$ or $t \not\models P$. Because the simply typed λ-calculus is strongly normalizing, it follows that there is an m such that tw reduces to normal form using at most m β-reductions (whatever the reduction strategy). For any position $n[r]\theta$ of a play of $\mathsf{G}(t, P)$ we say that it m-holds (m-fails) if $r = \exists$ ($r = \forall$) and when not final, by cases on the label at n(where look-up tables become delayed substitutions and we elide between nodes, subtrees and terms)

- @ then $n1n2 =_\beta r$ ($n1n2 \neq_\beta r$) and $n1n2$ normalizes with m β-reductions
- λ then $n1\theta =_\beta r$ ($n1\theta \neq_\beta r$) and $n1\theta$ normalizes with m β-reductions
- $\lambda y_1 \dots y_k$ then $n1\theta =_\beta r$ ($n1\theta \neq_\beta r$) and $n1\theta$ normalizes with $(m - k)$ β-reductions
- f then $n\theta =_\beta r$ ($n\theta \neq_\beta r$) and $n\theta$ normalizes with m β-reductions
- $y_j : \mathbf{0}$ if $n' \downarrow n$ and $\theta(n') = ((n_1, \dots, n_l), \theta')$ then $n_j\theta' =_\beta r$ ($n_j\theta' \neq_\beta r$) and $n_j\theta'$ normalizes with m β-reductions
- $y_j : (B_1, \dots, B_k, \mathbf{0})$ if $n' \downarrow n$ and $\theta(n') = ((n_1, \dots, n_l), \theta')$ then $t' =_\beta r$ ($t' \neq_\beta r$) where $t' = (n_j\theta')n1\theta \dots nk\theta$ and t' normalizes with m β-reductions

Initially, play is at n labelled @ with state [u] and the empty look-up table: therefore, as either $tm =_\beta u$ or $tm \neq_\beta u$ it follows that for some m, either $n[u]\theta_1$ m-holds or m-fails. The following invariants are easy to show by case analysis.

1. If $n[r]\theta$ m-holds (m-fails), n labels $\lambda y_1 \dots y_k$ and $n'[r']\theta'$ is the next position then it $(m - k)$-holds ($(m - k)$-fails)
2. If $n[r]\theta$ m-holds (m-fails), n labels λ and $n'[r']\theta'$ is the next position then it m-holds (m-fails)
3. If $n[r]\theta$ m-holds and n labels f and $n'[r']\theta'$ is any next position then it m'-holds for $m' \leq m$
4. If $n[r]\theta$ m-fails and n labels f then some next position $n'[r']\theta'$ m'-fails for some $m' \leq m$
5. If $n[r]\theta$ m-holds (m-fails) and n labels y_j and $n'[r']\theta'$ is the next next position then it m-holds (m-fails)

From these invariants it follows first that if a non-final position m-holds then any next position m'-holds for some $m' \leq m$ and second if a non-final position $n[r]\theta$ m-fails then there is a next position that m'-fails for some $m' \leq m$. Moreover, there cannot be an infinite sequence of positions (as the index m strictly decreases with a move at a node labelled $\lambda y_1 \dots y_k$, $k > 0$, and must be 0 at a node labelled with a constant $a : \mathbf{0}$). Therefore, the result follows. □

In the following we let $p \in \mathsf{G}(t, P)$ abbreviate that p is a position in some play of $\mathsf{G}(t, P)$. If such a position p is at a node labelled with a variable then we identify the earlier position at the node labelled with its binder when the value of that binder in the look-up table at p is defined.

Definition 18. *Assume $p_1 = n_1q_1\theta_1, \dots, p_l = n_lq_l\theta_l$ is a play of $\mathsf{G}(t, P)$ and n_j is labelled with a variable. Position p_i is a parent of p_j iff $n_i \downarrow n_j$ and $\theta_i(n_i) = \theta_j(n_i)$.*

Fact 5. *If $p \in \mathsf{G}(t, P)$ is at a node labelled with a variable then there is a unique $q \in \mathsf{G}(t, P)$ that is the parent of p.*

We now extend the notion of a successor in a tree to positions in a play.

Definition 19. *Assume $p_1 = n_1 q_1 \theta_1, \ldots, p_l = n_l q_l \theta_l$ is a play of $\mathsf{G}(t, P)$, node n_m is a successor of n_k (so, for some j, $n_m = n_k j$) and $1 \leq k < m < l$. Position p_m succeeds position p_k if either $m = k + 1$ or n_k is labelled @, or n_k is labelled with a variable and p_{k+1} is the parent of p_{m-1}.*

Proposition 6. *Assume $p_1 = n_1 q_1 \theta_1, \ldots, p_l = n_l q_l \theta_l$ is a play of $\mathsf{G}(t, P)$ and p_m is a position with $m < l$. There is a unique subsequence of positions $p_{i_1}, \ldots, p_{i_k}$ such that $i_1 = 1$, $i_k = m$ and for all $j : 1 \leq j < k$ position $p_{i_{j+1}}$ succeeds p_{i_j} and for any c if $n_{i_c} \downarrow n_{i_j}$ then p_{i_c} is the parent of p_{i_j}.*

Proof. Assume that $p_1 = n_1 q_1 \theta_1, \ldots, p_l = n_l q_l \theta_l$ is a play of $\mathsf{G}(t, P)$ and p_m is a position with $m < l$. Consider the branch of the interpolation tree from the root labelled @ to n_m. We now pick out the subsequence of positions at these nodes backwards starting with p_m at n_m. Suppose $p_{i_{j+1}}$ is given. If $n_{i_{j+1}}$ is labelled x_i, f or a then p_{i_j} is $p_{i_{j+1}-1}$. If n_{i_j} is labelled $\lambda\overline{y}$ and its immediate predecessor is f then p_{i_j} is also $p_{i_{j+1}-1}$. If n_{i_j} is labelled $\lambda\overline{y}$ and its immediate predecessor is x_i and p_l is the parent of $p_{i_{j+1}-1}$ then p_{i_j} is p_{l-1}. The argument that if $n_{i_c} \downarrow n_{i_j}$ then p_{i_c} is the parent of p_{i_j} is also straightforward. □

Definition 20. *If $p = n[r]\theta \in \mathsf{G}(t, P)$ and n is labelled y_j then its associated variable assumption, $V(p)$, is defined by induction on the type of y_j. If $y_j : \mathbf{0}$ then $V(p) = (y_j, r, \emptyset)$. If $y_j : (B_1, \ldots, B_k, \mathbf{0})$ and p' is the next position after p then $V(p) = (y_j, r, \Gamma)$ where $\Gamma = \{V(q) \mid q \in \mathsf{G}(t, P)$ and p' is the parent of $q\}$.*

Definition 21. *If $p = n[r]\theta \in \mathsf{G}(t, P)$ then $M(p)$ is the mode at node n associated with p defined by cases on the label at n (and which uses Definition 20). If @, f or a then $M(p) = (r, \emptyset)$. If y_j and $V(p) = (y_j, r, \Gamma)$ then $M(p) = (r, \Gamma)$. If $\lambda\overline{y}$ then $M(p) = (r, \Gamma)$ where $\Gamma = \{V(q) \mid q \in \mathsf{G}(t, P)$ and p is the parent of $q\}$.*

Theorem 2 is a corollary (via Lemma 1) of the following result.

Theorem 3. *Assume P is $xw = u$, Σ is the alphabet and A_P is the dependency Σ-tree automaton in Definition 13. For any canonical Σ-term t, $\forall$ loses $\mathsf{G}(t, P)$ iff A_P accepts the tree @tw.*

Proof. Assume $\forall$ loses $\mathsf{G}(t, P)$. We show that there is a successful run of A_P on @tw via Proposition 6 and Definition 21. More precisely, the successful run tree is built in such a way that for any of its nodes $(n, \alpha(r, \Gamma))$ there is a play $p_1 = n_1 q_1 \theta_1, \ldots, p_l = n_l q_l \theta_l$ of $\mathsf{G}(t, P)$ and a position p_m with $m < l$ such that if $p_{i_1}, \ldots, p_{i_k}$ is the subsequence identified in Proposition 6 then the branch from the root to $(n, \alpha(r, \Gamma))$ consists of nodes $n'_1, \ldots, n'_k$ where $n'_j = (n_{i_j}, M(p_{i_1}) \ldots M(p_{i_j}))$, $1 \leq j \leq k$. Initially this is true as $(n_1, (u, \emptyset))$

is the root node of the run tree when $n_1[u]\theta_1$ is the initial position (of any play). It is now an easy exercise to show that there is always an application of a transition rule of A_P of Definition 13 to a nonterminal node $(n, \alpha(r, \Gamma))$ that preserves this property.

For the other direction assume $\forall$ wins $\mathsf{G}(t, P)$ but there is a successful run of A_P on $@tw$. There is a winning play $p_1 = n_1[r_1]\theta_1, \ldots, p_l = n_l[r_l]\theta_l$ of $\mathsf{G}(t, P)$ for $\forall$. So, n_{l-1} is labelled $a : \mathbf{0}$ or $f : (B_1, \ldots, B_k, \mathbf{0})$ and $r_{l-1} \neq a$ or $r_{l-1} \neq f r_1 \ldots r_k$ because $r_l = \forall$. Let p_m be the earliest position in this play such that there are positions $p_{i_1}, \ldots, p_{i_k}$ of Proposition 6 such that there is a branch of the successful run tree of A_P on $@tw$ consisting of nodes $n'_1, \ldots, n'_{k-1}$ with $n'_{i_j} = (n_{i_j}, \alpha_j(r_{i_j}, \Gamma_j))$ for some α_j and Γ_j, $1 \leq j < k$ but no successor of n'_{k-1} of the form $(n_m, \alpha(r_m, \Gamma))$. We know that there is such a position p_m, $1 < m < l$, because the root of the run tree has the form $(n_1, (r_1, \emptyset))$ and by the transition rules 3 and 4 of Definition 13 there cannot be a node of a successful run tree $(n_{l-1}, \alpha(r_{l-1}, \Gamma))$ for any α and Γ. A case analysis on the label at node n_m shows that if there is such a position p_m then there is an even earlier position with this property which is a contradiction. $\square$

5 Conclusion

We introduced nondeterministic and alternating dependency tree automata for recognising terms with binding. Decidability of nonemptiness is shown for the nondeterministic automata. There are significant open questions for the alternating automata: are they more expressive than the nondeterministic automata and is their nonemptiness problem decidable? We also provided an application of the alternating automata to characterise solutions to a higher-order matching problem. We need to see if there are other applications of these automata.

To save space, we assumed that a right term u in an interpolation problem does not contain bound variables. We handle them as in [14,13] by including new corresponding constants which are not allowed to occur in solution terms. If u is $f(\lambda x_1 x_2 x_3.x_1 x_3)a$ then c_1, c_2 and c_3 are included where each c_i has the same type as x_i. Definition 11 is refined to only allow closed subterms of base type by replacing bound variables by their corresponding constants: for u above we include a, $c_1(c_3)$ and c_3. A new kind of variable assumption is included, a triple of the form (z_i, r, c) where c is one of the new constants and look-up tables are extended to include entries of the form $\theta_m(z) = c$. Transition rules for the automaton and the game moves are extended accordingly. For instance, in 4 of Figure 3 there is also the case when $r_j = \lambda x_1 \ldots x_m.r'$ and n_j is labelled $\lambda y_1 \ldots y_m$: so the next position is $nj[r'\{c_1/x_1, \ldots, c_m/x_m\}]\theta'$ where $\theta' = \theta\{c_1/y_1, \ldots, c_m/y_m\}$.

References

1. Alur, R., Madhusudan, P.: Adding nested structure to words. In: H. Ibarra, O., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 1–13. Springer, Heidelberg (2006)
2. Alur, R., Chaudhuri, S., Madhusudan, P.: Languages of nested trees. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 329–342. Springer, Heidelberg (2006)

3. Barendregt, H.: Lambda calculi with types. In: Abramsky, S., Gabbay, D., Maibaum, T. (eds.) Handbook of Logic in Computer Science, vol. 2, pp. 118–309. Oxford University Press, Oxford (1992)
4. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications. Draft Book (2002), http://l3ux02.univ-lille3.fr/tata/
5. Comon, H., Jurski, Y.: Higher-order matching and tree automata. In: Nielsen, M. (ed.) CSL 1997. LNCS, vol. 1414, pp. 157–176. Springer, Heidelberg (1998)
6. Joly, T.: The finitely generated types of the lambda calculus. In: Abramsky, S. (ed.) TLCA 2001. LNCS, vol. 2044, pp. 240–252. Springer, Heidelberg (2001)
7. Ong, C.-H.L.: On model-checking trees generated by higher-order recursion schemes. In: Procs. LICS 2006, pp. 81–90 (2006); (Longer version available from Ong's web page, 55 pages (preprint, 2006)
8. Padovani, V.: Decidability of fourth-order matching. Mathematical Structures in Computer Science, vol. 10(3), pp. 361–372 (2001)
9. Segoufin, L.: Automata and logics for words and trees over an infinite alphabet. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 41–57. Springer, Heidelberg (2006)
10. Schubert, A.: Linear interpolation for the higher-order matching problem. In: Bidoit, M., Dauchet, M. (eds.) CAAP 1997, FASE 1997, and TAPSOFT 1997. LNCS, vol. 1214, pp. 441–452. Springer, Heidelberg (1997)
11. Statman, R.: Completeness, invariance and λ-definability. The Journal of Symbolic Logic 47, 17–26 (1982)
12. Stirling, C.: Higher-order matching and games. In: Ong, L. (ed.) CSL 2005. LNCS, vol. 3634, pp. 119–134. Springer, Heidelberg (2005)
13. Stirling, C.: A game-theoretic approach to deciding higher-order matching. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 348–359. Springer, Heidelberg (2006)
14. Stirling, C.: Higher-order matching, games and automata. In: Procs. LICS 2007, pp. 326–335 (2007)
15. Støvring, K.: Higher-order beta matching with solutions in long beta-eta normal form. Nordic Journal of Computing 13, 117–126 (2006)

On Global Model Checking Trees Generated by Higher-Order Recursion Schemes

Christopher Broadbent and Luke Ong

Oxford University Computing Laboratory

Abstract. Higher-order recursion schemes are systems of rewrite rules on typed non-terminal symbols, which can be used to define infinite trees. The *Global Modal Mu-Calculus Model Checking Problem* takes as input such a recursion scheme together with a modal μ-calculus sentence and asks for a finite representation of the set of nodes in the tree generated by the scheme at which the sentence holds. Using a method that appeals to game semantics, we show that for an order-n recursion scheme, one can effectively construct a non-deterministic order-n collapsible pushdown automaton representing this set. The level of the automaton is strict in the sense that in general no non-deterministic order-$(n-1)$ automaton could do likewise (assuming the requisite hierarchy theorem). The question of determinisation is left open. As a corollary we can also construct an order-n collapsible pushdown automaton representing the constructible winning region of an order-n collapsible pushdown parity game.

Keywords: Recursion Scheme, Model Checking, Game Semantics, Collapsible Pushdown Automaton, Parity Game.

1 Introduction

Whilst *local* model checking asks whether a property holds at the root of a structure, a *global* model checking algorithm is designed to return a finite representation of the set of states in a structure at which a property holds.

Our own focus is on model checking modal μ-calculus properties of (possibly infinite) ranked trees generated by *higher-order recursion schemes*, which are systems of rewrite rules on typed non-terminals. A number of results exist concerning the local version [13,14] and it turns out that for an order-n recursion scheme the local problem is n-EXPTIME complete [14]. The computationally intensive part of our algorithm for the global result in fact consists of solving a local version of the problem. We have to compute the winning region of a finite parity game arising from Ong's method [14]. Algorithms for solving such games from a given node usually follow the global paradigm and compute the winning region in the process.

Owing to equivalences between recursion schemes and various flavours of higher-order pushdown automata (PDA) [13,10], the present work is very much related to computing winning regions of parity games played over the configuration graphs of such automata. Cachat and Serre independently showed that the winning regions of parity games over 1-PDA and prefix-recognizable graphs are regular [16,2]. Piterman and Vardi [15] have also presented a generalisation of automata-theoretic techniques used to solve the local problem over these graphs to obtain the same result. (Indeed we borrow an aspect of their method with what we call in the sequel *'the versatile automaton'*.) It was subsequently discovered by Carayol et al. that the winning regions of order-n pushdown parity games are regular and that the problem is n-EXPTIME complete [4]. As a corollary to this they show that the global model checking problem for order-n recursion schemes satisfying a syntactic constraint called *'safety'* can be solved in n-EXPTIME, with solution represented by a *deterministic* order-n pushdown automaton.

An analogous approach to general recursion schemes would require a regular characterisation of the winning region of a *collapsible pushdown* parity game [10], as provided by Serre[1]. The approach we consider here, however, does not go via collapsible pushdown parity games. Despite the difference in method, our final result is similar insofar as our algorithm represents the required set of tree nodes using an order-n collapsible pushdown automaton (CPDA). There is an unfortunate difference, however, in that our CPDA is *non-deterministic*. Even if this diminishes the practical utility of the output of our algorithm, our result nevertheless establishes that the μ-calculus definable node-sets of trees generated by order-n recursion schemes can themselves be generated by an order-n recursion scheme. In doing so we show how two different incarnations of a game-semantic approach to the Local Problem [14,10] can be merged.

As a corollary we are able to characterise the configurations with *constructible stacks* that are winning in a collapsible pushdown parity game using a CPDA. Constructible stacks are represented by sequences of stack operations that generate them. This resembles a similar result by Carayol and Slaats for (non-collapsible) PDA [5], although our version lacks the canonicity exhibited in *op. cit.*

An Outline Proof of the Local Problem. Space constraints limit the degree to which we can introduce apparatus from Ong's original paper [14] but we try to refer to a section of the *long version* for the reader interested in more details.

Fix a space of types formed from a single ground type o and the arrow-constructor $\rightarrow$. The order and arity of a type are given their standard definitions. An ***order-n recursion scheme*** ([14], Sec. 1.2) is a 5-tuple $\langle \Sigma, \mathcal{N}, \mathcal{V}, \mathcal{R}, \mathcal{S} \rangle$ where Σ is a finite ranked alphabet with a symbol of arity k given the order-1 type of arity k; $\mathcal{N}$ is a finite set of non-terminals assigned types of order no greater than n; $\mathcal{V}$ is a finite set of typed variables; $\mathcal{S} \in \mathcal{N}$ is a distinguished 'initial symbol' with type o and $\mathcal{R}$ is a finite set of rewrite rules of the form $F\zeta_1 \ldots \zeta_m \rightarrow t(\zeta_1, \ldots, \zeta_m)$ where $F \in \mathcal{N}$ and $\zeta_1, \ldots, \zeta_m \in \mathcal{V}$; $F\zeta_1, \ldots, \zeta_m$ has type o as does the term $t(\zeta_1, \ldots, \zeta_m)$, which is formed from variables $\zeta_1, \ldots, \zeta_m$, non-terminals from $\mathcal{N}$ and Σ-symbols. There should be precisely one rule for each non-terminal. The ***value-tree*** $[\![G]\!]$ defined by a recursion scheme G is the tree with nodes labelled in Σ that is the limit of the recursion scheme as it unfolds from $\mathcal{S}$ ([14], p. 7).

Given a μ-calculus sentence ϕ and a recursion scheme G, the local problem asks whether ϕ holds at the root of $[\![G]\!]$. Ong's proof of decidability for this [14] makes use of ideas from innocent game semantics [11] via the notion of ***traversal***. A traversal is a sequence of nodes obeying certain rules in an infinite lambda term $\lambda(G)$, called the

[1] Private communication with Olivier Serre, 7 October 2008.

computation tree, which represents the recursion scheme G ([14], Sec. 2). The manner in which a traversal hops around the computation tree can be viewed as both a form of evaluation of the scheme (linear head reduction) and a manifestation of its game-semantic denotation.

Thanks to Emerson and Jutla [7], we can convert the μ-calculus sentence ϕ to an *alternating parity tree automaton (APT)* $\mathcal{B}$, which we refer to as ***the property APT***, such that $\mathcal{B}$ has an accepting run-tree on the value-tree $[\![G]\!]$ generated by G just in case $[\![G]\!] \vDash \phi$.[2] We can map accepting run-trees to accepting ***traversal trees*** (and *vice versa*), where the latter allow $\mathcal{B}$ to jump over $\lambda(G)$ according to the rules for traversals ([14], Definition 2.11). We can then simulate such traversal trees using a ***traversal simulating APT*** $\mathcal{C}$ that reads $\lambda(G)$ in a 'normal' top-down manner ([14], Sec. 3). Since $\lambda(G)$ is regular, as witnessed by a finite graph $\mathrm{Gr}(G)$ ([14], p. 51), it can be decided whether $\mathcal{C}$ accepts $\lambda(G)$ and this gives the result.

Overview. Fix a ranked alphabet Σ, a tree-generating recursion scheme G and a μ-calculus sentence ϕ. Let $\mathcal{B}$ be the *property APT* associated with ϕ. The *Global Model-Checking Problem* asks for a finite representation of the set of nodes in the Σ-labelled tree $[\![G]\!]$ at which ϕ holds. We explicate a method that, given an order-n recursion scheme, constructs an n-CPDA word acceptor that accepts precisely these nodes, where nodes are represented in the standard way as strings over a 'directions alphabet'.

We actually establish a slightly stronger result than we need. We provide a finite representation of the set of ordered pairs (q, α), where q is a state of $\mathcal{B}$ and α is a node of $[\![G]\!]$, such that $\mathcal{B}$ accepts the subtree of $[\![G]\!]$ rooted at α starting from state q. The solution to the Global Model-Checking Problem for G and ϕ is then provided by restricting this set to those pairs of the form (q_0, α), where q_0 is the initial state of $\mathcal{B}$.

The construction begins with what we describe as the *versatile property APT*, $\mathcal{B}^{\perp}$, which is able to navigate to an arbitrary node in $[\![G]\!]$ before proceeding to adopt the behaviour of $\mathcal{B}$ starting at an arbitrary state q. Since $\mathcal{B}^{\perp}$ is just an ordinary APT, there exists a *traversal-simulating* APT $\mathcal{C}^{\perp}$ for $\mathcal{B}^{\perp}$. We can thus move on to consider the *finite* parity game $\mathcal{G}_{G,\mathcal{C}^{\perp}}$ induced by $\mathcal{C}^{\perp}$ and the computation graph $\mathrm{Gr}(G)$ of G. The two players of parity games are named Éloïse and Abelard. Éloïse can be viewed as trying to establish a μ-calculus formula whilst Abelard is trying to refute it. We can use a standard algorithm to find the winning region of $\mathcal{G}_{G,\mathcal{C}^{\perp}}$ and thereby label with a symbol 'W' the nodes of $\mathcal{G}_{G,\mathcal{C}^{\perp}}$ from which Éloïse has a winning strategy. The annotated graph is called $\mathcal{G}^{\mathcal{W}}_{G,\mathcal{C}^{\perp}}$.

Since $\mathcal{G}^{\mathcal{W}}_{G,\mathcal{C}^{\perp}}$ is induced (in part) by $\mathrm{Gr}(G)$, it makes sense to speak of traversals over $\mathcal{G}^{\mathcal{W}}_{G,\mathcal{C}^{\perp}}$. Consider the set of traversals of $\mathcal{G}^{\mathcal{W}}_{G,\mathcal{C}^{\perp}}$ travelling only over nodes labelled with W and halting at a node corresponding to a point where $\mathcal{B}^{\perp}$ starts to simulate $\mathcal{B}$ from state q at node n of the tree. It turns out that this set, when projected to the Σ-labelled nodes, corresponds to the set of ordered pairs (q, α) that we want to finitely represent. Since we can program an n-CPDA to navigate traversals of $\mathrm{Gr}(G)$ [10], we can also program it to navigate the traversals of $\mathcal{G}^{\mathcal{W}}_{G,\mathcal{C}^{\perp}}$ in the set. This provides the requisite n-CPDA word acceptor.

[2] For an introduction to the modal μ-calculus and parity automata / games we direct the reader to Bradfield and Stirling's survey [1].

2 The Versatile Property APT and Its Simulation

By convention the nodes of a (ranked and ordered) Σ-labelled tree, $T : dom(T) \longrightarrow \Sigma$ (say), are represented in the standard way by strings in Dir^* where $\text{Dir} = \mathbb{N}$, so that $dom(T) \subseteq \text{Dir}^*$; elements of $\text{Dir} \cup \{\epsilon\}$ (ϵ the empty string) are referred to as ***directions***. Thus the label of a node α is $T(\alpha)$ (e.g. $[\![G]\!](\alpha)$ means the label of the node α in a value-tree $[\![G]\!]$). A *path* p in a tree is viewed as a sequence of nodes such that the successor of an element of a sequence is its child. The *trace* $\text{trace}(p)$ of a path $p = (p_i)_{i \in I}$ is the sequence $(T(p_i))_{i \in I}$. For a node r of a tree T we write T_r for the maximal subtree of T rooted at r.

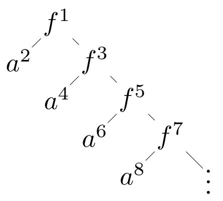


Fig. 1. A Value Tree

An APT that operates over a Σ-labelled tree T is a 5-tuple $\langle \Sigma, Q, \delta, q_0, \Omega \rangle$ consisting of a finite-set Q of control-states, transition function $\delta : (\Sigma \times Q) \to B^+((\text{Dir} \cup \{\epsilon\}) \times Q)$ (where $B^+(S)$ is the set of positive boolean formulae with set of atoms S), initial state $q_0 \in Q$ and priority function $\Omega : Q \to \mathbb{N}$. Whilst reading a node u of T in state q the automaton will pick a *minimal* set $S \subseteq ((\text{Dir} \cup \{\epsilon\}) \times Q)$ satisfying $\delta(T(u), q)$, and for each $(i, q') \in S$ will spawn an automaton in state q' reading the ith child of u, the ϵ-child of a node being itself. A ***run-tree*** of the APT is an unranked $(dom(T) \times Q)$-labelled tree representing such a branching run starting in state q_0 at the root of T. It is deemed accepting if the Q-projection $q_1\ q_2\ \ldots$ of the trace of every path satisfies ***the parity condition*** meaning that $\max(\{\Omega(q) : q \in \inf(q_1\ q_2\ \ldots)\})$ is even, where $\inf(\sigma)$ is the set of states occurring infinitely often in σ.

Suppose that the property APT $\mathcal{B}$ has initial state q_0 so that $\mathcal{B} = \langle \Sigma, Q, \delta, q_0, \Omega \rangle$. A ***template for an APT*** is a quadruple $\mathfrak{B} = \langle \Sigma, Q, \delta, \Omega \rangle$ with Q a finite set such that for any $q \in Q$ it is the case that $\mathcal{B}_q = \langle \Sigma, Q, \delta, q, \Omega \rangle$ is an APT. We may view $\mathfrak{B}$ as representing the family of automata: $\mathfrak{B} = \{ \mathcal{B}_q : q \in Q \}$.

Consider a ranked and ordered tree T. It is possible to construct an automaton $\mathcal{B}_\perp$ that can behave as any member of $\mathfrak{B}$ acting on any ranked and ordered subtree of T. We call this automaton *the versatile property APT for* $\mathfrak{B}$. The versatile APT traverses the tree T starting at its root whilst in a kind of 'nascent state' $\perp$. Once it reaches the desired node r of the tree, it switches into the required state q and starts behaving as though it were $\mathcal{B}_q$. We call this point *q-initialisation*.

Definition 1. *Let $\mathfrak{B} = \langle \Sigma, Q, \delta, \Omega \rangle$ be a template for an APT. The* ***versatile automaton*** *$\mathcal{B}^\perp$ for $\mathfrak{B}$ is the APT $\mathcal{B}^\perp$ given by:*

$$\mathcal{B}^\perp = \left\langle \Sigma, Q \uplus \{\perp\}, \delta^\perp, \perp, \Omega^\perp \right\rangle$$

where $\delta^\perp$ extends δ by the rule: $\delta^\perp : (\perp, f) \mapsto \bigvee_{1 \le i \le \text{ar}(f)} (i, \perp) \vee \bigvee_{q \in Q} (\epsilon, q)$ and $\Omega^\perp$ extends Ω with $\Omega^\perp(\perp) := -1$.

So the APT $\mathcal{B}^\perp$ has an 'initialisation phase' during which it is in state $\perp$.

Definition 2. *Let t be a run-tree of the versatile APT $\mathcal{B}^\perp$ on a Σ-labelled tree T. Let $t^\perp$ be the unique path in t consisting of precisely the nodes associated with $\perp$.*

*Let p be the unique path in T corresponding to the path in t of the form $t^{\perp}\ \beta$ where β is a node in t with label (α, q) such that $q \in Q$, the state space of the template $\mathfrak{B}$ (i.e. $q \neq \perp$). We then say that q-**initialization occurs at (the path)** p or **at (the node)** β. We call the path $t^{\perp}$ **the initialisation phase** of the automaton (during the run t).*

The following lemma summarises the significance of $\mathcal{B}^{\perp}$.

Lemma 1. *Let $\mathcal{B}^{\perp}$ be a versatile automaton and let T be a Σ-labelled tree. Given a state q of $\mathfrak{B}$ and a node r of T, it is the case that $\mathcal{B}_q$ accepts T_r if and only if $\mathcal{B}^{\perp}$ has an accepting run-tree on T with q-initialisation taking place at r.*

Example 1. We use the following automaton as our working example. It acts on trees with nodes labelled by f and a with arities 2 and 0 respectively. It has state space $\{q_0, q_1, q_2\}$ each of which is given priority 2.

$$(f, q_0) \mapsto (1, q_1) \wedge (2, q_1) \qquad\qquad (a,\ \_\ ) \mapsto \mathrm{t}$$
$$(f, q_1) \mapsto (1, q_1) \wedge (1, q_2) \wedge (2, q_1)$$

We additionally use as an example the recursion scheme with initial non-terminal $\mathcal{S} : o$, non-terminal $F : (o \to o) \to o$ and rules: $\mathcal{S} \to F(fa)$ and $F\phi \to \phi(F(fa))$. The scheme's value tree and computation tree are illustrated in Figures 1 and 2 respectively.

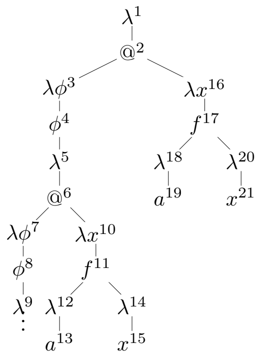


Fig. 2. A Computation Tree

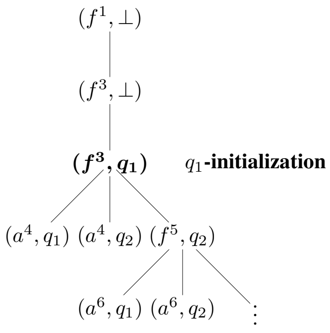


Fig. 3. A Run-Tree of a Versatile APT

Example 2. The versatile APT for the property APT in Example 1 has state space $\{q_0, q_1, q_2\} \cup \{\perp\}$ with $\perp$ the initial state. All states of the form q_i have priority 2 but $\perp$ has priority -1. Its transition function is given by:

$$(f, q_0) \mapsto (1, q_1) \wedge (2, q_1) \qquad (f, \perp) \mapsto (1, \perp) \vee (2, \perp) \vee \bigvee_{0 \le i \le 2} (\epsilon, q_i)$$
$$(f, q_1) \mapsto (1, q_1) \wedge (1, q_2) \wedge (2, q_1) \qquad (a,\ q_{\_}) \mapsto \mathbf{t}$$

with $\mathbf{t}$ the positive boolean formula that is always true (i.e. the empty conjunction) and $\mathbf{f}$ is that which is always false (i.e. the empty disjunction).

A run-tree of this versatile APT on the value tree in Figure 1 is given in Figure 3.

Traversals and the Versatile APT. Now consider a Σ-labelled tree $[\![G]\!]$ generated by some higher-order recursion scheme G. Let us fix a versatile APT $\mathcal{B}^{\perp}$ that can run on Σ-labelled trees.

We now make use of the notions of *traversals* and *the traversal tree* of an APT on the computation tree $\lambda(G)$ of the recursion scheme. We speak interchangeably of traversals over the computation graph $\mathrm{Gr}(G)$, which unravels to form $\lambda(G)$. The *Path-Traversal Correspondence Theorem* from the proof of the decidability of the Local Model-Checking Problem ([14], Thm. 7) ensures that the following definition is well-defined, which for any node α in $[\![G]\!]$ gives the corresponding node α^{Λ} in $\lambda(G)$:

Definition 3. *Let $\alpha = a_1 \dots a_m$ (with $a_i \in \mathrm{Dir}$ for $1 \leq i \leq m$) be a node in $[\![G]\!]$. Let t_α be a traversal of $\lambda(G)$ and for $1 \leq i \leq m+1$ let us name v_i the i-th occurrence of a terminal-labelled node in t_α. Suppose that v_1 bears the same label as the root of $[\![G]\!]$ and for $i \geq 2$, $[\![G]\!](a_1 \dots a_{i-1}) = [\![\lambda(G)]\!](v_i)$ (where $a_1 \dots a_{i-1}$ is a node in $[\![G]\!]$). Further assume that for $1 \leq i \leq m$ the successor of v_{i-1} in t_α is its a_i-th child in $\lambda(G)$. We define α^{Λ} to be v_{m+1}.*

Note that α^{Λ} also makes sense when speaking of traversals over $\mathrm{Gr}(G)$ except that in this case it should be viewed as a *particular instance* of a $\mathrm{Gr}(G)$-node in a traversal.

We can also speak of q-initialisation in a traversal tree of $\mathcal{B}^{\perp}$ in a completely analogous way – q-initialisation is the point in the traversal at which the automaton switches from being in state $\perp$ to being in state q. We illustrate in Figure 4 a traversal tree of the versatile APT in Example 2 on the computation tree in Figure 2.

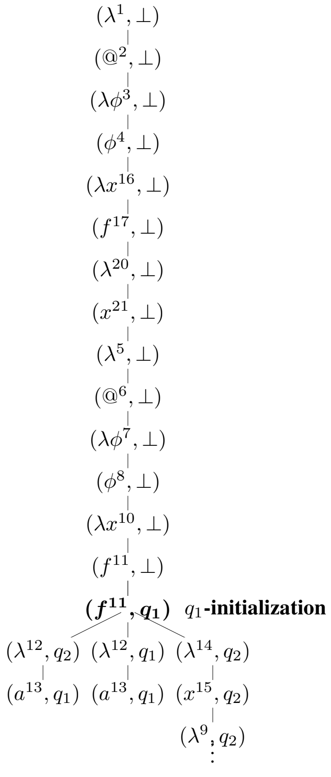
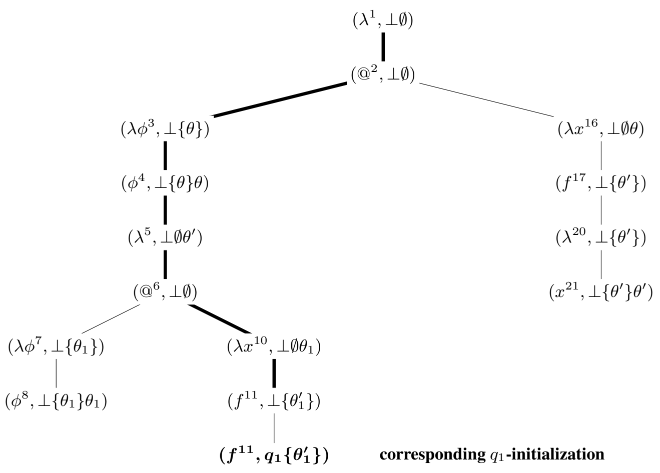


Fig. 4. A Traversal Tree of a Versatile APT

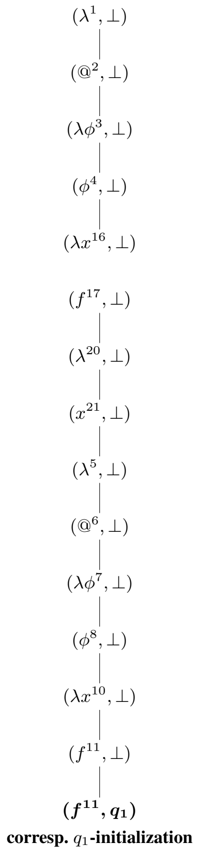
Now we make use of the ***traversal-simulating APT*** $\mathcal{C}^{\perp}$ associated with $\mathcal{B}^{\perp}$ (in the sense of Ong ([14], sec. 3)). The essential property of $\mathcal{C}^{\perp}$ is that it is possible to convert an accepting *traversal tree* of $\mathcal{B}^{\perp}$ on $\lambda(G)$ into an accepting *run-tree* of $\mathcal{C}^{\perp}$ on $\lambda(G)$ and conversely an accepting *run-tree* of $\mathcal{C}^{\perp}$ can be converted into an accepting *traversal tree* of $\mathcal{B}^{\perp}$.

Each state s of $\mathcal{C}^{\perp}$ includes a component $\mathrm{sim}(s)$ which is the state of $\mathcal{B}^{\perp}$ that is being simulated. Similarly for a sequence of states $\sigma = (s_i)_{i \in X}$ we write $\mathrm{sim}(\sigma)$ to denote the sequence $(\mathrm{sim}(s_i))_{i \in X}$. In contrast to $\mathcal{B}^{\perp}$, however, the transition function of $\mathcal{C}^{\perp}$ maps $\perp$-states to boolean formulae that are not necessarily disjunctions so that it can both guess how the simulated traversal will evolve as well as later verify these guesses. As a result q-initialization cannot be considered unique for $\mathcal{C}^{\perp}$ and so we adjust the definition appropriately.

Definition 4. *Let t be a run-tree of $\mathcal{C}^{\perp}$ on $\lambda(G)$ and q be a state of $\mathfrak{B}$. For a* finite *path p in $\lambda(G)$ (starting at the root), we say that* ***an instance of q-initialisation occurs at*** *p if there exists some path $t^{\perp}$ in t such that $\mathrm{sim}(\mathrm{trace}(t^{\perp}))$ consists only of $\perp$ and β is some node such that $t^{\perp}\beta$ is also a path in t with the projection of $\mathrm{trace}(t^{\perp})$ onto the*


$(\lambda^1, \bot\emptyset)$
$(@^2, \bot\emptyset)$
$(\lambda\phi^3, \bot\{\theta\})$
$(\phi^4, \bot\{\theta\}\theta)$
$(\lambda^5, \bot\emptyset\theta')$
$(@^6, \bot\emptyset)$
$(\lambda\phi^7, \bot\{\theta_1\})$
$(\phi^8, \bot\{\theta_1\}\theta_1)$
$(\lambda x^{10}, \bot\emptyset\theta_1)$
$(f^{11}, \bot\{\theta_1'\})$
$(f^{11}, q_1\{\theta_1'\})$
corresponding q_1-initialization
$(\lambda x^{16}, \bot\emptyset\theta)$
$(f^{17}, \bot\{\theta'\})$
$(\lambda^{20}, \bot\{\theta'\})$
$(x^{21}, \bot\{\theta'\}\theta')$

Part of a run-tree of the traversal-simulating APT on $\lambda(G)$. The states of the traversal-simulating automaton are either of the form $q\theta$ or $qS\theta$ where q represents the property APT state being simulated and S and θ describe how the automaton has guessed the traversal being simulated should evolve ([14], Sec. 3). The fragment of the run-tree illustrated here is precisely the fragment that corresponds to the fragment of the traversal tree illustrated to the right.



The part of the traversal tree up to the point of q-initialisation.

The traversal associated with the two diagrams above is:

$$\begin{tikzcd}[column sep=tiny]
\lambda^1 & @^2 & \lambda\phi^3 \arrow[l, bend right, "0"'] & \phi^4 \arrow[l, bend right, "1"'] & \lambda x^{16} \arrow[lll, bend right, "1"'] & f^{17} & \lambda^{20} \arrow[l, bend right, "2"'] & x^{21} \arrow[lll, bend right, "1"'] & \lambda^5 \arrow[lllll, bend right, "1"'] & @^6 & \lambda\phi^7 \arrow[l, bend right, "0"'] & \phi^8 \arrow[l, bend right, "1"'] & \lambda x^{10} \arrow[lll, bend right, "1"'] & f^{11}
\end{tikzcd}$$

which has P-View:

$$\begin{tikzcd}[column sep=tiny]
\lambda^1 & @^2 & \lambda\phi^3 \arrow[l, bend right, "0"'] & \phi^4 \arrow[l, bend right, "1"'] & \lambda^5 \arrow[l, bend right, "1"'] & @^6 & \lambda x^{10} \arrow[l, bend right, "1"'] & f^{11}
\end{tikzcd} .$$

The path corresponding to this P-view in the traversal-simulating APT run-tree has been highlighted.

Fig. 5. An illustration of Lemma 2

nodes of $\lambda(G)$ equal to p and β labelled (s, α) where α is the last node in p and s is a state of $\mathcal{C}^\perp$ such that $\mathrm{sim}(s) = q$.

Given a sequence of nodes $s := (\alpha_1, p_1) \ldots (\alpha_m, p_m)$ in $\mathrm{Gr}(G) \times Q_{\mathcal{C}^\perp}$ let us write $\pi_{\mathrm{Gr}(G)}(s)$ to mean $\alpha_1, \ldots, \alpha_m$ and $\pi^{\mathrm{NDup}}_{\mathrm{Gr}(G)}(s)$ to mean the largest subsequence of $\pi_{\mathrm{Gr}(G)}(s)$ whose adjacent elements are pairwise distinct. Given a traversal t we write $\ulcorner t \urcorner$ to denote its *P-view* ([14], Def. 2.5) which is a subsequence of t of a certain game-semantic significance.

Lemma 2. *Let α be a node in $[\![G]\!]$ and q be a state of $\mathfrak{B}$. The following are equivalent:*

1. *Property APT $\mathcal{B}_q \in \mathfrak{B}$ accepts $[\![G]\!]_\alpha$.*
2. *The APT $\mathcal{B}^\perp$ has an accepting traversal tree t with q-initialisation occurring at α^Λ.*
3. *There exists a finite subtree T_α of a run-tree of $\mathcal{C}^\perp$, all of whose nodes are associated with states simulating $\perp$. For some traversal t_α on $\lambda(G)$ ending in α^Λ it is the case that*

$$\{\pi^{\mathrm{NDup}}_{\mathrm{Gr}(G)}(p) : p \text{ is a path in } T_\alpha\} = \{\ulcorner \pi_{\mathrm{Gr}(G)}(s) \urcorner : s \text{ is an initial segment of } t_\alpha\}\ .$$

In particular there is a maximal branch b in T_α such that $\ulcorner \pi^{\mathrm{NDup}}_{\mathrm{Gr}(G)}(b) \urcorner = \ulcorner t_\alpha \urcorner$. Moreover, T_α can be extended to an accepting run-tree of $\mathcal{C}^\perp$ such that q-initialisation occurs on the tip of b.

Proof. The equivalence of 1 and 2 is given by the Path-Traversal Correspondence Theorem of Ong ([14], Thm 7). The equivalence of 2 and 3 is given by the inter-translations between traversal trees of $\mathcal{B}^\perp$ and run-trees of $\mathcal{C}^\perp$ given in *op. cit.* (Sec 4 and 5) together with the result from that same paper ([14], Prop. 6) stating that there is a $1-1$ correspondence between P-views of traversals and paths in the computation tree. This is illustrated in Figure 5. □

3 The Versatile Parity Game

We now move on to consider the parity game induced by $\mathcal{C}^\perp$ acting on $\mathrm{Gr}(G)$. Let us call this parity game the *versatile parity game* $\mathcal{G}_{G,\mathcal{C}^\perp}$. To retain a simple description, we assume that $\mathcal{C}^\perp$ is presented in such a form that the image of its transition function consists of just pure disjunctions and pure conjunctions. Let us write

$$\mathcal{C}^\perp \quad := \quad \langle \Lambda_G, Q_{\mathcal{C}^\perp}, \delta_{\mathcal{C}^\perp}, p_{0\mathcal{C}^\perp}, \Omega_{\mathcal{C}^\perp} \rangle$$

for the traversal-simulating automaton in such a form. This means that every element in the image of $\delta_{\mathcal{C}^\perp}$ can be written as $\bigwedge_{i \in I}(d_i, p_i)$ or $\bigvee_{i \in I}(d_i, p_i)$.

Definition 5. *Let N be the set of nodes in* $\mathrm{Gr}(G)$ *and let $Q_{\mathcal{C}^\perp}$ be the state space of $\mathcal{C}^\perp$. The **versatile parity game** is the parity game played on a directed graph with nodes in $N \times Q_{\mathcal{C}^\perp}$ such that:*

1. *The start node of the game is $(n_0, p_{0\mathcal{C}^\perp})$ where n_0 is the root of* $\mathrm{Gr}(G)$.
2. *There is an edge from (n,p) to (n',p') just in case*

$$\delta_{\mathcal{C}^\perp}(l,p) = \bigwedge_{i\in I}(d_i,p_i) \quad \textit{or} \quad \delta_{\mathcal{C}^\perp}(l,p) = \bigvee_{i\in I}(d_i,p_i)$$

where l is the label of n and for some $i \in I$, $p_i = p'$ and n' is the d_ith child of n. Note that we may have $d_i = \epsilon$ (the automaton does not move in the tree), in which case $n' = n$.
3. *A game node (n,p) is owned by Éloïse if it is mapped by $\delta_{\mathcal{C}^\perp}$ onto a $\bigvee$-formula and is owned by Abelard if it is mapped onto a $\bigwedge$-formula.*
4. *The only nodes in the game are those reachable from its start node.*
5. *The priority of a node (n,p) is $\Omega_{\mathcal{C}^\perp}(p)$.*

We write $\mathcal{G}_{G,\mathcal{C}^\perp}$ to denote this parity game and let us write $\mathbf{legalMove}((n,p),(n',p'))$ *if there is an edge from (n,p) to (n',p').*

We refer to a run-tree of $\mathcal{C}^\perp$, whose nodes associated with a $\bigvee$-state must have a unique child, as ***strategies for Éloïse*** in the game $\mathcal{G}_{G,\mathcal{C}^\perp}$. Such a strategy is termed ***winning*** just in case it is an accepting run-tree. A *finite* subtree of a strategy is called a ***partial strategy***.

By the definition of a traversal-simulating APT we may assume the following w.l.o.g:

Lemma 3

1. *Suppose that (n,p) is a node in $\mathcal{G}_{G,\mathcal{C}^\perp}$ such that the label of n is either an @ symbol or a variable ϕ. It is then the case that (n,p) is owned by Abelard and $n' \neq n$ for* every *(n',p') such that* $\mathbf{legalMove}((n,p),(n',p'))$ *if and only if there exists* some *(n',p') such that $n \neq n'$ and* $\mathbf{legalMove}((n,p),(n',p'))$.
2. *A game node (n,p) with n labelled by a terminal $f \in \Sigma$ and* $\mathrm{sim}(p) = \perp$ *is owned by Éloïse. The successor nodes of (n,p) include nodes of the form:*
   (a) (n,p_q) *with* $\mathrm{sim}(p_q) = q$ *for each $q \in Q$, the state space of $\mathcal{B}$.*
   (b) (n_i,p_i) *for* $1 \le i \le \mathrm{ar}(f)$ *where n_i is the ith child of n and* $\mathrm{sim}(p_i) = \perp$.
3. *Any node (n,p) in $\mathcal{G}_{G,\mathcal{C}^\perp}$ such that the label of n is a λ-node is owned by Éloïse.*

We can extend the notion of traversal on $\mathrm{Gr}(G)$ (or $\lambda(G)$) to $\mathcal{G}_{G,\mathcal{C}^\perp}$. Such traversals must respect the edge-relation of $\mathcal{G}_{G,\mathcal{C}^\perp}$ in the sense that they could be 'reassembled' into a tree embeddable in $\mathcal{G}_{G,\mathcal{C}^\perp}$.

Definition 6. *Consider a finite sequence of nodes $(n_1,p_1),\ldots,(n_m,p_m)$ in $\mathcal{G}_{G,\mathcal{C}^\perp}$ such that an element (n_i,p_i) might be endowed with an integer labelled pointer to an element (n_j,p_j) for $1 \le j < i$. We say that such a sequence is a **traversal of** $\mathcal{G}_{G,\mathcal{C}^\perp}$ just in case all of the following conditions hold:*

1. *(n_1,p_1) is the initial node of $\mathcal{G}_{G,\mathcal{C}^\perp}$ (so n_1 will have label λ).*
2. *The sequence $n_1,\ldots,n_m$ together with pointers is a traversal of* $\mathrm{Gr}(G)$, *which we refer to as the* underlying traversal.

3. *Suppose that the traversal includes an instance of*

$$\begin{tikzcd} \ldots (n,p) & \ldots & (n',p') \arrow[ll, bend right, "i"'] & \ldots \end{tikzcd}$$

where n' is a λ-node, or $\ldots$ (n,p) (n',p') $\ldots$ where n is a λ-node and (n',p') may or may not source a pointer.

We require that there is a path from (n,p) to (n',p') in $\mathcal{G}_{G,\mathcal{C}^\perp}$. Note that this path will necessarily be of the form

$$(n,p),\ (s_1,p_1),\ \ldots,\ (s_l,p_l),\ (s_{l+1},p_{l+1}),\ \ldots,\ (s_k,p_k),\ (n',p')$$

for some $l,k \in \mathbb{N}$ such that for all $1 \le i \le l$ we have the label of s_i being the label of n and for all $l+1 \le j \le k$ we have the label of s_j being the label for n'.

4. *Every occurrence of (n_i,p_i) such that n_i is an @ or a variable node is owned by Abelard.*

Our definition of traversal respects the rules of $\mathcal{G}_{G,\mathcal{C}^\perp}$, equivalently the transition function of $\mathcal{C}^\perp$, in the following sense:

Lemma 4. *For every traversal t of $\mathcal{G}_{G,\mathcal{C}^\perp}$ there exists a partial strategy T for Éloïse in $\mathcal{G}_{G,\mathcal{C}^\perp}$ such that*

$$\{\pi^{\mathrm{NDup}}_{\mathrm{Gr}(G)}(\mathrm{trace}(r)) \;:\; r \text{ is a path in } T\} = \{\ulcorner \pi_{\mathrm{Gr}(G)}(s)\urcorner \;:\; s \text{ is an initial segment of } t\}\ .$$

Traversals of $\mathcal{G}_{G,\mathcal{C}^\perp}$ consisting of nothing but nodes (n,p) with $\mathrm{sim}(p) = \perp$ are particularly pleasant because nodes associated with a terminal $f \in \Sigma$ never occur in immediate succession; they also allow access to arbitrary children of the f labelled node in $\mathrm{Gr}(G)$. We are also interested in such traversals that then finish with a node (n,p) with $\mathrm{sim}(p) = q$ for $q \in Q$, the state space of $\mathcal{B}$.

Definition 7. *A $\perp$-**traversal** of $\mathcal{G}_{G,\mathcal{C}^\perp}$ is a traversal of $\mathcal{G}_{G,\mathcal{C}^\perp}$ consisting entirely of nodes (n,p) that satisfy $\mathrm{sim}(p) = \perp$. If q is a state of $\mathcal{B}$, then a q-**tipped traversal** of $\mathcal{G}_{G,\mathcal{C}^\perp}$ is a traversal of the form $s\,(n,p)$ where s is a $\perp$-traversal but $\mathrm{sim}(p) = q$.*

Using known algorithms (such as Jurdziński's [12]) we can compute the winning region for Éloïse of finite parity games. We may thus effectively annotate with the symbol 'W' the states of $\mathcal{G}_{G,\mathcal{C}^\perp}$ from which Éloïse has a winning strategy. Let us write $\mathcal{G}^{\mathcal{W}}_{G,\mathcal{C}^\perp}$ for this annotated game and refer to it as the ***decorated game***. We interchangeably refer to traversals as being over $\mathcal{G}_{G,\mathcal{C}^\perp}$ and $\mathcal{G}^{\mathcal{W}}_{G,\mathcal{C}^\perp}$. A traversal of $\mathcal{G}^{\mathcal{W}}_{G,\mathcal{C}^\perp}$ containing only nodes annotated with W is referred to as a ***winning traversal***. In particular, the following lemma is useful to us and comes as a corollary to Lemmas 3 and 4 together with the fact that a partial strategy labelled everywhere with W can be extended to a winning strategy:

Lemma 5. *Let α be a node in $[\![G]\!]$ and q a state of the APT template $\mathfrak{B}$, and let t_α be a* winning *q-tipped traversal t_α in $\mathcal{G}^{\mathcal{W}}_{G,\mathcal{C}^\perp}$ whose final element is of the form (α^Λ,p) (and $\mathrm{sim}(p) = q$). There is a minimal partial strategy T_α for Éloïse on $\mathcal{G}^{\mathcal{W}}_{G,\mathcal{C}^\perp}$* that can be extended to a winning strategy (accepting run-tree of $\mathcal{C}^\perp$ on $\lambda(G)$) *satisfying*

$$\{\pi^{\mathrm{NDup}}_{\mathrm{Gr}(G)}(\mathrm{trace}(r)) \ : \ r \textit{ is a path in } T_\alpha\} \ = \ \{\ulcorner \pi_{\mathrm{Gr}(G)}(s) \urcorner \ : \ s \textit{ initial segment of } t_\alpha\} \ .$$

In particular there is a maximal branch b in T_α such that

$$\pi^{\mathrm{NDup}}_{\mathrm{Gr}(G)}(\mathrm{trace}(b)) = \ulcorner \pi_{\mathrm{Gr}(G)}(t_\alpha) \urcorner$$

with the tip of b being the sole node to be given a label (n, p) where $\mathrm{sim}(p) = q$.

Note that α^Λ is well-defined as used above due to the second observation in Lemma 3, which ensures that during the initialization phase of a traversal of $\mathcal{G}_{G,\mathcal{C}^\perp}$ one can leave a terminal node in any direction and the only time at which an ϵ-transition may be made at a terminal node is to q-initialize.

We now make the following claim:

Lemma 6. *Let α be a node in $[\![G]\!]$ and q a state of the property APT template $\mathfrak{B}$. TFAE:*

1. *Property APT $\mathcal{B}_q \in \mathfrak{B}$ accepts $[\![G]\!]_\alpha$.*
2. *There exists a winning q-tipped traversal in $\mathcal{G}^{\mathcal{W}}_{G,\mathcal{C}^\perp}$ whose final element is of the form (α^Λ, p) (and $\mathrm{sim}(p) = q$).*

The proof from 2 to 1 just consists of combining Lemmas 5 and 2. To go in the other direction, we use the second equivalence in Lemma 5 and then use Ong's construction of an accepting run tree of $\mathcal{C}^\perp$ from the accepting traversal tree of $\mathcal{B}^\perp$ ([14], Sec. 5). We appeal to observations concerning @ and variable nodes in $\mathcal{G}^{\mathcal{W}}_{G,\mathcal{C}^\perp}$ made in Lemma 3 to ensure that Abelard owns the @ and variable elements of the q-tipped traversal.

4 Construction of an n-CPDA Recogniser

Let us formally consider what it means to have an automaton as a solution to the Global Model-Checking Problem for a tree generated by a higher-order recursion scheme.

Definition 8. *Let $\mathfrak{B}$ be a template for an APT with state space $Q_{\mathfrak{B}}$ and let G be a higher-order recursion scheme. Let $n \in \mathbb{N}$ be the maximal rank of any terminal occurring in G and let $\mathrm{Dir}(\Sigma) = \{1, \ldots, n\}$ be the corresponding set of directions (so that nodes in $[\![G]\!]$ are denoted by elements in $\mathrm{Dir}(\Sigma)^*$).*

Now let $\mathcal{A}$ be an automaton (of any type) that reads (finite) words over the alphabet $\mathrm{Dir}(\Sigma)$ with a finite state-set $Q_{\mathcal{A}}$ (and possibly additional memory of some kind such as a stack). We say that $\mathcal{A}$ is an ***automaton-solution to the Global Model Checking Problem*** *(GMCP) for (the tree generated by) G with respect to $\mathfrak{B}$ just in case it can be endowed with a map $\mathcal{Q} : Q_{\mathcal{A}} \longrightarrow Q_{\mathcal{B}} \cup \{\perp\}$ such that the following set equality holds for every state $q \in Q_{\mathfrak{B}}$:*

$$\{w \in \mathrm{Dir}(\Sigma)^* \ : \ \mathcal{B}_q \textit{ accepts } [\![G]\!]_w\} = \{w \in \mathrm{Dir}(\Sigma)^* : \exists s \in \mathrm{ctl}(w) \,.\, \mathcal{Q}(s) = q\}$$

where $\mathrm{ctl}(w)$ is the set of control *states of $\mathcal{A}$ that are reachable on reading word w.*[3]

In particular if we have an APT $\mathcal{B}_{q_0}$ (with initial state q_0) then we can represent the set of subtrees of $[\![G]\!]$ accepted by $\mathcal{B}_{q_0}$ with an automaton-solution $\mathcal{A}$ to the Global Model-

[3] The $\perp$ label of a state in $\mathcal{A}$ allows one to avoid a state in $\mathcal{A}$ being associated with any element of $\mathcal{B}$. That is, $\mathcal{Q}$ could be viewed as a *partial* function from $\mathcal{A}$ to $\mathcal{B}$.

Checking Problem which is given final states $\{s \in Q_{\mathcal{A}} : \mathcal{Q}(s) = q_0\}$ and the standard acceptance condition for finite strings.

An ***order-n pushdown automaton (n-PDA)*** is an automaton equipped with a stack of $(n-1)$-stacks where a 1-stack contains only atoms and a $(k+1)$-stack is a stack of k-stacks. For $m \geq 2$ an order-m *push* operation copies the top-most $(m-1)$-stack whilst an order-m *pop* operation discards it. The order-1 *push* and *pop* are the standard pushdown operations acting on the top-most 1-stack. We write top_1 to denote the top-most element of the top-most 1-stack. An ***order-n*** **collapsible** ***pushdown automaton (n-CPDA)*** [10] has an n-stack that allows a *pointer* from any atomic element (stack symbol) to a k-stack below it (where $1 \leq k < n$). It has a *collapse* operation that discards the stack's contents above the target of top_1's pointer.

We claim that an order-n collapsible pushdown automaton (n-CPDA) can be a solution to the GMCP for an order-n recursion scheme. Moreover we claim that it is possible to effectively construct the requisite n-CPDA from $\mathfrak{B}$ (or $\mathcal{B}_q$) and G. We adapt the automaton $\mathrm{CPDA}(G)$ introduced by Hague et al. [10] that is able to compute traversals of $\mathrm{Gr}(G)$ so that instead it computes *winning traversals of* $\mathcal{G}^{\mathcal{W}}_{G,\mathcal{C}^{\perp}}$. Its direction at terminal symbols is guided by reading a node α of $[\![G]\!]$ (which is just a word in $\mathrm{Dir}(\Sigma)^*$). By Lemma 6 this enables the automaton to fulfil the task demanded of it.

Theorem 1. *Let G be an order-n recursion scheme and $\mathfrak{B}$ a property APT template. We can construct an n-CPDA that is a solution to the associated Global Model Checking Problem in n-EXPTIME. The constructed automaton has n-exponential size.*

Note that deciding whether an n-CPDA accepts a given finite-word turns out to be $(n-1)$-EXPTIME complete in the size of the n-CPDA. We can establish this by first showing the emptiness problem to be $(n-1)$-EXPTIME complete.

In general an $(n-1)$-CPDA cannot provide a solution to GMCP for order-n schemes unless $(n-1)$-CPDA are equi-expressive with n-CPDA.

Lemma 7. *Let G be a (non-deterministic) order-n recursion scheme that generates a finite-word language $\mathcal{L}$ over an alphabet Σ. There exists a* deterministic *order-n recursion scheme G' generating a ranked and ordered tree together with a μ-calculus sentence ϕ such that the language $\mathcal{L}' := \{\, \alpha \in \mathrm{Dir}^* : [\![G]\!]_\alpha \vDash \phi \,\}$ can be viewed as being over an alphabet Σ' with $\Sigma \subseteq \Sigma'$ and*

$$\mathcal{L}' \upharpoonright_{\Sigma} := \{\, w \in \Sigma^* \ : \ w \textit{ is the maximal } \Sigma\textit{-sub-sequence of an element of } \mathcal{L}' \,\} \ = \ \mathcal{L}$$

Proof. Let G be a *non-deterministic* order-n recursion scheme generating a finite word language $\mathcal{L}$ over a finite alphabet Σ. The elements of Σ can be viewed as terminals of arity 1 and we also have an *end-of-string marker* $e \notin \Sigma$ with arity 0. The rules of G will be of the form $F_i \zeta^i_1 \ldots \zeta^i_{m_i} \quad \longrightarrow \quad t^i_1 \mid \ldots \mid t^i_{k_i}$ for each non-terminal F_i where i ranges in $1 \leq i \leq N$ (say). Let K be the least integer with $k_i \leq K$ for all $1 \leq i \leq N$.

We form a deterministic recursion scheme G' that generates a single tree. The ranked alphabet Γ used by G' consists of two arity-0 terminals e and b together with a terminal h of arity $|\Sigma| + K$. Let $\sigma \ : \ \Sigma \longrightarrow \{\, i \ : \ 1 \leq i \leq |\Sigma| \,\}$ be some bijection.

We give G' a terminal F'_i for every terminal F_i in G such that F'_i has the same type as F_i. We take the initial non-terminal of G' to be S'; we further provide a non-terminal C_c with type $o \rightarrow o$ for each $c \in \Sigma$. The rules of G' are as follows:

$$F'_i\, \zeta^i_1 \ldots \zeta^i_{m_i} \longrightarrow h\ \underbrace{b \ldots b}_{|\Sigma| \text{ times}}\ t^{i\,\star}_1 \ldots t^{i\ \star}_{k_i}\ \ \underbrace{b \ldots b}_{(K - k_i) \text{ times}}$$

$$C_c\, x \longrightarrow h\ \ \underbrace{b \ldots b}_{\sigma(c) - 1 \text{ times}}\ \ x\ \ \underbrace{b \ldots b}_{|\Sigma| - \sigma(c) \text{ times}}\ \underbrace{b \ldots b}_{K \text{ times}}$$

for $1 \leq i \leq N$ and $c \in \Sigma$, where $t^{i\star}_l$ ($1 \leq l \leq k_i$) is formed from t^i_l by replacing each occurrence of a terminal $c \in \Sigma$ with a non-terminal C_j. Note that the end-of-string marker e is never replaced as our convention leaves it out of Σ.

Let α be a node of $[\![G]\!]$. Let us identify Σ with the first $|\Sigma|$ directions $\{1, \ldots, |\Sigma|\}$ of h. By the construction of G' from G we have $[\![G]\!](\alpha) = e$ if and only if $\alpha \upharpoonright_{\Sigma}\ \in \mathcal{L}$. The μ-calculus formula ϕ asserting a node is labelled with e then gives the result. $\square$

The language $\mathcal{L}'$ in the above Lemma is the set that should be recognised by the solution to the GMCP for G' and ϕ. If an n-CPDA can recognise $\mathcal{L}'$, then there must exist an n-CPDA that can recognise $\mathcal{L}' \upharpoonright_{\Sigma}$. There exists a hierarchy theorem of Damm [6] for PDA and modulo the assumption of a similar theorem for CPDA we obtain the following:

Theorem 2. *Assuming that the CPDA generated word-languages form a strict hierarchy, there exists an order-n recursion scheme G and a μ-calculus sentence ϕ such that no m-CPDA with $m < n$ can be a solution to the corresponding GMCP.*

5 Winning Region of a Collapsible Pushdown Game

We characterise the *constructible* winning region of a collapsible pushdown parity game in terms of the sequences of stack operations that can generate the winning configurations. We refer to the automaton-generator of the underlying digraph as a ***collapsible pushdown system*** (CPDS) and its configuration graph as a ***CPDS graph***.

Some stacks cannot be constructed by operations on the empty-stack: $[\,[\,a\,]\,[\,a\,]\,[\,b\,]\,]$.

The Unravelling of a CPDS Graph and Winning Condition APT. The *unravelling* of a CPDS graph $\mathcal{G}$ is a tree $\mathrm{unrav}(G)$ formed by labelling each node (q, s) (q a control state and s a stack) of the configuration graph with $(q, top_1\ s)$ and then unfolding from the initial configuration. We can view this tree as being ranked and ordered by giving a label (q, a) an arity equal to the size of the set $\{\,(q', \theta) : (q, a, q', \theta) \in \Delta\,\}$, where $\Delta \subseteq Q \times \Gamma \times Q \times Op_n$ is the transition relation and Op_n the set of order-n stack operations. We make the tree ordered by placing a linear order on the set.

For any CPDS parity game with underlying CPDS graph $\mathcal{G}$ the ownership $O(q)$ and priority $\Omega(q)$ of a configuration (q, s) are given entirely by q. We can thus [7] construct an APT $\mathcal{B}$ that, for a given node r in $\mathrm{unrav}(\mathcal{G})$ corresponding to a configuration (q, s) in $\mathcal{G}$, accepts $\mathrm{unrav}(\mathcal{G})_r$ if and only if Éloïse has a winning strategy from (q, s). Whenever $\mathcal{B}$ reads a node labelled (q, a), it transitions to a state with priority $\Omega(q)$ that is a $\bigvee$ state if $O(q)$ is Abelard and $\bigwedge$ otherwise. We call $\mathcal{B}$ ***the winning condition APT (WCAPT)***.

The Versatile CPDS Parity Game. Let us fix an n-CPDS parity game $\mathcal{A}$. We convert it to a game $\mathcal{A}^{\mathbf{0}}$, which by analogy with the work in previous sections is referred to as

the versatile CPDS parity game. The game $\mathcal{A}^{\mathbf{0}}$ extends $\mathcal{A}$ with a single control state $\mathbf{0}$. The priority and owner of $\mathbf{0}$ does not matter and so may be arbitrarily selected.

We make $\mathbf{0}$ the initial control state of $\mathcal{A}^{\mathbf{0}}$. Whilst in control state $\mathbf{0}$, Éloïse is allowed to perform arbitrary stack operations whilst remaining at $\mathbf{0}$. She may also opt at any point to transition from $\mathbf{0}$ into a control state q of $\mathcal{A}$ without performing any stack operation. After doing so, play proceeds as in $\mathcal{A}$. Consider the set:

$$S_{\mathbf{0}} := \{(\mathbf{0}, \theta) \ : \theta \text{ a stack operation}\} \cup \{(q, id) \ : \ q \text{ a control state of } \mathcal{A}\}$$

where id is the stack operation that leaves the stack unchanged. Let $\mathcal{G}^{\mathbf{0}}$ be the underlying CPDS-graph of $\mathcal{A}^{\mathbf{0}}$. The directions emanating from a node r in $\mathrm{unrav}(\mathcal{G}^{\mathbf{0}})$ having label $(\mathbf{0}, a)$, for any stack symbol a, are in $1-1$ correspondence with $S_{\mathbf{0}}$. We may thus label a direction of such a node r with θ if this direction corresponds to a transition $(\mathbf{0}, \theta)$ and q (q a control state of $\mathcal{A}$) if it corresponds to a transition (q, id).

Consider a finite path $p = p_0 \; p_1 \; \ldots \; p_m \; p'_m$ in $\mathrm{unrav}(\mathcal{G}^{\mathbf{0}})$, where p_0 is the root of the tree, with trace of the form $(\mathbf{0}, a_1) \; (\mathbf{0}, a_2) \; \ldots \; (\mathbf{0}, a_m) \; (q, a_m)$ such that q is a control state of $\mathcal{A}$. The node p'_m is represented as a string of directions, but this string can be represented by a string of the form $\theta_1 \; \ldots \; \theta_m \; q$. The final element p'_m of p will correspond to a configuration (q, s) in $\mathcal{G}^{\mathbf{0}}$ where s is a stack produced from the empty stack by performing the composite operation $\theta_1; \; \ldots \; ; \theta_m$. Conversely, for any sequence of stack operations followed by a control state q of $\mathcal{A}$ there must exist a node in $\mathrm{unrav}(\mathcal{G}^{\mathbf{0}})$ represented by this sequence which corresponds to a configuration (q, s) with s formed by the sequence of stack operations starting at the empty stack.

Éloïse has a winning strategy from such a configuration (q, s) in $\mathcal{A}$ if and only if she has a winning strategy from (q, s) in $\mathcal{A}^{\mathbf{0}}$, since the games proceed identically from this configuration. Let us write $\mathcal{B}^{\mathbf{0}}$ for the WCAPT of $\mathcal{A}^{\mathbf{0}}$. Suppose further that s can be formed from the empty stack by a sequence $\theta_1 \; \ldots \; \theta_m$ of stack operations. It follows that (q, s) is a winning configuration in $\mathcal{A}$ if and only if $\mathcal{B}^{\mathbf{0}}$ accepts the tree $\mathrm{unrav}(\mathcal{G}^{\mathbf{0}})_{\theta_1 \; \ldots \; \theta_m \; q}$, viewing $\theta_1 \; \ldots \; \theta_m \; q$ as a string of directions – i.e. a node.

The Constructible Winning Region of an n-CPDS Parity Game. It has been shown by Hague et al. [10] that the *unravelling* of a CPDS graph can be generated by a deterministic n-CPDA and consequently by a (deterministic) order-n recursion scheme. Let $G^{\mathbf{0}}$ be such a recursion scheme for our n-CPDS parity game $\mathcal{A}^{\mathbf{0}}$. Let us apply Theorem 1 to generate a solution $\mathcal{D}$ for the GMCP with $G^{\mathbf{0}}$ and the property expressed by $\mathcal{B}^{\mathbf{0}}$. We then restrict $\mathcal{D}$ to form an automaton $\mathcal{D}^{-}$ that only accepts words of the form $\theta_1 \; \ldots \; \theta_m \; q$ that are also accepted by $\mathcal{D}$. The automaton $\mathcal{D}^{-}$ witnesses the following:

Theorem 3. *Let $\mathcal{A}$ be an n-CPDS parity game with stack operations Op_n and control states Q. We can construct in n-EXPTIME an n-CPDA that recognises a subset $\mathcal{L}$ of $(Op_n)^* Q$ such that Éloïse has a winning strategy from a configuration (q, s) with s constructible (via operations in Op_n) from the empty stack, if and only if for every operation sequence $\theta_1; \; \ldots \; ; \; \theta_m$ generating s from the empty stack, $\theta_1 \; \ldots \; \theta_m \; q \in \mathcal{L}$.*

Given any configuration (q, s) with constructible stack s we can thus determine whether it is a winning configuration by picking *any* operation sequence $\theta_1 \; \ldots \; \theta_m$ witnessing the constructibility of s and deciding whether $\theta_1 \; \ldots \; \theta_m \; q$ is accepted by the automaton.

Further Directions. A pressing question is whether one can construct a more succinct and *deterministic* n-CPDA providing a solution to the GMCP for the trees in question.

Theorem 3 is weak as it stands. Carayol and Slaats [5] have shown that constructible n-PDS (non-collapsible) parity game winning regions are *'n-regular'* [3,9] and admit a *canonical representation*. An analogous result for CPDS games would be good.

Acknowledgements. We acknowledge the use of William Blum's `pstring.sty` and Frank Drewes' `graph.sty` packages, and thank Olivier Serre and Matthew Hague for useful discussions, and the anonymous referees for helpful and thorough comments.

References

1. Bradfield, J., Stirling, C.P.: Modal logics and mu-calculi: an introduction. In: Handbook of Process Algebra, pp. 293–332. Elsevier, North-Holland (2001)
2. Cachat, T.: Uniform solution of parity games on prefix-recognizable graphs. In: Proc. VISS. ENTCS, vol. 68. Elsevier, Amsterdam (2002)
3. Carayol, A.: Regular sets of higher-order pushdown stacks. In: Jedrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 168–179. Springer, Heidelberg (2005)
4. Carayol, A., Hague, M., Meyer, A., Ong, C.-H.L., Serre, O.: Winning regions of higher-order pushdown games. In: Proc. LICS, pp. 193–204. IEEE Computer Society, Los Alamitos (2008)
5. Carayol, A., Slaats, M.: Positional strategies for higher-order pushdown parity games. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 217–228. Springer, Heidelberg (2008)
6. Damm, W.: The IO- and OI -hierarchy. Theoretical Computer Science 20, 95–207 (1982)
7. Emerson, E.A., Jutla, C.S.: Tree automata, mu-calculus and determinacy. In: Proc. FOCS, pp. 368–377. IEEE computer society, Los Alamitos (1991)
8. Engelfriet, J.: Iterated pushdown automata and complexity classes. In: Proc. STOC, pp. 365–373. ACM, New York (1983)
9. Frantani, S.: Automates à piles de piles...de piles. PhD thesis (2005)
10. Hague, M., Murawski, A.S., Ong, C.-H.L., Serre, O.: Collapsible pushdown automata and recursion schemes. In: Proc. LICS. IEEE Computer Society, Los Alamitos (2008)
11. Hyland, M., Ong, C.-H.L.: On full abstraction for PCF: I, II and III. Information and computation 163(2), 285–408 (2000)
12. Jurdziński, M.: Small progress measures for solving parity games. In: Reichel, H., Tison, S. (eds.) STACS 2000. LNCS, vol. 1770, p. 290. Springer, Heidelberg (2000)
13. Knapik, T., Niwinski, D., Urzyczyn, P.: Higher-order pushdown trees are easy. In: Nielsen, M., Engberg, U. (eds.) FOSSACS 2002. LNCS, vol. 2303, pp. 205–222. Springer, Heidelberg (2002)
14. Ong, C.-H.L.: On model-checking trees generated by higher-order recursion schemes. In: IEEE Computer Society, IEEE Computer Society, Los Alamitos (2006); Journal version, `users.comlab.ox.ac.uk/luke.ong/publications/ntrees.pdf`
15. Piterman, N., Vardi, M.Y.: Global model-checking of infinite-state systems. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 387–400. Springer, Heidelberg (2004)
16. Serre, O.: Note on winning positions on pushdown games with ω-regular conditions. Information Processing Letters 85, 285–291 (2003)
17. Stirling, C.P.: Bisimulation, model checking and other games. In: Notes for the Mathfit instructional meeting on games and Computation (1997)

A Kleene Theorem for Polynomial Coalgebras

Marcello Bonsangue[1,2], Jan Rutten[2,3], and Alexandra Silva[2,⋆]

[1] LIACS - Leiden University
[2] Centrum voor Wiskunde en Informatica (CWI)
[3] Vrije Universiteit Amsterdam (VUA)

Abstract. For polynomial functors G, we show how to generalize the classical notion of regular expression to G-coalgebras. We introduce a language of expressions for describing elements of the final G-coalgebra and, analogously to Kleene's theorem, we show the correspondence between expressions and finite G-coalgebras.

1 Introduction

Regular expressions were first introduced by Kleene [8] to study the properties of neural networks. They are an algebraic description of languages, offering a declarative way of specifying the strings to be recognized and they define exactly the same class of languages accepted by deterministic (and non-deterministic) finite state automata: the regular languages. The correspondence between regular expressions and (non-)deterministic automata has been widely studied and a translation between these two different formalisms is presented in most books on automata and language theory [10,6].

Formally, a deterministic automaton consists of a set of states S equipped with a transition function $\delta : S \to 2 \times S^A$ determining for each state whether or not it is final and assigning to each input symbol a next state.

Deterministic automata can be generalized to coalgebras for an endofunctor G on the category **Set**. A coalgebra is a pair (S, g) consisting of a set of states S and a transition function $g : S \to GS$, where the functor G determines the type of the dynamic system under consideration and is the base of the theory of universal coalgebra [14]. The central concepts in this theory are homomorphism of coalgebras, bisimulation equivalence and final coalgebra. These can be seen, respectively, as generalizations of automata homomorphism, language equivalence and the set of all languages. In fact, in the case of deterministic automata, the functor G would be instantiated to $2 \times Id^A$ and the usual notions would be recovered. In particular, note that the final coalgebra for this functor is precisely the set 2^{A^*} of all languages over A [15].

Given the fact that coalgebras can be seen as generalizations of deterministic automata, it is natural to investigate whether there exists an appropriate

⋆ Partially supported by the Fundação para a Ciência e a Tecnologia, Portugal, under grant number SFRH/BD/27482/2006.

notion of regular expression in this setting. More precisely: is it possible to define a language of expressions that represents precisely the behaviour of finite G-coalgebras, for a given functor G?

In this paper, we show how to define such a language for coalgebras of polynomial functors G (a functor is polynomial if it is built inductively from the identity and constant functors, using product, coproduct and exponential). We introduce a language of expressions for describing elements of the final G-coalgebra (Section 3). Analogously to Kleene's theorem, we show the correspondence between expressions and finite G-coalgebras. In particular, we show that every state of a finite G-coalgebra corresponds to an expression in the language (Section 4) and, conversely, we give a compositional synthesis algorithm which transforms every expression into a finite G-coalgebra (Section 5).

Related Work. Regular expressions have been originally introduced by Kleene [8] as a mathematical notation for describing languages recognized by deterministic finite automata. In [15], deterministic automata, the sets of formal languages and regular expressions are all presented as coalgebras of the functor $2 \times Id^A$ (where A is the alphabet, and 2 is the two element set). It is then shown that the standard semantics of language acceptance of automata and the assignment of languages to regular expressions both arise as the unique homomorphism into the final coalgebra of formal languages. The coalgebra structure on the set of regular expressions is determined by their so-called *Brzozowski* derivatives [4]. In the present paper, the set of expressions for the functor $F(S) = 2 \times S^A$ differs from the classical definition in that we do not have Kleene star and full concatenation (sequential composition) but, instead, the least fixed point operator and action prefixing. Modulo that difference, the definition of a coalgebra structure on the set of expressions in both [15] and the present paper is essentially the same. All in all, one can therefore say that standard regular expressions and their treatment in [15] can be viewed as a special instance of the present approach. This is also the case for the generalization of the results in [15] to automata on guarded strings [11]. Finally, the present paper extends the results in our FoSSaCS'08 paper [3], where a sound and complete specification language and a synthesis algorithm for Mealy machines is given. Mealy machines are coalgebras of the functor $(B \times Id)^A$, where A is a finite input alphabet and B is a finite meet semilattice for the output alphabet.

2 Preliminaries

We give the basic definitions on polynomial functors and coalgebras and introduce the notion of bisimulation.

First we fix notation on sets and operations on them. Let **Set** be the category of sets and functions. Sets are denoted by capital letters $X, Y, \ldots$ and functions by lower case $f, g, \ldots$. The collection of functions from a set X to a set Y is denoted by Y^X. Given functions $f : X \to Y$ and $g : Y \to Z$ we write their composition as $g \circ f$. The product of two sets X, Y is written as $X \times Y$, with projection functions $X \xleftarrow{\pi_1} X \times Y \xrightarrow{\pi_2} Y$ .The set 1 is a singleton set typically written as $1 = \{*\}$ and it can be regarded as the empty product. We define $X + Y$ as the set $X \uplus Y \uplus \{\bot, \top\}$, where $\uplus$ is the disjoint union of sets, with injections $X \xrightarrow{\kappa_1} X \uplus Y \xleftarrow{\kappa_2} Y$. Note that the set $X + Y$ is different from the classical coproduct of X and Y, because of the two extra elements $\bot$ and $\top$. These extra elements will later be used to represent, respectively, underspecification and inconsistency in the specification of some systems. The intuition behind the need of these extra elements will become clear when we present our language of expressions and concrete examples, in Section 5.3, of systems whose type involves $+$.

Polynomial Functors. In our definition of polynomial functors we will use constant sets equipped with an information order. In particular, we will use join-semilattices. A (bounded) join-semilattice is a set B equipped with a binary operation $\vee_B$ and a constant $\bot_B \in B$, such that $\vee_B$ is commutative, associative and idempotent. The element $\bot_B$ is neutral w.r.t. $\vee_B$. As usual, $\vee_B$ gives rise to a partial ordering $\leq_B$ on the elements of B:

$$b_1 \leq_B b_2 \Leftrightarrow b_1 \vee_B b_2 = b_2$$

Every set S can be transformed into a join-semilattice by taking B to be the set of all finite subsets of S with union as join.

We are now ready to define the class of polynomial functors. They are functors $G : \mathbf{Set} \to \mathbf{Set}$, built inductively from the identity and constants, using $\times$, $+$ and $(-)^A$. Formally, the class PF of *polynomial functors* on **Set** is inductively defined by putting:

$$G ::= Id \mid B \mid G + G \mid G \times G \mid G^A$$

where B is a finite join-semilattice and A is a finite set.

Typical examples of polynomial functors are $D = 2 \times Id^A$ and $P = (1 + Id)^A$. These functors, which we shall later use as our running examples, represent, respectively, the type of *deterministic* and *partial deterministic* automata. It is worth noting that although when we mentioned the type of deterministic automata in the introduction, we did not made explicit that the set 2 was a join semilattice, which is in fact the case. Also the set of classical regular expressions has a join-semilattice structure, which provides also intuition for the differences in our definition of polynomial functors, when compared with [13,7], in the use of a join-semilattice as constant and in the definition of $+$. If we want to generalize regular expressions to polynomial functors then we must guarantee that they also have such structure, namely by imposing it in the constant and $+$ functors. For the $\times$ and $(-)^A$ we do not need to add extra elements because the semilattice structure is compositionally inherited.

Next, we give the definition of the ingredient relation, which relates a polynomial functor G with its *ingredients*, *i.e.* the functors used in its inductive construction. We shall use this relation later for typing our expressions.

Let $\lhd \subseteq PF \times PF$ be the least reflexive and transitive relation such that

$$G_1 \lhd G_1 \times G_2, \quad G_2 \lhd G_1 \times G_2, \quad G_1 \lhd G_1 + G_2, \quad G_2 \lhd G_1 + G_2, \quad G \lhd G^A$$

Here and throughout this document we use $F \lhd G$ as a shorthand for $\langle F, G \rangle \in \lhd$. If $F \lhd G$, then F is said to be an *ingredient* of G. For example, 2, Id, Id^A and D itself are all the ingredients of the deterministic automata functor D.

Coalgebras. For a functor G on **Set**, a G-coalgebra is a pair (S, f) consisting of a set of *states* S together with a function $f : S \to GS$. The functor G, together with the function f, determines the *transition structure* (or dynamics) of the G-coalgebra [14]. Deterministic automata and partial automata are, respectively, coalgebras for the functors $D = 2 \times Id^A$ and $P = (1 + Id)^A$.

A *G-homomorphism* from a G-coalgebra (S, f) to a G-coalgebra (T, g) is a function $h\colon S \to T$ preserving the transition structure, *i.e.*, such that $g \circ h = Gh \circ f$. A G-coalgebra (Ω, ω) is said to be *final* if for any G-coalgebra (S, f) there exists a unique G-homomorphism $[\![\cdot]\!] : S \to \Omega$. For every polynomial functor G there exists a final G-coalgebra (Ω_G, ω_G) [14]. For instance, as we already mentioned in the introduction, the final coalgebra for the functor D is the set of languages 2^{A^*} over A, together with a transition function $d : 2^{A^*} \to 2 \times (2^{A^*})^A$ defined as $d(\phi) = \langle \phi(\epsilon), \lambda a \lambda w.\phi(aw) \rangle$. Here ϵ denotes the empty sequence and aw denotes the word resulting from prefixing w with the letter a. The notion of finality will play a key role later in providing a semantics to expressions.

Let (S, f) and (T, g) be two G-coalgebras. We call a relation $R \subseteq S \times T$ a *bisimulation* [1] if there exists a map $e : R \to GR$ such that the projections π_1 and π_2 are coalgebra homomorphisms, *i.e.* the following diagram commutes.

$$\begin{tikzcd}
S \arrow[d, "f"'] & R \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "\exists e"'] & T \arrow[d, "g"] \\
GS & GR \arrow[l, "G\pi_2"] \arrow[r, "G\pi_1"'] & GT
\end{tikzcd}$$

We write $s \sim_G t$ whenever there exists a bisimulation relation containing (s, t) and we call $\sim_G$ the bisimilarity relation. We shall drop the subscript G whenever the functor G is clear from the context. For G-coalgebras (S, f) and (T, g) and $s \in S, t \in T$, it holds that if $s \sim t$ then $[\![s]\!] = [\![t]\!]$.

3 A Language of Expressions for Polynomial Coalgebras

In this section we generalize the classical notion of regular expressions to polynomial coalgebras. We start by introducing an untyped language of expressions and then we single out the well-typed ones via an appropriate typing system, associating expressions to polynomial functors.

Let A be a finite set, B a finite join-semilattice and X a set of fixpoint variables. The set of all *expressions* is given by the following grammar:

$$\varepsilon ::= \emptyset \mid x \mid \varepsilon \oplus \varepsilon \mid \mu x.\gamma \mid b \mid l(\varepsilon) \mid r(\varepsilon) \mid l[\varepsilon] \mid r[\varepsilon] \mid a(\varepsilon)$$

where γ is a *guarded expression* given by:

$$\gamma ::= \emptyset \mid \gamma \oplus \gamma \mid \mu x.\gamma \mid b \mid l(\varepsilon) \mid r(\varepsilon) \mid l[\varepsilon] \mid r[\varepsilon] \mid a(\varepsilon)$$

A *closed expression* is an expression without free occurrences of fixpoint variables x. We denote the set of guarded and closed expressions by Exp.

Intuitively, expressions denote elements of the final coalgebra. The expressions $\emptyset$, $\varepsilon_1 \oplus \varepsilon_2$ and $\mu x.\, \varepsilon$ will play a similar role to, respectively, the empty language, the union of languages and the Kleene star in classical regular expressions for deterministic automata. The expressions $l(\varepsilon)$, $r(\varepsilon)$, $l[\varepsilon]$, $r[\varepsilon]$ and $a(\varepsilon)$ refer to the left and right hand-side of products and sums (*i.e.*, represent projections and injections), and function application, respectively. We shall soon illustrate, by means of examples, the role of these expressions.

Our language does not have any operator denoting intersection or complement (it only includes the sum operator $\oplus$). This is a natural restriction, very much in the spirit of Kleene's regular expressions for deterministic finite automata. We will prove that this simple language is expressive enough to denote exactly all finite coalgebras.

Next, we present a typing assignment system for associating expressions to polynomial functors. This will associate with each functor G the expressions $\varepsilon \in \mathit{Exp}$ that are valid specifications of G-coalgebras. The typing proceeds following the structure of the expressions and the ingredients of the functors.

We type expressions ε using the ingredient relation, as follows:

$$\frac{}{\vdash \emptyset : F \lhd G} \qquad \frac{}{\vdash b : B \lhd G} \qquad \frac{}{\vdash x : G \lhd G}$$

$$\frac{\vdash \varepsilon : G \lhd G}{\vdash \varepsilon : \mathit{Id} \lhd G} \qquad \frac{\vdash \varepsilon_1 : F \lhd G \quad \vdash \varepsilon_2 : F \lhd G}{\vdash \varepsilon_1 \oplus \varepsilon_2 : F \lhd G} \qquad \frac{\vdash \varepsilon : G \lhd G}{\vdash \mu x.\varepsilon : G \lhd G}$$

$$\frac{\vdash \varepsilon : F_1 \lhd G}{\vdash l(\varepsilon) : F_1 \times F_2 \lhd G} \qquad \frac{\vdash \varepsilon : F_2 \lhd G}{\vdash r(\varepsilon) : F_1 \times F_2 \lhd G} \qquad \frac{\vdash \varepsilon : F \lhd G}{\vdash a(\varepsilon) : F^A \lhd G}$$

$$\frac{\vdash \varepsilon : F_1 \lhd G}{\vdash l[\varepsilon] : F_1 + F_2 \lhd G} \qquad \frac{\vdash \varepsilon : F_2 \lhd G}{\vdash r[\varepsilon] : F_1 + F_2 \lhd G}$$

This type system is simple and most rules are self-explanatory. However, for full clarification some remarks should be made. (1) Intuitively, $\varepsilon : F \lhd G$ means that ε is an element (up to bisimulation) of $F(\Omega_G)$. (2) As expected, there is a rule for each expression construct. The extra rule involving $\mathit{Id} \lhd G$ reflects the isomorphism between the final coalgebra Ω_G and $G(\Omega_G)$. (3) Only fixpoints at the outermost level of the functor are allowed. This does not mean however that we disallow nested fixpoints. For instance, $\mu x.\, a(x \oplus \mu y.\, a(y))$ would be a well-typed expression for the functor D of deterministic automata, as it will become clear below, when we will present more examples of well-typed and non-well-typed expressions. (4) The presented type system is decidable (expressions are of finite length and the system is recursive).

We can now formally define the set of G-expressions: well-typed expressions associated with a polynomial functor G.

Definition 1 (G-expressions). *Let G be a polynomial functor and F an ingredient of G. We denote by $Exp_{F \lhd G}$ the following set:*

$$Exp_{F \lhd G} = \{\varepsilon \in Exp \ \mid \vdash \varepsilon : F \lhd G\}\,.$$

*We define the set Exp_G of well-typed G-*expressions *by $Exp_{G \lhd G}$.*

For the functor D, examples of well-typed expressions include $r(a(0))$, $l(1) \oplus r(a(l(0)))$ and $\mu x.r(a(x)) \oplus l(1)$. The expressions $l[1]$, $l(1) \oplus 1$ and $\mu x.1$ are examples of non well-typed expressions, because the functor D does not involve $+$, the subexpressions in the sum have different type, and recursion is not at the outermost level (1 has type $2 \lhd D$), respectively.

Let us instantiate the definition of expressions to the functors of deterministic automata $D = 2 \times Id^A$ and partial automata $P = (1 + Id)^A$.

Example 2 (Deterministic expressions). Let A be a finite set of input actions and let X be a set of (recursion or) fixpoint variables. The set of *deterministic expressions* is given by the following BNF syntax. For $a \in A$ and $x \in X$:

$$\varepsilon ::= \emptyset \mid x \mid r(a(\varepsilon)) \mid l(1) \mid l(0) \mid \varepsilon \oplus \varepsilon \mid \mu x.\varepsilon$$

where ε is closed and occurrences of fixpoint variables are within the scope of an input action.

It is easy to see that the closed (and guarded) expressions generated by the grammar presented above are exactly the elements of Exp_D. One can easily see that $l(1)$ and $l(0)$ are well-typed expressions for $D = 2 \times Id^A$ because both 1 and 0 are of type $2 \lhd D$. For the expression $r(a(\varepsilon))$ note that $a(\varepsilon)$ has type $Id^A \lhd D$ as long as ε has type $Id \lhd D$. And the crucial remark here is that, by definition of $\vdash$, $Exp_{Id \lhd G} = Exp_G$. Intuitively, this can be explained by the fact that for a polynomial functor G, if Id is one of the ingredients of G, then it is functioning as a pointer to the functor being defined:

$$\begin{tikzcd} G & = & \boxed{\cdots \quad Id \quad \cdots} \arrow[ll, bend left] \end{tikzcd}$$

Therefore, ε has type $Id \lhd D$ if it is of type $D \lhd D$, or more precisely, if $\varepsilon \in Exp_D$, which explains why the grammar above is correct.

At this point, we should remark that the syntax of our expressions differs from the classical regular expressions in the use of μ and action prefixing $a(\varepsilon)$ instead of star and full concatenation. We shall prove later that these two syntactically different formalisms are equally expressive (Theorems 5 and 6).

Without additional explanation we present next the syntax for the expressions in Exp_P.

Example 3 (Partial automata expressions). Let A be a finite set of input actions and X be a set of (recursion or) fixpoint variables. The set of *partial expressions* is given by the following BNF syntax. For $a \in A$ and $x \in X$:

$$\varepsilon ::= \emptyset \mid x \mid a(\varepsilon) \mid a\uparrow \mid \varepsilon \oplus \varepsilon \mid \mu x.\varepsilon$$

where ε is closed and occurrences of fixpoint variables are within the scope of an input action. For simplicity, $a\uparrow$ and $a(\varepsilon)$ abbreviate $a(l[*])$ and $a(r[\varepsilon])$.

We have now defined a language of expressions which gives us an algebraic description of systems. In the remainder of the paper, we want to present a generalization of Kleene's theorem for polynomial coalgebras (Theorems 5 and 6). Recall that, for regular languages, the theorem states that a language is regular if and only if it is recognized by a finite automaton.

3.1 Expressions Are Coalgebras

In this section, we show that the set of G-expressions for a given polynomial functor G has a coalgebraic structure $\lambda_G : Exp_G \to G(Exp_G)$. We proceed by induction on the ingredients of G. More precisely we are going to define a function

$$\lambda_{F \lhd G} : Exp_{F \lhd G} \to F(Exp_G)$$

and then set $\lambda_G = \lambda_{G \lhd G}$. Our definition of the function $\lambda_{F \lhd G}$ will make use of the following.

(i) We define a constant $Empty_{F \lhd G} \in F(Exp_G)$ by induction on the syntactic structure of F:

$$\begin{aligned}
Empty_{Id \lhd G} &= \emptyset\\
Empty_{B \lhd G} &= \bot_B\\
Empty_{F_1 \times F_2 \lhd G} &= \langle Empty_{F_1 \lhd G}, Empty_{F_2 \lhd G}\rangle\\
Empty_{F_1 + F_2 \lhd G} &= \bot\\
Empty_{F^A \lhd G} &= \lambda a. Empty_{F \lhd G}
\end{aligned}$$

(ii) We define $Plus_{F \lhd G} : F(Exp_G) \times F(Exp_G) \to F(Exp_G)$ by induction on the syntactic structure of F:

$$\begin{aligned}
Plus_{Id \lhd G}(\varepsilon_1, \varepsilon_2) &= \varepsilon_1 \oplus \varepsilon_2\\
Plus_{B \lhd G}(b_1, b_2) &= b_1 \vee_B b_2\\
Plus_{F_1 \times F_2 \lhd G}(\langle \varepsilon_1, \varepsilon_2\rangle, \langle \varepsilon_3, \varepsilon_4\rangle) &= \langle Plus_{F_1 \lhd G}(\varepsilon_1, \varepsilon_3), Plus_{F_2 \lhd G}(\varepsilon_2, \varepsilon_4)\rangle\\
Plus_{F_1 + F_2 \lhd G}(\kappa_i(\varepsilon_1), \kappa_i(\varepsilon_2)) &= \kappa_i(Plus_{F_i \lhd G}(\varepsilon_1, \varepsilon_2)), \quad i \in \{1,2\}\\
Plus_{F_1 + F_2 \lhd G}(\kappa_i(\varepsilon_1), \kappa_j(\varepsilon_2)) &= \top \quad i,j \in \{1,2\} \ \textit{and} \ \ i \neq j\\
Plus_{F_1 + F_2 \lhd G}(x, \top) &= Plus_{F_1 + F_2 \lhd G}(\top, x) = \top\\
Plus_{F_1 + F_2 \lhd G}(x, \bot) &= Plus_{F_1 + F_2 \lhd G}(\bot, x) = x\\
Plus_{F^A \lhd G}(f, g) &= \lambda a.\ Plus_{F \lhd G}(f(a), g(a))
\end{aligned}$$

Now we have all we need to define $\lambda_{F \lhd G}$. This function will be defined by double induction on the maximum number $N(\varepsilon)$ of nested unguarded occurrences of μ-expressions in ε and on the length of the proofs for typing expressions. We define $N(\varepsilon)$ as follows:

$$\begin{array}{ll} N(\emptyset) & = N(b) = N(a(\varepsilon)) = N(l(\varepsilon)) = N(r(\varepsilon)) = N(l[\varepsilon]) = N(r[\varepsilon]) = 0 \\ N(\varepsilon_1 \oplus \varepsilon_2) & = max\{N(\varepsilon_1),\ N(\varepsilon_2)\} \qquad\qquad N(\mu x.\varepsilon) = 1 + N(\varepsilon) \end{array}$$

For every ingredient F of a polynomial functor G and expression $\varepsilon \in Exp_{F \lhd G}$, $\lambda_{F \lhd G}(\varepsilon)$ is defined as follows:

$$\begin{array}{l} \lambda_{F \lhd G}(\emptyset) = Empty_{F \lhd G} \\ \lambda_{F \lhd G}(\varepsilon_1 \oplus \varepsilon_2) = Plus_{F \lhd G}(\lambda_{F \lhd G}(\varepsilon_1), \lambda_{F \lhd G}(\varepsilon_2)) \\ \lambda_{G \lhd G}(\mu x.\varepsilon) = \lambda_{G \lhd G}(\varepsilon[\mu x.\varepsilon/x]) \\ \lambda_{Id \lhd G}(\varepsilon) = \varepsilon \quad \text{for } G \neq Id \\ \lambda_{B \lhd G}(b) = b \\ \lambda_{F_1 \times F_2 \lhd G}(l(\varepsilon)) = \langle \lambda_{F_1 \lhd G}(\varepsilon), Empty_{F_2 \lhd G} \rangle \\ \lambda_{F_1 \times F_2 \lhd G}(r(\varepsilon)) = \langle Empty_{F_1 \lhd G}, \lambda_{F_2 \lhd G}(\varepsilon) \rangle \\ \lambda_{F_1 + F_2 \lhd G}(l[\varepsilon]) = \kappa_1(\lambda_{F_1 \lhd G}(\varepsilon)) \\ \lambda_{F_1 + F_2 \lhd G}(r[\varepsilon]) = \kappa_2(\lambda_{F_2 \lhd G}(\varepsilon)) \\ \lambda_{F^A \lhd G}(a(\varepsilon)) = \lambda a'. \begin{cases} \lambda_{F \lhd G}(\varepsilon) & a = a' \\ Empty_{F \lhd G} & otherwise \end{cases} \end{array}$$

Here, $\varepsilon[\mu x.\varepsilon/x]$ denotes syntactic substitution, replacing every free occurrence of x in ε by $\mu x.\varepsilon$.

In order to see that the definition of $\lambda_{F \lhd G}$ is well-formed, note the interplay between the two inductions: the length of the typing proof of the arguments in the recursive calls is strictly decreasing, except in the case of $\mu x.\varepsilon$; but, in this case we have that $N(\varepsilon) = N(\varepsilon[\mu x.\varepsilon/x])$, which can easily be proved by (standard) induction on the syntactic structure of ε, since ε is guarded (in x), and it guarantees that $N(\varepsilon[\mu x.\varepsilon/x]) < N(\mu x.\varepsilon)$. Also note that clause 4 of the above definition overlaps with clauses 1 and 2 (by taking $F = Id$). However, they give the same result and thus the definition is correct.

Definition 4. *We can now define, for each polynomial functor G, a G-coalgebra*

$$\lambda_G : Exp_G \to G(Exp_G)$$

by putting $\lambda_G = \lambda_{G \lhd G}$.

This means that we can define the subcoalgebra generated by an expression $\varepsilon \in Exp_G$, by repeatedly applying λ_G, which seems to be the correspondent of *half* of Kleene's theorem — the language represented by a given regular expression can be recognized by a finite state automaton. However, it is important to remark that the subcoalgebra generated by an expression $\varepsilon \in Exp_G$ by repeatedly applying λ_G is, in general, infinite. Take for instance the deterministic expression $\varepsilon_1 = \mu x.\, r(a(x \oplus \mu y.\, r(a(y))))$ and observe that:

$$\begin{array}{ll} \lambda_D(\varepsilon_1) & = \langle 0, \varepsilon_1 \oplus \mu y.\, r(a(y)) \rangle \\ \lambda_D(\varepsilon_1 \oplus \mu y.\, r(a(y))) & = \langle 0, \varepsilon_1 \oplus \mu y.\, r(a(y)) \oplus \mu y.\, r(a(y)) \rangle \\ & \vdots \end{array}$$

As one would expect, all the new states are bisimilar and can be identified. However, the function λ_D does not make any state identification and thus yields an infinite coalgebra.

The observation that the set of expressions has a coalgebra structure will be crucial for the proof of the generalized Kleene theorem, as will be shown in the next two sections.

4 Expressions Are Expressive

Having a G-coalgebra structure on Exp_G has two advantages. First, it provides us, by finality, directly with a natural semantics because of the existence of a (unique) homomorphism $[\![\cdot]\!] : Exp_G \to \Omega_G$, that assigns to every expression ε an element $[\![\varepsilon]\!]$ of the final coalgebra Ω_G.

The second advantage of the coalgebra structure on Exp_G is that it lets us use the notion of G-bisimulation to relate G-coalgebras (S, g) and expressions $\varepsilon \in Exp_G$. If one can construct a bisimulation relation between an expression ε and a state s of a given coalgebra, then the behaviour represented by ε is equal to the behaviour determined by the transition structure of the coalgebra applied to the state s. This is the analogue of computing the language $L(r)$ represented by a given regular expression r and the language $L(s)$ accepted by a state s of a finite state automaton and checking whether $L(r) = L(s)$.

The following theorem states that the every state in a finite G-coalgebra can be represented by an expression in our language. This generalizes *half* of Kleene's theorem: if a language is accepted by a finite automaton then it is regular. The generalization of the other *half* of the theorem (if a language is regular then it is accepted by a finite automaton) will be presented in Section 5.

Theorem 5. *Let G be a polynomial functor and (S, g) a G-coalgebra. If S is* finite *then there exists for any $s \in S$ an expression $\varepsilon_s \in Exp_G$ such that $\varepsilon_s \sim s$ (which implies $[\![\varepsilon_s]\!] = [\![s]\!]$).*

Proof. We construct, for a state $s \in S$, an expression $\varepsilon_s \sim s$. If $G = Id$, $\varepsilon_s = \emptyset$. Otherwise we proceed in the following way. Let $S = \{s_1, s_2, \ldots, s_n\}$, where $s_1 = s$. We associate with each state s_i a variable $x_i \in X$ and an equation $\varepsilon_i = \mu x_i.\gamma^G_{g(s_i)}$, where $\gamma^G_{g(s_i)}$ is defined as follows. For $F \lhd G$ and $s' \in FS$, the expression $\gamma^F_{s'} \in Exp_{F \lhd G}$ is defined by induction on the structure of F:

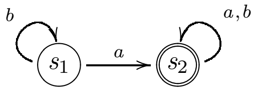
$$
\begin{array}{ll}
\gamma^{Id}_s = x_s & \gamma^B_b = b \\
\gamma^{F_1 \times F_2}_{\langle s, s' \rangle} = l(\gamma^{F_1}_s) \oplus r(\varepsilon^{F_2}_{s'}) & \\
\gamma^{F_1 + F_2}_{\kappa_1(s)} = l[\gamma^{F_1}_s] & \gamma^{F_1 + F_2}_{\kappa_2(s)} = r[\gamma^{F_2}_s] \\
\gamma^{F_1 + F_2}_{\bot} = \emptyset & \gamma^{F_1 + F_2}_{\top} = l[\emptyset] \oplus r[\emptyset] \\
\gamma^{F^A}_f = \bigoplus_{a \in A} a(\gamma^F_{f(a)}) &
\end{array}
$$

Note that the choice of $l[\emptyset] \oplus r[\emptyset]$ to represent inconsistency is arbitrary but *canonical*, in the sense that any other expression involving sum of $l[\varepsilon_1]$ and $r[\varepsilon_2]$ will be bisimilar.

Next, we eliminate all free occurrences of $x_1, \ldots, x_n$ from the system of equations $\varepsilon_1 = \mu x_1 . \gamma^G_{g(s_1)}, \ldots, \varepsilon_n = \mu x_n . \gamma^G_{g(s_n)}$ by first replacing x_n by ε_n in the equations for $\varepsilon_1, \ldots, \varepsilon_{n-1}$. Next, we replace x_{n-1} by ε_{n-1} in the equations for $\varepsilon_1, \ldots, \varepsilon_{n-2}$. Continuing in this way, we end up with an equation $\varepsilon_1 = \varepsilon$, where ε no longer contains any free variable. We then take $\varepsilon_s = \varepsilon$.

Moreover, $s \sim \varepsilon_s$, because the relation $R_G = \{\langle \varepsilon_s, s\rangle \mid s \in S\}$ is a bisimulation (for every functor G). Due to space restrictions we omit the proof of this fact, which can be found in [2].

Let us illustrate the construction above by some examples. Consider the following deterministic automaton over a two letter alphabet $A = \{a, b\}$, whose transition function is depicted in the following picture ($\textcircled{\textcircled{s}}$ represents that the state s is final):



Now define $\varepsilon_1 = \mu x_1 .\, \varepsilon$ and $\varepsilon_2 = \mu x_2 .\, \varepsilon'$ where

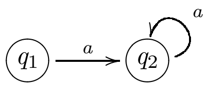
$$\varepsilon = l(0) \oplus r(b(x_1) \oplus a(x_2)) \qquad \varepsilon' = l(1) \oplus r(a(x_2) \oplus b(x_2))$$

Substituting x_2 by ε_2 in ε_1 then yields

$$\varepsilon_1 = \mu x_1 .\, l(0) \oplus r(b(x_1) \oplus a(\varepsilon_2)) \qquad \varepsilon_2 = \mu x_2 .\, l(1) \oplus r(a(x_2) \oplus b(x_2))$$

By construction we have $s_1 \sim \varepsilon_1$ and $s_2 \sim \varepsilon_2$.

As another example, take the following partial automaton, also over a two letter alphabet $A = \{a, b\}$:



In the graphical representation of a partial automaton (S, p) we omit transitions for which $p(s)(a) = \kappa_1(*)$. In this case, this happens for both states for the input letter b.

We define $\varepsilon_1 = \mu x_1 .\, \varepsilon$ and $\varepsilon_2 = \mu x_2 .\, \varepsilon'$ where $\varepsilon = \varepsilon' = b{\uparrow} \oplus a(x_2)$. Substituting x_2 by ε_2 in ε_1 then yields

$$\varepsilon_1 = \mu x_1 .\, b{\uparrow} \oplus a(\varepsilon_2) \qquad \varepsilon_2 = \mu x_2 .\, b{\uparrow} \oplus a(x_2)$$

Again we have $s_1 \sim \varepsilon_1$ and $s_2 \sim \varepsilon_2$.

5 Finite Systems for Expressions

We now give a construction to prove the converse of Theorem 5, that is, we describe a synthesis process that produces a *finite* G-coalgebra from an arbitrary regular G-expression ε. The states of the resulting G-coalgebra will consist of a finite subset of expressions, including an expression ε' such that $\varepsilon \sim_G \varepsilon'$.

5.1 Formula Normalization

We saw in Section 3.1 that the set of expressions has a coalgebra structure. We observed however that the subcoalgebra generated by an expression is in general infinite.

In order to guarantee the termination of the synthesis process we need to identify some expressions. In fact, as we will formally show later, it is enough to identify expressions that are provably equivalent using only the following axioms:

$$\begin{array}{ll}
(\mathit{Idempotency}) & \varepsilon \oplus \varepsilon = \varepsilon \\
(\mathit{Commutativity}) & \varepsilon_1 \oplus \varepsilon_2 = \varepsilon_2 \oplus \varepsilon_1 \\
(\mathit{Associativity}) & \varepsilon_1 \oplus (\varepsilon_2 \oplus \varepsilon_3) = (\varepsilon_1 \oplus \varepsilon_2) \oplus \varepsilon_3 \\
(\mathit{Empty}) & \emptyset \oplus \varepsilon = \varepsilon
\end{array}$$

This group of axioms gives to the set of expressions the structure of a join-semilattice. One easily shows that if two expressions are provably equivalent using these axioms then they are bisimilar (soundness).

For instance, it is easy to see that the deterministic expressions

$$r(a(\emptyset)) \oplus l(1) \oplus \emptyset \oplus l(1) \text{ and } r(a(\emptyset)) \oplus l(1)$$

are equivalent using the equations (*Idempotency*) and (*Empty*).

We thus work with normalized expressions in order to eliminate any syntactic redundancy present in the expression: in a sum, $\emptyset$ can be eliminated and, by idempotency, the sum of two syntactically equivalent expressions can be simplified. The function $\mathit{norm}_G : \mathit{Exp}_G \to \mathit{Exp}_G$ encodes this procedure. We define it by induction on the expression structure as follows:

$$\begin{array}{ll}
\mathit{norm}_G(\emptyset) & = \emptyset \\
\mathit{norm}_G(\varepsilon_1 \oplus \varepsilon_2) & = \mathit{plus}(\mathit{rem}(\mathit{flatten}(\mathit{norm}_G(\varepsilon_1) \oplus \mathit{norm}_G(\varepsilon_2)))) \\
\mathit{norm}_G(\mu x.\varepsilon) & = \mu x.\varepsilon \\
\mathit{norm}_B(b) & = b \\
\mathit{norm}_{G_1 \times G_2}(l(\varepsilon)) & = l(\varepsilon) \\
\mathit{norm}_{G_1 \times G_2}(r(\varepsilon)) & = r(\varepsilon) \\
\mathit{norm}_{G_1 + G_2}(l[\varepsilon]) & = l[\varepsilon] \\
\mathit{norm}_{G_1 + G_2}(r[\varepsilon]) & = r[\varepsilon] \\
\mathit{norm}_{G^A}(a(\varepsilon)) & = a(\varepsilon)
\end{array}$$

Here, the function *plus* takes a list of expressions $[\varepsilon_1, \dots, \varepsilon_n]$ and returns the expression $\varepsilon_1 \oplus \dots \oplus \varepsilon_n$ (*plus* applied to the empty list yields $\emptyset$), *rem* removes duplicates in a list and *flatten* takes an expression ε and produces a list of expressions by omitting brackets and replacing $\oplus$-symbols by commas:

$$\begin{array}{ll}
\mathit{flatten}(\varepsilon_1 \oplus \varepsilon_2) & = \mathit{flatten}(\varepsilon_1) \cdot \mathit{flatten}(\varepsilon_2) \\
\mathit{flatten}(\emptyset) & = [] \\
\mathit{flatten}(\varepsilon) & = [\varepsilon],\ \varepsilon \in \{b, a(\varepsilon_1), l(\varepsilon_1), r(\varepsilon_1), l[\varepsilon_1], r[\varepsilon_1], \mu x.\varepsilon_1\}
\end{array}$$

In this definition, $\cdot$ denotes list concatenation and $[\varepsilon]$ the singleton list containing ε. Note that any occurrence of $\emptyset$ in a sum is eliminated because $\mathit{flatten}(\emptyset) = []$.

For example, the normalization of the two deterministic expressions above results in the same expression: $r(a(\emptyset)) \oplus l(1)$.

Note that $norm_G$ only normalizes one level of the expression and still distinguishes the expressions $\varepsilon_1 \oplus \varepsilon_2$ and $\varepsilon_2 \oplus \varepsilon_1$. To simplify the presentation of the normalization algorithm, we decided not to identify these expressions, since it does not influence termination. In the examples below, this situation will never occur.

5.2 Synthesis Procedure

Given an expression $\varepsilon \in Exp_G$ we will generate a finite G-coalgebra by applying repeatedly $\lambda_G : Exp_G \to Exp_G$ and normalizing the expressions obtained at each step. We will use the function Δ, which takes an expression $\varepsilon \in Exp_G$ and returns a G-coalgebra, and which is defined as follows:

$$\Delta_G(\varepsilon) = (dom(g), g) \quad \textbf{where } g = D_G(\{norm_G(\varepsilon)\}, \emptyset)$$

Here, *dom* returns the domain of a finite function and D_G applies λ_G, starting with state $norm_G(\varepsilon)$, to the new states (after normalization) generated at each step, repeatedly, until all states in the coalgebra have their transition structure fully defined. The arguments of D_G are two sets of states: $sts \subseteq Exp_G$, the states that still need to be processed and $vis \subseteq Exp_G$, the states that already have been visited (synthesized). For each $\varepsilon \in sts$, D_G computes $\lambda_G(\varepsilon)$ and produces an intermediate transition function (possibly partial) by taking the union of all those $\lambda_G(\varepsilon)$. Then, it collects all new states appearing in this step, normalizing them, and recursively computes the transition function for those.

$$D_G(sts, vis) = \begin{cases} \emptyset & sts = \emptyset \\ trans \cup D_G(newsts, vis') & otherwise \end{cases}$$
$$\begin{aligned} \textbf{where } trans &= \{\langle \varepsilon, \lambda_G(\varepsilon)\rangle \mid \varepsilon \in sts\} \\ sts' &= collectStates_G(\pi_2(trans)) \\ vis' &= sts \cup vis \\ newsts &= sts' \setminus vis' \end{aligned}$$

Here, $collectStates_G : GExp_G \to \mathcal{P}Exp_G$ is a function that collects and normalizes the G-expressions appearing in a *structured* state $\lambda_G(\varepsilon) \in GExp_G$. We can now formulate the converse of Theorem 5.

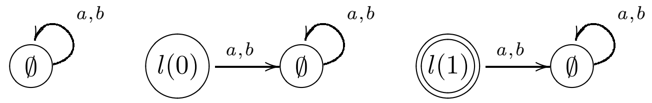
Theorem 6. *Let G be a polynomial functor. For every $\varepsilon \in Exp_G$, $\Delta_G(\varepsilon) = (S, g)$ is such that S is finite and there exists $s \in S$ with $\varepsilon \sim s$.*

Proof. First note that $\varepsilon \sim norm_G(\varepsilon)$ and $norm_G(\varepsilon) \in S$, by the definition of Δ_G and D_G. For space reasons, we omit the proof that S is finite, *i.e.* that $D_G(\{norm_G(\varepsilon)\}, \emptyset)$ terminates (details can be found in [2]).

5.3 Examples

In this subsection we will illustrate the synthesis algorithm presented above. For simplicity, we will consider deterministic and partial automata expressions over $A = \{a, b\}$.

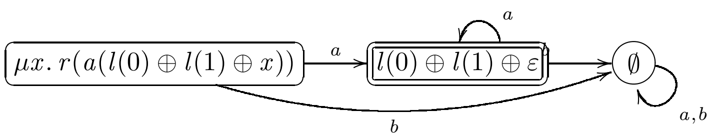
Let us start by showing the synthesised automata for the most simple deterministic expressions – $\emptyset$, $l(0)$ and $l(1)$.



It is interesting to make the parallel with the traditional regular expressions and remark that the first two automata recognize the empty language $\{\}$ and the last the language $\{\epsilon\}$ containing only the empty word.

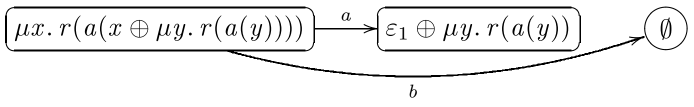
An important remark is that the automata generated are not minimal (for instance, the automata $l(0)$ and $\emptyset$ are bisimilar). Our goal has been to generate a finite automaton from a regular expression. From this the minimal automaton can always be obtained by identifying bisimilar states.

For an example of an expression containing fixpoints, consider $\varepsilon = \mu x.\ r(a(l(0) \oplus l(1) \oplus x))$. One can easily compute the synthesised automaton:

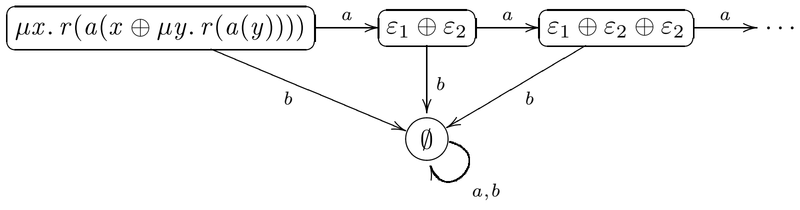

$\mu x.\ r(a(l(0) \oplus l(1) \oplus x))$
a
$l(0) \oplus l(1) \oplus \varepsilon$
a
$\emptyset$
a, b
b

and observe that it recognizes the language aa^*. Here, the role of the join-semilattice structure is also visible: $l(0) \oplus l(1) \oplus \varepsilon$ specifies that the current state is both final and non-final. Because $1 \vee 0 = 1$ the state is set to be final.

As a last example of deterministic expressions consider $\varepsilon_1 = \mu x.\ r(a(x \oplus \mu y.\ r(a(y))))$. Applying λ_D to ε_1 one gets the following (partial) automaton:


$\mu x.\ r(a(x \oplus \mu y.\ r(a(y))))$
a
$\varepsilon_1 \oplus \mu y.\ r(a(y))$
$\emptyset$
b

Calculating $\lambda_D(\varepsilon_1 \oplus \mu y.\ r(a(y)))(a)$ yields $\langle 0, \varepsilon_1 \oplus \mu y.\ r(a(y)) \oplus \mu y.\ r(a(y))\rangle$. When applying *collectStates*$_G$, the expression $\varepsilon_1 \oplus \mu y.\ r(a(y)) \oplus \mu y.\ r(a(y))$ will be normalized to $\varepsilon_1 \oplus \mu y.\ r(a(y))$, which is a state that already exists. Remark here the role of *norm* in guaranteeing termination. Without normalization, one would get the following infinite coalgebra ($\varepsilon_2 = \mu y.\ r(a(y))))$):


$\mu x.\ r(a(x \oplus \mu y.\ r(a(y))))$
a
$\varepsilon_1 \oplus \varepsilon_2$
a
$\varepsilon_1 \oplus \varepsilon_2 \oplus \varepsilon_2$
a
$\cdots$
b
b
b
$\emptyset$
a, b

Let us now see a few examples of synthesis for partial automata expressions, where we will illustrate the role of $\perp$ and $\top$. As before, let us first present the corresponding automata for simple expressions – $\emptyset$, $a\uparrow$, $a(\emptyset)$ and $a\uparrow \oplus b\uparrow$.

$$\begin{tikzcd}
\emptyset \arrow[r, "{a,b}"] & \boxed{\bot} & a\!\uparrow \arrow[r, "b"] & \boxed{\bot} & a(\emptyset) \arrow[r, "a"] \arrow[rr, bend left, "b"] & \emptyset \arrow[r, "{a,b}"] & \boxed{\bot} & \boxed{a\!\uparrow \oplus b\!\uparrow}
\end{tikzcd}$$

In the graphical representation of a partial automata (S, p), whenever $g(s)(a) \in \{\bot, \top\}$ we represent a transition, but note that $\bot \notin S$ and $\top \notin S$ (thus, the square box) and have no defined transitions.

Here, one can now observe how $\bot$ is used to encode underspecification, working as a kind of deadlock state. Note that in the first three expressions the behaviour for one or both of the inputs is missing, whereas in the last expression the specification is complete. The element $\top$ is used to deal with inconsistent specifications. For instance, consider the expression $a\!\uparrow \oplus\, b\!\uparrow \oplus\, a(a\!\uparrow \oplus\, b\!\uparrow)$. All inputs are specified, but note that at the outermost level input a appears in two different sub-expressions – $a\!\uparrow$ and $a(a\!\uparrow \oplus\, b\!\uparrow)$ – specifying at the same time that input a leads to successful termination and that it leads to a state where $a\!\uparrow \oplus\, b\!\uparrow$ holds, which is contradictory, giving rise to the following automaton.

$$\begin{tikzcd}
\boxed{a\!\uparrow \oplus b\!\uparrow \oplus a(a\!\uparrow \oplus b\!\uparrow)} \arrow[r, "a"] & \boxed{\top}
\end{tikzcd}$$

6 Conclusions

We have presented a generalization of Kleene's theorem for polynomial coalgebras. More precisely, we have introduced a language of expressions for polynomial coalgebras and we have shown that they constitute a precise syntactic description of deterministic systems, in the sense that every expression in the language is bisimilar to a state of a finite coalgebra and vice-versa (Theorems 5 and 6).

The language of expressions presented in this paper can be seen as an alternative to the classical regular expressions for deterministic automata and to KAT expressions [11] and as a generalization of previous work of the authors on Mealy machines [3].

As was pointed out by the referees of the present paper, Theorem 5 is closely related to the well known fact that, for polynomial functors, an element in a finite subcoalgebra of the final coalgebra can be characterised as a "finite tree with loops". This could in turn give rise to a different language of expressions $\varepsilon ::= x \mid \mu x.\varepsilon \mid \sigma(\varepsilon_1, ..., \varepsilon_n)$, where σ is an n-ary operation symbol in the signature corresponding to a polynomial functor (e.g., if $GX = 1 + X + X + X^2$ then the signature has one constant, two unary and one binary operation symbol). This alternative approach might seem simpler than the one taken in this paper but does not provide an operator for combining specifications as our $\oplus$ operator, and, more importantly, will not allow for an easy and modular axiomatization of bisimulation. Providing such a complete finite axiomatization generalizing the results presented in [9,5] is subject of our current research. This will provide a generalization of Kleene algebra to polynomial coalgebras.

Further, we would like to deal with non-deterministic systems (which amounts to include the powerset in our class of functors) and probabilistic systems.

In our language we have a fixpoint operator, $\mu x.\varepsilon$, and action prefixing, $a(\varepsilon)$, opposed to the use of star E^* and sequential composition E_1E_2 in classical regular expressions. We would like to study in more detail the precise relation between these two (equally expressive) syntactic formalisms. Ordinary regular expressions are closed under intersection and complement. We would like to study whether a similar result can be obtained for our language.

Coalgebraic modal logics (CML) [12] have been presented as a general theory for reasoning about transition systems. The connection between our language and CML is also subject of further study.

Acknowledgements. The authors are grateful to Dave Clarke, Helle Hansen, Clemens Kupke, Yde Venema and the anonymous referees for useful comments.

References

1. Aczel, P., Mendler, N.: A Final Coalgebra Theorem. In: Dybjer, P., Pitts, A.M., Pitt, D.H., Poigné, A., Rydeheard, D.E. (eds.) Category Theory and Computer Science. LNCS, vol. 389, pp. 357–365. Springer, Heidelberg (1989)
2. Bonsangue, M., Rutten, J., Silva, A.: Regular expressions for polynomial coalgebras. CWI Technical report E0703 (2007)
3. Bonsangue, M., Rutten, J., Silva, A.: Coalgebraic logic and synthesis of mealy machines. In: Amadio, R. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 231–245. Springer, Heidelberg (2008)
4. Brzozowski, J.A.: Derivatives of regular expressions. Journal of the ACM 11(4), 481–494 (1964)
5. Ésik, Z.: Axiomatizing the equational theory of regular tree languages (extended abstract). In: Meinel, C., Morvan, M. (eds.) STACS 1998. LNCS, vol. 1373, pp. 455–465. Springer, Heidelberg (1998)
6. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 3rd edn. Addison-Wesley Longman Publishing Co., Inc., Boston (2006)
7. Jacobs, B.: Many-sorted coalgebraic modal logic: a model-theoretic study. ITA 35(1), 31–59 (2001)
8. Kleene, S.: Representation of events in nerve nets and finite automata. Automata Studies, 3–42 (1956)
9. Kozen, D.: A completeness theorem for Kleene algebras and the algebra of regular events. In: Logic in Computer Science, pp. 214–225 (1991)
10. Kozen, D.: Automata and Computability. Springer, New York (1997)
11. Kozen, D.: On the coalgebraic theory of Kleene algebra with tests. Technical Report, Computing and Information Science, Cornell University (March 2008), `http://hdl.handle.net/1813/10173`
12. Kurz, A.: Coalgebras and Their Logics. SIGACT News 37(2), 57–77 (2006)
13. Rößiger, M.: Coalgebras and modal logic. ENTCS, 33 (2000)
14. Rutten, J.: Universal coalgebra: a theory of systems. TCS 249(1), 3–80 (2000)
15. Rutten, J.: Automata and coinduction (an exercise in coalgebra). In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 194–218. Springer, Heidelberg (1998)

Coalgebraic Hybrid Logic

Rob Myers^{1}, Dirk Pattinson^{1,*}, and Lutz Schröder^{2,**}

^{1} Department of Computing, Imperial College London
^{2} DFKI Bremen and Department of Computer Science, Universität Bremen

Abstract. We introduce a generic framework for hybrid logics, i.e. modal logics additionally featuring *nominals* and *satisfaction operators*, thus providing the necessary facilities for reasoning about individual states in a model. This framework, coalgebraic hybrid logic, works at the same level of generality as coalgebraic modal logic, and in particular subsumes, besides normal hybrid logics such as hybrid K, a wide variety of logics with non-normal modal operators such as probabilistic, graded, or coalitional modalities and non-monotonic conditionals. We prove a generic finite model property and an ensuing weak completeness result, and we give a semantic criterion for decidability in PSPACE. Moreover, we present a fully internalised PSPACE tableau calculus. These generic results are easily instantiated to particular hybrid logics and thus yield a wide range of new results, including e.g. decidability in PSPACE of probabilistic and graded hybrid logics.

Introduction

The ability to represent and reason about individuals is a core feature of many formalisms in the field of logic-based knowledge representation. Individuals may be persons, parts of the human body, or even positions in a strategic game. Reasoning about individuals is a prominent feature in description logics [3], and is supported by a number of reasoning engines, including Fact, Racer and Pellet [33,19,31]. Both description logics and the associated reasoning tools are based on *relational models*, usually Kripke structures with a fixed number of relations. While this is adequate for a large number of applications, description logics can neither formulate nor reason about statements of a non-relational nature, such as assertions involving quantitative uncertainty ('The likelihood that John is a son of Mary is greater than 23 %') or non-monotonic conditionals ('John normally goes to work on Mondays' — unless e.g. he decides to call in sick).

Features of this kind are usually catered for by specific logics, such as probabilistic modal logic [20] or conditional logic [14], neither of which admits a semantics in terms of Kripke structures. On the other hand, these logics cannot be used off the shelf in most applications in knowledge representation, as they lack facilities to represent and reason about individuals. Of course, one may opt to study extensions of these logics on a case-by-case basis. However, we can do better: both probabilistic modal logic and conditional logic, as well as many others including coalition logic [26], graded modal

* Partially supported by EPSRC grant EP/F031173/1.
** Work performed as part of the DFG project *Generic Algorithms and Complexity Bounds in Coalgebraic Modal Logic* (SCHR 1118/5-1).

logic [18], and logics for counterfactual reasoning [22], as well as most description logics naturally fit under the umbrella of *coalgebraic modal logic* [24]. On the semantical side, this is achieved by replacing the notion of *model* of a particular logic by *coalgebras* for an endofunctor T on sets. As we illustrate by means of examples, the semantics of particular logics then arises by instantiating the framework with a concrete endofunctor T. This opens the road for a uniform treatment of a large class of modal logics. In particular, reasoning about individuals in coalgebraic logic is tantamount to extending a large class of logics simultaneously.

This is the starting point of this paper. We introduce *coalgebraic hybrid logic*, that is, an extension of coalgebraic logics with both nominals (that denote individual states) and satisfaction operators (which are used to formulate assertions concerning specific individuals) in the same way that hybrid logic [2] extends the modal logic K. Our main results are concerned with completeness and complexity of coalgebraic hybrid logics. We do not treat the downarrow binder $\downarrow$, as hybrid logics involving $\downarrow$ are known to be undecidable, even in the special case of relational (Kripke) semantics [1]. As we are working in a general framework, these statements are formulated in terms of abstract *coherence conditions* that relate the syntax (modal operators and proof rules) and the semantics, which is abstractly given in terms of coalgebras. Conceptually, we show that the *same* coherence conditions that give rise to completeness and complexity results in the non-hybrid case also guarantee completeness and (the same) complexity bounds for the hybrid extension. In more detail, we prove completeness of hybrid coalgebraic logics both with respect to a Hilbert-style proof calculus in Section 2 and a cut-free sequent system 3. We exploit the latter to show that the satisfiability problem is decidable in polynomial space by means of backward proof search in Section 4 before we give a purely semantical account of complexity in Section 5. As the coherence conditions that guarantee completeness and complexity are already known to hold for a large class of logics, instantiations of the general framework give rise to a number of new results concerning particular logics:

– Both *hybrid conditional logic* and *hybrid probabilistic logic* are complete and decidable in PSPACE
– Moreover, *graded hybrid logic* is decidable in polynomial space with numbers coded in binary, and these results immediately carry over to a setting with role hierarchies, giving a new (tight) PSPACE upper bound for the description logic $\mathcal{ALCHOQ}$ over the empty TBox.
– The semantic analysis yields previously unknown PSPACE-upper bounds for *Presburger hybrid logic* [16] and a version of probabilistic modal logic featuring linear inequalities [17].

In a nutshell, the addition of nominals and satisfaction operators greatly increases the expressive power of (coalgebraic) modal logics and is still decidable in PSPACE, i.e. the same complexity class as the modal logic K. In particular, the ability to cater for a large number of logical features, such as quantitative uncertainty and non-monotonic conditionals in the coalgebraic framework offers a new (coalgebraic) perspective on description logics, with the vision of being able to represent a large class of concepts of information-theoretic relevance in a single framework.

1 Syntax and Semantics of Coalgebraic Hybrid Logic

To make our treatment parametric in the concrete syntax of any given modal logic, we fix a modal similarity type Λ consisting of modal operators with associated arities throughout. For given countably infinite and disjoint sets P of propositional variables and N of nominals, the set $\mathcal{F}(\Lambda)$ of *hybrid Λ-formulas* is given by the grammar

$$\mathcal{F}(\Lambda) \ni \phi, \psi ::= p \mid i \mid \phi \wedge \psi \mid \neg\phi \mid \heartsuit(\phi_1, \dots, \phi_n) \mid @_i\phi$$

where $p \in \mathsf{P}$, $i \in \mathsf{N}$ and $\heartsuit \in \Lambda$ is an n-ary modal operator. We use the standard definitions for the other propositional connectives $\rightarrow, \leftrightarrow, \vee$. The set of nominals occurring in a formula ϕ is denoted by $\mathsf{N}(\phi)$, and the nesting depth of modal operators (excluding satisfaction operators) by $\mathrm{rank}(\phi)$. A formula of the form $@_i\phi$ is called an *$@$-formula*. Semantically, nominals i denote individual states in a model, and an $@$-formula $@_i\phi$ stipulates that ϕ holds at i.

To reflect parametricity in the particular underlying logic also semantically, we equip hybrid logics with a *coalgebraic semantics* extending the standard coalgebraic semantics of modal logics [24]: we fix throughout a *Λ-structure* consisting of an endofunctor $T : \mathsf{Set} \rightarrow \mathsf{Set}$ on the category of sets, together with an assignment of an *n-ary predicate lifting* $[\![\heartsuit]\!]$ to every n-ary modal operator $\heartsuit \in \Lambda$, i.e. a set-indexed family of mappings $([\![\heartsuit]\!]_X : \mathcal{P}(X)^n \rightarrow \mathcal{P}(TX))_{X \in \mathsf{Set}}$ that satisfies

$$[\![\heartsuit]\!]_X \circ (f^{-1})^n = (Tf)^{-1} \circ [\![\heartsuit]\!]_Y$$

for all $f : X \rightarrow Y$. In categorical terms, $[\![\heartsuit]\!]$ is a natural transformation $\mathcal{Q}^n \rightarrow \mathcal{Q} \circ T^{op}$ where $\mathcal{Q} : \mathsf{Set}^{op} \rightarrow \mathsf{Set}$ is the contravariant powerset functor.

In this setting, T-coalgebras play the roles of *frames*. A *T-coalgebra* is a pair (C, γ) where C is a set of *states* and $\gamma : C \rightarrow TC$ is the *transition function*. If clear from the context, we identify a T-coalgebra (C, γ) with its state space C. A *(hybrid) T-model* (C, γ, π) consists of a T-coalgebra (C, γ) together with a *hybrid valuation* π, i.e. a map $\mathsf{P} \cup \mathsf{N} \rightarrow \mathcal{P}(C)$ that assigns singleton sets to all nominals $i \in \mathsf{N}$. We often identify the singleton set $\pi(i)$ with its unique element.

The semantics of $\mathcal{F}(\Lambda)$ is a satisfaction relation $\models$ between states $c \in C$ in hybrid T-models $M = (C, \gamma, \pi)$ and formulas $\phi \in \mathcal{F}(\Lambda)$, inductively defined as follows. For $x \in \mathsf{N} \cup \mathsf{P}$ and $i \in \mathsf{N}$, put

$$c, M \models x \text{ iff } c \in \pi(x) \qquad c, M \models @_i\phi \text{ iff } \pi(i), M \models \phi.$$

Modal operators are interpreted using their associated predicate liftings, that is,

$$c, M \models \heartsuit(\phi_1, \dots, \phi_n) \iff \gamma(c) \in [\![\heartsuit]\!]_C([\![\phi_1]\!]_M, \dots, [\![\phi_n]\!]_M)$$

where $\heartsuit \in \Lambda$ n-ary and $[\![\phi]\!]_M = \{c \in C \mid M, c \models \phi\}$ denotes the truth-set of ϕ relative to M. A formula ϕ is *satisfiable* if it is satisfied in some state in some model, and *valid*, written $\models \phi$, if it is satisfied in all states in all models.

The distinguishing feature of the coalgebraic approach to hybrid and modal logics is the parametricity in both the logical language and the notion of frame: concrete instantiations of the general framework, in other words a choice of modal operators Λ and a Λ-structure T, capture the semantics of a wide range of modal logics, as witnessed by the following examples.

Example 1

1. The hybrid version of the modal logic K, *hybrid* K for short, has a single unary modal operator $\Box$, interpreted over the structure given by the powerset functor $\mathcal{P}$ that takes a set X to its powerset $\mathcal{P}(X)$ and $[\![\Box]\!]_X(A) = \{B \in \mathcal{P}(X) \mid B \subseteq A\}$. It is clear that $\mathcal{P}$-coalgebras $(C, \gamma : C \to \mathcal{P}(C))$ are in 1-1 correspondence with Kripke frames, and that the coalgebraic definition of satisfaction specialises to the usual semantics of the box operator.

2. *Graded hybrid logic* has modal operators $\Diamond_k$ 'in more than k successors, it holds that'. It is interpreted over the functor $\mathcal{B}$ that takes a set X to the set $\mathcal{B}(X)$ of multisets over X, i.e. maps $B : X \to \mathbb{N} \cup \{\infty\}$, by $[\![\Diamond_k]\!]_X(A) = \{B \in \mathcal{B}(X) \mid \sum_{x \in A} B(x)\}$. This captures the semantics of graded modalities over *multigraphs* [15], which are precisely the $\mathcal{B}$-coalgebras. Unlike in the modal case [27], the multigraph semantics does not engender the same notion of satisfiability as the more standard Kripke semantics of graded modalities, as the latter validates all formulas $\neg\Diamond_1 i$, $i \in \mathsf{N}$. One can however polynomially encode Kripke semantics into multigraph semantics: a graded hybrid formula ϕ, w.l.o.g. an @-formula, is satisfiable over Kripke frames iff $\phi \wedge \bigwedge_{j \in \mathsf{N}(\phi)} @_j \bigwedge_{n < \mathrm{rank}(\phi)} \Box_0^n \bigwedge_{i \in \mathsf{N}(\phi)} \neg\Diamond_1 i$, where $\Box_k$ is defined as $\neg\Diamond_k\neg$ and $\Box_0^n$ denotes n boxes, is satisfiable over multigraphs. Thus, our completeness and complexity results for multigraph semantics derived below transfer to Kripke semantics. In particular they apply to many description logics, which commonly feature both nominals and graded modal operators (*qualified number restrictions*).

3. *Probabilistic hybrid logic*, the hybrid extension of probabilistic modal logic [21], has modal operators L_p 'in the next step, it holds with probability at least p that', for $p \in [0, 1] \cap \mathbb{Q}$. It is interpreted over the functor D_ω that maps a set X to the set of finitely-supported probability distributions on X by putting $[\![L_p]\!]_X(A) = \{P \in D_\omega(X) \mid PA \geq p\}$. Coalgebras for D_ω are just Markov chains. A simple valid formula of hybrid probabilistic logic is $L_p i \wedge L_q j \to L_q i \vee L_1(\neg(i \wedge j))$, $i, j \in \mathsf{N}$.

4. *Hybrid CK*, the hybrid extension of the basic conditional logic CK, has a single binary modal operator $\Rightarrow$, written in infix notation. Hybrid CK is interpreted over the functor Cf that maps a set X to the set of $\mathcal{P}(X) \to \mathcal{P}(X)$, whose coalgebras are selection function models [14], by putting $[\![\Rightarrow]\!]_X(A, B) = \{f : \mathcal{P}(X) \to \mathcal{P}(X) \mid f(A) \subseteq B\}$.

2 A Generic Complete Hilbert System

We proceed to present a Hilbert-system for coalgebraic hybrid logics, and prove its soundness and its weak completeness over finite models, provided that the logic at hand satisfies certain coherence conditions between the axiomatisation and the semantics — in fact the *same* conditions as in the modal case, which are easily verified *local* properties that can be verified without reference to T-models and are already known to hold for a large variety of logics [24,27]. We recall notation from earlier work:

Notation 2. For a set or multiset Σ of formulas and a set O of operators, we write $O\Sigma$ or $O(\Sigma)$ for the set or multiset of formulas arising by prefixing elements of Σ with exactly one operator from O; e.g. $\Lambda(\Sigma) = \{\heartsuit(\phi_1, \dots, \phi_n) \mid \heartsuit \in \Lambda\ n\text{-ary}, \phi_1, \dots, \phi_n \in$

$\Sigma\}$ and $@\Sigma := \{@_i \mid i \in \mathsf{N}\})(\Sigma) = \{@_i\phi \mid i \in \mathsf{N}, \phi \in \Sigma\}$. Moreover, $\mathsf{Prop}(\Sigma)$ denotes the set of propositional combinations of Σ-formulas. For a propositional formula $\phi \in \mathsf{Prop}(\mathsf{P} \cup \mathsf{N})$, we write $\kappa \models \phi$ if $\kappa : \mathsf{P} \cup \mathsf{N} \to 2 = \{\bot, \top\}$ is a satisfying valuation for ϕ, and $X, \tau \models \phi$ if ϕ evaluates to $\top$ in the boolean algebra $\mathcal{P}(X)$ under a hybrid valuation $\tau : \mathsf{P} \cup \mathsf{N} \to \mathcal{P}(X)$. For $\psi \in \mathsf{Prop}(\Lambda(\mathsf{P} \cup \mathsf{N}))$, the interpretation $[\![\psi]\!]_{TX,\tau}$ of ψ in the boolean algebra $\mathcal{P}(TX)$ under τ is the inductive extension of the assignment $[\![\heartsuit(p_1, \dots, p_n)]\!]_{TX,\tau} = [\![\heartsuit]\!]_X(\tau(p_1), \dots, \tau(p_n))$. We write $TX, \tau \models \psi$ if $[\![\psi]\!]_{TX,\tau} = TX$, and $t \models_{TX,\tau} \psi$ if $t \in [\![\psi]\!]_{TX,\tau}$.

Proof systems for coalgebraic logics are most conveniently described in terms of one-step rules, for which the announced coherence conditions take the following form.

Definition 3. A *one-step rule* over Λ is an $n+1$-tuple $r = (\Gamma_1, \dots, \Gamma_n, \Gamma_0)$, written

$$\frac{\Gamma_1 \dots \Gamma_n}{\Gamma_0},$$

where $\Gamma_0 \subseteq \Lambda(\mathsf{P}) \cup \neg\Lambda(\mathsf{P})$ and $\Gamma_1, \dots, \Gamma_n \subseteq \mathsf{P} \cup \neg\mathsf{P}$. The rule $\Gamma_1 \dots \Gamma_n / \Gamma_0$ is *one-step sound* if $TX, \tau \models \bigvee \Gamma_0$ whenever $X, \tau \models \bigvee \Gamma_i$ for all $i = 1, \dots, n$. A set $\mathcal{R}$ of one-step rules is *one-step (cut-free) complete* if whenever $TX, \tau \models \bigvee \Delta$ for $\Delta \subseteq \Lambda(\mathsf{P}) \cup \neg\Lambda(\mathsf{P})$ then $\bigvee \Delta$ is propositionally entailed by formulas $\bigvee \Gamma_0\sigma$ ($\Delta \supseteq \Gamma_0$ is a super-sequent of a single sequent Γ_0) where $\Gamma_1 \dots \Gamma_n/\Gamma_0$ is in $\mathcal{R}$ and $\sigma : \mathsf{P} \to \mathsf{Prop}(\mathsf{P})$ is a substitution such that $X, \tau \models \bigvee \Gamma_i\sigma$ for all i.

Example 4. One-step cut-free complete rule sets, which can just be inherited from the corresponding modal systems, for graded and probabilistic logics, conditional logics, and many others are found in [28,25]. We recall that the one-step cut-free complete rule set for (hybrid) K consists of the rules

$$\frac{\neg a_1, \dots, \neg a_n, b}{\neg\Box a_1, \dots, \neg\Box a_n, \Box b}\ (n \geq 0)$$

A set $\mathcal{R}$ of one-step rules now gives rise to a Hilbert-system $\mathsf{H}\mathcal{R}$ by adjoining the congruence rule (replacement of equivalents under modal operators), propositional tautologies, modus ponens, uniform substitution, standard axioms for satisfaction operators $@_i$ stating that $@_i$ commutes with all propositional connectives ($\neg @_i\bot$, $\neg @_i\phi \leftrightarrow @_i\neg\phi$, $@_i(\phi \wedge \psi) \leftrightarrow (@_i\phi \wedge @_i\psi)$) and that $i \sim j :\equiv @_i j$ defines an equivalence relation on nominals ($@_i i$, $@_i j \leftrightarrow @_j i$, $@_i k \wedge @_j p \to @_i p$), the @-generalisation rule $a/@_i a$, and the axioms

$$\begin{array}{rl} \text{(in)} & i \wedge \phi \to @_i\phi \\ \text{(mob)} & @_i p \to (\heartsuit(q_1, \dots, q_n) \leftrightarrow \heartsuit(@_i p \wedge q_1, \dots, @_i p \wedge q_n)) \end{array}$$

called @-*introduction* (in) and *make-or-break* (mob), respectively. We write $\mathsf{H}\mathcal{R} \vdash \phi$ if ϕ is derivable in this system. The (mob) axiom captures the fact that the truth set of an @-formula is either empty or the whole model; in the case of hybrid K, it is deductively equivalent to the standard back axiom $@_i\phi \to \Box @_i\phi$.

From @-introduction one easily derives @-elimination $i \wedge @_i a \to a$. Moreover, the (mob) axiom readily generalises to any @-formula in place of $@_i p$. Hence we can derive a *relativised congruence rule*

$$(@cong)\ \frac{\phi \to ((a_1 \leftrightarrow b_1) \wedge \cdots \wedge (a_n \leftrightarrow b_n))}{\phi \to (\heartsuit(a_1, \dots, a_n) \leftrightarrow \heartsuit(b_1, \dots, b_n))}\ (\phi\ @\text{-formula})$$

It is clear that $\mathsf{H}\mathcal{R}$ is sound if all rules in $\mathcal{R}$ are one-step sound. For the remainder of the section, we assume that $\mathcal{R}$ *is one-step complete* and proceed to prove weak completeness of $\mathsf{H}\mathcal{R}$ over finite models by extending the finite model construction of [27], i.e. by constructing a model for a consistent formula ϕ whose states are maximally consistent subsets of a suitable finite closure of ϕ.

In the following, let Σ be a finite and *closed* set of formulas, i.e. closed under subformulas (where we count i as a subformula of $@_i\rho$), negation, and $@_i$ for $i \in \mathsf{N}(\Sigma)$, where we identify $\neg\neg\phi$ with ϕ, $\neg @_i\phi$ with $@_i\neg\phi$, and $@_i@_j\phi$ with $@_j\phi$. Denote by $@\Sigma$ the set of @-formulas in Σ. Fix a maximally $\mathsf{H}\mathcal{R}$-consistent subset K of $@\Sigma$. Let S denote the set of *atoms*, i.e. maximally $\mathsf{H}\mathcal{R}$-consistent subsets of Σ, and let S_K be the set of K*-atoms*, i.e. atoms containing K.

Lemma 5. *Let $\phi \in \mathsf{Prop}(\mathsf{P})$, and let σ be a Σ-substitution. Then $K \to \phi\sigma$ is derivable iff $S_K, \tau \models \phi$, where τ is the $\mathcal{P}(S_K)$-valuation $\tau(a) = \{A \in S_K \mid \sigma(a) \in A\}$.*

We define a hybrid $\mathcal{P}(S_K)$-valuation π in the standard way by $\pi(a) = \{A \mid a \in A\}$ for $a \in \mathsf{P} \cup \mathsf{N}(\Sigma)$, and taking $\pi(i)$ to be an arbitrary singleton set otherwise.

Lemma 6. *The valuation τ is hybrid, i.e. $\tau(i)$ is a singleton for each $i \in \mathsf{N}(\Sigma)$, namely $\tau(i) = \{K_i\}$, where $K_i = \{\phi \in \Sigma \mid @_i\phi \in K\}$.*

Call a coalgebra (S_K, ξ) *coherent* if for all $\heartsuit(\phi_1, \dots, \phi_n) \in \Sigma$, $A \in S_K$,

$$\xi(A) \in [\![\heartsuit]\!](\hat{\phi}_1, \dots, \hat{\phi}_n) \iff \heartsuit(\phi_1, \dots, \phi_n) \in A$$

where $\hat{\phi} = \{A \in S_K \mid \phi \in A\}$. Making crucial use of the relativised congruence rule (@-cong), one proves

Lemma 7 (Relativised existence lemma). *There exists a coherent coalgebra (S_K, ξ).*

It is then straightforward to establish

Lemma 8 (Truth lemma). *If (S_K, ξ) is coherent, then $A \models_{(S_K, \xi, \pi)} \phi$ iff $\phi \in A$ for all $\phi \in \Sigma$, $A \in S_K$.*

Theorem 9 (Weak completeness of $\mathsf{H}\mathcal{R}$ over finite models). *Every $\mathsf{H}\mathcal{R}$-consistent hybrid formula ϕ is satisfiable in a T-model with at most $2^{|\phi|}$ states.*

The above theorem establishes weak completeness of the Hilbert calculus not only for hybrid K, but also for graded and probabilistic hybrid logic and hybrid CK, as well as hybrid versions of many other modal logics treated e.g. in [27,28].

3 Hybrid Sequent Calculi and Cut Elimination

We now introduce a sound and complete sequent calculus for coalgebraic hybrid logics. Completeness of the calculus, initially with the cut-rule, is proved using completeness of the Hilbert system of the previous section by showing that both provability predicates coincide. Subsequently, we prove cut elimination.

The sequent calculus shares one characteristic trait with the tableaux calculus studied in [6]: sequents are composed of @-prefixed formulas. As a consequence, the sequent calculus presented here can be understood as (the dual of) a labelled tableau [34] by reading an @-prefixed formula $@_i\phi$ as a labelled sequent $i : \phi$. As universal validity of $\phi \in \mathcal{F}(\Lambda)$ is equivalent to validity of $@_t\phi$ for some $t \in \mathsf{N}$ which doesn't occur in ϕ, this suffices to obtain completeness. The main difference between our calculus and those found in the literature is its modularity: it comes about by extending a standard, cut-free sequent calculus for the underlying logic by means of proof rules for nominals and satisfaction operators. We first consider a sequent calculus with the cut-rule and relegate cut-elimination to the next section. We begin by fixing our notation regarding sequent calculi; in the tradition of the sequent calculus literature, we denote formulas by capital letters $A, B, \ldots$ for the next two sections.

Definition 10. If $\Sigma \subseteq @\mathcal{F}(\Lambda)$ is a set of formulas, a *Σ-sequent* is a finite multiset of formulas in Σ. We write $\mathcal{S}(\Sigma)$ for the set of Σ-sequents, and $\mathcal{S}$ for the set of $@\mathcal{F}(\Lambda)$-sequents. If $\Gamma, \Delta \in \mathcal{S}$, then Γ, Δ denotes their multiset union, and we identify the singleton sequent $\{A\}$ with A for $A \in @\mathcal{F}(\Lambda)$. We put $\mathrm{rank}(\Gamma) = \max\{\mathrm{rank}(A) \mid A \in \Gamma\}$.

By virtue of the above definition, all elements of a sequent are necessarily @-prefixed formulas. As the modal logics we consider are extensions of classical propositional logic, it is most convenient to use a right-handed (or Gentzen-Schütte) calculus where a sequent intuitively stands for the disjunction of its elements. The rules for propositional reasoning and reasoning about names then take the following form:

$$(\mathrm{Ax})\quad @_t\neg A, @_t A, \Gamma \qquad (\mathrm{Ref})\quad @_t t, \Gamma \qquad (@\top)\quad @_t\top, \Gamma$$

$$(\neg\neg)\quad \frac{@_t A, \Gamma}{@_t\neg\neg A, \Gamma} \qquad\qquad (\wedge)\quad \frac{@_t A, \Gamma \qquad @_t B, \Gamma}{@_t(A \wedge B), \Gamma}$$

$$(\neg\wedge)\quad \frac{@_t\neg A, @_t\neg B, \Gamma}{@_t\neg(A \wedge B), \Gamma} \qquad\qquad (\mathrm{At})\quad \frac{@_t A, \Gamma}{@_s @_t A, \Gamma}$$

$$(\mathrm{Sd})\quad \frac{@_s\neg A, \Gamma}{@_t\neg @_s A, \Gamma} \qquad\qquad (\mathrm{Eq})\quad \frac{\Gamma[t := i]}{@_t\neg i, \Gamma}$$

In the above, $s, t \in \mathsf{N}$, $A, B \in @\mathcal{F}(\Lambda)$, and $\Gamma \in \mathcal{S}(@\mathcal{F}(\Lambda))$.

In combination with a set $\mathcal{R}$ of one-step sound one-step rules, which we fix throughout, we obtain the following notion of derivability.

Definition 11. The set of $\mathcal{R}$-derivable sequents is the least set that

- contains all instances of (Ax), (Ref) and $(@\top)$
- is closed under the rules $(\neg\neg)$, $(\wedge)$, $(\neg\wedge)$, (At), (Sd) and (Eq)
- is closed under the rules

$$(R) \quad \frac{@_n\Gamma_1\sigma, @_t\Gamma_0\sigma, \Delta \quad \ldots \quad @_n\Gamma_k\sigma, @_t\Gamma_0\sigma, \Delta}{@_t\Gamma_0\sigma, \Delta} (n \notin \Gamma_0\sigma, \ldots, \Gamma_k\sigma, \Delta)$$

where $\Gamma_1, \ldots, \Gamma_k/\Gamma_0 \in \mathcal{R}$, σ is a substitution and $\Delta \in \mathcal{S}$. In the side condition, $n \notin \Sigma$ denotes that the nominal n does not occur in the sequent Σ. The instances of the rule schema (R) are called *modal rules* and the remaining rules are referred to as *static rules*.

We write $\mathsf{G}\mathcal{R} \vdash \Gamma$ if Γ is $\mathcal{R}$-derivable and $\mathsf{G}\mathcal{R}$ for the above set of rules. If Γ is $\mathcal{R}$-derivable with additional help of the cut-rule

$$(\mathrm{cut}) \quad \frac{\Gamma, @_tA \qquad \Delta, @_t\neg A}{\Gamma, \Delta}$$

then this is denoted by $\mathsf{G}\mathcal{R}\mathsf{C} \vdash \Gamma$. We write $\mathsf{G}\mathcal{R} \vdash_n \Gamma$ if there exists a proof tree with end sequent Γ where (R) has been applied at most n times on any branch.

Proposition 12 (Soundness). *$\bigvee \Gamma$ is valid whenever $\mathsf{G}\mathcal{R}\mathsf{C} \vdash \Gamma$.*

The completeness of $\mathsf{G}\mathcal{R}\mathsf{C}$ is witnessed by the fact that every proof in the (complete) system $\mathsf{H}\mathcal{R}$ can be simulated. More precisely:

Theorem 13. *Let $\mathsf{H}\mathcal{R} \vdash A$. Then for every $t \notin A$, $\mathsf{G}\mathcal{R}\mathsf{C} \vdash @_tA$.*

Together with the completeness theorem for the Hilbert-system, we obtain completeness as an easy corollary.

Theorem 14. *Let $\mathcal{R}$ be one-step complete. Then for every $\Gamma \in \mathcal{S}$, $\mathsf{G}\mathcal{R}\mathsf{C} \vdash \Gamma$ whenever $\models \bigvee \Gamma$.*

Our main motivation for introducing the system $\mathsf{G}\mathcal{R}$ is to determine the complexity of the satisfiability problem by means of proof search. Hence, completeness of $\mathsf{G}\mathcal{R}\mathsf{C}$ is insufficient, as the use of the cut-rule leads to an infinite search space. Our next goal is therefore cut-elimination.

Cut elimination is subject to a number of structural properties that are readily established inductively. We begin with the inversion lemma.

Lemma 15 (Inversion Lemma). *All static rules of $\mathsf{G}\mathcal{R}$ are invertible, i.e. if $\Gamma_1 \ldots \Gamma_k/\Gamma_0$ is a static rule of $\mathsf{G}\mathcal{R}$, then $\mathsf{G}\mathcal{R} \vdash_k \Gamma_0$ iff $\mathsf{G}\mathcal{R} \vdash_k \Gamma_i$ for all $i = 1, \ldots, n$ and all $k \geq 0$.*

It is easy to see that weakening is also admissible, and moreover does not increase the number of applications of (R) in a proof tree.

Concerning the admissibility of cut and contraction, we adapt the standard double induction method [32] and use an additional outermost induction on the modal depth of the endsequent. The key point here is to observe that an instance of cut or contraction in the principal sequent of an instance of (R) can be eliminated by virtue of one-step cutfree completeness using a different instance of (R).

Theorem 16. *Contraction and cut are admissible in $\mathsf{G}\mathcal{R}$, that is $\mathsf{G}\mathcal{R} \vdash_n \Gamma, A$ whenever $\mathsf{G}\mathcal{R} \vdash_n \Gamma, A, A$, and $\mathsf{G}\mathcal{R} \vdash_n \Gamma, \Delta$ whenever $\mathsf{G}\mathcal{R} \vdash_n \Gamma, A$ and $\mathsf{G}\mathcal{R} \vdash_n \Delta, \neg A$, for all $\Gamma, \Delta \in \mathcal{S}$ and all $A \in @\mathcal{F}(\Lambda)$.*

The proof establishes moreover that neither cut elimination nor contraction increase the number of applications of (R) along any branch of the proof tree.

4 Complexity of Proof Search

While cutfreeness is clearly essential to establish complexity bounds by means of proof search, the duplication of the conclusion in the application of the sequent version of a one-step rule remains a further potential source of non-termination. We now proceed to show that the height of a proof tree in the sequent calculus of Section 3 can be bounded polynomially in the size of the endsequent. As a consequence, we obtain a polynomial space bound for proof search, and – dually – for the satisfiability problem.

Definition 17. A *pseudo-subformula* of Γ is of the form B or $\neg B$ for a subformula B of A with $@_t A \in \Gamma$. The set of pseudo-subformulas of Γ is denoted by $\mathsf{PSF}(\Gamma)$. A *core formula* of Γ is a formula $@_t A$ such that $t \in \mathsf{N}(\Gamma)$ and A can be obtained from $B \in \mathsf{PSF}(\Gamma)$ by means of a sequence of renamings that only affect nominals in $\mathsf{N}(\Gamma)$, formally $A = B[i_1 := j_1] \dots [i_n := j_n]$ where $i_1, \dots, i_n, j_1, \dots, j_k \in \mathsf{N}(\Gamma)$ and $\mathsf{N}(\Gamma) = \bigcup\{\mathsf{N}(A) \mid A \in \Gamma\}$.

Lemma 18. *Let $\mathsf{G}\mathcal{R} \vdash \Gamma$. Then the collection of formulas appearing on any branch of a proof tree of Γ contains at most $|\mathsf{PSF}(\Gamma)| \cdot |\mathsf{N}(\Gamma)|^2$ core formulas.*

The previous lemma allows us to argue that we can eliminate branches of the proof tree in case they do not add new core formulas.

Theorem 19. *Let $\mathsf{G}\mathcal{R} \vdash \Gamma$. Then Γ has a proof where at most $|\mathsf{N}(\Gamma)|^2 \cdot |\mathsf{PSF}(\Gamma)| \cdot \mathrm{rank}(\Gamma)$ instances of (R) are applied on every branch.*

For the proof of this theorem, we adopt the technique of [8] and attach a label $(t \to n)$ to an instance of the rule (R) as in Definition 11. Note that the modal depth of the formula decreases when we move from conclusion to premise. We next show that it suffices to consider applications of (R) where either a new core formula is introduced or a new nominal which is used higher up in the branch. In this situation, the collection of labels $(t \to n)$ forms a forest with at most $|\mathsf{N}(\Gamma)|^2 \cdot |\mathsf{PSF}(\Gamma)|$ leaves by Lemma 18, as every leaf corresponds to an application of (R) that either introduces a new core formula or has empty premise. As the modal depth decreases with each application of (R), this gives an overall bound of $|\mathsf{N}(\Gamma)|^2 \cdot |\mathsf{PSF}(\Gamma)| \cdot \mathrm{rank}(\Gamma)$ on the number of times (R) was applied.

This bound on the proof depth implies that proof search is in PSPACE, provided the rules are such that the set of possible premises of rule applications can be computed from the conclusions in nondeterministic polynomial time. We adapt the treatment of [28] to the sequent format of one-step rules used here. Recall that a nondeterministic polynomial time multivalued (NPMV) function [9] is a function $f : \Sigma^* \to \mathcal{P}\Delta^*$, where Σ and Δ are alphabets, such that there exists a polynomial p such that $|y| \leq p(|x|)$ for all $y \in f(x)$, where $|\cdot|$ denotes size, and the graph $\{(x, y) \mid y \in f(x)\}$ of f is in NP. With a view to implementing proof search on an alternating turing machine [13], this leads to the following definition.

Definition 20. Let sequents be represented in Σ^* for a finite alphabet Σ. The rule set $\mathcal{R}$ is *tractable* if there are NPMV functions $f : \Sigma^* \to \mathcal{P}(\Sigma^*)$ and $g : \Sigma^* \to \mathcal{P}(\Sigma^*)$ such that $\{\{\Gamma_1, \dots, \Gamma_n\} \mid \Gamma_1, \dots, \Gamma_n / \Gamma \in \mathsf{G}\mathcal{R}\} = \{g(x) \mid x \in f(\Gamma)\}$ for all $\Gamma \in \mathcal{S}$.

This allows us to formulate the main result of the present section as follows:

Theorem 21. *Let $\mathcal{R}$ be one-step sound, one-step cut-free complete and tractable. Then satisfiablity of $A \in \mathcal{F}(\Lambda)$ is in $PSPACE$.*

Example 22. As all hybrid logics of Example 1 have previously [28,25] been equipped with rule sets satisfying the assumptions of Theorem 21, they are decidable in polynomial space. In the case of hybrid K, this re-proves a known result [2]. The PSPACE-bounds for graded and probabilistic hybrid logic and for hybrid CK are new.

5 Shallow Models and PSPACE Algorithms

Next, we establish a semantics-based criterion for a hybrid logic to be decidable in PSPACE, thus complementing the sequent calculus based approach above by a method that applies also to logics for which no tractable cut-free axiomatisation is known. To this end, we extend the shallow model construction of [29] to the hybrid case. The extension is quite non-trivial for two reasons: shallow models are now forest-shaped rather than tree-shaped; and moreover they are not perfect forests in that they may have loops into the roots. The coalgebraic constructions require the novel concept of *fragments*, i.e. models based on partial coalgebras. We assume that Λ is equipped with a size measure, thus inducing a size measure $|\cdot|$ on $\mathcal{F}(\Lambda)$, with numbers coded in binary. As we now leave the sequent calculus context, we return to designating formulas by small greek letters.

Definition 23. A model *satisfies* a set K of @-formulas if its states satisfy K. A formula ϕ is *K-satisfiable* if it is satisfiable in some model satisfying K, and *K-valid* if it is valid in all models satisfying K.

Let ϕ be satisfied in a model M. Then ϕ is K-satisfiable, where K is the set of @-formulas satisfied in M. Moreover, ϕ is satisfiable iff $@_t\phi$ is satisfiable for t fresh. Thus, we assume w.l.o.g. that ϕ is an @-formula. Then K entails ϕ, so that we can forget about ϕ and concentrate on models satisfying K. In the following we fix a finite closed set Σ and a maximally satisfiable set $K \subseteq @\Sigma$, and we put $N = \mathsf{N}(\Sigma)$. For $i \in N$, we put $K_i = \{\rho \mid @_i\rho \in K\}$ as in Section 2.

Definition 24. A hybrid formula is *@-free* if it does not contain occurrences of @. A set of @-formulas is *@-eliminated* if it consists of formulas $@_i\rho$ with ρ @-free. For $\rho \in \Sigma$, $\rho[K]$ denotes the @-free formula obtained by replacing every subformula $@_i\chi$ of ρ not contained in further occurrences of @ by $\top$ if $@_i\chi \in K$, and by $\bot$ otherwise. The *@-eliminated form* $K[K]$ of K is the @-eliminated set $\{@_i\rho[K] \mid @_i\rho \in K\}$.

Lemma 25 (@-Elimination). *A model satisfies K iff it satisfies $K[K]$.*

By @-elimination, *we may henceforth assume that K is @-eliminated* and hence that the K_i are @-free. We wish to construct a model which satisfies K and which is shallow in the following sense:

Definition 26. A *supporting Kripke frame* of a T-model (C, γ, π) is a Kripke frame (C, R) such that $\gamma(c) \in T\{d \mid cRd\} \subseteq TC$ for all $c \in C$. A T-model of K is *shallow* if it has a supporting Kripke frame which is a forest of depth at most the rank of K up to loops into the roots, and whose roots have names in N.

We now introduce the crucial notion of fragment:

Definition 27. A *K-fragment* is a *partial T-model*, i.e. a triple $F = (C, \gamma, \pi)$ consisting of the same data as a T-model except that $\gamma : C \to TC$ is a *partial* map, with $K_i \in C$ for all $i \in N$, such that γ is undefined precisely on the K_i and $\pi(i) = \{K_i\}$ for $i \in N$. We define *K-fragmentary satisfaction* $c \models_F^K \rho$ of $\rho \in \Sigma$ in $c \in C$ by

$$K_i \models_F^K \rho \quad \text{iff} \quad \rho \in K_i$$

for $i \in N$, and by the usual recursive clauses for the other states. We put $[\![\rho]\!]_F^K = \{c \mid c \models_F^K \rho\}$. We say that F is a *shallow K-fragment model* of $\rho \in \Sigma$ if F has a supporting Kripke frame (defined in analogy to Definition 26) which is, up to possible isolated states K_i, a tree with root r of depth at most the rank of ρ such that $r \models_F^K \rho$.

Lemma 28. *Every @-free K-satisfiable $\psi \in \mathsf{Prop}(\Sigma)$ has an exponentially branching shallow K-fragment model.*

The proof is largely analogous to the modal shallow model theorem [29], the crucial point being that one need not yet define the coalgebra structure on the named states K_i.

Definition 29. Let $\kappa : N \to (V \cup N) \to 2$ be an N-indexed family of valuations for $V \cup N$, where $V \subseteq \mathsf{P}$. A *one-step (κ-)model* (X, τ, t) over V consists of a set X, a hybrid $\mathcal{P}(X)$-valuation τ for $V \cup N$, and $t \in TX$ (such that $\tau(i) \subseteq \tau(a)$ iff $\kappa(i)(a) = \top$ for all $i \in N, a \in V$). A *one-step N-pair* (η, ψ) over V consists of formulas $\psi \in \mathsf{Prop}(\Lambda(V \cup N))$ and $\eta \in \mathsf{Prop}(V \cup N)$. We say that (X, τ, t) is a *one-step (κ-)model of* (η, ψ) if $X, \tau \models \eta$ and $t \models_{TX,\tau} \psi$. The pair (η, ψ) is *one-step (κ-)satisfiable* if it has a one-step (κ-)model, and *one-step satisfiable over* (X, τ) if it has a one-step model of the form (X, τ, t).

Lemma 30. *A one-step N-pair (η, ψ) over V is one-step κ-satisfiable for $\kappa : N \to (V \cup N) \to 2$ iff it is one-step κ-satisfiable over the set*

$$X = \{\iota : V \cup N \to 2 \mid \iota \models \eta; \iota(i) = \top \Rightarrow \iota(a) = \kappa(i)(a) \textit{ for all } i \in N, a \in V\},$$

equipped with the (hybrid) valuation $\tau(b) = \{\iota \in X \mid \iota(b) = \top\}$ for all $b \in V \cup N$.

Definition 31. For $x \in X$ and a hybrid $\mathcal{P}(X)$-valuation τ for $V \cup N$, where $V \subseteq \mathsf{P}$, we put $\mathsf{Th}_\tau(x) \equiv \bigwedge_{x \in \tau(a)} a \wedge \bigwedge_{x \notin \tau(a)} \neg a$, where a ranges over $V \cup N$.

Theorem 32. *The (satisfiable) set K is satisfiable in an exponentially branching shallow T-model.*

In the proof, one constructs a forest-shaped model with roots K_i. One generates suitable exponential-size one-step models according to Lemma 30 that induce the coalgebra structure on the K_i, and then attaches shallow K-fragments obtained from Lemma 28 at the arising successor states, merging identically named states throughout the process.

This construction informs the design of a decision procedure for satisfiability of a hybrid formula ϕ. The algorithm first performs a non-deterministic reduction to satisfiability of $K \subseteq @\Sigma$ as above and @-elimination, and then reduces satisfiability of K to fragmentary K-satisfiability of certain formulas $\psi \in \mathsf{Prop}(\Sigma)$. The latter is decided by the following recursive procedure.

Algorithm 33 (Check fragmentary K-satisfiability of $\psi \in \mathsf{Prop}(\Sigma)$)

1. Decompose $\psi \equiv \psi_0\sigma$ with $\psi_0 \in \Lambda(V) \cup \neg\Lambda(V)$ and σ a Σ-substitution for $V \subseteq \mathsf{P}$.
2. Recursively compute the *propositional theory* $\eta \in \mathsf{Prop}(V \cup N)$, $N = \mathsf{N}(\Sigma)$, of σ as the disjunction of all maximal conjunctive clauses χ over $V \cup N$ (i.e. for each $b \in V \cup N$, χ contains either b or $\neg b$) such that $\chi\sigma$ is fragmentarily K-satisfiable.
3. Check that (η, ψ_0) is one-step κ-satisfiable, where for $i \in N$ and $a \in V$, $\kappa(i)(a) = \top$ iff $@_i\sigma(a) \in K$.

In the implementation of this algorithm (and analogously in the reduction step that precedes it), one cannot keep the whole (potentially exponential-sized) formula η in memory at once. Instead, read access to η by the one-step satisfiability checking procedure queries whether individual conjunctive clauses χ belong to η. Since the recursion depth is bounded by $\mathrm{rank}(\psi)$, such an implementation will run in PSPACE if one-step satisfiability checking can be performed in polynomial space. The exact definition of the relevant decision problem is the following:

Definition 34. The *strict one-step satisfiability* problem is to decide whether a one-step N-pair (η, ψ) over V is one-step κ-satisfiable for a given $\kappa : N \to (V \cup N) \to 2$, where the input size is defined to be $|\psi|$, and η is represented as a disjunctive set of maximal conjunctive clauses over $V \cup N$ and stored on an input tape (which does not count towards space consumption). The *lax one-step satisfiability* problem is the same decision problem, but with input size $|(\eta, \psi)|$.

We thus have

Theorem 35. *If strict one-step satisfiability is in PSPACE, then the satisfiability problem of $\mathcal{L}$ is in PSPACE.*

In concrete applications, the above condition may be established either directly or with the help of a local small model property:

Definition 36. The *one-step polysize model property (OSPMP)* holds if there is a polynomial p such that every one-step κ-satisfiable pair (η, ψ), $\kappa : N \to (V \cup N) \to 2$, has a one-step κ-model (X, τ, t) such that $|X| \leq p(|\psi| + |N|)$.

Theorem 37. *Under the OSPMP, strict one-step satisfiability is in PSPACE iff lax one-step satisfiability is in PSPACE.*

Example 38. In all hybrid logics of Example 1 (and many others), strict one-step satisfiability is in PSPACE, which reproves the PSPACE upper bounds of Example 22. This is established in each case in essentially the same way as in the purely modal version as carried out in [29]. In most examples, this involves application of Theorem 37, with the crucial step being the proof of the OSPMP (lax one-step satisfiability is typically even in NP [27]), the only essential change w.r.t. the modal case being that states with names in N are retained in the construction of small submodels of given one-step models. This applies moreover to the extension of probabilistic hybrid logic with linear inequalities [17], which is thus newly established to be in PSPACE.

One notable case that requires a direct proof that strict one-step satisfiability is in PSPACE is graded hybrid logic. Although this, too, is largely analogous to the modal case, we briefly sketch the argument for the sake of illustration. Thus, let (η, ψ) be a one-step N-pair over V in graded hybrid logic, and let $\kappa : N \to (V \cup N) \to 2$. One-step satisfiability of ψ over given (X, τ) amounts to solvability of a system of integer linear inequalities [27] whose coefficients occur in ψ. Such a system is solvable iff it has a solution whose components are of size at most $p(|\psi|)$, where p is a polynomial [23]. By Lemma 30, a non-deterministic algorithm which traverses a multiset over η, regarded as a set of maximal conjunctive clauses χ, by successively guessing the multiplicity of each χ and adding up the multiplicity for each $a \in V \cup N$ in order to finally check satisfaction of ψ, decides one-step κ-satisfiability of (η, ψ); the algorithm clearly uses polynomial space in $|\psi|$. This establishes the new result that *graded hybrid logic is in PSPACE* with numbers coded in binary.

These arguments extend straightforwardly to the hybrid extension of Presburger modal logic [16], which in generalisation of graded modalities features linear inequalities between satisfaction multiplicities of formulas. It is equally straightforward to add multiple agents, or multiple roles in description logic parlance, and role hierarchies, i.e. inclusion axioms between roles. In summary, we obtain the new result that *Presburger hybrid logic with role hierarchies is in PSPACE*. This logic is substantially stronger than e.g. the description logic $\mathcal{ALCHOQ}$, which features role hierarchies, nominals, and qualified number restrictions but not linear inequalities or satisfaction operators, so that as a corollary we obtain that *concept satisfiability over the empty TBox in $\mathcal{ALCHOQ}$ is in PSPACE*, a tight upper bound. According to the description logic complexity navigator [35], this bound was previously unknown (a PSPACE upper bound for the sublogic $\mathcal{ALCOQ}$, which excludes role hierarchies, is proved for unary coding of numbers, and claimed to extend to binary coding, in [4]). Note that Presburger-type logics and probabilistic logics with linear inequalities [17] are presently not amenable to syntactic complexity analysis, e.g. using the sequent calculus method presented above, as no cut-free axiomatisation is known.

6 Conclusions and Related Work

There is quite a large variety of different proof calculi for hybrid logics: one sees fully internalised [30] and labelled calculi [7], as well as natural deduction systems [10]. Apart from the fact that the results of this paper are applicable to a much larger variety of logics, the construction of the sequent calculus introduced in Section 3 is canonical, in the sense that the hybrid sequent rules correspond to a system of cut-free sequent rules for the non-hybrid system, which raises hopes that the same rules might also serve in even more expressive logical systems. The analysis of the sequent calculus is complemented by a semantic analysis which confirms the syntactic PSPACE bounds and yields new ones where the syntactic approach has not reached (yet); the precise relationship between the semantics-based algorithms and proof search remains to be explored in detail. The hybrid proof search algorithms will be integrated into the generic reasoner

CoLoSS [11]. Our complexity results allow inferring identical complexity bounds for a coalgebraic modal logic and its hybrid companion from the same set of conditions. In the more restrictive realm of relational semantics, one also has so-called *transfer* results that in particular allow inferring the complexity of a hybrid logic from that of its modal companion [5,12]. Coalgebraic transfer results are the subject of future investigation.

Acknowledgments. The authors wish to thank Erwin R. Catesbeiana for detailed explanations of his views on consistent formulas.

References

1. Areces, C., Blackburn, P., Marx, M.: A road-map on complexity for hybrid logics. In: Flum, J., Rodríguez-Artalejo, M. (eds.) CSL 1999. LNCS, vol. 1683, pp. 307–321. Springer, Heidelberg (1999)
2. Areces, C., ten Cate, B.: Hybrid logics. In: Blackburn, P., Wolter, F., van Benthem, J. (eds.) Handbook of Modal Logics. Elsevier, Amsterdam (2006)
3. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook. Cambridge University Press, Cambridge (2003)
4. Baader, F., Milicic, M., Lutz, C., Sattler, U., Wolter, F.: Integrating description logics and action formalisms for reasoning about web services. LTCS-Report LTCS-05-02, Dresden University of Technology (2005), `http://lat.inf.tu-dresden.de/research/reports.html`
5. Bezhanishvili, N., ten Cate, B.: Transfer results for hybrid logic. Part 1: the case without satisfaction operators. J. Logic Comput. 16, 177–197 (2006)
6. Blackburn, P., Marx, M.: Tableaux for quantified hybrid logic. In: Egly, U., Fermüller, C. (eds.) TABLEAUX 2002. LNCS (LNAI), vol. 2381, pp. 38–52. Springer, Heidelberg (2002)
7. Bolander, T., Blackburn, P.: Termination for hybrid tableaus. J. Logic Comput. 17, 517–554 (2007)
8. Bolander, T., Braüner, T.: Tableau-based decision procedures for hybrid logic. J. Log. Comput. 16(6), 737–763 (2006)
9. Book, R., Long, T., Selman, A.: Quantitative relativizations of complexity classes. SIAM J. Computing 13, 461–487 (1984)
10. Braüner, T.: Natural deduction for first-order hybrid logic. J. Logic, Language and Information 14, 173–198 (2005)
11. Calin, G., Myers, R., Pattinson, D., Schröder, L.: CoLoSS: The coalgebraic logic satisfiability solver (system description). In: Methods for Modalities, M4M-5. ENTCS. Elsevier, Amsterdam (to appear, 2008)
12. ten Cate, B.: Model theory for extended modal languages. PhD thesis, University of Amsterdam, ILLC Dissertation Series DS-2005-01 (2005)
13. Chandra, A., Stockmeyer, L.: Alternation. J. ACM 28, 114–133 (1981)
14. Chellas, B.: Modal Logic. Cambridge University Press, Cambridge (1980)
15. D'Agostino, G., Visser, A.: Finality regained: A coalgebraic study of Scott-sets and multisets. Arch. Math. Logic 41, 267–298 (2002)
16. Demri, S., Lugiez, D.: Presburger modal logic is PSPACE-complete. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 541–556. Springer, Heidelberg (2006)
17. Fagin, R., Halpern, J.Y.: Reasoning about knowledge and probability. J. ACM 41, 340–367 (1994)
18. Fine, K.: In so many possible worlds. Notre Dame J. Formal Logic 13, 516–520 (1972)

19. Haarslev, V., Möller, R., van der Straeten, R., Wessel, M.: Extended query facilities for racer and an application to software-engineering problems. In: Description Logics, DL 2004, pp. 148–157 (2004)
20. Heifetz, A., Mongin, P.: Probabilistic logic for type spaces. Games and Economic Behavior 35, 31–53 (2001)
21. Larsen, K., Skou, A.: Bisimulation through probabilistic testing. Inform. Comput. 94, 1–28 (1991)
22. Mares, E.: Relevant Logic: A Philosophical Interpretation. Cambridge University Press, Cambridge (2004)
23. Papadimitriou, C.: On the complexity of integer programming. J. ACM 28, 765–768 (1981)
24. Pattinson, D.: Coalgebraic modal logic: Soundness, completeness and decidability of local consequence. Theoret. Comput. Sci. 309, 177–193 (2003)
25. Pattinson, D., Schröder, L.: Admissibility of cut in coalgebraic logics. In: Coalgebraic Methods in Computer Science, CMCS 2008. ENTCS, vol. 203, pp. 221–241. Elsevier, Amsterdam (2008)
26. Pauly, M.: A modal logic for coalitional power in games. J. Logic Comput. 12, 149–166 (2002)
27. Schröder, L.: A finite model construction for coalgebraic modal logic. J. Log. Algebr. Prog. 73, 97–110 (2007)
28. Schröder, L., Pattinson, D.: PSPACE reasoning for rank-1 modal logics. In: Logic in Computer Science, LICS 2006, pp. 231–240. IEEE, Los Alamitos (2006); Full version to appear in ACM TOCL
29. Schröder, L., Pattinson, D.: Shallow models for non-iterative modal logics. In: Dengel, A.R., Berns, K., Breuel, T.M., Bomarius, F., Roth-Berghofer, T.R. (eds.) KI 2008. LNCS (LNAI), vol. 5243, pp. 324–331. Springer, Heidelberg (2008)
30. Seligman, J.: Internalization: The case of hybrid logics. J. Logic Comput. 11, 671–689 (2001)
31. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. J. Web Semantics (2006)
32. Troelstra, A., Schwichtenberg, H.: Basic Proof Theory. Cambridge University Press, Cambridge (1996)
33. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 292–297. Springer, Heidelberg (2006)
34. Tzakova, M.: Tableau calculi for hybrid logics. In: Murray, N.V. (ed.) TABLEAUX 1999. LNCS, vol. 1617, pp. 278–292. Springer, Heidelberg (1999)
35. Zolin, E.: The description logic complexity navigator (2007), http://www.cs.man.ac.uk/~ezolin/dl

A Description of Iterative Reflections of Monads (Extended Abstract)

Jiří Adámek[1], Stefan Milius[1], and Jiří Velebil[2,⋆]

[1] Institute of Theoretical Computer Science, Technical University, Braunschweig, Germany
adamek@iti.cs.tu-bs.de, mail@stefan-milius.eu

[2] Faculty of Electrical Engineering, Czech Technical University of Prague, Czech Republic
velebil@math.feld.cvut.cz

Abstract. For ideal monads in Set (e. g. the finite list monad, the finite bag monad etc.) we have recently proved that every set generates a free iterative algebra. This gives rise to a new monad. We prove now that this monad is iterative in the sense of Calvin Elgot, in fact, this is the iterative reflection of the given ideal monad. This shows how to freely add unique solutions of recursive equations to a given algebraic theory. Examples: the monad of free commutative binary algebras has the monad of binary rational unordered trees as iterative reflection, and the finite list monad has the iterative reflection given by adding an absorbing element.

1 Introduction

The semantics of recursive definitions is a topic at the heart of theoretical computer science. Iterative theories of Calvin Elgot are a well-established formalism in which recursive equation systems can be solved. So far, iterative theories were considered over arbitrary signatures [9,10] or arbitrary endofunctors [3] but without studying the effect of equational laws on given operations. For example, Elgot et al. described in [10] the free iterative theory on a signature Σ as the theory $\mathbb{R}_\Sigma$ of all rational Σ-trees (that is, Σ-trees with only finitely many subtrees up to isomorphism). The free iterative theory can be thought of as the closure of the theory formed by all Σ-terms under unique solutions of recursive equations. In our present paper we attend to the influence that equations have on iteration. This topic is relevant e. g. for process algebra where processes are defined recursively and operations on processes typically satisfy equational laws such as associativity, commutativity or idempotency. Let us consider the simple case of one binary operation: by the above result the free iterative theory is the theory of rational binary trees. What happens if the operation is required to be commutative? The answer is simple: the free iterative theory consists of all rational non-ordered binary trees. This has been known before since commutativity can be expressed by working with algebras for an endofunctor H, and in

⋆ The third author acknowledges the support of the Grant Agency of the Czech Republic under the Grant No. 201/06/664.

that case the free iterative theory was described in [2] as follows: one applies the given equations to rational Σ-trees possibly infinitely often. Next question: what happens if the given binary operation is required to be associative? That is, the theory we start with is the theory of finite lists. It follows from our results in the present paper that the free iterative theory is just the extension of the finite-list theory by a single absorbing element. (Informally, for every infinite binary tree one gets, by applying the associative law infinitely often, the complete binary tree viz. the unique solution of $x \approx x \cdot x$.) The same answer is true for a commutative and associative binary operation, in other words, for the finite-bag theory. Last question: what about an idempotent binary operation? We cannot provide an answer to this question because the corresponding theory is not *ideal*, and we have only iterative reflections for ideal theories—in fact, the question makes no sense for general theories.

We are going to work with finitary monads $\mathbb{M}$ rather than equational theories—recall that the underlying functor M of the monad assigns to every set X the free algebra MX on X for the given equational theory, and that the inclusion of generators forms a natural transformation $\eta : \mathrm{Id} \to M$. Elgot [9] called $\mathbb{M}$ ideal if M is a coproduct of Id and a subfunctor M' such that η is one of the coproduct injections, and the monad multiplication $\mu : MM \to M$ has a restriction to $\mu' : M'M \to M'$; in the language of theories that means that the presentation of M by operations modulo equations is such that the property of a term not being equivalent to a variable is preserved by substitution. Commutativity and associativity of a binary operation are examples of equational specifications yielding ideal monads, idempotency is not.

We already know that every ideal monad has an iterative reflection; this states in category-theoretic terms that unique solutions of guarded recursive equations can be added freely to the given monad. This was proved in [5] under much less restrictive side conditions than those required below. However, a concrete description of the iterative reflection was missing: we proved that for a given ideal monad $\mathbb{M}$ every object X generates a free iterative algebra $\widehat{M}X$, and thus, we obtain a new monad $\widehat{\mathbb{M}}$. Here we prove that $\widehat{\mathbb{M}}$ is iterative and that it is the desired iterative reflection of $\mathbb{M}$.

Although the statement *"the iterative reflection is the monad of free iterative algebras"* may sound almost tautological, we have not found an easy proof. In fact, the proof presented in our paper is not only technically involved, it also requires at one point that every strong epimorphism be split—this forces us to restrict our attention to monads in categories such as sets or vector spaces. In contrast, the existence of iterative reflections was proved for ideal monads in all extensive locally finitely presentable categories, see [5].

In this extended abstract we omit most of the technical details and provide only sketches of proofs. The full details can be found in [6].

2 Ideal and Iterative Monads

In this section we recall the concepts introduced by Calvin Elgot [9] in the language of monads in lieu of algebraic theories used originally. Recall first that a functor is called *finitary* if it preserves filtered colimits. Throughout this section $\mathbb{M} = (M, \eta, \mu)$ denotes a finitary monad on a category $\mathcal{A}$ (that is, the underlying endofunctor M is finitary). Recall further that given another monad $\overline{\mathbb{M}} = (\overline{M}, \overline{\eta}, \overline{\mu})$, a *monad morphism* is a natural transformation $h : M \to \overline{M}$ such that $h \cdot \eta = \overline{\eta}$ and $h \cdot \mu = \overline{\mu} \cdot (h * h)$.

Assumption 2.1. Throughout the paper we assume that the base category $\mathcal{A}$ is locally finitely presentable and has split strong epimorphisms. More detailed, we assume that (1) $\mathcal{A}$ has colimits, (2) for every strong epimorphism $e : X \to Y$ there exists $m : Y \to X$ with $e \cdot m = \mathrm{id}$ (where "strong" means the diagonal fill-in property w. r. t. all monomorphisms), and (3) $\mathcal{A}$ has a set $\mathcal{A}_{\mathrm{fp}}$ of finitely presentable objects (i. e., their hom-functors are finitary) whose closure under filtered colimits is all of $\mathcal{A}$.

For example, the categories of sets and many-sorted sets satisfy the above assumptions, with $\mathcal{A}_{\mathrm{fp}}$ formed by all finite sets. For every field K the category Vec_K of vector spaces on K also satisfies (1)–(3) and $\mathcal{A}_{\mathrm{fp}}$ is formed by the finite-dimensional spaces. Finally, every algebraic lattice $\mathcal{A}$ considered as a category fulfills (1)–(3); the finitely presentable objects are the compact elements.

Notation 2.2. The category of (Eilenberg-Moore) algebras for the monad $\mathbb{M}$ is denoted by $\mathcal{A}^{\mathbb{M}}$. Recall that its objects are algebras $a : MA \to A$ for the functor M such that $a \cdot \eta_A = \mathrm{id}$ and $a \cdot Ma = a \cdot \mu_A$. And the morphisms of $\mathcal{A}^{\mathbb{M}}$ are the usual M-algebra homomorphisms, i. e. h is a homomorphism from an algebra (A, a) to (B, b) if $h \cdot a = b \cdot Mh$.

Definition 2.3. *(C. Elgot [9]) An* ideal monad *consists of a finitary monad $\mathbb{M} = (M, \eta, \mu)$, a finitary subfunctor $m : M' \hookrightarrow M$ such that $M = M' + \mathrm{Id}$ with injections m and η, and a natural transformation $\mu' : M'M \to M'$ such that the square below commutes:*

$$\begin{tikzcd} M'M \arrow[r, "\mu'"] \arrow[d, "mM"'] & M' \arrow[d, "m"] \\ MM \arrow[r, "\mu"'] & M \end{tikzcd} \tag{2.1}$$

Definition 2.4. *Let $\mathbb{M}$ be an ideal monad.*

(1) By a (finitary) equation morphism *we mean a morphism $e : X \to M(X + Y)$, where X is a finitely presentable object (of variables) and Y is an arbitrary object (of parameters).*

(2) We call e guarded *provided that it factorizes through the summand $M'(X + Y) + Y$ of $M(X + Y) = M'(X + Y) + X + Y$:*

$$\begin{tikzcd} X \arrow[r, "e"] \arrow[dr, dashed, "e_0"'] & M(X+Y) \\ & M'(X+Y)+Y \arrow[u, "{[m_{X+Y}, \eta_{X+Y} \cdot \mathsf{inr}]}"'] \end{tikzcd} \tag{2.2}$$

Remark 2.5. Recall that if $\mathcal{A} = \mathsf{Set}$, then for every finitary monad $\mathbb{M}$ there exists an equational presentation such that $\mathbb{M}$ is the associated free-algebra monad. That is, for every set Z we can consider MZ as the set of all terms with the free variables in Z modulo the given equations.

(1) If $X = \{ x_1, \dots, x_n \}$ then the equation morphism e can be regarded as the system of recursive equations

$$\begin{aligned} x_1 &\approx t_1(x_1, \dots, x_n, p_1, \dots, p_k) \\ &\vdots \\ x_n &\approx t_n(x_1, \dots, x_n, p_1, \dots, p_k) \end{aligned}$$

whose right-hand sides $t_i = e(x_i)$ are $\mathbb{M}$-terms in the variables from X and parameters $p_1, \dots, p_k \in Y$.

(2) The concept of a guarded equation morphism forbids equations such as $x_1 \approx x_1$. Evelyn Nelson [14] introduced iterative algebras for a signature in which guarded systems of equations have unique solutions—see also the related concept by Jerzy Tiuryn [15]:

Definition 2.6. *We say that the algebra (A, a) for the ideal monad $\mathbb{M}$ is* iterative *provided that every guarded equation morphism $e : X \to M(X + A)$ with parameter object $Y = A$ has a unique solution, i. e., a unique morphism $e^\dagger : X \to A$ for which*

$$e^\dagger = a \cdot M[e^\dagger, \mathrm{id}_A] \cdot e\,. \tag{2.3}$$

Remark 2.7. For every ideal monad $\mathbb{M}$ the following was proved in [5]:

(1) Iterative algebras form a full subcategory of $\mathcal{A}^{\mathbb{M}}$; the reason why we consider the usual homomorphisms as the "right" morphisms of iterative algebras is that homomorphisms automatically preserve solutions.

(2) Every object X generates a free iterative $\mathbb{M}$-algebra which we denote by $\widehat{MX}$ with the structure and the universal arrow given by $\rho_X : M\widehat{MX} \to \widehat{MX}$ and $\widehat{\eta}_X : X \to \widehat{MX}$. In other words, the forgetful functor of the category of iterative $\mathbb{M}$-algebras has a left adjoint $X \mapsto (\widehat{MX}, \rho_X)$.

(3) We obtain a new monad $\widehat{\mathbb{M}} = (\widehat{M}, \widehat{\eta}, \widehat{\mu})$ and a monad morphism $\kappa : \mathbb{M} \to \widehat{\mathbb{M}}$ with the components $\kappa_X = \rho_X \cdot M\widehat{\eta}_X$. We also proved that the ideal monad $\mathbb{M}$ has an iterative reflection—and in the present paper we prove that this is $\kappa : \mathbb{M} \to \widehat{\mathbb{M}}$.

In [5] we assumed that the base category is extensive, and therefore we worked with ideal (rather than guarded) equation morphisms. However, all the results remain valid under our present assumption. In particular, our proof of the existence of an iterative reflection (Theorem 2.13) contained in Remark 2.14 uses split epimorphisms and does not use extensivity. The proof is based on the fact (proved in [5]) that iterative algebras are closed under limits—by inspecting the proof one sees immediately that this is true for iterativity based, as in Definition 2.6 above, on guarded equation morphisms. Also extensivity plays no rôle here.

Examples 2.8. We mention some well-known ideal monads $\mathbb{M}$ in Set for which a description of the free iterative $\mathbb{M}$-algebra $\widehat{MX}$ on X is known, see [5].

(1) The monad $MX = X^*$ of finite lists in X whose $\mathbb{M}$-algebras are monoids. A free iterative algebra is described in [5]: one adds to X^*, the monoid of finite lists with concatenation, an absorbing element $\perp$; this means that the binary operation is extended by $w\perp = \perp = \perp w$ for all lists w. Shortly: $\widehat{M}X = X^* + \{\perp\}$.

(2) The monad $\mathbb{M}$ assigning to a set X the set of finite trees with leaves labelled in X. The $\mathbb{M}$-algebras are precisely the Σ-algebras for the signature Σ of one n-ary operation symbol for every natural number $n = 1, 2, 3, \ldots$. The free iterative algebra was described by Evelyn Nelson [14] using the concept of *rational tree* of Susanna Ginali [12], meaning a tree with finitely many subtrees up to isomorphism: $\widehat{M}X$ consists of all finitely branching rational trees with leaves labelled in X.

(3) Analogously, let MX consist of all finite non-ordered binary trees on X, so $\mathbb{M}$ is the monad of commutative binary algebras. In this case $\widehat{M}X$ consists of all rational non-ordered binary trees on X, see [2].

(4) The monad $\mathbb{M}$ assigning to a set X the set of finite bags (= multisets) in X has as algebras the commutative monoids. A free iterative algebra, as proved in [14], is $\widehat{M}X = MX + \{\perp\}$ where $\perp$ is an absorbing element.

(5) The monad $MX = X \times \Sigma^*$ of free unary algebras with operations from the set Σ has the free iterative algebras $\widehat{M}X = (X \times \Sigma^*) + \Sigma^p$ where $\Sigma^p = \Sigma^*(\Sigma^*)^\omega$ are the words in Σ which are eventually periodic, see [14]. For example, $\widehat{M}X = X \times \mathbb{N} + 1$ in case $\Sigma = \{0\}$.

(6) The monad $MX = X \times \mathbb{N}^\Sigma$ of free unary algebras with commuting operations from the set Σ (i. e., $a(b(x)) = b(a(x))$ for all $a, b \in \Sigma$) has the free iterative algebras $\widehat{M}X = (X \times \mathbb{N}^\Sigma) + \{\perp\}$ where $\perp$ is a fixed point of all operations. In case of two operations this was pointed out to us by Bruno Courcelle, and a detailed proof is presented in [5]. The general case is completely analogous.

(7) For every finitary endofunctor H we have described in [3] a free iterative monad on H as the monad of free iterative algebras for the functor H. Moreover, a free iterative H-algebra on the object Y is given as the colimit of all finite coalgebras for $H(-) + Y$. We know from Michael Barr [8] that H-algebras are precisely the monadic algebras for the free monad $\mathbb{M}$ on H, and it is not difficult to prove that iterative algebras for the functor H in the sense of [3] (see Definition 2.5 and Theorem 4.4) are precisely the iterative algebras for $\mathbb{M}$ as defined in 2.6. Thus, the free iterative monad on H is the monad $\widehat{\mathbb{M}}$. Special cases include (2) above for $HX = \coprod_{n\in\mathbb{N}} X^n$, (3) for $HX = \{\{x, y\} \mid x, y \in X\}$ and (5) above for $HX = X \times \Sigma$. In [3] we called $\widehat{\mathbb{M}}$ the *rational monad* of H.

Remark 2.9. In contrast to Example 2.8(4), the monad $\mathcal{P}_{\mathsf{fin}}$ of finite subsets is not ideal: consider $x \neq y$ in X and a function $f : X \to X'$ with $f(x) = f(y)$. Then $\{x, y\} \in \mathcal{P}_{\mathsf{fin}}X \setminus \eta_X[X]$ but $\mathcal{P}_{\mathsf{fin}}f\{x, y\} \in \eta_{X'}[X']$.

Definition 2.10. *[9] An ideal monad* $\mathbb{M}$ *is called* iterative *if every guarded equation morphism* $e : X \to M(X+Y)$ *has a unique* solution w. r. t. $\mathbb{M}$*, which means a morphism* $e^\dagger : X \to MY$ *such that the equation* $e^\dagger = \mu_Y \cdot M[e^\dagger, \eta_Y] \cdot e$ *holds.*

Proposition 2.11. [5] *An ideal monad $\mathbb{M}$ is iterative if and only if all of its free algebras (MX, μ_X) are iterative algebras.*

Example 2.12. The monad $\widehat{M}X = X \times \mathbb{N} + 1$ of 2.8(5) is iterative: its free algebras are the free algebras on one unary operation σ extended by a unique fixed point of σ. In general, an algebra for $\widehat{\mathbb{M}}$ is an algebra with a unary operation with a chosen fixed point of that operation, see [4]. So there exist many algebras for $\widehat{\mathbb{M}}$ that are not iterative, e. g., all those with more than one fixed point for σ. In fact, in an iterative algebra the equation $x \approx \sigma(x)$ must have a unique solution.

Definition 2.13. *Suppose we have two ideal monads $\mathbb{M} = (M, \eta, \mu, M', m, \mu')$ and $\overline{\mathbb{M}} = (\overline{M}, \overline{\eta}, \overline{\mu}, \overline{M}', \overline{m}, \overline{\mu}')$. By an* ideal monad morphism *we understand a monad morphism $h : (M, \eta, \mu) \to (\overline{M}, \overline{\eta}, \overline{\mu})$ such that there exists a domain-codomain restriction $h' : M' \to \overline{M}'$ of h with $\overline{m} \cdot h' = h \cdot m$ (which is necessarily unique, recall that $\overline{m}$ is a monomorphism).*

Notation 2.14. In the category of all finitary monads on $\mathcal{A}$ we now consider (a) the non-full subcategory of all ideal monads and ideal monad morphisms, denoted by $\mathsf{FM}_{\mathrm{id}}(\mathcal{A})$ and (b) its full subcategory of all iterative monads $\mathsf{IFM}(\mathcal{A})$.

3 A Characterization of Free Iterative Algebras

Assumption 3.1. Throughout the rest of the paper $\mathbb{M}$ denotes an ideal monad on $\mathcal{A}$.

Remark 3.2. In [3] we described for every endofunctor H of a locally finitely presentable category, the free iterative H-algebra on an object Y as a colimit of the diagram Eq_Y of all *flat* equation morphisms, i. e., morphisms $e : X \to HX + Y$ with X finitely presentable. The connecting morphisms of that diagram Eq_Y are simply the coalgebra homomorphisms for the endofunctor $H(-) + Y$. The fact that Eq_Y is a filtered diagram whose colimit is the free iterative H-algebra on Y turned out to be the basic step for describing the rational monad of H, see Example 2.8(7). The proof of this fact was technically rather involved.

In the present section we prove the analogous result for algebras for an ideal monad $\mathbb{M}$: we form the diagram of all guarded equation morphisms $e : X \to M(X + Y)$ with X finitely presentable, but unfortunately, in lieu of coalgebra homomorphisms for $M((-) + Y)$ we need more general "solution homomorphisms" here. To make sure that Eq_Y is a filtered diagram we need the restrictive side condition that strong epimorphisms split.

Notation 3.3. Given an equation morphism $e : X \to M(X + Y)$, every morphism $h : Y \to Y'$ yields a new equation morphism (by changing parameters)

$$h \bullet e \equiv X \xrightarrow{e} M(X + Y) \xrightarrow{M(X+h)} M(X + Y').$$

In particular, use the universal arrow $\widehat{\eta}_Y : Y \to \widehat{M}Y$ of Remark 2.7 to turn every "abstract" equation morphism $e : X \to M(X + Y)$ into a "concrete" equation morphism $\widehat{\eta}_Y \bullet e : X \to M(X + \widehat{M}Y)$ in the free iterative algebra $(\widehat{M}Y, \rho_Y)$.

The latter has, whenever e is guarded, a unique solution in $\widehat{M}Y$ which, by abuse of notation, we denote by $e^\dagger : X \to \widehat{M}Y$. Thus, for every guarded equation morphism $e : X \to M(X+Y)$ we define $e^\dagger$ by

$$e^\dagger = \rho_Y \cdot M[e^\dagger, \widehat{\eta}_Y] \cdot e. \tag{3.1}$$

Definition 3.4. *Let $e : X \to M(X+Y)$ and $f : X' \to M(X'+Y)$ be guarded equation morphisms. By a* solution homomorphism *is meant a morphism $h : X \to X'$ in $\mathcal{A}$ for which the equation $f^\dagger \cdot h = e^\dagger : X \to \widehat{M}Y$ holds.*

Notation 3.5. For every object Y we denote by EQ_Y the category of all guarded equation morphisms with parameter object Y and all solution homomorphisms.

We also denote by $\mathrm{Eq}_Y : \mathsf{EQ}_Y \to \mathcal{A}$ the forgetful functor assigning to $e : X \to M(X+Y)$ the object X.

Example 3.6. Whenever $h : X \to X'$ is a coalgebra homomorphism, i. e., whenever we have $f \cdot h = M(h+Y) \cdot e$, then h is a solution homomorphism. Indeed, $f^\dagger \cdot h = e^\dagger$ follows from the uniqueness of solutions since $f^\dagger \cdot h$ solves e. To see this consider the diagram below:

$$\begin{tikzcd}
X \arrow[r, "h"] \arrow[d, "e"'] \arrow[rr, bend left, "f^\dagger \cdot h"] & X' \arrow[r, "f^\dagger"] \arrow[d, "f"] & \widehat{M}Y \\
M(X+Y) \arrow[r, "M(h+Y)"'] \arrow[rr, bend right, "M[f^\dagger \cdot h, \widehat{\eta}_Y]"'] & M(X'+Y) \arrow[r, "M[f^\dagger, \widehat{\eta}_Y]"'] & M\widehat{M}Y \arrow[u, "\rho_Y"']
\end{tikzcd}$$

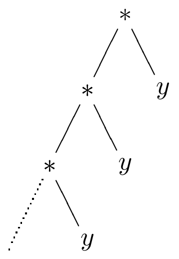
The right-hand square commutes by (3.1), the left-hand one by assumption and the upper and lower parts obviously do. So the outside of the diagram commutes, showing that $f^\dagger \cdot h$ is a solution of e as desired.

Remark 3.7. In the coalgebraic construction of the free iterative monad on an endofunctor H in [3] we used the category EQ_Y of all flat equation morphisms. This category is trivially filtered because it is closed under finite colimits in the category of all coalgebras, and so the corresponding forgetful functor Eq_Y is a filtered diagram whose colimit yields the object assignment of the desired free iterative monad on H.

Our present setting is more subtle: here we cannot work with coalgebra homomorphism (for $M((-)+Y)$) because they are insufficient to relate all equations with the same solution in the corresponding diagram. To see this we consider the example of a signature with one binary operation $*$. The associated free monad on that signature is the finite binary tree monad $\mathbb{M}$. Now let, just in this example, EQ'_Y denote the category of guarded equation morphisms and coalgebra homomorphisms. Consider the two recursive equations (trees are written as terms here)

$$x \approx x * y \qquad \text{and} \qquad x \approx (x * y) * y$$

which give rise to two different equation morphism $e, f : \{x\} \to M(\{x\}+\{y\})$. These two equations specify the same rational binary tree on $\{y\}$:



However, the above two equations will lead to two distinct elements in the colimit of the diagram given by the forgetful functor on EQ'_Y—this is due to the fact that any morphism in EQ'_Y preserves the height of the binary trees on the right-hand side of recursive equations.

Lemma 3.8. *The category* EQ_Y *is filtered.*

Theorem 3.9. *The free iterative algebra* $\widehat{M}Y$ *is a filtered colimit of the diagram* $\mathrm{Eq}_Y : \mathsf{EQ}_Y \to \mathcal{A}$ *of all guarded equation morphisms in* Y*, shortly,* $\widehat{M}Y = \operatorname{colim}\mathrm{Eq}_Y$.

The overall structure of the proof is somewhat similar to the structure of the proof of Theorem 3.3 in [3]. However, the technical details are different and more involved.

Sketch of Proof. We denote by $\widetilde{M}Y$ a colimit of the filtered diagram Eq_Y in $\mathcal{A}$ with colimit morphisms $e^\sharp : X \to \widetilde{M}Y$ for all $e : X \to M(X+Y)$ in EQ_Y. We first turn $\widetilde{M}Y$ into an $\mathbb{M}$-algebra. Recall that since $\mathcal{A}$ is locally finitely presentable, the object $M\widetilde{M}Y$ is a colimit of the canonical diagram of all morphisms $p : P \to M\widetilde{M}Y$ with P finitely presentable. Thus, we can define a morphism $\widetilde{\rho}_Y : M\widetilde{M}Y \to \widetilde{M}Y$ by specifying its composites $\widetilde{\rho}_Y \cdot p$ with any $p : P \to M\widetilde{M}Y$ with P finitely presentable. Since M is finitary and X is finitely presentable, for every p there exists $e : X \to M(X+Y)$ in EQ_Y and a morphism $p_0 : P \to MX$ such that $p = Me^\sharp \cdot p_0$. Now denote by $[\![p, e]\!]$ the guarded equation morphism

$$P + X \xrightarrow{[p_0,\eta]} MX \xrightarrow{Me} MM(X+Y) \xrightarrow{\mu} M(X+Y) \xrightarrow{M\mathsf{inr}} M(P+X+Y).$$

One then proves that the morphisms $[\![p, e]\!]^\sharp \cdot \mathsf{inl} : P \to \widetilde{M}Y$ are independent of the choice of p_0, and that they form a cocone of the canonical diagram. Thus, there exists a unique morphism $\widetilde{\rho}_Y : M\widetilde{M}Y \to \widetilde{M}Y$ such that $\widetilde{\rho}_Y \cdot p = [\![p_0, e]\!]^\sharp \cdot \mathsf{inl}$ holds for all p as above. One can now prove that $\widetilde{\rho}_Y$ is the structure morphism of an $\mathbb{M}$-algebra.

Next consider for a finitely presentable object Y the trivial guarded equation morphism $\eta_{Y+Y} \cdot \mathsf{inr} : Y \to M(Y+Y)$ and let $\widetilde{\eta}_Y = (\eta_{Y+Y} \cdot \mathsf{inr})^\sharp : Y \to \widetilde{M}Y$.

Observe that the morphisms $e^\dagger : X \to \widehat{M}Y$ (e in EQ_Y) form a cocone of the diagram Eq_Y, thus, there exists a unique morphism $i : \widetilde{M}Y \to \widehat{M}Y$ such that $i \cdot e^\sharp = e^\dagger$ holds for all e in EQ_Y. One now proves that $i : (\widetilde{M}Y, \widetilde{\rho}_Y) \to (\widehat{M}Y, \rho_Y)$ is a homomorphism of $\mathbb{M}$-algebras such that $i \cdot \widetilde{\eta}_Y = \widehat{\eta}_Y$.

Using this property of i one then verifies that for each finitely presentable object Y and for each $e : X \to M(X+Y)$ in EQ_Y the colimit injection $e^\sharp : X \to \widetilde{M}Y$ is the unique morphism such that $e^\sharp = \widetilde{\rho}_Y \cdot M[e^\sharp, \widetilde{\eta}_Y] \cdot e$. And this fact is the crucial ingredient for the verification that the $\mathbb{M}$-algebra $(\widetilde{M}Y, \widetilde{\rho}_Y)$ is iterative.

For this verification suppose that $e : X \to M(X + \widetilde{M}Y)$ is a guarded equation morphism with a factor $e_0 : X \to M'(X + \widetilde{M}Y) + \widetilde{M}Y$. Since M' is finitary, the object $M'(X+\widetilde{M}Y)+\widetilde{M}Y$ is a colimit of $M'(X+\mathrm{Eq}_Y)+\mathrm{Eq}_Y$. Since X is finitely presentable, we can choose some equation morphism $f : V \to M(V+Y)$ in EQ_Y and a factorization $w' : X \to M'(X+V)+V$ such that $e_0 = (M'(X+f^\sharp)+f^\sharp)\cdot w'$. For $w = [m_{X+V}, \eta_{X+V} \cdot \mathsf{inr}] \cdot w'$ define the equation morphism $\overline{e}$ by

$$X+V \xrightarrow{[w,\eta\cdot\mathsf{inr}]} M(X+V) \xrightarrow{M(\eta+f)} M(MX+M(V+Y)) \xrightarrow{\mu\cdot M\mathsf{can}} M(X+V+Y).$$

It is not difficult to prove that $\overline{e}$ is guarded, whence an object of EQ_Y. Now let $e^\dagger = \overline{e}^\sharp \cdot \mathsf{inl} : X \to \widetilde{M}Y$. One then verifies that $e^\dagger$ is a unique solution of e in the algebra $\widetilde{M}Y$.

Since $\widehat{M}Y$ is a free iterative algebra on Y we obtain a unique $\mathbb{M}$-algebra homomorphism $j : \widehat{M}Y \to \widetilde{M}Y$ such that $j \cdot \widehat{\eta}_Y = \widetilde{\eta}_Y$. Then we immediately have $i \cdot j = \mathrm{id}$, and to see that $j \cdot i = \mathrm{id}$ it is sufficient to establish $j \cdot i \cdot e^\sharp = e^\sharp$ for all e in EQ_Y. This completes the proof for finitely presentable objects Y, and the result also readily extends to arbitrary objects Y by using the canonical diagram with colimit Y. □

4 Rational Equation Morphisms

In this section we prove that iterative algebras have a stronger property of solving equations than stated in their definition. More precisely, it is our goal to show that every iterative algebra for $\mathbb{M}$ is also an iterative algebra for $\widehat{\mathbb{M}}$. As an example consider the monad $\mathbb{M}$ of finite binary trees, for which an algebra is a set A with a binary operation. The algebra A is iterative iff every guarded system of equations

$$x_i \approx t_i(x_1, \dots, x_n, a_1, \dots, a_k) \qquad i = 1, \dots, n$$

where each t_i is a finite binary tree on $\{\, x_i \mid i = 1, \dots n \,\} + \{\, a_j \mid j = 1, \dots, k \,\}$ has a unique solution. However, in lieu of finite trees we can as well take rational infinite trees on the right-hand sides. That is, in lieu of equation morphisms of the form $e : X \to M(X + A)$ we are allowed to consider all $e : X \to \widehat{M}(X + A)$, where $\widehat{\mathbb{M}}$ is the monad of free iterative $\mathbb{M}$-algebras (as constructed in Section [3]).

Definition 4.1. *By a* rational equation morphism *is meant a morphism* $: X \to \widehat{M}(X + Y)$ *with* X *finitely presentable.*

The concept of a solution in an iterative $\mathbb{M}$-algebra is based on the following

Notation 4.2. For an iterative $\mathbb{M}$-algebra (A, a) we denote by $\widehat{a} : \widehat{M}A \to A$ the unique homomorphism extending the identity:

$$\widehat{a} \cdot \rho_A = a \cdot M\widehat{a} \qquad \text{and} \qquad \widehat{a} \cdot \widehat{\eta}_A = \text{id}. \tag{4.1}$$

This is just the counit of the adjunction between the category of iterative algebras for $\mathbb{M}$ and the category $\mathcal{A}$. In other words, $(A, \widehat{a})$ is the corresponding algebra for the monad $\widehat{\mathbb{M}}$. In particular, for the free iterative algebras $(\widehat{M}Y, \rho_Y)$ of Theorem 3.9 the corresponding homomorphism $\widehat{\rho_Y}$ is the monad multiplication $\widehat{\mu}_Y : \widehat{M}\widehat{M}Y \to \widehat{M}Y$.

Definition 4.3. *By a* solution *of a rational equation morphism $e : X \to \widehat{M}(X + A)$ in an iterative $\mathbb{M}$-algebra (A, a) is meant a morphism $e^\ddagger$ such that $e^\ddagger = \widehat{a} \cdot \widehat{M}[e^\ddagger, A] \cdot e$ holds.*

Remark 4.4. In order to state the theorem about unique solutions of rational equation morphisms e, we need to introduce the concept of e being guarded. This would be easy if we knew that the monad $\widehat{\mathbb{M}}$ is ideal. Although this is actually true, and we prove this below (see Theorem 5.5), we are in no position for proving this now. In lieu of the desired equality $\widehat{M} = \widehat{M}' + \text{Id}$, we will now simply introduce a (seemingly arbitrary) subfunctor $\widehat{m} : \widehat{M}' \to \widehat{M}$ of $\widehat{M}$, and relate our concept of guarded equation morphism to $\widehat{M}'$—for distinction from the "real thing" we call this notion "weakly guarded" equation morphism. At the end of our paper we will indeed verify $\widehat{M} = \widehat{M}' + \text{Id}$.

Notation 4.5.

1. We denote by $\rho : M\widehat{M} \to \widehat{M}$ the natural transformation whose components are the algebra morphisms $\rho_Y : M\widehat{M}Y \to \widehat{M}Y$ of the free iterative $\mathbb{M}$-algebras $\widehat{M}Y$, see Remark 2.7.
2. The monad $\widehat{\mathbb{M}} = (\widehat{M}, \widehat{\eta}, \widehat{\mu})$ of free iterative $\mathbb{M}$-algebras has the unit $\widehat{\eta}$ given by universal morphisms of Remark 2.7 and the multiplication $\widehat{\mu}$ with $\widehat{\mu}_Y = \widehat{\rho_Y}$, see Notation 4.2.

Remark 4.6. Recall from [7] that in every locally finitely presentable category every morphism can be factorized as a strong epimorphism followed by a monomorphism.

Definition 4.7. *We define the subfunctor $\widehat{M}'$ of $\widehat{M}$ to be the image of the natural transformation $\rho \cdot m\widehat{M} : M'\widehat{M} \to \widehat{M}$. More precisely, for every object X we have a strong epimorphism γ_X and a monomorphism $\widehat{m}_X$ such that*

$$\rho_X \cdot m_{\widehat{M}X} = \widehat{m}_X \cdot \gamma_X \tag{4.2}$$

holds. Obviously, $\gamma : M'\widehat{M} \to \widehat{M}'$ and $\widehat{m} : \widehat{M}' \to \widehat{M}$ are natural transformations with $\widehat{m} \cdot \gamma = \rho \cdot m\widehat{M}$.

Definition 4.8. *A rational equation morphism $e : X \to \widehat{M}(X+Y)$ is called* weakly guarded *if it factorizes through $[\widehat{m}_{X+Y}, \widehat{\eta}_{X+Y} \cdot \mathsf{inr}] : \widehat{M}'(X+Y) + Y \to \widehat{M}(X+Y)$ as shown below:*

$$\begin{tikzcd}
X \arrow[r, "e"] \arrow[dr, "e'"'] & \widehat{M}(X+Y) \\
& \widehat{M}'(X+Y)+Y \arrow[u, "{[\widehat{m},\widehat{\eta}\cdot\mathsf{inr}]}"']
\end{tikzcd} \tag{4.3}$$

Theorem 4.9. *In every iterative $\mathbb{M}$-algebra every weakly guarded rational equation morphism has a unique solution.*

Sketch of Proof. This result generalizes Theorem 4.6 of [3]. Suppose we are given an iterative $\mathbb{M}$-algebra (A, a) and a weakly guarded rational equation morphism e as in (4.3). Since $\gamma_{X+A} : M'\widehat{M}(X+A) \to \widehat{M}'(X+A)$ is a strong epimorphism we have, by Assumption 2.1, a morphism $s : \widehat{M}'(X+A) \to M'\widehat{M}(X+A)$ with $\gamma_{X+A} \cdot s = \mathrm{id}$. We define $e_0 = (s+A) \cdot e' : X \to M'\widehat{M}(X+A) + A$. Now apply Theorem 3.9 and use the fact that M' is finitary to see that $M'\widehat{M}(X+A)+A = \mathrm{colim}(M'\mathrm{Eq}_{X+A} + A)$. Thus, by the finite presentability of X, there exists an object $g : W \to M(W+X+A)$ in EQ_{X+A} and a factorization $w' : X \to M'W+A$ of e_0 through the colimit injection $M'g^\sharp + A$, i. e., $e_0 = (M'g^\sharp + A) \cdot w'$. Let $w = (m_W + A) \cdot w' : X \to MW + A$ and define an equation morphism $\langle e \rangle : W + X \to M(W+X+A)$ in EQ_A by its components as follows

$$\begin{tikzcd}
W \arrow[d, "g"'] & X \arrow[dd, "w"] \\
M(W+X+A) \arrow[d, "{M[\mathsf{inl}\cdot\eta_W, w, \mathsf{inr}]}"'] & \\
M(MW+A) \arrow[d, "{M[M\mathsf{inl}, \eta\cdot\mathsf{inr}]}"'] & MW+A \arrow[d, "{[M\mathsf{inl}, \eta\cdot\mathsf{inr}]}"] \\
MM(W+X+A) \arrow[r, "\mu"'] & M(W+X+A)
\end{tikzcd}$$

One readily proves that $\langle e \rangle$ is a guarded equation morphism, and therefore there exists a unique solution $\langle e \rangle^\dagger : W + X \to A$. Now let $e^\ddagger = \langle e \rangle^\dagger \cdot \mathsf{inr} : X \to A$. The technical part of the proof is the verification that $e^\ddagger$ is a unique solution of e. □

5 The Iterative Reflection

We are ready to prove that for every ideal monad $\mathbb{M}$ the monad $\widehat{\mathbb{M}}$ of free iterative algebras (see Remark 2.7) is the free iterative reflection. More detailed:

(1) $\widehat{M} = \widehat{M}' + \mathrm{Id}$ with coproduct injections $\widehat{m}$ (Definition 4.7) and $\widehat{\eta}$,
(2) the multiplication $\widehat{\mu}$ has a restriction $\widehat{\mu}' : \widehat{M}'\widehat{M} \to \widehat{M}'$,
(3) every guarded equation morphism $e : X \to \widehat{M}(X + A)$ has a unique solution,
(4) the natural transformation

$$\kappa \equiv M \xrightarrow{M\widehat{\eta}} M\widehat{M} \xrightarrow{\rho} \widehat{M} \tag{5.1}$$

is an ideal monad morphism, and
(5) κ has the following universal property: for every ideal monad morphism $\lambda : \mathbb{M} \to \mathbb{S}$, where $\mathbb{S}$ is an iterative monad, there exists a unique ideal monad morphism $\overline{\lambda} : \widehat{\mathbb{M}} \to \mathbb{S}$ with $\overline{\lambda} \cdot \kappa = \lambda$.

We have to leave (1) to the end and prove the other properties first. We will use the same terminology as in Section 4: in (3) we speak about weakly guarded equation morphisms meaning those with a factorization as in (4.3). In (4) and (5) we use the following notion of weakly ideal monads.

Definition 5.1

1. *A* weakly ideal monad *consists of a finitary monad* $\mathbb{M} = (M, \eta, \mu)$*, a finitary subfunctor* $m : M' \hookrightarrow M$*, and a natural transformation* μ' *such that the square (2.1) commutes.*
2. *Suppose we have two weakly ideal monads* $\mathbb{M} = (M, \eta, \mu, M', m, \mu')$ *and* $\overline{\mathbb{M}} = (\overline{M}, \overline{\eta}, \overline{\mu}, \overline{M}', \overline{m}, \overline{\mu}')$. *By a* weakly ideal monad morphism *we understand a monad morphism* $h : (M, \eta, \mu) \to (\overline{M}, \overline{\eta}, \overline{\mu})$ *such that there exists a domain-codomain restriction* $h' : M' \to \overline{M}'$ *of* h *with* $\overline{m} \cdot h' = h \cdot m$.
3. *A weakly ideal monad is called* weakly iterative *if every weakly guarded equation morphism has a unique solution.*

Lemma 5.2. *The monad* $\widehat{\mathbb{M}}$ *of free iterative algebras for* $\mathbb{M}$ *is weakly ideal.*

Proof. We only need to supply the restriction $\widehat{\mu}' : \widehat{M}'\widehat{M} \to \widehat{M}'$ of the monad multiplication $\widehat{\mu} : \widehat{M}\widehat{M} \to \widehat{M}$. Then $\widehat{\mathbb{M}} = (\widehat{M}, \widehat{\eta}, \widehat{\mu}, \widehat{M}', \widehat{m}, \widehat{\mu}')$ is a weakly ideal monad.

Observe first that the diagram

$$\begin{tikzcd}
M'\widehat{M}\widehat{M} \arrow[r, "m\widehat{M}\widehat{M}"] \arrow[d, "M'\widehat{\mu}"'] \arrow[rr, bend left, "\widehat{m}\widehat{M}\cdot\gamma\widehat{M}"] & M\widehat{M}\widehat{M} \arrow[r, "\rho\widehat{M}"] \arrow[d, "M\widehat{\mu}"'] & \widehat{M}\widehat{M} \arrow[d, "\widehat{\mu}"] \\
M'\widehat{M} \arrow[r, "m\widehat{M}"'] \arrow[rr, bend right, "\widehat{m}\cdot\gamma"'] & M\widehat{M} \arrow[r, "\rho"'] & \widehat{M}
\end{tikzcd}$$

commutes. In fact, the left-hand square commutes by naturality of m, the rest follows from Notation 4.5 and Equation (4.2). Thus, by diagonal fill-in there exists a unique natural transformation $\widehat{\mu}' : \widehat{M}' \to \widehat{M}$ such that the diagram

$$\begin{tikzcd}
\widehat{M'}\widehat{M}\widehat{M} \arrow[r, two heads, "\gamma\widehat{M}"] \arrow[d, "\widehat{M'}\widehat{\mu}"'] & \widehat{M'}\widehat{M} \arrow[d, "\widehat{m}\widehat{M}"] \arrow[ddl, dashed, "\widehat{\mu}'"'] \\
\widehat{M'}\widehat{M} \arrow[d, "\gamma"'] & \widehat{M}\widehat{M} \arrow[d, "\widehat{\mu}"] \\
\widehat{M'} \arrow[r, tail, "\widehat{m}"'] & \widehat{M}
\end{tikzcd} \tag{5.2}$$

commutes. The lower triangle shows that μ' is the required restriction of μ (cf. (2.1)). □

Lemma 5.3. *The monad* $\widehat{\mathbb{M}}$ *of free iterative algebras for* $\mathbb{M}$ *is weakly iterative.*

Proof. It is trivial to see that Proposition 2.11 extends to weakly ideal monads. Then the desired result follows from Theorem 4.9. □

Theorem 5.4

1. *The natural transformation* $\kappa : M \to \widehat{M}$ *is a weakly ideal monad morphism.*
2. *Let* $\mathbb{S}$ *be an iterative monad. For every weakly ideal monad morphism* $\lambda : \mathbb{M} \to \mathbb{S}$ *there exists a unique weakly ideal monad morphism* $\overline{\lambda} : \widehat{\mathbb{M}} \to \mathbb{S}$ *with* $\lambda = \overline{\lambda} \cdot \kappa$.

Sketch of Proof. We omit the proof of item 1 and we sketch the proof of item 2.

For every object Y, we show that SY is an iterative $\mathbb{M}$-algebra. In fact, since $\lambda : M \to S$ is a monad morphism we obtain an $\mathbb{M}$-algebra $\mu^S_Y \cdot \lambda_{SY} : MSY \to SY$. Suppose that $e : X \to M(X + SY)$ is a guarded equation morphism. Then we form an equation morphism $\overline{e}$ with respect to the iterative monad $\mathbb{S}$ as follows:

$$X \xrightarrow{e} M(X + SY) \xrightarrow{\lambda * (\eta^S_X + SY)} S(SX + SY) \xrightarrow{\mu^S \cdot S\mathsf{can}} S(X + Y)\,.$$

It is not difficult to see that $\overline{e}$ is guarded, and that there is a 1-1 correspondence between solutions of e in the algebra SY and solutions of $\overline{e}$ with respect to $\mathbb{S}$. Since the latter exists uniquely, so does the former. Now the freeness of $\widehat{M}Y$ as an iterative algebra implies the existence of a unique homomorphism $\overline{\lambda}_Y$ of $\mathbb{M}$-algebras from $(\widehat{M}Y, \rho_Y)$ to $(SY, \mu^S_Y \cdot \lambda_{SY})$ such that $\overline{\lambda}_Y \cdot \widehat{\eta}_Y = \eta^S_Y$. One then proves that $\overline{\lambda}$ is a weakly ideal monad morphism with $\lambda = \overline{\lambda} \cdot \kappa$, and that $\overline{\lambda}$ is uniquely determined. □

Theorem 5.5. *The iterative reflection of an ideal monad* $\mathbb{M}$ *is the monad* $\widehat{\mathbb{M}}$ *of free iterative* $\mathbb{M}$*-algebras.*

Proof. In view of the preceding results this amounts to proving that $\widehat{M} = \widehat{M'} + \mathrm{Id}$ with injections $\widehat{m}$ and $\widehat{\eta}$.

It is known that every weakly ideal monad $\mathbb{S}$ has an ideal coreflection $c : \mathbb{S}^* \to \mathbb{S}$ and $\mathbb{S}^*$ is iterative whenever $\mathbb{S}$ is weakly iterative (see [3], Proposition 6.7). More detailed: let $\mathbb{S}$ be weakly ideal with the corresponding subfunctor $s : S' \hookrightarrow S$. Then for the functor $S^* = S' + \mathrm{Id}$ there is a structure of a monad $\mathbb{S}^*$ with unit $\mathsf{inr} : \mathrm{Id} \to S' + \mathrm{Id}$ and multiplication $\mu^* : S^*S^* \to S^*$ such that the morphism $c = [s, \eta] : S' + \mathrm{Id} \to S$ is a weakly ideal monad morphism from $\mathbb{S}^*$ to $\mathbb{S}$. Moreover, every weakly ideal monad morphism from an ideal monad into $\mathbb{S}$ uniquely factorizes through c. We now apply this to $\mathbb{S} = \widehat{\mathbb{M}}$: we obtain an iterative monad $\widehat{\mathbb{M}}^* = (\widehat{M'} + \mathrm{Id}, \mathsf{inr}, \widehat{\mu}^*)$ and a weakly ideal monad morphism $c = [\widehat{m}, \widehat{\eta}] : \widehat{\mathbb{M}}^* \to \widehat{\mathbb{M}}$. We prove that c is an isomorphism—this implies the desired statement $\widehat{M} = \widehat{M'} + \mathrm{Id}$.

Since $\mathbb{M}$ is an ideal monad, the weakly ideal monad morphism $\kappa : \mathbb{M} \to \widehat{\mathbb{M}}$ factorizes as $\kappa = c \cdot \kappa^*$ for an ideal monad morphism $\kappa^* : \mathbb{M} \to \widehat{\mathbb{M}}^*$. By the universal property of Theorem 5.4 we obtain an ideal monad morphism $d : \widehat{\mathbb{M}} \to \widehat{\mathbb{M}}^*$ such that $d \cdot \kappa = \kappa^*$. Then we get $c \cdot d \cdot \kappa = \kappa$ from which we immediately conclude that $c \cdot d = \mathrm{id}$. Now, $d \cdot c$ is an ideal monad endomorphism on the ideal coreflection $\widehat{M'} + \mathrm{Id}$ of $\widehat{M}$. Thus, the equality $c \cdot d \cdot c = c$ proves that $d \cdot c = \mathrm{id}$. This establishes that $\widehat{M}$ is a coproduct of $\widehat{M'}$ and Id with injections $\widehat{m} : \widehat{M'} \to \widehat{M}$ and $\widehat{\eta} : \mathrm{Id} \to \widehat{M}$ as desired. □

Corollary 5.6. *The full embedding of the category* $\mathsf{IFM}(\mathcal{A})$ *of iterative monads to the category* $\mathsf{FM}_{\mathrm{id}}(\mathcal{A})$ *of ideal monads has a left adjoint.*

6 Conclusions and Future Research

For every ideal monad $\mathbb{M}$ of Set we proved that the monad $\widehat{\mathbb{M}}$ of free iterative Eilenberg-Moore algebras for $\mathbb{M}$ is iterative. In fact, $\widehat{\mathbb{M}}$ is the iterative reflection of $\mathbb{M}$. We thus derive a number of examples of iterative monads, cf. Examples 2.8:

1. For the finite list monad $MX = X^*$ we obtain the iterative reflection $\widehat{M}X = X^* \cup \{ \bot \}$ where $\bot$ is an absorbing element.
2. Analogously, for the finite bag monad $\mathbb{M}$ we have $\widehat{M}X = MX \cup \{ \bot \}$ where $\bot$ is an absorbing element.
3. For the finite tree monad $\mathbb{M}$, the reflection is the monad $\widehat{\mathbb{M}}$ of rational trees.
4. An analogous example works for non-ordered finite trees: here $\widehat{\mathbb{M}}$ is the monad of rational unordered trees. This follows from results in [2].
5. The iterative reflection of the unary algebra monad $MX = X \times \Sigma^*$ is the monad $\widehat{M}X = X \times \Sigma^* + \Sigma^*(\Sigma^*)^\omega$.

Although the existence of iterative reflections was established in [5] for all ideal monads on extensive, locally finitely presentable categories, in the present paper we restricted our attention to monads in Set-like categories. The reason was purely technical: at two stages, in the proofs of Lemma 3.8 and Theorem 4.9, we needed the hypothesis that every strong epimorphism splits. We do not know whether our results hold in general extensive, locally finitely presentable categories. And for the development of our paper those results seem indispensable. The generalization of our results to a larger collection of base categories is therefore left as an open problem.

In the future we intend to work on a connection of our results to iteration as applied in process algebra.

Acknowledgment. We are grateful to one referee who suggested several improvements of the presentation of our paper.

References

1. Adámek, J., Börger, R., Milius, S., Velebil, J.: Iterative algebras: How iterative are they? Theory Appl. Categ. 19, 61–92 (2008)
2. Adámek, J., Milius, S.: Terminal coalgebras and free iterative Theories. Inform. and Comput. 204, 1139–1172 (2006)
3. Adámek, J., Milius, S., Velebil, J.: Iterative algebras at work. Math. Structures Comput. Sci. 16.6, 1085–1131 (2006)
4. Adámek, J., Milius, S., Velebil, J.: Elgot algebras. Logical Methods Comput. Sci. 2(5:4), 31 (2006)
5. Adámek, J., Milius, S., Velebil, J.: Iterative reflections of monads. Math. Structures Comput. Sci. (accepted for publication)
6. Adámek, J., Milius, S., Velebil, J.: A description of iterative reflections of monads, http://www.stefan-milius.eu
7. Adámek, J., Rosický, J.: Locally presentable and accessible categories. Cambridge University Press, Cambridge (1994)
8. Barr, M.: Coequalizers and free triples. Math. Z. 116, 307–322 (1970)
9. Elgot, C.C.: Monadic computation and iterative algebraic theories. In: Rose, H.E., Shepherdson, J.C. (eds.) Logic Colloquium 1973. North-Holland Publishers, Amsterdam (1975)
10. Elgot, C.C., Bloom, S., Tindell, R.: On the algebraic structure of rooted trees. J. Comput. System Sci. 16, 361–399 (1978)
11. Gabriel, P., Ulmer, F.: Lokal präsentierbare Kategorien. Lecture N. Math., vol. 221. Springer, Berlin (1971)
12. Ginali, S.: Regular trees and the free iterative theory. J. Comput. System Sci. 18, 228–242 (1979)
13. Mac Lane, S.: Categories for the working mathematician, 2nd edn. Springer, Heidelberg (1998)
14. Nelson, E.: Iterative algebras. Theoret. Comput. Sci. 25, 67–94 (1983)
15. Tiuryn, J.: Unique fixed points vs. least fixed points. Theoret. Comput. Sci. 12, 229–254 (1980)

Tighter Bounds for the Determinisation of Büchi Automata*

Sven Schewe

University of Liverpool
sven.schewe@liverpool.ac.uk

Abstract. The introduction of an efficient determinisation technique for Büchi automata by Safra has been a milestone in automata theory. To name only a few applications, efficient determinisation techniques for ω-word automata are the basis for several manipulations of ω-tree automata (most prominently the nondeterminisation of alternating tree automata) as well as for satisfiability checking and model synthesis for branching- and alternating-time logics. This paper proposes a determinisation technique that is simpler than the constructions of Safra, Piterman, and Muller and Schupp, because it separates the principle acceptance mechanism from the concrete acceptance condition. The principle mechanism intuitively uses a Rabin condition on the transitions; we show how to obtain an equivalent Rabin transition automaton with approximately $(1.65\,n)^n$ states from a nondeterministic Büchi automaton with n states. Having established this mechanism, it is simple to develop translations to automata with standard acceptance conditions. We can construct standard Rabin automata whose state-space is bilinear in the size of the input alphabet and the state-space of the Rabin transition automaton, or, for large input alphabets, contains approximately $(2.66\,n)^n$ states, respectively. We also provide a flexible translation to parity automata with $O(n!^2)$ states and $2n$ priorities based on a later introduction record, and hence connect the transformation of the acceptance condition to other record based transformations known from the literature.

1 Introduction

Automata over infinite words have been introduced by Büchi in his proof that the monadic second-order logic of one successor (S1S) is decidable [1]. Büchi automata are an adaptation of finite automata to languages over infinite sequences. They differ from finite automata only with respect to their acceptance condition: While finite runs of finite automata are accepting if a final state is visited at the end of the run, an infinite run of a Büchi automaton is accepting if a final state is visited infinitely many times. Unfortunately, this close relationship between finite and Büchi automata does not imply that automata manipulations

* This work was partly supported by the EPSRC through the grand EP/F033567/1 *Verifying Interoperability Requirements in Pervasive Systems.*

for Büchi automata are equally simple as those for finite automata [2]. In particular, Büchi automata are not closed under determinisation: While a simple subset construction suffices to efficiently determinise finite automata [2], deterministic Büchi automata are strictly less expressive[1] than nondeterministic Büchi automata. Determinisation therefore requires automata with more involved acceptance mechanisms [3,4,5], such as automata with Muller's subset condition [6,7] or Rabin's [3,4] accepting pair condition. Also, an $n^{\Omega(n)}$ lower bound for the determinisation of Büchi automata has been established [8] even if we allow for Muller objectives, which implies that a simple subset construction cannot suffice.

The development of determinisation techniques for Büchi automata was inspired by the problem of synthesising reactive systems [9,10], a problem originally introduced by Church [9] in 1962: Given a relation $R \subseteq (2^I)^\omega \times (2^O)^\omega$ represented by a Büchi automaton (or an S1S or LTL formula), we want to find a function $p : (2^I)^\omega \rightarrow (2^O)^\omega$ such that $(\pi, p(\pi)) \in R$ satisfies the relation for all infinite sequences $\pi \in (2^I)^\omega$. Church's problem was solved independently by Rabin [11], and Büchi and Landweber [12,13] in 1969. Since their seminal works, the relation [14] between finite automata over infinite structures [11] and finite games of infinite duration [12,13] became apparent.

Determinisation is a key ingredient in these proofs. Rabin's extension of the correspondence between automata and monadic logic to the case of trees [11], for example, builds on McNaughton's determinisation theorem [7], and Muller and Schupp's [4] efficient nondeterminisation technique for alternating tree automata is closely linked to the determinisation of nondeterministic word automata. Indeed, the standard technique to nondeterminise an alternating automaton $\mathcal{A}$ with a memoryless acceptance conditions (such as a parity or Rabin automata [15]) is to enrich the input tree with a (guessed) memoryless strategy. Nondeterminising $\mathcal{A}$ can then be reduced to determinise the resulting universal automaton [4,14], and projecting away the strategy. Improved determinisation techniques thus have a considerable impact in automata theory and its application to module checking [16], satisfiability checking [1,11,17,18], and open synthesis [10].

Contribution. This paper contributes a determinisation technique for Büchi automata that simplifies the constructions of Safra [3] and Piterman [5] by separating the principle data structure of the algorithm — the history trees proposed in Section 3 — from the acceptance mechanism. It is my believe that this separations eases teaching and understanding the principles, but it also provides better bounds on the size of the resulting automata.

The central advancement of the proposed method over the previous leading determinisation techniques [3,4,5] is that we abandon the introduction of explicit names for the nodes. One positive effect of this decision is that it yields a leaner and simpler core data structure: The number $hist(n)$ of history trees for Büchi automata with n states is in $o\big((1.65\,n)^n\big)$. We use this observation to construct a deterministic Rabin automaton with only $hist(n)$ states whose pairs are defined

[1] Deterministic Büchi automata cannot, for example, recognise the simple ω-regular language that consists of all infinite words that contain only finitely many a's.

on the *transitions*. As Rabin tree automata have a memoryless accepting run if they accept a tree [15], this implies a $hist(n)$ bound on the size of a program that solves Church's problem as well as an $l \cdot hist(n)$ bound on the size of an ordinary deterministic Rabin automata on alphabets with l letters.

If we want the size of the Rabin automaton to be independent of the alphabet size, or if we want to construct a deterministic parity automaton because of the computational advantages attached to parity objectives, we have to add memory to the history trees. The required amount of memory depends on the acceptance mechanism. For Rabin automata, it suffices to store the acceptance information from the last transition, which only leads to a minor blow-up of the state-space to $o\big((2.66\,n)^n\big)$ states.

For parity automata, we turn to the proved method of keeping a record of the most recent relevant events in the tradition of later [19] and index appearance records [4]: We store (an abstraction of) the order in which the nodes of the current history tree have been introduced in a *later introduction record*.

The separation of concerns allows us to phrase our procedure as a nondeterministic determinisation technique: While the update rule for history trees is strict, the update rule for the later introduction record offers some leeway. This leeway is likely to reduce the size of a deterministic automaton in practice.

Waving this advantage, we still yield a determinisation procedure similar to Piterman's [5], but with an improved complexity analysis ($O(n!^2)$ vs. $O(n^n\,n!)$). However, a reviewer has pointed me to unpublished work of Liu and Wang [20], who independently[2] improved Piterman's complexity analysis to a similar bound.

Organisation of the Paper. In the following section, we recapitulate the different types of automata used in this paper. Section 3 then introduces history trees, which serve as the main data structure used in the proposed determinisation techniques, transitions between them, and a principle approach to exploit this data structure in an efficient determinisation technique. Finally, we use this blueprint of a determinisation technique in Sections 4 and 5 to devise different translations from nondeterministic Büchi tree automata to deterministic Rabin automata, and one to deterministic parity automata, respectively.

2 Preliminaries — Rabin, Parity and Büchi Automata

Nondeterministic Rabin automata are used to represent ω-regular languages $L \subseteq \Sigma^\omega = \omega \to \Sigma$ over a finite alphabet Σ. A nondeterministic Rabin automaton $\mathcal{A} = (\Sigma, Q, I, \delta, \{(A_i, R_i) \mid i \in J\})$ is a five tuple, consisting of a finite alphabet Σ, a finite set Q of states with a non-empty subset $I \subseteq Q$ of initial states, a transition function $\delta : Q \times \Sigma \to 2^Q$ that maps states and input letters to sets of successor states, and a finite family $\{(A_i, R_i) \in 2^Q \times 2^Q \mid i \in J\}$ of Rabin pairs.

[2] I was not aware of the unpublished work of Liu and Wang [20] when writing this paper. While their improvement of Piterman's analysis was submitted after the acceptance of this paper, I would like to point out that their work is older.

Nondeterministic Rabin automata are interpreted over infinite sequences $\alpha : \omega \to \Sigma$ of input letters. An infinite sequence $\rho : \omega \to Q$ of states of $\mathcal{A}$ is called a *run* of $\mathcal{A}$ on an input word α if the first letter $\rho(0) \in I$ of ρ is an initial state, and if, for all $i \in \omega$, $\rho(i+1) \in \delta\big(\rho(i), \alpha(i)\big)$ is an $\alpha(i)$-successor state of $\rho(i)$.

A run $\rho : \omega \to Q$ is *accepting* if, for some index $i \in J$, some state $q \in A_i$ in the acceptance set A_i of the Rabin pair (A_i, R_i), but no state $q' \in R_i$ from the rejecting set R_i of this Rabin pair appears infinitely often in ρ. ($\exists i \in J.\ \mathit{inf}(\rho) \cap A_i \neq \emptyset \wedge \mathit{inf}(\rho) \cap R_i = \emptyset$ for $\mathit{inf}(\rho) = \{q \in Q \mid \forall i \in \omega\ \exists j > i$ such that $\rho(j) = q\}$). A word $\alpha : \omega \to \Sigma$ is *accepted* by $\mathcal{A}$ if $\mathcal{A}$ has an accepting run on α, and the set $\mathcal{L}(\mathcal{A}) = \{\alpha \in \Sigma^\omega \mid \alpha$ is accepted by $\mathcal{A}\}$ of words accepted by $\mathcal{A}$ is called its *language*.

For technical convenience we also allow for finite runs $q_0 q_1 q_2 \ldots q_n$ with $\delta\big(q_n, \alpha(n)\big) = \emptyset$. Naturally, no finite run satisfies the Rabin condition; finite runs are therefore rejecting, and have no influence on the language of an automaton.

Two particularly simple types of Rabin automata are of special interest: parity (or Rabin chain) and Büchi automata. We call a Rabin condition a *Rabin chain* condition if J is an initial sequence of the natural numbers ω, and if $R_i \subset A_i$ and $A_i \subset R_{i+1}$ holds for all indices. The Rabin chain condition is nowadays usually referred to by the term *parity* condition, because it can be represented by a priority function $\mathit{pri} : Q \to \omega$ that maps a state q to $2i+2$ (called the priority of q) if it appears in A_i but not in R_i, and to $2i+1$ if it appears in R_i but not in A_{i-1}. A run ρ of $\mathcal{A}$ then defines an infinite trace of priorities, and the parity of the lowest priority occurring infinitely often determines if ρ is accepting. That is, ρ is accepting if $\min(\mathit{inf}(\mathit{pri}(\rho)))$ is even. We denote parity automata $\mathcal{A} = (\Sigma, Q, I, \delta, \mathit{pri})$, using this priority function. Büchi automata are even simpler: they are Rabin automata with only one accepting pair $(F, \emptyset)$ that has an empty set of rejecting states (or, likewise, parity automata with a priority function pri whose codomain is $\{0, 1\}$. A Büchi automaton is denoted $\mathcal{A} = (\Sigma, Q, I, \delta, F)$, and the states in F are called *final* states.

A Rabin, parity, or Büchi automaton is called *deterministic*, if it has a single initial state and its transition function is deterministic. (That is, if $|\delta(q, \sigma)| \leq 1$ holds true for all states $q \in Q$ and all input letters $\sigma \in \Sigma$ of the automata $\mathcal{A}$.)

3 Büchi Determinisation

The determinisation technique discussed in this section is a variant of Safra's [3] determinisation technique, and the main data structure — the history trees proposed in the first subsection — can be viewed as a simplification of Safra trees [3].

3.1 History Trees

History trees are an abstraction of the possible initial sequences of runs of a Büchi automaton $\mathcal{A}$ on an input word α. They can be viewed as a simplification and abstraction of Safra trees [3]. The main difference between Safra trees and the simpler history trees introduced in this paper is the omission of explicit names for the nodes.

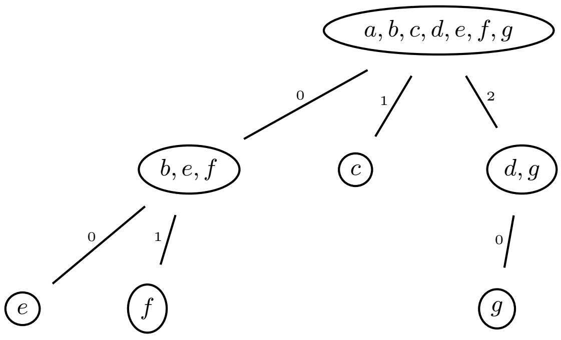


Fig. 1. Example History Tree. The labels of the children of every node are disjoint, and their union is a strict subset of their parent's label. The label of the root node contains the reachable states of the Büchi automaton $\mathcal{A}$ on the input seen so far.

An *ordered tree* $T \subseteq \omega^*$ is a finite prefix and order closed subset of finite sequences of natural numbers. That is, if a sequence $\tau = t_0, t_1, \ldots t_n \in T$ is in T, then all sequences $s_0, s_1, \ldots s_m$ with $m \leq n$ and, for all $i \leq m$, $s_i \leq t_i$, are also in T. For a node $\tau \in T$ of an ordered tree T, we call the number of children of τ its *degree*, denoted by $\deg_T(\tau) = |\{i \in \omega \mid \tau \cdot i \in T\}|$.

A *history tree* (cf. Figure 1) for a given nondeterministic Büchi automaton $\mathcal{A} = (\Sigma, Q, I, \delta, F)$ is a labelled tree $\langle T, l \rangle$, where T is an ordered tree, and $l : T \to 2^Q \setminus \{\emptyset\}$ is a labelling function that maps the nodes of T to non-empty subsets of Q, such that

– the label of each node is a proper superset of the union of the labels of its children, and
– the labels of different children of a node are disjoint.

We call a node τ the *host node* of a state q, if $q \in l(\tau)$ is in the label of τ, but not in the label of any child of τ.

Our estimation of the number of history trees for a given Büchi automaton draws from an estimation of Temme [21] (in the representation of Friedgut, Kupferman, and Vardi [22]) for the number of functions from a set with n elements onto a set with βn elements, where $\beta \in]0, 1[$ is a positive rational number smaller than 1: For the unique positive real number x that satisfies $\beta x = 1 - e^{-x}$, and for $a = -\ln x + \beta \ln(e^x - 1) - (1 - \beta) + (1 - \beta) \ln\left(\frac{1-\beta}{\beta}\right)$, the number of these functions is in $[(1+o(1))M(\beta)n]^n$ for $M(\beta) = \left(\frac{\beta}{1-\beta}\right)^{1-\beta} e^{(a-\beta)}$. This simplifies to

$$m(x) = \frac{1}{ex}(e^x - 1)^{\beta(x)}$$

for $\beta(x) = \frac{1-e^{-x}}{x}$ and $m(x) = M\big(\beta(x)\big)$ when using $e^{a-\beta} = \frac{1}{ex}(e^x - 1)^\beta \left(\frac{1-\beta}{\beta}\right)^{1-\beta}$, where x can be any strictly positive real number.

To estimate the number $hist(n)$ of history trees for Büchi automata with n states, the number $order(m)$ of trees with m nodes can be estimated by 4^m. (More precisely, $order(m) = \frac{(2m-2)!}{m!(m-1)!}$ is the $(m-1)$-st Catalan number [5].) The

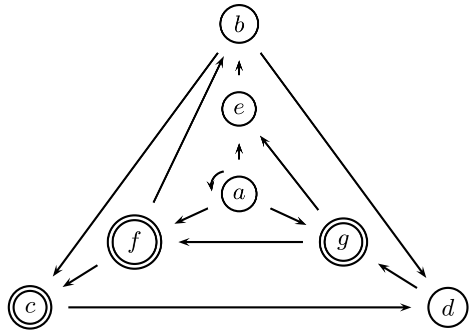


Fig. 2. Relevant Fragment of a Büchi Automaton. This figure captures all transitions for an input letter σ from the states in the history tree from Figure 1. The double lines indicate that the states c, f, and g are final states.

number of history trees with m nodes for a Büchi automaton with n states is the product of the number $\mathit{order}(m)$ of ordered trees with m nodes and functions from the set of n states onto the set of m nodes (if the root is mapped to all states of $\mathcal{A}$), plus the functions the automata states to a set with $(m+1)$ elements. Together with the estimation from above, we can numerically estimate

$$\mathit{hist}(n) \in \sup_{x>0} O\big(m(x) \cdot 4^{\beta(x)}\big) \subset o\big((1.65\,n)^n\big).$$

3.2 History Transitions

For a given nondeterministic Büchi automaton $\mathcal{A} = (\Sigma, Q, I, \delta, F)$, history tree $\langle T, l\rangle$, and input letter $\sigma \in \Sigma$, we construct the *σ-successor* $\langle \widehat{T}, \widehat{l}\rangle$ of $\langle T, l\rangle$ in four steps. (An example transition for the history tree shown in Figure 1 for the σ-transition of an automaton $\mathcal{A}$ shown in Figure 2 is described in Figures 3–6.)

In a first step (shown in Figure 3), we construct the labelled tree $\langle T', l' : T' \to 2^Q\rangle$ such that

- $\tau \in T' \supset T$ is a node of T' if, and only if, $\tau \in T$ is in T or $\tau = \tau' \cdot \deg_T(\tau')$ is formed by appending the degree $\deg_T(\tau')$ of a node $\tau' \in T$ in T to τ',
- the label $l'(\tau) = \delta(l(\tau), \sigma)$ of an old node $\tau \in T$ is the set $\delta(l(\tau), \sigma) = \bigcup_{q \in l(\tau)} \delta(q, \sigma)$ of σ-successors of the states in the label of τ, and
- the label $l'(\tau \cdot \deg_T(\tau')) = \delta(l(\tau), \sigma) \cap F$ of a new node $\tau \cdot \deg_T(\tau)$ is the set of *final* σ-successors of the states in the label of τ.

After this step, each old node is labelled with the σ-successors of the states in its old label, and every old node τ has spawned a new sibling $\tau' = \tau \cdot \deg(\tau)$, which is labelled with the final states $l'(\tau') = l'(\tau) \cap F$ in the label of its parent τ.

The new tree is not necessarily a history tree: (1) nodes may be labelled with an empty set (like node 000 of Figure 3), (2) the labels of siblings do not need to be disjoint (f and g are, for example, in the intersection of the labels of nodes 2 and 3 in Figure 3), and (3) the union of the children's labels do not need to form a proper subset of their parent's label (the union of the labels of node 20 and 21, for example, equals the label of node 2 in Figure 3).

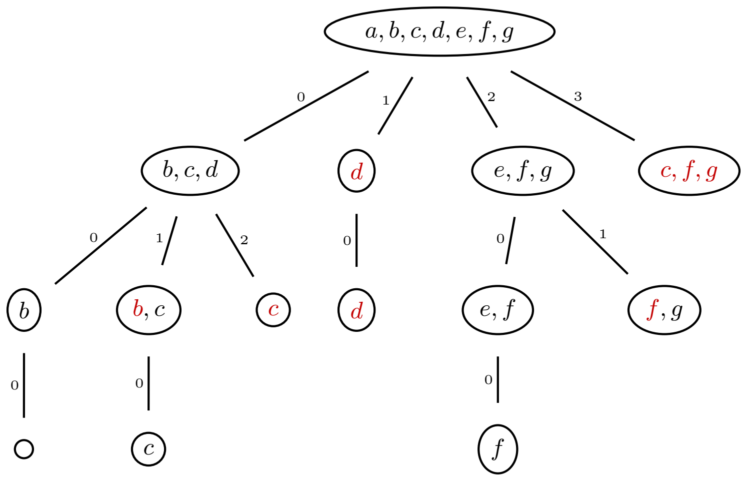


Fig. 3. First Step of the History Transition. This figure shows the tree resulting from the history tree of Figure 1 for the Büchi automaton and transition from Figure 2 alter the first step of the history transition. Every node of the tree from Figure 3 has spawned a new child, whose label may be empty (like the label of node 10) if no final state is reachable upon the read input letter from any state in the label of the parent node. (States printed in red are deleted from the respective label in the second step.)

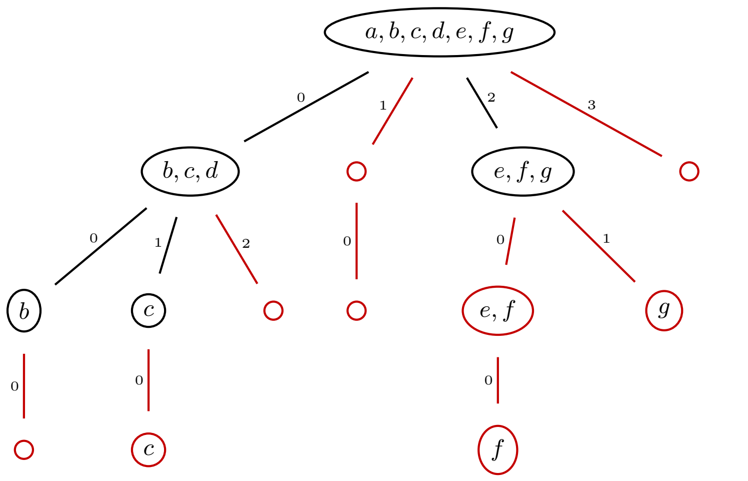


Fig. 4. Second Step of the History Transition. This figure shows the labelled tree that results from the second step of the history transition. the states from the labels of the tree shown in Figure 3 that also occur in the label of an older sibling (like the state f from the old label of the node 21) or in the label of an older sibling of an ancestor of the node (like the state d from the old label of the node 10) are deleted from the label. In this tree, the labels of the siblings are pairwise disjoint, but may be empty, and the union of the label of the children of a node are not required to form a *proper* subset of their parent's label. (The nodes colour coded red are deleted in the third step.)

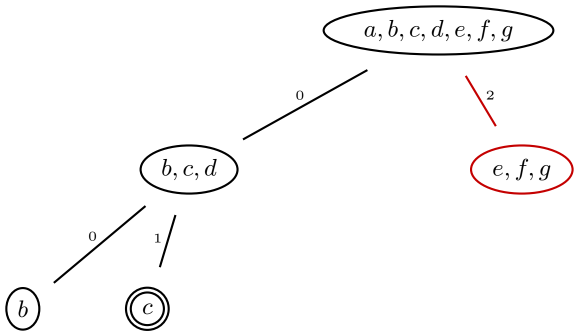


Fig. 5. Third Step of the History Transition. The nodes with (a) an empty label (nodes 000, 02, 1, 10 and 3 from the tree shown in Figure 4) and (b) the descendants of nodes whose children's labels decomposed their own label (nodes 010, 20, 200 and 21) have been deleted from the tree. The labels of the siblings are pairwise disjoint, and form a proper subset of their parent's label, but the tree is not order closed. The nodes that are renamed when establishing order closedness in the final step are depicted in red. Node 01 is the only accepting node (indicated by the double line): Its siblings have been removed due to (b), and, different to node 2, node 01 is stable.

In the second step, property (2) is re-established. We construct the tree $\langle T', l'' : T' \to 2^Q\rangle$, where l'' is inferred from l' by removing all states in the label of a node $\tau' = \tau \cdot i$ and all its descendants if it appears in the label $l'(\tau \cdot j)$ of an older sibling ($j < i$). In Figure 3, the states that are deleted by this rule are depicted in red, and the tree resulting from this deletion is shown in Figure 4.

Properties (1) and (3) are re-established in the third transformation step. In this step, we construct the tree $\langle T'', l'' : T'' \to 2^Q\rangle$ by (a) removing all nodes τ with an empty label $l''(\tau) = \emptyset$, and (b) removing all descendants of nodes whose label is disintegrated by the labels of its descendants from T'. (We use l'' in spite of the type mismatch, strictly speaking we should use its restriction to T''.) The part of the tree that is deleted during the third step is depicted in red in Figure 4, and the tree resulting from this transformation step is shown in Figure 5.

We call the greatest prefix and order closed subset of T'' the set of *stable* nodes and the stable nodes whose descendants have been deleted due to rule (b) *accepting*. In Figure 5, the unstable node 2 is depicted in red, and the accepting leaf 01 is marked by a double line. (Note that only leaves can be accepting.)

The tree resulting from this transformation satisfies the properties (1)–(3), but it is no longer order closed. The tree from Figure 5, for example, has a node 2, but no node 1. In order to obtain a proper history tree, the order closedness is re-established in the final step of the transformation. We construct the σ-successor $\langle \widehat{T}, \widehat{l} : \widehat{T} \to 2^Q \smallsetminus \{\emptyset\}\rangle$ of $\langle T, l\rangle$ by "compressing" T'' to a an order closed tree, using the compression function $comp : T'' \to \omega^*$ that maps the empty word ε to ε, and $\tau \cdot i$ to $comp(\tau) \cdot j$, where $j = |\{k < i \mid \tau \cdot k \in T''\}|$ is the number of older siblings of $\tau \cdot i$. For this function $comp : T'' \to \omega^*$, we simply set $\widehat{T} = \{comp(\tau) \mid \tau \in T''\}$ and $\widehat{l}(comp(\tau)) = l''(\tau)$ for all $\tau \in T''$. The nodes

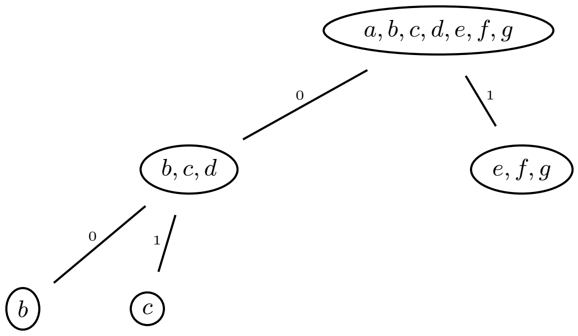


Fig. 6. Final Step of the History Transition. The history tree that results from the complete history transition, has the shape and labelling of the tree from Figure 5, but the former node 2 has been renamed to 1 in order to re-establishing order closedness.

that are renamed during this step are exactly those which are unstable. In our example transformation this is node 2 (depicted in red in Figure 5).

Figure 6 shows the σ-successor for the history tree of Figure 1 and an automaton with σ-transitions as shown in Figure 2.

3.3 Deterministic Acceptance Mechanism

For a nondeterministic Büchi automaton $\mathcal{A} = (\Sigma, Q, I, \delta, F)$, we call the history tree $\langle T_0, l_0 \rangle = \langle \{\varepsilon\}, \varepsilon \mapsto I \rangle$ that contains only the empty word and maps it to the initial states I of $\mathcal{A}$ the initial history tree.

For an input word $\alpha : \omega \to \Sigma$ we call the sequence $\langle T_0, l_0 \rangle, \langle T_1, l_1 \rangle, \ldots$ of history trees that start with the initial history tree $\langle T_0, l_0 \rangle$ and where, for every $i \in \omega$, $\langle T_i, l_i \rangle$ is followed by $\alpha(i)$-successor $\langle T_{i+1}, l_{i+1} \rangle$ the *history trace* or α. A node τ in the history tree $\langle T_{i+1}, l_{i+1} \rangle$ is called stable or accepting, respectively, if it is stable or accepting in the $\alpha(i)$-transition from $\langle T_i, l_i \rangle$ to $\langle T_{i+1}, l_{i+1} \rangle$.

Proposition 1. *An ω-word α is accepted by a nondeterministic Büchi automaton $\mathcal{A}$ if, and only if, there is a node $\tau \in \omega^*$ such that τ is eventually always stable and always eventually accepting in the history trace of α.*

Proof. For the "if" direction, let $\tau \in \omega^*$ be a node that is eventually always stable and always eventually accepting, and let $i_0 < i_1 < i_2 < \ldots$ be an ascending chain of indices such that τ is stable for the $\alpha(j)$-transitions from $\langle T_j, l_j \rangle$ to $\langle T_{j+1}, l_{j+1} \rangle$ for all $j \geq i_0$, and accepting for the $\alpha(i-1)$-transition from $\langle T_{i-1}, l_{i-1} \rangle$ to $\langle T_i, l_i \rangle$ for all indices i in the chain.

By definition of the σ-transitions, for every $j \in \omega$, the *finite* automaton $\mathcal{A}_j = (\Sigma, Q, l_{i_j}(\tau), \delta, F)$ has, for every state $q \in l_{i_{j+1}}(\tau)$, a run ρ_j^q on the finite word $\alpha(i_j)\alpha(i_{j+1})\alpha(i_{j+2}) \ldots \alpha(i_{j+1} - 1)$ that contains an accepting state and ends in q. Also, $\mathcal{A} = (\Sigma, Q, I, \delta, F)$ read as a finite automaton has, for every state $q \in l_{i_0}(\tau)$, a run ρ^q on the finite word $\alpha(0)\alpha(1)\alpha(2) \ldots \alpha(i_0 - 1)$ that ends in q. Let us fix such runs, and define a tree $T \subseteq Q^*$ that contains, besides the empty word and the initial states, a node iq_0 of length 2 if q_0 is in $l_{i_{j+1}}(\tau)$ and i is the first letter of ρ^{q_0}, and a node $iq_0q_1q_2\dots q_kq_{k+1}$ of length $k+1>2$ if $iq_0q_1q_2\dots q_k$ is in T, q_{k+1} is in $l_{i_{k+1}}(n)$ and q_k is the first letter of $\rho_k^{q_{k+1}}$. By construction, T is an infinite tree with finite branching degree, and therefore contains an infinite path $iq_0q_1q_2\dots$ by König's Lemma. By construction, $\rho^{q_0}\rho_0^{q_1}\rho_1^{q_2}\dots$ is a run of $\mathcal{A}$ on α that visits some accepting state infinitely many times.

To demonstrate the "only if" direction, let us fix an accepting run, $\rho = q_0q_1\dots$ of $\mathcal{A}$ on an input word α. Then we can define the sequence $\vartheta = \tau_0\tau_1\dots$ of nodes such that, for the history trace $\langle T_0, l_0\rangle, \langle T_1, l_1\rangle, \dots$, τ_i is the host node of $q_i \in l_i(\tau_i)$ for the history tree $\langle T_i, l_i\rangle$. Let l be the shortest length $|\tau_i|$ of these nodes that occurs infinitely many times.

It is easy to see that the initial sequence of length l of the nodes in ϑ eventually stabilises: Let $i_0 < i_1 < i_2 < \dots$ be an infinite ascending chain of indices such that the length $|\tau_j| \geq l$ of the j-th node is not smaller than l for all $j \geq i_0$, and equal to $l = |\tau_i|$ for all indices $i \in \{i_0, i_1, i_2, \dots\}$ in this chain. This implies that $\tau_{i_0}, \tau_{i_1}, \tau_{i_2}, \dots$ is a descending chain when the single nodes τ_i are compared by lexicographic order, and hence almost all $\tau_i := \pi$ are equal. This also implies that π is eventually always stable.

Let us assume that π is accepting only finitely many times. Then we can chose an index i from the chain $i_0 < i_1 < i_2 < \dots$ such that $\tau_j = \pi$ holds for all indices $j \geq i$, and π is not accepting for any $j \geq i$. (Note that every time the length of τ_j is *reduced* to l, τ_j is unstable, which we excluded, or accepting, which violates the assumption.) But now we can pick an index $i' > i$ such that $q_{i'} \in F$ is a final state, which, together with $\tau_{i'} = \pi$, implies that π is accepting for $\big(\langle T_{i'-1}, l_{i'-1}\rangle, \alpha(i'-1), \langle T_{i'}, l_{i'}\rangle\big)$. (Note that $q_{i'}$ is in the label of $\pi\cdot\deg_{T_{i'-1}}(\pi)$ in the labelled tree $\langle T'_{i'-1}, l'_{i'-1}\rangle$ resulting from the first step of the σ-transition of history trees.) $\lightning$ $\square$

4 From Nondeterministic Büchi Automata to Deterministic Rabin Automata

In this section, we discuss three determinisation procedures for nondeterministic Büchi automata. First we observe that the acceptance mechanism from the previous section already describes a deterministic automaton with a Rabin condition, but the Rabin condition is on the *transitions*. This provides us with the first corollary:

Corollary 1. *For a given nondeterministic Büchi automaton with n states, we can construct a deterministic Rabin transition*[3] *automaton with $o\big((1.65\,n)^n\big)$ states and 2^n-1 accepting pairs that recognises the language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$.* $\square$

[3] A transition automaton records the history of transitions in addition to the history of states. For such a history of transitions, we can translate the acceptance condition $1:1$ by using the nodes as index set, and (A_τ, R_τ) where A_τ are the transitions where τ is accepting, and R_τ are the transitions where τ is unstable as Rabin pairs.

To see that the number of accepting pairs is bounded by $2^n - 1$, note that the labels of siblings are disjoint, and that the label of every node contains a state not in the label of any of its children. Thus, the number of ancestors and their older siblings of every node is strictly smaller than n. Thus, a node $i_0 i_1 i_2 \ldots i_n$ can be represented by a sequence of i_0 0's followed by a 1, followed by i_1 0's and so on, such that every node that can be accepting is representable by a sequence of strictly less than n 0's and 1's.

There are two obvious ways to transform an automaton with a Rabin condition on the transitions to an automaton with Rabin conditions on the states. The first option is to "postpone" the transitions by one step. The new states are (with the exception of one dedicated initial state $\widehat{q_0}$) pairs, consisting of a state of the transition automaton and the input letter read in the previous round. Thus, if the deterministic Rabin transition automaton has the run ρ on an input word α, then the resulting ordinary deterministic Rabin automaton has the run $\rho' = \widehat{q_0}, \big(\rho(0), \alpha(0)\big), \big(\rho(1), \alpha(1)\big), \big(\rho(2), \alpha(2)\big), \ldots$.

Corollary 2. *For a given nondeterministic Büchi automaton $\mathcal{A}$ with n states over an alphabet with l letters, we can construct a deterministic Rabin automaton with $l \cdot o\big((1.65\, n)^n\big)$ states and $2^n - 1$ accepting pairs that recognises the language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$.* □

Given that the alphabets tend to be small in practice — in particular compared to $(1.65\, n)^n$ — a blow-up linear in the alphabet size is usually acceptable. However, an alphabet may, in principle, have up to 2^{n^2} distinguishable letters, and the imposed bound is not very good for extremely large alphabets. (Two letters σ_1 and σ_2 can be considered equivalent or indistinguishable for a Büchi automaton $\mathcal{A} = (\Sigma, Q, I, \delta, F)$ if $\delta(q, \sigma_1) = \delta(q, \sigma_2)$ holds true for all states $q \in Q$ of the automaton $\mathcal{A}$.) As an alternative to preserving one input letter in the state-space, we enrich the history trees with information about which node of the resulting tree was accepting or unstable in the third step of the transition.

To estimate the number of different enriched history trees with n nodes, we have to take into account that the unstable and accepting nodes are not arbitrarily distributed over the tree: Only leaves can be accepting, and if a node of the tree in unstable, then all of its descendants and all of its younger siblings are unstable, too. Furthermore, only stable nodes can be accepting and the root cannot be unstable. (An unstable root implies that the Büchi automaton has no run for this word. Instead of allowing for an unstable root, we use a partial transition function.)

The number $eOrder(n)$ of ordered trees enriched with this information can be recursively computed using the following case distinction: If the eldest child 0 of the root is unstable, then all nodes but the root are unstable. Hence, the number of trees of this form is $order(n) = \frac{(2n-2)!}{n!(n-1)!}$. For the case that the eldest child 0 of the root is stable, there are $eOrder(n-1)$ trees where the size of the sub-tree rooted in 0 is $n-1$, and $eOrder(i) \cdot eOrder(n-i)$ trees where the sub-tree rooted in 0 contains $i \in \{1, \ldots, n-2\}$ nodes. (Every tree can be uniquely defined by the tree rooted in 0, and the remaining tree. The special treatment of the case that

0 has no younger siblings is due to the fact that the root cannot be accepting if it has a child.) Thus, we have $eOrder(1) = 2$ (as a leaf, the root can be accepting or stable but not accepting), and

$$eOrder(n) = eOrder(n-1) + order(n) + \sum_{i=1}^{n-2} eOrder(i)\,eOrder(n-1)$$

for $n \geq 2$. A numerical analysis[4] of this sequence shows that $eOrder(n) < 6.738^n$. This allows for an estimation of the number $eHist(n)$ of enriched history trees for a Büchi automaton with n states similar to the estimation of the number $hist(n)$ of history trees:

$$eHist(n) \in \sup_{x>0} O\big(m(x) \cdot 6.738^{\beta(x)}\big) \subset o\big((2.66\,n)^n\big).$$

Corollary 3. *Given a nondeterministic Büchi automaton $\mathcal{A}$ with n states, we can construct a deterministic Rabin automaton with $o\big((2.66\,n)^n\big)$ states and $2^n - 1$ accepting pairs that recognises the language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$.* □

5 From Nondeterministic Büchi Automata to Deterministic Parity Automata

From a practical point of view, it is often preferable to trade state-space for simpler acceptance conditions. Algorithms that solve Rabin games, for example, are usually exponential in the index, while the index of the constructions discussed in the previous sections is exponential in the size to the Büchi automaton we want to determinise.

While a reasonable index has been a side product of previous determinisation techniques [3,4,5], the smaller state-spaces resulting from the determinisation techniques discussed in Sections 3 and 4 are partly paid for by a higher index.

Traditional techniques for the transformation of Muller and Rabin or Streett to parity acceptance conditions use later [19] and index appearance records [4], respectively. However, using index (or later) appearance records would result in an exponential blow-up of the state-space, and hence in a doubly exponential construction. We therefore introduce the later introduction record as a record tailored for ordered trees.

A *later introduction record* (LIR) stores the order in which the nodes of the ordered trees have been introduced. For an ordered tree T with m nodes, a later introduction record is a sequence $\tau_1, \tau_2, \ldots \tau_m$ that contains the nodes of T, such that every node appears after its parent and older siblings.

To analyse the effect of adding a later introduction record to a history tree on the state-space, we slightly change the representation: We represent the tree structure of a tree with m nodes *and* its later introduction record by a sequence

[4] $\frac{eOrder(n+1)}{eOrder(n)}$ is growing, and $\big(\frac{eOrder(n+1)}{eOrder(n)}\big)\big(1 + \frac{2}{n}\big)$ is falling for growing $n \geq 2$.

of $m-1$ integers $i_2, i_3, \dots i_m$, such that i_j points to the position $< j$ of the parent of node τ_j in the later introduction record $\tau_1, \tau_2, \dots \tau_m$. (The root τ_1 has no parent.) There are $(m-1)!$ such sequences.

The labelling function of a history tree $\langle T, l\rangle$ whose root is labelled with the complete set Q of states of the Büchi automaton can be represented by a function from Q onto $\{1, \dots, m\}$ that maps each state $q \in Q$ to the positions of its host node in the LIR. Let $t(n,m)$ denote the number of trees and later introduction record pairs for such history trees with m nodes and $n = |Q|$ states in the label of the root. First, $t(n,n) = (n-1)!n!$ holds: There are $(n-1)!$ ordered-tree / LIR pairs, and $n!$ functions from a set with n elements onto itself. For every $m \leq n$, a coarse estimation[5] provides $t(n, m-1) \leq \frac{1}{2}t(n,m)$. Hence, $\sum_{i=1}^{n} t(n,i) \leq 2(n-1)!n!$.

Likewise, the labelling function of a history tree $\langle T, l\rangle$ whose root is labelled with the complete set Q of states of the Büchi automaton can be represented by a function from Q onto $\{1, \dots, m\}$ that maps each state $q \in Q$ to the positions of its host node in the LIR, or to 0 if the state is not in the label of the root. Let $t'(n,m)$ denote the number of history tree / LIR pairs for such history trees with m nodes for a Büchi automaton with n states. We have $t'(n,n-1) = (n-2)!n!$ and, by an argument similar to the one used in the analysis of t, we also have $t'(n,m-1) \leq \frac{1}{2}t'(n,m)$ for every $m < n$, and hence $\sum_{i=1}^{n-1} t'(n,i) \leq 2(n-2)!n!$.

Proposition 2. *For a given nondeterministic Büchi automaton $\mathcal{A}$ with n states, we can build a deterministic parity automaton with $O(n!^2)$ states and $2n$ priorities that recognises the language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$.*

Proof. We construct a deterministic parity automaton, whose states consist of the history tree / LIR pairs, and an explicitly represented priority. The priority is determined by the position i of the first node in the previous LIR that is either unstable or accepting in the σ-transition: If it is accepting, the priority is $2i$, if it is unstable, the priority is $2i-1$. If no node is unstable or accepting, the priority is $2n+1$. The automaton has at most the priorities $\{2, 3, \dots, 2n+1\}$ and $O(n!^2)$ states — $O\big((n-1)!n!\big)$ history tree / LIR pairs times $2n$ priorities.

Let α be a word in the language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$. Then there is by Proposition 1 a node τ that is always eventually accepting and eventually always stable in the history tree, and will hence eventually always remain in the same position p in the LIR and be stable. (A stable node can only move further to the front of the LIR, which can only happen finitely many times.) From that time onward, no node with a smaller position $p' < p$ is deleted (this would result τ to move further to the front of the record), nor is the node τ on position p unstable. Hence, no odd number $< 2p$ occurs infinitely many times. Also from that time

[5] If we connect functions by letting a function g from Q onto $\{1, \dots, m-1\}$ be the successor of a function f from Q onto $\{1, \dots, m\}$ if there is an index $i \in \{1, \dots, m-1\}$ such that $g(q) = i$ if $f(q) = m$ and $g(q) = f(q)$ otherwise, then the functions onto m have $(m-1)$ successors, while every function onto $m-1$ has at least two predecessors. Hence, the number of labelling functions growth at most by a factor of $\frac{m-1}{2}$, while the number of ordered tree / LIR pairs is reduced by a factor of $m-1$.

onward, the node τ is accepting infinitely many times, which results in visiting a priority $\leq 2p$ by our prioritisation rule. Hence the smallest number occurring infinitely many times is even.

Let, on the other hand, $2i$ be the dominating priority of the run of our deterministic parity automaton. Then eventually no lower priority than $2i$ appears, which implies that all positions $\leq i$ remain unchanged in the LIR, and the respective nodes remain stable from that time onward. Also, the node that is from that time onward on position i is accepting infinitely many times, which implies by Proposition 1 that α is in the language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$. $\square$

While the separation of concerns does not generate the same theoretical benefit with respect to state-space reduction when we construct parity automata instead of Rabin automata, the practical advantage might be comparable. While the update rule for history trees is strict, the update rule for LIR's is much less so: The only property of LIR updates used in the proof of Proposition 2 is that the position of accepting positions is reduced, and strictly reduced if there was an unstable node on a smaller position of the previous LIR. This leaves much leeway for updating the LIR — any update that satisfies this constraint will do.

Usually only a fragment of the state-space is reachable, and determinisation algorithms tend to construct the state-space of the automaton on the fly. The simplest way to exploit the leeway in the update rule for LIR's is to check if there is a suitable LIR such that a state with an appropriate history tree / LIR pair has already been constructed. If this is the case, then we can, depending on the priority of that state, turn to this state or construct a new state that differs only in the priority, which allows us to ignore the new state in the further expansion of the state-space. It is my belief that such a nondeterministic determinisation procedure will result in a significant state-space reduction compared to any deterministic rule.

References

1. Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Proceedings of the International Congress on Logic, Methodology, and Philosophy of Science, 1960, Berkeley, California, USA, pp. 1–11. Stanford University Press (1962)
2. Rabin, M.O., Scott, D.S.: Finite automata and their decision problems. IBM Journal of Research and Development 3, 115–125 (1959)
3. Safra, S.: On the complexity of ω-automata. In: Proceedings of the 29th Annual Symposium on Foundations of Computer Science (FOCS 1988), White Plains, New York, USA, pp. 319–327. IEEE Computer Society Press, Los Alamitos (1988)
4. Muller, D.E., Schupp, P.E.: Simulating alternating tree automata by nondeterministic automata: new results and new proofs of the theorems of Rabin, McNaughton and Safra. Theoretical Computer Science 141, 69–107 (1995)
5. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. Journal of Logical Methods in Computer Science 3 (2007)
6. Muller, D.E.: Infinite sequences and finite machines. In: Proceedings of the 4th Annual Symposium on Switching Circuit Theory and Logical Design (FOCS 1963), Chicago, Chicago, Illinois, USA, pp. 3–16. IEEE Computer Society Press, Los Alamitos (1963)

7. McNaughton, R.: Testing and generating infinite sequences by a finite automaton. Information and Control 9, 521–530 (1966)
8. Yan, Q.: Lower bounds for complementation of *omega*-automata via the full automata technique. Journal of Logical Methods in Computer Science 4 (2008)
9. Church, A.: Logic, arithmetic and automata. In: Proceedings of the International Congress of Mathematicians, Institut Mittag-Leffler, Djursholm, Sweden, 1962 (Stockholm 1963), 15–22 August, pp. 23–35 (1962)
10. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Proceedings of the 16th Annual ACM Symposium on Principles of Programming Languages (POPL 1989), Austin, Texas, USA, pp. 179–190. ACM Press, New York (1989)
11. Rabin, M.O.: Decidability of second order theories and automata on infinite trees. Transaction of the American Mathematical Society 141, 1–35 (1969)
12. Büchi, J.R., Landweber, L.H.: Solving sequential conditions by finite-state strategies. Transactions of the American Mathematical Society 138, 295–311 (1969)
13. Büchi, J.R., Landweber, L.H.: Definability in the monadic second-order theory of successor. Journal of Symbolic Logic 34, 166–170 (1969)
14. Wilke, T.: Alternating tree automata, parity games, and modal μ-calculus. Bulletin of the Belgian Mathematical Society 8 (2001)
15. Emerson, E.A.: Automata, tableaux and temporal logics. In: Parikh, R. (ed.) Logic of Programs 1985. LNCS, vol. 193, pp. 79–88. Springer, Heidelberg (1985)
16. Kupferman, O., Vardi, M.: Module checking revisited. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 36–47. Springer, Heidelberg (1997)
17. Emerson, E.A., Jutla, C.S.: Tree automata, μ-calculus and determinacy. In: Proceedings of the 32nd Annual Symposium on Foundations of Computer Science (FOCS 1991), San Juan, Puerto Rico, pp. 368–377. IEEE Computer Society Press, Los Alamitos (1991)
18. Schewe, S., Finkbeiner, B.: Satisfiability and finite model property for the alternating-time μ-calculus. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 591–605. Springer, Heidelberg (2006)
19. Gurevich, Y., Harrington, L.: Trees, automata, and games. In: Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC 1982), San Francisco, California, USA, pp. 60–65. ACM Press, New York (1982)
20. Liu, W., Wang, J.: A tigher analysis of Piterman's Büchi determinization. Information Processing Letters (submitted, 2009)
21. Temme, N.M.: Asymptotic estimates of Stirling numbers. Studies in Applied Mathematics 89, 233–243 (1993)
22. Friedgut, E., Kupferman, O., Vardi, M.Y.: Büchi complementation made tighter. International Journal of Foundations of Computer Science 17, 851–867 (2006)

Lower Bounds on Witnesses for Nonemptiness of Universal Co-Büchi Automata

Orna Kupferman[1] and Nir Piterman[2,⋆]

[1] Hebrew University
[2] Imperial College London

Abstract. The nonemptiness problem for nondeterministic automata on infinite words can be reduced to a sequence of reachability queries. The length of a shortest witness to the nonemptiness is then polynomial in the automaton. Nonemptiness algorithms for alternating automata translate them to nondeterministic automata. The exponential blow-up that the translation involves is justified by lower bounds for the nonemptiness problem, which is exponentially harder for alternating automata. The translation to nondeterministic automata also entails a blow-up in the length of the shortest witness. A matching lower bound here is known for cases where the translation involves a $2^{O(n)}$ blow up, as is the case for finite words or Büchi automata.

Alternating co-Büchi automata and witnesses to their nonemptiness have applications in model checking (complementing a nondeterministic Büchi word automaton results in a universal co-Büchi automaton) and synthesis (an LTL specification can be translated to a universal co-Büchi tree automaton accepting exactly all the transducers that realize it). Emptiness algorithms for alternating co-Büchi automata proceed by a translation to nondeterministic Büchi automata. The blow up here is $2^{O(n \log n)}$, and it follows from the fact that, on top of the subset construction, the nondeterministic automaton maintains ranks to the states of the alternating automaton. It has been conjectured that this super-exponential blow-up need not apply to the length of the shortest witness. Intuitively, since co-Büchi automata are memoryless, it looks like a shortest witness need not visit a state associated with the same set of states more than once. A similar conjecture has been made for the width of a transducer generating a tree accepted by an alternating co-Büchi tree automaton. We show that, unfortunately, this is not the case, and that the super-exponential lower bound on the witness applies already for universal co-Büchi word and tree automata.

1 Introduction

Finite automata on infinite objects were first introduced in the 60's. Motivated by decision problems in mathematics and logic, Büchi, McNaughton, and Rabin developed a framework for reasoning about infinite words and trees [2,11,16]. The framework has proven to be very powerful. Automata, and their tight relation to second-order monadic logics were the key to the solution of several fundamental decision problems in mathematics and logic [17]. Indeed, for many highly expressive logics, it is possible to translate

⋆ Supported by the UK EPSRC project Complete and Efficient Checks for Branching-Time Abstractions (EP/E028985/1).

a formula in the logic to an automaton accepting exactly all the models satisfying the formula. The formula is then satisfiable iff the language of the automaton is not empty. Thus, decidability can be reduced to the emptiness problem.

Today, automata on infinite objects are used for specification and verification of nonterminating systems [18,9,19]. The emptiness problem plays a key role also in these more modern applications. Two important examples are model checking and synthesis. Model checking a system with respect to a specification is reduced to checking the emptiness of the product of the system with an automaton accepting exactly all models that violate the specification [19]. Synthesis of a reactive system that satisfies a desired specification is reduced to checking the emptiness of a tree automaton accepting all possible strategies that realize the specification [15].

In the case of finite nondeterministic automata on finite words, the emptiness problem is simple: The automaton accepts some word if there is a path from an initial state to an accepting state (c.f., [4]). Thus, the automaton is viewed as a graph, its alphabet is ignored, and emptiness is reduced to reachability in finite graphs. An important and useful outcome of this simplicity is the fact that when the language of the automaton is not empty, it is easy to return a witness to the nonemptiness — a word v that labels a path from an initial state to a final states. Clearly, reachability may be checked only along simple paths, thus the length of a witness is bounded by the number of states of the automaton.

The case of finite nondeterministic automata on infinite words is similar. Acceptance in such automata depends on the set of states that a run visits infinitely often. For example, in the Büchi acceptance condition, some states are designated as accepting, and in order for a run to be accepting it has to visit at least one of these states infinitely often. Nonemptiness is slightly more complicated, but again, the automaton is viewed as a graph, its alphabet is ignored, and emptiness is reduced to a sequence of reachability queries in finite graphs. Now, the witness to the nonemptiness is a word of the form $v \cdot u^\omega$, where the word v labels a path from an initial state to some accepting state, and the word u labels a path from this accepting state to itself. Since both v and u are extracted from reachability queries on the graph, their lengths are bounded by the number of states of the automaton.[1] For acceptance conditions more complicated than Büchi, the emptiness test is more involved, but still, as long as we consider nondeterministic automata, emptiness can be reduced to a sequence of reachability queries on the graph of the automaton, and a nonempty automaton has a witness of the form $v \cdot u^\omega$ for v and u polynomial in the number of states of the automaton.

Alternating automata enrich the branching structure of the automaton by combining universal and existential branching. In the presence of alternation, we can no longer ignore the alphabet when reasoning about emptiness. Indeed, the different copies of the automaton have to agree on the letters they read on the same position of the word. The standard solution is to remove alternation by translating the automaton to an equivalent nondeterministic automaton, and checking the emptiness of the latter. This simple solution is optimal, as the exponential blow-up that the translation involves is justified by lower bounds for the nonemptiness problem, which is exponentially harder in the alternating setting (c.f., NLOGSPACE vs. PSPACE for nondeterministic vs. alternating automata on finite words).

[1] In fact, it can be shown that even the sum of their lengths is bounded by the number of states of the automaton [6].

The translation to nondeterministic automata also entails an exponential blow-up in the length of the shortest witness. Can this blow up be avoided? A negative answer for this question is known for alternating automata on finite words and alternating Büchi automata. There, removing alternation from an alternating automaton with n states results in a nondeterministic automaton with $2^{O(n)}$ states [3,12], and it is not hard to prove a matching lower bound [1]. Note also that a polynomial witness would have led to the nonemptiness problem being in NP, whereas it is known to be PSPACE-complete.

Things become challenging when the removal of alternation involves a super-exponential blow up. In particular, emptiness algorithms for alternating co-Büchi automata proceed by a translation to nondeterministic Büchi automata, and the involved blow up is $2^{O(n \log n)}$. Alternating co-Büchi automata have been proven useful in model checking (complementing a nondeterministic Büchi word automaton results in a universal co-Büchi automaton) and synthesis (an LTL specification can be translated to a universal co-Büchi tree automaton accepting exactly all the transducers that realize it [8,5]). In the case of model checking, the witness to the nonemptiness is a computation that violates the property. In the case of synthesis, the witness is a system that realizes the specification). Thus, we clearly seek shortest witnesses.

The $2^{O(n \log n)}$ blow up follows from the fact that, on top of the subset construction, the nondeterministic automaton maintains ranks to the states of the alternating automaton. It has been conjectured that this super-exponential blow-up need not apply to the length of the shortest witness. Intuitively, since co-Büchi automata are memoryless, it seems as if a shortest witness need not visit a state associated with the same set of states more than once. This intuition suggests that a shortest witness need not be longer than $2^{O(n)}$. A similar conjecture has been made for the width of a transducer[2] generating a tree accepted by an alternating co-Büchi tree automaton [8].

In this paper we show that, unfortunately, this is not the case, and the super-exponential blow-up in the translation of alternating co-Büchi automata to nondeterministic Büchi automata is carried over to a super-exponential lower bound on the witness to the nonemptiness. In fact, the lower bound applies already for universal co-Büchi automata. We start with the linear framework. There, we show that for every odd integer $n \geq 1$, there exists a universal co-Büchi word automaton $\mathcal{A}_n$ with n states such that the shortest witness to the nonemptiness of $\mathcal{A}_n$ has a cycle of length $\frac{n+1}{2}!$.

In the branching framework, the witness to the nonemptiness is a transducer that generates a tree accepted by the automaton. The linear case trivially induces a lower bound on the size of such a transducer. In the branching framework, however, it is interesting to consider also the width of the witness transducer. In particular, the LTL synthesis algorithm in [8], which is based on checking the nonemptiness of a universal co-Büchi tree automaton, is incremental, and it terminates after k iterations, with k being an upper bound on the width of a transducer generating a tree accepted by the automaton. The bound used in [8] is super-exponential, and has been recently tightened to $2n(n!)^2$ [14,10]. It is conjectured in [8] that the bound can be improved to $2^{O(n)}$. As in the word case, the intuition is convincing: The alternating automaton may send a set of states to a subtree of the input tree, in which case the subtree should be accepted by all the states in the set. The memoryless nature of the co-Büchi condition suggests that if in an accepting run of the automaton the

[2] Essentially, the width of a transducer is the number of different states that the transducer may be at after reading different input sequences of the same length.

same set of states is sent to different subtrees, then there is also an accepting run on a tree in which these subtrees are identical. Thus, we do not need more than 2^n different subtrees in a single level of the input tree. We show that, unfortunately, this intuition fails, and there is a lower bound of $\frac{n+1}{2}!$ on the width of the transducer. Formally, we show that for every odd integer $n \geq 1$, there exists a universal co-Büchi tree automaton $\mathcal{B}_n$ with n states such that every tree accepted by $\mathcal{B}_n$ is such that, all levels beyond a finite prefix have at least $\frac{n+1}{2}!$ different subtrees. Thus, the minimal width of a transducer that generate a tree accepted by $\mathcal{B}_n$ has width at least $\frac{n+1}{2}!$.

Our constructions use a very large alphabet. Indeed, the alphabet of the automata $\mathcal{A}_n$ and $\mathcal{B}_n$ has $\frac{n+1}{2}!$ letters. In the case of words, the word accepted by the automaton is a cycle consisting of all these letters ordered in some fixed order (say, lexicographically). The case of trees is similar. We were not able to reduce the size of the alphabet. While the question of a smaller alphabet is very interesting, it is of less practical importance: Constructions for removal of alternation introduce an exponential alphabet in an intermediate step (where the exponent is quadratic in the number of states). The larger alphabet is discarded at a later stage but the degree of nondeterminism induced by it remains in the resulting nondeterministic automaton. Furthermore, the size of the alphabet does not play a role in these constructions, and obviously does not play a role when checking the emptiness of a nondeterministic automaton.

2 Universal Co-Büchi Word Automata

A *word automaton* is $\mathcal{A} = \langle \Sigma, Q, \delta, Q_{in}, \alpha \rangle$, where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, $Q_{in} \subseteq Q$ is a set of initial states, and α is an acceptance condition that defines a subset of Q^ω.

Given an input word $w = \sigma_0 \cdot \sigma_1 \cdots$ in Σ^ω, a *run* of $\mathcal{A}$ on w is a word $r = q_0, q_1, \ldots$ in Q^ω such that $q_0 \in Q_{in}$ and for every $i \geq 0$, we have $q_{i+1} \in \delta(q_i, \sigma_i)$; i.e., the run starts in the initial state and obeys the transition function. Since the transition function may specify many possible transitions for each state and letter, $\mathcal{A}$ may have several runs on w. A run is accepting iff it satisfies the acceptance condition α. We consider here the *Büchi* acceptance condition, where $\alpha \subseteq Q$ is a subset of Q. For a run r, let $inf(r)$ denote the set of states that r visits infinitely often. That is, $inf(r) = \{q \in Q : q_i = q \text{ for infinitely many } i \geq 0\}$. A run r is accepting iff $inf(r) \cap \alpha \neq \emptyset$. That is, r is accepting if some state in α is visited infinitely often. The *co-Büchi* acceptance condition dualizes the Büchi condition. Thus, again α is a subset of Q, but a run r is accepting if $inf(r) \cap \alpha = \emptyset$. Thus, r visits all the states in α only finitely often.

If the automaton $\mathcal{A}$ is *nondeterministic*, then it accepts an input word w iff it has an accepting run on w. If $\mathcal{A}$ is *universal*, then it accepts w iff all its runs on w are accepting. The *language* of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$ is the set of words that $\mathcal{A}$ accepts. Dualizing a nondeterministic Büchi automaton (NBW, for short) amounts to viewing it as a universal co-Büchi automaton (UCW, for short). It is easy to see that by dualizing $\mathcal{A}$, we get an automaton that accepts its complementary language.

In [7], Kupferman and Vardi analyze runs of UCW in terms of a ranking function one can associate with their run DAG. In the rest of this section, we describe their analysis.

Let $\mathcal{A} = \langle \Sigma, Q, Q_{in}, \delta, \alpha \rangle$ be a universal co-Büchi automaton with α. Let $|Q| = n$. The runs of $\mathcal{A}$ on a word $w = \sigma_0 \cdot \sigma_1 \cdots$ can be arranged in an infinite DAG (directed acyclic graph) $G = \langle V, E \rangle$, where

– $V \subseteq Q \times \mathbb{N}$ is such that $\langle q, l \rangle \in V$ iff some run of $\mathcal{A}$ on w has $q_l = q$. For example, the first level of G contains the vertices $Q_{in} \times \{0\}$.
– $E \subseteq \bigcup_{l \geq 0} (Q \times \{l\}) \times (Q \times \{l+1\})$ is such that $E(\langle q, l \rangle, \langle q', l+1 \rangle)$ iff $\langle q, l \rangle \in V$ and $q' \in \delta(q, \sigma_l)$.

Thus, G embodies exactly all the runs of $\mathcal{A}$ on w. We call G the *run* DAG of $\mathcal{A}$ on w. We say that a vertex $\langle q, l \rangle$ in G is an *α-vertex* iff $q \in \alpha$. We say that G is *accepting* if each path p in G contains only finitely many α-vertices. It is easy to see that $\mathcal{A}$ accepts w iff G is accepting.

Let $[2n]$ denote the set $\{0, 1, \ldots, 2n\}$. A *ranking* for G is a function $f : V \to [2n]$ that satisfies the following conditions:

1. For all vertices $\langle q, l \rangle \in V$, if $f(\langle q, l \rangle)$ is odd, then $q \notin \alpha$.
2. For all edges $\langle \langle q, l \rangle, \langle q', l+1 \rangle \rangle \in E$, we have $f(\langle q', l+1 \rangle) \leq f(\langle q, l \rangle)$.

Thus, a ranking associates with each vertex in G a rank in $[2n]$ so that ranks along paths decrease monotonically, and α-vertices cannot get an odd rank. Note that each path in G eventually gets trapped in some rank. We say that the ranking f is an *odd ranking* if all the paths of G eventually get trapped in an odd rank. Formally, f is odd iff for all paths $\langle q_0, 0 \rangle, \langle q_1, 1 \rangle, \langle q_2, 2 \rangle, \ldots$ in G, there is $l \geq 0$ such that $f(\langle q_l, l \rangle)$ is odd, and for all $l' \geq l$, we have $f(\langle q_{l'}, l' \rangle) = f(\langle q_l, l \rangle)$. Note that, equivalently, f is odd if every path of G has infinitely many vertices with odd ranks.

We now analyze the form of accetping run DAGs. The following three lemmata relate to DAGs induced by words accepted by $\mathcal{A}$. Consider a (possibly finite) DAG $G' \subseteq G$. We say that a vertex $\langle q, l \rangle$ is *finite* in G' iff only finitely many vertices in G' are reachable from $\langle q, l \rangle$. We say that a vertex $\langle q, l \rangle$ is *α-free* in G' iff all the vertices in G' that are reachable from $\langle q, l \rangle$ are not α-vertices. Note that, in particular, $\langle q, l \rangle$ is not an α-vertex.

We define an infinite sequence of DAGs $G_0 \supseteq G_1 \supseteq G_2 \supseteq G_3 \supseteq \ldots$ as follows.

– $G_0 = G$.
– $G_{2i+1} = G_{2i} \setminus \{\langle q, l \rangle \mid \langle q, l \rangle \text{ is finite in } G_{2i}\}$.
– $G_{2i+2} = G_{2i+1} \setminus \{\langle q, l \rangle \mid \langle q, l \rangle \text{ is } \alpha\text{-free in } G_{2i+1}\}$.

Lemma 1. *For every $i \geq 0$, there exists l_i such that for all $l \geq l_i$, there are at most $n - i$ vertices of the form $\langle q, l \rangle$ in G_{2i}.*

Lemma 1 implies that G_{2n} is finite, and G_{2n+1} is empty.

Each vertex $\langle q, l \rangle$ in G has a unique $i \geq 1$ such that $\langle q, l \rangle$ is either finite in G_{2i} or α-free in G_{2i+1}. This induces a function $f : V \to [2n]$ defined as follows.

$$f(\langle q, l \rangle) = \begin{bmatrix} 2i & \text{If } \langle q, l \rangle \text{ is finite in } G_{2i}. \\ 2i+1 & \text{If } \langle q, l \rangle \text{ is } \alpha\text{-free in } G_{2i+1}. \end{bmatrix.$$

Lemma 2. *For every two vertices $\langle q, l \rangle$ and $\langle q', l' \rangle$ in G, if $\langle q', l' \rangle$ is reachable from $\langle q, l \rangle$, then $f(\langle q', l' \rangle) \leq f(\langle q, l \rangle)$.*

Lemma 3. *For every infinite path in G, there exists and a vertex $\langle q, l \rangle$ such that all the vertices $\langle q', l' \rangle$ on the path that are reachable from $\langle q, l \rangle$ have $f(\langle q', l' \rangle) = f(\langle q, l \rangle)$.*

We can now conclude with Theorem 1 below.

Theorem 1. [7] *The* DAG G *is accepting iff it has an odd ranking.*

Proof. Assume first that there is an odd ranking for G. Then, every path in G eventually gets trapped in some odd rank. Hence, as α-vertices cannot get this rank, the path visits α only finitely often, and we are done.

For the other direction, note that Lemma 2, together with the fact that a vertex gets an odd rank only if it is α-free, imply that the function f described above is a ranking. Lemma 3 then implies that the ranking is odd. □

3 Lower Bound on Length of Accepted Words

In this section we construct, for every odd $n \geq 1$, a UCW $\mathcal{A}_n$ with n states such that the shortest words accepted by $\mathcal{A}$ have a cycle of length $\frac{n+1}{2}!$. The alphabet Σ_n of $\mathcal{A}_n$ has $\frac{n+1}{2}!$ letters, and there is an ordering $\leq$ of all the letters in Σ_n such that $\mathcal{A}_n$ accepts exactly all words vu^ω, where $v \in \Sigma_n^*$ and $u \in (\Sigma_n)^{\frac{n+1}{2}!}$ has all the letters in Σ_n ordered according to $\leq$.

Formally, given an odd $n \geq 1$, let $\mathcal{A}_n = \langle \Sigma_n, Q_n, \delta_n, Q_n, \alpha_n \rangle$, where

- Let Π_n be the set of permutations on $\{1, 3, 5, \ldots, n\}$ (the odd members of $\{1, \ldots, n\}$), and let $\leq$ be the lexicographic ordering[3] on the members of Π_n. Then, $\Sigma_n \subseteq \Pi_n \times \Pi_n$ is such that $\langle \pi, \pi' \rangle \in \Sigma_n$ iff π' is the (cyclic) successor of π in the order $\leq$. Thus, each letter of Σ_n is a pair $\langle \pi, \pi' \rangle$ of permutations, such that π' is the successor of π in the lexicographic order of Π_n. Note we refer to the order in a cyclic way, thus $\langle n \ldots 31, 13 \ldots n \rangle$ is a letter in Σ_n. For example, $\Pi_5 = \{135, 153, 315, 351, 513, 531\}$ and $\Sigma_5 = \{\langle 135, 153 \rangle, \langle 153, 315 \rangle, \langle 315, 351 \rangle, \langle 351, 513 \rangle, \langle 513, 531 \rangle, \langle 531, 135 \rangle\}$. Note that each permutation in Π_n contributes to Σ_n one letter, thus $|\Sigma_n| = |\Pi_n| = \frac{n+1}{2}!$.
- $Q_n = \{1, \ldots, n\}$.
- Consider a permutation $\pi \in \Pi_n$. An *even-extension* of π is a permutation σ of $\{1, 2, 3, \ldots, n\}$ obtained from π by using π for the odd positions and inserting in each even position e the even number e. For example, if $\pi = 153$, then $\sigma = 12543$.

  Let π and π' be such that $\langle \pi, \pi' \rangle \in \Sigma_n$, and let $\sigma = i_1 \cdots i_n$ and $\sigma' = j_1 \cdots j_n$ be the even extensions of π and π'. Then, for every $1 \leq k \leq n$, we define

$$\delta_n(i_k, \langle \pi, \pi' \rangle) = \begin{cases} \{j_1, \ldots, j_k\} & \text{if } k \text{ is odd} \\ \{j_1, \ldots, j_{k-1}\} & \text{if } k \text{ is even.} \end{cases}$$

  That is, when a state $h \in Q_n$ reads $\langle \pi, \pi' \rangle$, it checks its location in σ (this is the k for which $h = i_k$) and sends copies to all states in smaller (or equal, if k is odd) locations in σ' (these are the states h' for which $h' = j_{k'}$ for k' smaller than (or equal to) k. Note that for all even k's, we have $\delta_n(i_k, \langle \pi, \pi' \rangle) = \delta_n(i_{k-1}, \langle \pi, \pi' \rangle)$.

  For example, $\delta_5(3, \langle 135, 153 \rangle) = \{1, 2, 5\}$. Indeed, the location of 3 in 12345 is 3 and the states located in the first three positions in 12543 are 1, 2, and 5. The other transitions on the letter $\langle 135, 153 \rangle$ are defined similarly:

[3] The proof stays valid with every ordering.

  - $\delta_5(1, \langle 135, 153\rangle) = \delta_5(2, \langle 135, 153\rangle) = \{1\}$,
  - $\delta_5(3, \langle 135, 153\rangle) = \delta_5(4, \langle 135, 153\rangle) = \{1, 2, 5\}$, and
  - $\delta_5(5, \langle 135, 153\rangle) = \{1, 2, 3, 4, 5\}$.
- $\alpha_n = \{i \mid i \text{ is even}\}$. Thus, every infinite run of $\mathcal{A}_n$ has to visit only finitely many even states.

Note that for every word $v \in \Sigma^\omega$, the run DAG of $\mathcal{A}_n$ on v has all the states in Q_n appearing in every level of the DAG. This follows from the set of initial states of $\mathcal{A}_n$ being Q_n and the fact that for every letter $a = \langle \pi, \pi' \rangle \in \Sigma_n$, there exists one state q in Q_n (q is last number in π) for which the transition from q on a contains all the states in Q_n.

Let u be the word in $(\Sigma_n)^{\frac{n+1}{2}!}$ that contains all the letters in Σ_n ordered lexicographically. For example, when $n = 5$, we have that $u = \langle 135, 153\rangle \langle 153, 315\rangle \langle 315, 351\rangle \langle 351, 513\rangle \langle 513, 531\rangle \langle 531, 135\rangle$. We prove that $\mathcal{A}_n$ accepts the word u^ω. It follows that $\mathcal{A}_n$ accepts vu^ω for every word $v \in \Sigma^*$.

Lemma 4. $u^\omega \in L(\mathcal{A}_n)$.

Proof. Consider the run DAG G of $\mathcal{A}_n$ on u^ω. In Figure 1, we describe the accepting run DAG of $\mathcal{A}_5$ on u^ω. As argued above, each level l of G consists of all the vertices in

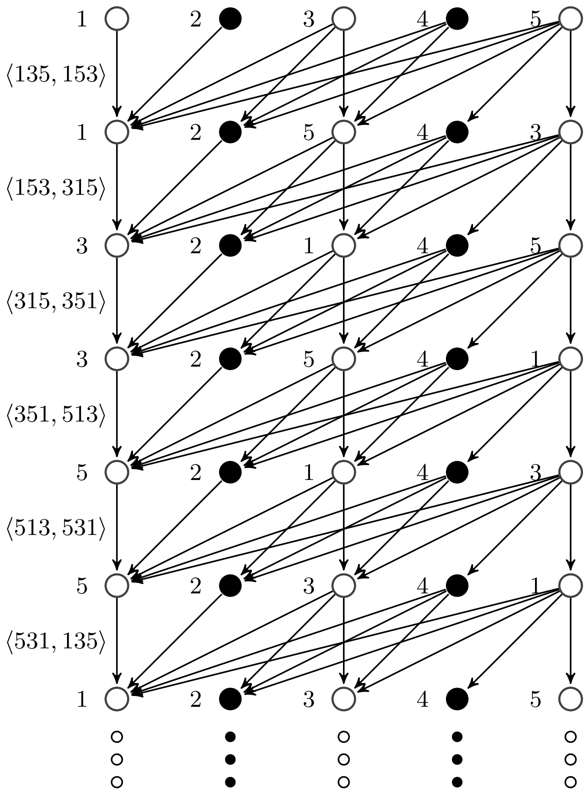


Fig. 1. The accepting run of $\mathcal{A}_5$ on u^ω

$Q_n \times \{l\}$. We arrange the vertices of G in columns numbered 1 to n. In the level that reads $\langle \pi, \pi' \rangle$, we arrange the vertices according to the position of the state component of each vertex in the even extension σ of π. For example, when we read $\langle 135, 153 \rangle$ in level 0, we consult the even extension 12345 of 135 and put the vertex $\langle 1, 0 \rangle$ in Column 1 (the leftmost), put $\langle 2, 0 \rangle$ in Column 2, and so on. Since u contains all the letters in Σ_n ordered lexicographically, the letter to be read in the next level is $\langle \pi', \pi'' \rangle$, and the vertices are arranged in columns in this level according to π'. By the definition of δ_n, the above implies that the edges in G go from columns to smaller or equal columns. Accordingly, all α-vertices appear in even columns and all other vertices appear in odd columns.

We prove that G has an odd ranking. For that, we prove, by induction on i, that the vertices in Column i, for $1 \leq i \leq n$, get rank i (independent of their level).

By definition, the set of successors of a vertex in Column 1 is a singleton containing the next vertex in Column 1. As all vertices in this column are not α-vertices, they are all α-free and they get rank 1. The set of successors of vertices in Column 2 is again a singleton containing only the next vertex in Column 1. Since vertices in Column 2 are α-vertices, they do not get rank 1. In the DAG G_2, however, these vertices have no successors. Thus, they are finite, and get rank 2.

The induction step is similar: the DAG G_i contains only vertices in Columns i to n. When i is odd, the vertices in Column i are α-free, and get rank i. When i is even, the vertices in Column i are finite, and get rank i too. □

Consider two letters $\langle \pi_1, \pi_1' \rangle$ and $\langle \pi_2, \pi_2' \rangle$ in Σ_n. We say that $\langle \pi_1, \pi_1' \rangle$ and $\langle \pi_2, \pi_2' \rangle$ are *gluable* if $\pi_1' = \pi_2$. Otherwise, $\langle \pi_1, \pi_1' \rangle$ and $\langle \pi_2, \pi_2' \rangle$ are *non-gluable*. We say that location $i \in \mathbb{N}$ is an *error* in w if letters i and $i+1$ in w are non-gluable. A word w is *bad* if w has infinitely many errors. The definition of non-gluable is extended to finite words in the obvious way. Consider a word $v \in \Sigma_n^*$. We denote by $\mathit{first}(v)$ the permutation $\pi \in \Pi_n$ such that the first letter of v is $\langle \pi, \pi' \rangle$, for the (lexicographic) successor π' of π. Similarly, we denote by $\mathit{last}(v)$ the permutation π' such that the last letter of v is $\langle \pi, \pi' \rangle$ for the predecessor π of π'. Given an even-extension $\sigma = i_1 \cdots i_n$ of a permutation, we say that the state i_k is the k-th state appearing in σ.

Consider a fragment of a run that starts in permutation π and ends in permutation π'. That is, the fragment reads the word v, the permutation π is $\mathit{first}(v)$, and the permutation π' is $\mathit{last}(v)$. We arrange the states in Q_n according to their order in the even extensions σ and σ' of π and π'. In the following lemma, we show that if q is the k-th state in σ, q' is the k'-th state in σ', and $k' \leq k$, then q' is reachable from q in this run fragment. Furthermore, if $k' < k$ then q' is reachable from q along a run that visits α.

Lemma 5. *Consider an infinite word $\sigma_0\sigma_1\cdots$ and a run* DAG *G of $\mathcal{A}_n$ on it. Let l be a level of G, let $l' > 0$ be an integer, and let $v = \sigma_l \cdots \sigma_{l+l'}$ be the subword of length l' read at the level l. Let k and k' be such that k is odd and $1 \leq k' \leq k \leq n$. Let q be the k-th state in the even extension of $\mathit{first}(v)$, and let q' be the k'-th state in the even extension of $\mathit{last}(v)$. Then, the vertex $\langle q', l+l' \rangle$ is reachable from the vertex $\langle q, l \rangle$ of G. Moreover, if $l' > 1$ and $k' < k$, then $\langle q', l+l' \rangle$ is reachable from $\langle q, l \rangle$ along a path that visits α.*

Proof. We start with the first part of the lemma and prove it by induction on l' (that is, the length of v). For $l' = 1$, the lemma follows from the definition of the transition function. For the induction step, consider a word $v = wa$. Let $\mathit{first}(w) = \pi_1$, $\mathit{last}(w) = \pi_2$ and

$a = \langle \pi_3, \pi_4 \rangle$. Let $i_1 \cdots i_n$, $j_1 \cdots j_n$, $c_1 \cdots c_n$, and $d_1 \cdots d_n$ be the even extensions of π_1, π_2, π_3, and π_4, respectively.

Consider the run DAG G of $\mathcal{A}_n$ on the input word. By the induction hypotheses, which holds for w, we know that for every odd k and for all $k' \leq k$, the vertex $\langle j_{k'}, l + |w| \rangle$ is reachable from the vertex $\langle i_k, l \rangle$. We consider now the edges of G reading the last letter a. We distinguish between two cases. If $\pi_2 = \pi_3$, the lemma follows from the definition of the transition function. If $\pi_2 \neq \pi_3$, consider the state c_k appearing in the k-th position in even extension of π_3. Let m be such that $j_m = c_k$. We again distinguish between two cases. If $m \leq k$, the lemma follows from the definition of the transition function. If $m > k$, then there exist $m' \leq k$ and $m'' > k$ such that $c_{m''} = j_{m'}$. By the induction hypothesis, $\langle j_{m'}, l + |w| \rangle$ is reachable from $\langle i_k, l \rangle$. As $j_{m'} = c_{m''}$, the transition of $c_{m''}$ reading $\langle \pi_3, \pi_4 \rangle$ implies that for every $k' < m''$ (and in particular for every $k' < k$) the vertex $\langle d_{k'}, l + |w| + 1 \rangle$ is reachable from $\langle i_k, l \rangle$.

We now prove the second part of the lemma. By the first part, the vertex $\langle j_{k-1}, l + l' - 1 \rangle$ is reachable from $\langle i_k, l \rangle$. As k is odd, $k - 1$ is even, thus, by the definition of an even-extension, $c_{k-1} = k - 1$, thus $\langle c_{k-1}, l + l' - 1 \rangle$ is an α-vertex. By the definition of the transition function, for every $k' < k - 1$, there is an edge from $\langle c_{k-1}, l + l' - 1 \rangle$ to $\langle d_{k'}, l + l' \rangle$. It follows that there is a path that visits α from $\langle i_k, l \rangle$ to $\langle d_{k'}, l + l' \rangle$. □

We use this result to show that bad words cannot be accepted by $\mathcal{A}_n$. Indeed, whenever there is a mismatch between the permutations, we find a state that reduces its position in the permutations. This state, gives rise to a fragment that visits α. If this happens infinitely often, we get a run that visits α infinitely often.

Lemma 6. *Every bad word u is rejected by $\mathcal{A}_n$.*

Proof. We start with the case that $u = vw^\omega$. Assume that $|w| > 1$. Otherwise, we replace w by $w \cdot w$. By the definition of bad words, the word w^ω contains two successive letters $\langle \pi_1, \pi_1' \rangle$ and $\langle \pi_2, \pi_2' \rangle$ such that $\pi_1' \neq \pi_2$. Let l be a level in the run DAG of $\mathcal{A}_n$ on vw^ω such that $l > |v|$ is such that $\langle \pi_1, \pi_1' \rangle$ is being read in level $l - 1$ and $\langle \pi_2, \pi_2' \rangle$ is being read in level l. Note that $\langle \pi_1, \pi_1' \rangle$ is then being read again at level $l + |w| - 1$.

We show that there exists a vertex $\langle q, l + |w| \rangle$ reachable from $\langle q, l \rangle$ such that the path from $\langle q, l \rangle$ to $\langle q, l + |w| \rangle$ visits an α-vertex. Since $\mathcal{A}_n$ is universal, the block of $|w|$ levels of G that starts in level l repeats forever, thus it follows that G has a path with infinitely many α-vertices.

Let w' be the word read between levels l and $l + |w|$. Note that w' is w shifted so that $\mathit{first}(w') = \pi_2$, and $\mathit{last}(w') = \pi_1'$. Let $\sigma = i_1, \ldots, i_n$ and $\sigma' = j_1, \ldots, j_n$ be the even-extensions of π_2 and π_1', respectively. Since $\pi_2 \neq \pi_1'$, there exists some odd k and k' such that $i_k = j_{k'}$ and $k' < k$. Let q be the state $i_k = j_{k'}$. The state q satisfies the conditions of Lemma 5 with respect to level l and length $l' = |w|$: it is the k-th state in $\mathit{first}(w')$ for an odd k, and it is also the k'-th state in $\mathit{last}(w')$. Hence, since $|w'| > 1$ and $k' < k$, we have that $\langle q, l + |w| \rangle$ is reachable from $\langle q, l \rangle$ along a path that visits α.

Consider some bad word $u \in \Sigma^\omega$ such that u does not have a cycle. It follows that u can be partitioned to infinitely many finite subwords that are non-gluable. Consider two such subwords w_1 and w_2. As w_1 and w_2 are non-gluable there exists some k and k' such that $k' < k$ and the k-th state in $\mathit{last}(w_1)$ is the k'-th state in $\mathit{first}(w)$. There are infinitely many subwords, we use Ramsey's Theorem to find infinitely many points that have the same k and k'. This defines a new partition to finite subwords. By using Lemma 5 we can show that the run on w contains a path with infinitely many visits to α. □

Corollary 1. *The language of $\mathcal{A}_n$ is $\{vu^\omega \mid v \in \Sigma_n^*\}$.*

In Figure 2, we describe a rejecting run of $\mathcal{A}_n$ on v^ω where v is obtained from u by switching the order of the letters $\langle 315, 351\rangle$ and $\langle 351, 513\rangle$. The pair $\langle 153, 315\rangle$ and $\langle 351, 513\rangle$ is non-gluable. In the run DAG G, the state 1 satisfies the conditions of Lemma 5 with $l = 2$ and $l' = 6$. To see this, note that the subword of v^ω of length 6 that is read at level 2 is $w = \langle 351, 513\rangle\langle 315, 351\rangle \langle 513, 531\rangle \langle 531, 135\rangle \langle 135, 153\rangle\langle 153, 315\rangle$, with $\mathit{first}(w) = 351$ and $\mathit{last}(w) = 315$. The state 1 is the 5-th state in the even extension 32541 of $\mathit{first}(w)$, thus $k = 5$, and is the 3-rd state in the even extension 32145 of $\mathit{last}(w)$, thus $k' = 3$. As promised in the lemma, the vertex $\langle 1, 8\rangle$ is reachable from the vertex $\langle 1, 2\rangle$ via a path that visits the α-vertex $\langle 2, 3\rangle$ — the rejecting path that is highlighted in bold in the figure.

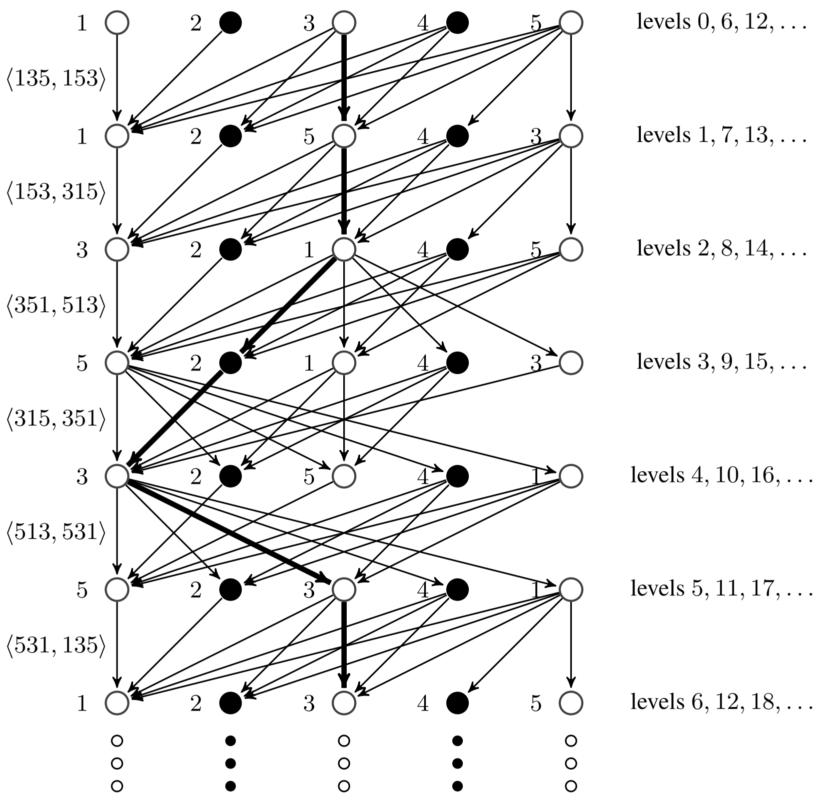

1
2
3
4
5
levels $0, 6, 12, \ldots$
$\langle 135, 153\rangle$
1
2
5
4
3
levels $1, 7, 13, \ldots$
$\langle 153, 315\rangle$
3
2
1
4
5
levels $2, 8, 14, \ldots$
$\langle 351, 513\rangle$
5
2
1
4
3
levels $3, 9, 15, \ldots$
$\langle 315, 351\rangle$
3
2
5
4
1
levels $4, 10, 16, \ldots$
$\langle 513, 531\rangle$
5
2
3
4
1
levels $5, 11, 17, \ldots$
$\langle 531, 135\rangle$
1
2
3
4
5
levels $6, 12, 18, \ldots$

Fig. 2. The rejecting run of $\mathcal{A}_5$ on $(\langle 135, 153\rangle\langle 153, 315\rangle\langle 351, 513\rangle\langle 315, 351\rangle\langle 513, 531\rangle \langle 531, 135\rangle)^\omega$

We can now conclude with the statement of the lower bound for the linear case.

Theorem 2. *There is a $\frac{n+1}{2}!$ lower bound on the length of a witness accepted by a UCW with n states.*

Proof. Consider the sequence of UCWs $\mathcal{A}_1, \mathcal{A}_3, \ldots$ defined above. By the above, the language of $\mathcal{A}_n$ is $\{vu^\omega \mid v \in \Sigma_n^*\}$, where u is the word in $(\Sigma_n)^{\frac{n+1}{2}!}$ that contains all the letters in Σ_n ordered lexicographically. Thus, the length of witnesses is at least $\frac{n+1}{2}!$. □

4 Universal Co-Büchi Tree Automata

Given an alphabet Σ and a set D of directions, a *Σ-labeled D-tree* is a pair $\langle T, \tau\rangle$, where $T \subseteq D^*$ is a tree over D and $\tau : T \to \Sigma$ maps each node of T to a letter in Σ. A *transducer* is a labeled finite graph with a designated start node, where the edges are labeled by D and the nodes are labeled by Σ. A Σ-labeled D-tree is *regular* if it is the unwinding of some transducer. More formally, a transducer is a tuple $\mathcal{T} = \langle D, \Sigma, S, s_{in}, \eta, L\rangle$, where D is a finite set of directions, Σ is a finite alphabet, S is a finite set of states, $s_{in} \in S$ is an initial state, $\eta : S \times D \to S$ is a deterministic transition function, and $L : S \to \Sigma$ is a labeling function. We define $\eta : D^* \to S$ in the standard way: $\eta(\varepsilon) = s_{in}$, and for $x \in D^*$ and $d \in D$, we have $\eta(x \cdot d) = \eta(\eta(x), d)$. Intuitively, a Σ-labeled D-tree $\langle D^*, \tau\rangle$ is regular if there exists a transducer $\mathcal{T} = \langle D, \Sigma, S, s_{in}, \eta, L\rangle$ such that for every $x \in D^*$, we have $\tau(x) = L(\eta(x))$. We denote by $\mathcal{T}_s$ the transducer $\langle D, \Sigma, S, s, \eta, L\rangle$, i.e., the transducer $\mathcal{T}$ with s as initial state. Given a transducer $\mathcal{T}$, let $reach_0(\mathcal{T}) = \{s_{in}\}$ and let $reach_{i+1}(\mathcal{T}) = \bigcup_{s \in reach_i(\mathcal{T})} \bigcup_{d \in D} \{\eta(s, d)\}$. The *width* of $\mathcal{T}$ is the minimal j such that $|reach_i(\mathcal{T})| = j$ for infinitely many i. That is, starting from some i_0, we have that $|reach_i(\mathcal{T})| \geq j$ for all $i \geq i_0$. Note that while the width of an infinite tree generated by a transducer is unbounded, the width of a transducer is always bounded by its number of states.

A universal co-Büchi tree automaton (UCT, for short) is a tuple $\mathcal{A} = \langle \Sigma, D, Q, Q_{in}, \delta, \alpha\rangle$, where Σ, Q, Q_{in}, and α are as in UCW, D is a set of directions, and $\delta : Q \times \Sigma \to 2^{(D \times Q)}$ is a transition function. When the language of $\mathcal{A}$ is not empty, it accepts a regular Σ-labeled D-tree [16,13]. It is convenient to consider runs of $\mathcal{A}$ on transducers.

Consider a transducer $\mathcal{T} = \langle D, \Sigma, S, s_{in}, \eta, L\rangle$. A run of $\mathcal{A}$ on $\mathcal{T}$ can be arranged in an infinite DAG $G = \langle V, E\rangle$, where

- $V \subseteq S \times Q \times \mathbb{N}$.
- $E \subseteq \bigcup_{l \geq 0} (S \times Q \times \{l\}) \times (S \times Q \times \{l+1\})$ is such that $E(\langle s, q, l\rangle, \langle s', q', l+1\rangle)$ iff there is $d \in D$ such that $(d, q') \in \delta(q, L(s))$ and $\eta(s, d) = s'$.

The run DAG G is accepting iff every path in it has only finitely many vertices in $S \times \alpha \times \mathbb{N}$. A transducer is accepted by $\mathcal{A}$ if its run DAG is accepting. In the sequel we restrict attention to binary trees, i.e., $D = \{0, 1\}$ and $T = \{0, 1\}^*$. All our ideas apply to larger branching degrees as well.

5 Lower Bound on Width of Accepted Transducers

In [8], it is shown that if a UCT with n states is not empty, then it accepts a transducer of width bounded by $(2n!)n^{2n}3^n(n+1)/n!$. An improved upper bound for determinization shows that the width reduces to $2n(n!)^2$ [14,10]. It is conjectured in [8] that this bound can be tightened to $2^{O(n)}$. Intuitively, it is conjectured there that if a UCT is not empty, then different states of a transducer it accepts that are visited by the same set of states of the UCT can be merged.

In this section we construct, for every odd $n \geq 1$, a UCT $\mathcal{B}_n$ with n states such that the language of $\mathcal{B}_n$ is not empty and yet the width of a transducer accepted by $\mathcal{B}_n$ is at least $\frac{n+1}{2}!$.

We extend the ideas in Section 3 to a tree automaton. The basic idea is to create a mismatch between the permutation the automaton has to send to the left successor of a node and the permutation the automaton has to send to the right successor. Doing so, we force the input tree to display all possible permutations in one level. Thus, the minimal width of a transducer generating such a tree is $\frac{n+1}{2}!$.

Recall the alphabet Σ_n defined in Section 3. We reuse this alphabet in the context of a tree. Whenever we refer to a letter $\langle \pi, \pi' \rangle \in \Sigma_n$ we assume that π' is the successor of π according to the lexicographic order. Consider a letter $\langle \pi, \pi' \rangle \in \Sigma_n$ and a node x labeled by $\langle \pi, \pi' \rangle$. Intuitively, when the automaton $\mathcal{B}_n$ reads the node x, it "sends" the permutation π' to the left successor of x and it "sends" the permutation π (i.e., the same permutation) to the right successor of x. Consider a Σ_n-labeled binary tree $\langle T, \tau \rangle$. Consider a node x and its two successors $x \cdot 0$ and $x \cdot 1$. Let $\tau(x)$ be $\langle \pi_x, \pi'_x \rangle$, $\tau(x \cdot 0)$ be $\langle \pi_{x0}, \pi'_{x0} \rangle$, and $\tau(x \cdot 1)$ be $\langle \pi_{x1}, \pi'_{x1} \rangle$. We say that the node x is *good* if $\pi_{x0} = \pi'_x$ and $\pi_{x1} = \pi_x$. That is, the left successor of x is labeled by the successor permutation π'_x (paired with its successor permutation) and the right successor of x is labeled by the same permutation π_x (paired with its successor permutation). A tree $\langle T, \tau \rangle$ is *good* if all vertices $x \in T$ are good. Given a permutation π there is a unique good tree whose root is labeled by $\langle \pi, \pi' \rangle$. We denote this tree by $\langle T, \tau_\pi \rangle$.

Lemma 7. *For every permutation π, the width of a transducer that generates $\langle T, \tau_\pi \rangle$ is $\frac{n+1}{2}!$.*

Proof. We can construct a transducer generating $\langle T, \tau_\pi \rangle$ with $\frac{n+1}{2}!$ states. Indeed, the states of such a transducer are the letters of Σ_n. The 0-successor of a state $\langle \pi, \pi' \rangle$ is the unique state $\langle \pi', \pi'' \rangle$, for the successor π'' of π', and its 1-successor is $\langle \pi, \pi' \rangle$.

Let $\pi_0, \ldots, \pi_{\frac{n+1}{2}!}$ be an enumeration of all permutations according to the lexicographic order. For simplicity we assume that $\pi = \pi_0$. We can see that $\langle \pi_0, \pi_1 \rangle$ appears in every level in $\langle T, \tau_\pi \rangle$. By induction, $\langle \pi_i, \pi_{i+1} \rangle$ appears for the first time in $\langle T, \tau_\pi \rangle$ in level $i-1$. It follows that $\langle \pi_i, \pi_{i+1} \rangle$ appears in $\langle T, \tau_\pi \rangle$ in all levels above $i-1$. In particular, in all levels after $\frac{n+1}{2}!$, all permutations appear. It follows that $|reach_j(\mathcal{T})| \geq \frac{n+1}{2}!$ for all transducers $\mathcal{T}$ that generate $\langle T, \tau_\pi \rangle$ and $j \geq \frac{n+1}{2}!$. □

Corollary 2. *Every transducer $\mathcal{T}$ that generates a tree that has a subtree $\langle T, \tau_\pi \rangle$, for some permutation π, has width at least $\frac{n+1}{2}!$.*

We now define $\mathcal{B}_n$ as a UCT variant of the UCW $\mathcal{A}_n$ constructed in Section 3. Essentially, every transducer accepted by $\mathcal{B}_n$ generates a tree that contains $\langle T, \tau_\pi \rangle$ as a subtree, for some permutation π of all the letters in Σ_n.

Let $\mathcal{B}_n = \langle \Sigma_n, \{0,1\}, Q_n, \delta_n, Q_n, \alpha_n \rangle$, where $Q_n = \{1, \ldots, n\}$, $\alpha_n = \{i \mid i \text{ is even}\}$, and $\delta : Q_n \times \Sigma_n \to 2^{\{0,1\} \times Q_n}$ is as follows. Let $\langle \pi, \pi' \rangle \in \Sigma_n$ and let $\sigma = i_1 \cdots i_n$ and $\sigma' = j_1 \cdots j_n$ be the even extensions of π and π'. Then, for every $1 \leq k \leq n$, we define

$$\delta_n(i_k, \langle \pi, \pi' \rangle) = \begin{cases} \{(0, j_1), \ldots, (0, j_k), (1, i_1), \ldots, (1, i_k)\} & \text{if } k \text{ is odd} \\ \{(0, j_1), \ldots, (0, j_{k-1}), (1, i_1), \ldots, (1, i_{k-1})\} & \text{if } k \text{ is even} \end{cases}$$

When going left, $\mathcal{B}_n$ treats the pair $\langle \pi, \pi' \rangle$ like the UCW $\mathcal{A}_n$ treats it. When going right, $\mathcal{B}_n$ mimics the same concept, this time, without changing the permutation. From state $q \in Q_n$, our automaton checks the location of q in σ and sends copies to all states in smaller (or equal, if k is odd) locations in σ' in direction 0 and all states in smaller (or equal) locations in σ in direction 1.

Consider a transducer $\mathcal{T} = \langle D, \Sigma_n, S, s_{in}, \eta, L \rangle$ accepted by $\mathcal{B}_n$. Given a permutation π, we say that π' is the 0-successor of π for the successor π' of π according to the lexicographic order (i.e., the unique π' such that $\langle \pi, \pi' \rangle \in \Sigma_n$) and we say that π is the 1-successor of π. Consider a path $p = s_0, a_0, s_1, a_1, \ldots \in (S \times D)^\omega$, where $s_{i+1} = \eta(s_i, a_i)$. We say that p is *good* if for all $i \geq 0$ we have $L(s_{i+1})$ is the a_i-successor of $L(s_i)$. We say that p is *bad* otherwise[4]. If p is bad, every location $i \in \mathbb{N}$ such that $L(s_i)$ is not the a_{i-1}-successor of $L(s_{i-1})$ is called an *error* in p.

Consider a transducer $\mathcal{T} = \langle D, \Sigma_n, S, s_{in}, \eta, L \rangle$ and an infinite path $p = s_0, a_0, s_1, a_1, \ldots \in (S \times D)^\omega$, where $s_{i+1} = \eta(s_i, a_i)$. Consider a sub-path $v = s_l, a_l, \ldots, s_{l'-1}, a_{l'-1}, s_{l'}$. We denote by *first*$(v)$ the permutation $\pi \in \Pi_n$ such that $\langle \pi, \pi' \rangle = L(s_l)$. We denote by *last*$(v)$ the permutation $\pi'' \in \Pi_n$ such that $L(s_{l'-1}) = \langle \pi, \pi' \rangle$ and $\pi'' = \pi$ if $a_{l'-1} = 1$ and $\pi'' = \pi'$ if $a_{l'-1} = 0$. That is, the last permutation read in v is determined by the last direction p takes in v.

Let G be the DAG run of $\mathcal{B}_n$ on $\mathcal{T}$, $p = s_0, a_0, s_1, a_1, \ldots$ an infinite path in T, and $v = s_l, a_l, \ldots, s_{l'-1}, a_{l'-1}, s_{l'}$ a sub-path of p. Consider the part of G consisting of all nodes in levels l to l' that read the states $s_l, \ldots, s_{l'}$. Let π be *first*(v) and π' be *last*(v). We arrange the states in Q according to their order in the even extensions σ and σ' of π and π'. The following lemma is the tree variant of Lemma 5. It shows that if q is the k-th state in σ and q' is the k'-th state in σ', then $k' \leq k$ implies that q' is reachable from q in this part of the run. Furthermore, if $k' < k$ then q' is reachable from q along a run that visits α. The proof is identical to that of Lemma 5.

Lemma 8. *Consider a transducer $\mathcal{T} = \langle D, \Sigma_n, S, s_{in}, \eta, L \rangle$ and the* DAG *run G of $\mathcal{B}_n$ on it. Let $p = s_0, a_0, s_1, a_1, \ldots$ be a path in T and let $v = s_l, a_l \ldots, s_{l'-1}, a_{l'-1}, s_{l'}$ be a sub-path of p. Let q be the k-th state in the even extension of first(v) for an odd k, and let q' be the k'-th state in the even extension of last(v), for $k' \leq k$. Then, the vertex $\langle s_{l'}, q', l' \rangle$ in G is reachable from the vertex $\langle s_l, q, l \rangle$. Moreover, if $l' - l > 1$ and $k' < k$, then a path connecting $\langle s_l, q, l \rangle$ to $\langle s_{l'}, q', l' \rangle$ visits α.*

The following Lemma resembles Lemma 6. It shows that in a transducer accepted by $\mathcal{B}_n$, every path has only finitely many errors.

Lemma 9. *For every path p in a transducer $\mathcal{T} \in L(\mathcal{B}_n)$, the path p contains finitely many errors.*

Proof. Let G be an accepting run of $\mathcal{B}_n$ on $\mathcal{T} = \langle D, \Sigma_n, S, s_{in}, \eta, L \rangle$. Assume that $p = s_0, a_0, s_1, a_1, \ldots$, where $s_{i+1} = \eta(s_i, a_i)$, is a path in $\mathcal{T}$ with infinitely many errors. Let $s_{l_0}, s_{l_1}, \ldots$ denote the error locations in p. By definition, for every $m \geq 0$ we have

[4] Notice that the definition of bad here is slightly different from the definition of bad in Section 3.

$L(s_{l_m})$ is not the a_{l_m-1}-successor of $L(s_{l_m-1})$. With every index l_m we associate a triplet $\langle \pi_m, \pi'_m, d_m\rangle$ such that $L(s_{l_m-1}) = \langle \pi, \pi'\rangle$ and π_m is the a_{l_m-1}-successor of π (i.e., π' if $a_{l_m-1} = 0$ and π otherwise), $L(s_{l_m}) = \langle \pi'_m, \pi'''\rangle$, and $d_m = a_{l_m-1}$. That is, we record the permutation π'_m labeling s_{l_m}, the unmatching π_m, which is the a_{l_m-1}-successor of the label of s_{l_m-1}, and the direction that takes from s_{l_m-1} to s_{l_m}. There are infinitely many errors and finitely many triplets. There is a triplet $\langle \pi, \pi', d\rangle$ associated with infinitely many indices. We abuse notations and denote by $s_{l_0}, s_{l_1}, \ldots$ the sub-sequence of locations associated with $\langle \pi, \pi', d\rangle$. Without loss of generality, assume that for all $m \geq 0$ we have $l_{m+1} - l_m > 1$.

For $m, m' \geq 0$ such that $m \neq m'$, let $v_{m,m'}$ denote the sub-path of p that starts in s_{l_m} and ends in $s_{l_{m'}}$. Then $\pi' = \mathit{first}(v_{m,m'})$ and $\pi = \mathit{last}(v_{m,m'})$. By assumption π' is not the d-successor of π. Let $\sigma = i_1, \ldots, i_n$ be the even extension of the d-successor of π and let $\sigma' = j_1, \ldots, j_n$ be the even extension of π'. Then there exists some odd k and k' such that $j_k = i_{k'}$ and $k' < k$. Let q be the state $j_k = i_{k'}$. The state q satisfies the condition of Lemma 8 with respect to $v_{m,m'}$: it is the k-th state in $\mathit{first}(v_{m,m'})$ for an odd k, and it is also the k'-th state in $\mathit{last}(v_{m,m'})$. Hence, since $l_{m'} - l_m > 1$ and $k' < k$, the node $\langle s_{l_{m'}}, q, l_{m'}\rangle$ in G is reachable from the node $\langle s_{l_m}, q, l_m\rangle$ along a path that visits α.

For every two different integers m and m' we identify one such state $q_{m,m'}$. By Ramsey's Theorem, there exist a state q and a sequence $l'_0, l'_1, \ldots$ such that for every $m \geq 0$ the sub-path $v_{l'_m, l'_{m+1}}$ connects state q to itself with a path that visits α. We have found a path in G that visits α infinitely often. □

We now show that every tree generated by $\mathcal{T}$ contains $\langle T, \tau_\pi\rangle$ for some π as a subtree.

Lemma 10. *For every $\mathcal{T} \in L(\mathcal{B}_n)$, there exists a permutation π and a state s reachable from s_{in} such that the transducer $\mathcal{T}_s$ generates $\langle T, \tau_\pi\rangle$.*

Proof. We add an annotation to the edges in $\mathcal{T}$. Every edge $s' = \eta(s, a)$ such that s' is an error in a path that contains s and s' is annotated by 1. Every other edge is annotated by 0. According to Lemma 9, every path in $\mathcal{T}$ is annotated by finitely many 1's.

We say that a state s is 1-free in $\mathcal{T}$ iff all the edges in $\mathcal{T}$ that are reachable from s are not labeled by 1. It is enough to find one such state s. Assume by contradiction that no such state s exists. We construct by induction a path that is labeled by infinitely many 1's.[5]

By assumption, s_{in} is not 1-free. Hence there is some state s_1 reachable from s_{in} and a direction a_1 such that the edge from s_1 to $\eta(s_1, a_1)$ is annotated by 1. By induction the path from s_{in} to $\eta(s_i, a_i)$ has at least i edges annotated by 1. By assumption $\eta(s_i, a_i)$ is not 1-free. There exists a node s_{i+1} reachable from $\eta(s_i, a_i)$ and a direction a_{i+1} such that the edge from s_{i+1} to $\eta(s_{i+1}, a_{i+1})$ is annotated by 1. It follows that the path from s_{in} to $\eta(s_{i+1}, a_{i+1})$ has at least $i+1$ edges annotated by 1. In the limit, we get a path in $\mathcal{T}$ that has infinitely many edges labeled 1. In contradiction to Lemma 9.

It follows that there exists a state s in $\mathcal{T}$ such that s is 1-safe. As s is 1-safe, the subtree generated by $\mathcal{T}_s$ contains no errors. Let π be the permutation such that $L(s) = \langle \pi, \pi'\rangle$. Then $\mathcal{T}_s$ generates $\langle T, \tau_\pi\rangle$. □

[5] Notice the resemblance to the definition of α-free in Section 2. Indeed, the proof of the existence of a 1-free state follows closely the similar proof in [7].

We can now conclude with the statement of the lower bound for the branching case.

Theorem 3. *There is a* $\frac{n+1}{2}!$ *lower bound on the width of a transducer accepted by a UCT with* n *states.*

Proof. Consider the sequence of UCTs $\mathcal{B}_1, \mathcal{B}_3, \ldots$ defined above. For every permutation π, the transducer that generates $\langle T, \tau_\pi \rangle$ is accepted by $\mathcal{B}_n$. By Lemma 10 and Corollary 2, every transducer accepted by $\mathcal{B}_n$ is of width at least $\frac{n+1}{2}!$. □

References

1. Brzozowski, J.A., Leiss, E.: Finite automata and sequential networks. TCS 10, 19–35 (1980)
2. Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Proc. Int. Congress on Logic, Method, and Philosophy of Science. 1960, pp. 1–12. Stanford Univ. Press (1962)
3. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. J. ACM 28(1), 114–133 (1981)
4. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 2nd edn. Addison-Wesley, Reading (2000)
5. Kupferman, O., Piterman, N., Vardi, M.Y.: Safraless compositional synthesis. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 31–44. Springer, Heidelberg (2006)
6. Kupferman, O., Sheinvald-Faragy, S.: Finding shortest witnesses to the nonemptiness of automata on infinite words. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 492–508. Springer, Heidelberg (2006)
7. Kupferman, O., Vardi, M.Y.: Weak alternating automata are not that weak. ACM ToCL 2(2), 408–429 (2001)
8. Kupferman, O., Vardi, M.Y.: Safraless decision procedures. In: Proc. 46th FOCS, pp. 531–540 (2005)
9. Kurshan, R.P.: Computer Aided Verification of Coordinating Processes. Princeton Univ. Press, Princeton (1994)
10. Liu, W.: A tighter analysis of Piterman determinization construction (2007), http://nlp.nudt.edu.cn/~lww/pubs.htm
11. McNaughton, R.: Testing and generating infinite sequences by a finite automaton. I&C 9, 521–530 (1966)
12. Miyano, S., Hayashi, T.: Alternating finite automata on ω-words. TCS 32, 321–330 (1984)
13. Muller, D.E., Schupp, P.E.: Simulating alternating tree automata by nondeterministic automata: New results and new proofs of theorems of Rabin, McNaughton and Safra. TCS 141, 69–107 (1995)
14. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. LMCS 3(3), 5 (2007)
15. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Proc. 16th POPL, pp. 179–190 (1989)
16. Rabin, M.O.: Decidability of second order theories and automata on infinite trees. Transaction of the AMS 141, 1–35 (1969)
17. Thomas, W.: Automata on infinite objects. In: Handbook of Theoretical Computer Science, pp. 133–191 (1990)
18. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification. In: Proc. 1st LICS, pp. 332–344 (1986)
19. Vardi, M.Y., Wolper, P.: Reasoning about infinite computations. I&C 115(1), 1–37 (1994)

Interrupt Timed Automata

Beatrice Bérard[1,⋆] and Serge Haddad[2,⋆]

[1] Université Pierre & Marie Curie, LIP6/MoVe, CNRS UMR 7606, Paris, France
Beatrice.Berard@lip6.fr
[2] Ecole Normale Supérieure de Cachan, LSV, CNRS UMR 8643, Cachan, France
Serge.Haddad@lsv.ens-cachan.fr

Abstract. In this work, we introduce the class of Interrupt Timed Automata (ITA), which are well suited to the description of multi-task systems with interruptions in a single processor environment. This model is a subclass of hybrid automata. While reachability is undecidable for hybrid automata we show that in ITA the reachability problem is in 2-EXPSPACE and in PSPACE when the number of clocks is fixed, with a procedure based on a generalized class graph. Furthermore we consider a subclass ITA$_-$ which still describes usual interrupt systems and for which the reachability problem is in NEXPTIME and in NP when the number of clocks is fixed (without any class graph). There exist languages accepted by an ITA$_-$ but neither by timed automata nor by controlled real-time automata (CRTA), another extension of timed automata. However we conjecture that CRTA is not contained in ITA. So, we combine ITA with CRTA in a model which encompasses both classes and show that the reachability problem is still decidable.

Keywords: Hybrid automata, timed automata, multi-task systems, interruptions, decidability of reachability.

1 Introduction

Context. The model of timed automata (TA), introduced in [1], has proved very successful due to the decidability of the emptiness test. A timed automaton consists of a finite automaton equipped with real valued variables, called clocks, which evolve synchronously with time, during the sojourn the states. When a discrete transition occurs, clocks can be tested by guards, which compare their values with constants, and reset. The decidability result was obtained through the construction of a finite partition of the state space into regions, leading to a finite graph which is time-abstract bisimilar to the original transition system, thus preserving reachability.

Hybrid automata have subsequently been proposed as an extension of timed automata [14], with the aim to increase the expressive power of the model. In this model, clocks are replaced by variables which evolve according to a differential

⋆ Work partly supported by project DOTS (ANR-06-SETI-003).

equation. Furthermore, guards consist of more general constraints on the variables and resets are extended into (possibly non deterministic) updates. However, since reachability is undecidable for this model, many classes have been defined, between timed and hybrid automata, to obtain the decidability of this problem. Examples of such classes are multi-rate or rectangular automata [2], some systems with piece-wise constant derivatives [3], controlled real-time automata [9], integration graphs [11], o-minimal hybrid systems [12,13], some updatable timed automata [6] or polygonal hybrid systems [4].

Contribution. In this paper, we define a subclass of hybrid automata, called Interrupt Timed Automata (ITA), well suited to the description of multi-task systems with interruptions in a single processor environement. In an ITA, the finite set of control states is organized according to *interrupt levels*, ranging from 1 to n, with exactly one active clock for a given level. The clocks from lower levels are suspended and those from higher levels are not yet defined. On the transitions, guards are linear constraints using only clocks from the current level or the levels below and the relevant clocks can be updated by linear expressions, using clocks from lower levels. For a transition increasing the level, the newly relevant clocks are reset. This model is rather expressive since it combines variables with rate 1 or 0 (usually called stopwatches) and linear expressions for guards or updates.

While the reachability problem is well known to be undecidable for automata with stopwatches [10,8,7], we prove that for ITA, it belongs to 2-EXPSPACE. The procedure significantly extends the classical region construction of [1] by associating with each state a family of orderings over linear expressions. Furthermore, we define a slight restriction of the model, leading to a subclass ITA_- for which reachability can be decided in NEXPTIME. Furthermore when the number of clocks is fixed, the complexity is greatly reduced for both classes: PSPACE (resp. NP) for ITA (resp. ITA_-).

We also investigate the expressive power of the class ITA, in comparison with the original model of timed automata and also with the more general controlled real-time automata (CRTA) proposed in [9]. In CRTA, clocks are also organized into a partition (according to colours) and may have different rates, but all active clocks in a given state have identical rate. We prove that there exist timed languages accepted by ITA (and also ITA_-) but not by a CRTA (resp. not by a TA). We conjecture that the classes ITA and CRTA are incomparable, which leads us to define a combination of the two models, the CRTA part describing a basic task at an implicit additional level 0. For this extended model denoted by ITA^+ (with ITA^+_- as a subclass), we show that reachability is still decidable with the same complexity.

Outline. In section 2, we define ITA and study its expressive power. Section 3 is devoted to the decidability of the reachability problem and section 4 extends the results for the models combining ITA and CRTA.

2 Interrupt Timed Automata

2.1 Definitions and Examples

The sets of natural numbers, rational numbers and real numbers are denoted respectively by $\mathbb{N}$, $\mathbb{Q}$ and $\mathbb{R}$, with $\mathbb{Q}_{\geq 0}$ (resp. $\mathbb{R}_{\geq 0}$) for the set of non negative rational (resp. real) numbers.

Let X be a set of clocks. A linear expression over X is a term of the form $\sum_{x \in X} a_x x + b$ where b and the a_xs are in $\mathbb{Q}$. We denote by $\mathcal{C}^+(X)$ the set of constraints obtained by conjunctions of atomic propositions of the form $C \bowtie 0$, where C is a linear expression and $\bowtie$ is in $\{<, \leq, \geq, >\}$. The subset of $\mathcal{C}^+(X)$ where linear expressions are restricted to the form $x + b$, for $x \in X$ and $b \in \mathbb{Q}$ is denoted by $\mathcal{C}(X)$. An update over X is a conjunction of the form $\bigwedge_{x \in X} x := C_x$ where C_x is a linear expression. We denote by $\mathcal{U}^+(X)$ the set of updates over X and by $\mathcal{U}(X)$ the subset of $\mathcal{U}^+(X)$ where for each clock x, the linear expression C_x is either x (value unchanged) or 0 (clock reset).

A clock valuation is a mapping $v : X \mapsto \mathbb{R}$ and we denote by $\mathbf{0}$ the valuation assigning the value 0 to all clocks. The set of all clock valuations is $\mathbb{R}^X$ and we write $v \models \varphi$ when valuation v satisfies the clock constraint φ. For an element d of $\mathbb{R}_{\geq 0}$, the valuation $v+d$ is defined by $(v+d)(x) = v(x)+d$, for each clock x in X. For a linear expression $C = \sum_{x \in X} a_x x + b$, the real number $v[C]$ is defined by $\sum_{x \in X} a_x v(x) + b$. For an update u defined by $\bigwedge_{x \in X} x := C_x$, the valuation $v[u]$ is defined by $v[u](x) = v[C_x]$ for x in X. The linear expression $C[u]$ is obtained by substituting in C every x by C_x.

The model of ITA is based on the principle of multi-task systems with interruptions, in a single processor environment. We consider a set of tasks with different priority levels, where a higher level task represents an interruption for a lower level task. At a given level, exactly one clock is active with rate 1, while the clocks for tasks of lower levels are suspended, and the clocks for tasks of higher levels are not yet activated.

Definition 1 (Interrupt Timed Automaton). *An* interrupt timed automaton *is a tuple $\mathcal{A} = (\Sigma, Q, q_0, F, X, \lambda, \Delta)$, where Σ is a finite alphabet, Q is a finite set of states, q_0 is the initial state, $F \subseteq Q$ is the set of final states, $X = \{x_1, \ldots, x_n\}$ consists of n interrupt clocks, the mapping $\lambda : Q \mapsto \{1, \ldots, n\}$ associates with each state its level and $\Delta \subseteq Q \times [\mathcal{C}^+(X) \times (\Sigma \cup \{\varepsilon\}) \times \mathcal{U}^+(X)] \times Q$ is the set of transitions.*

We call $x_{\lambda(q)}$ the active clock in state q. Let $q \xrightarrow{\varphi, a, u} q'$ in Δ be a transition with $k = \lambda(q)$ and $k' = \lambda(q')$. The guard φ contains only clocks from levels less than or equal to k: it is a conjunction of constraints of the form $\sum_{j=1}^{k} a_j x_j + b \bowtie 0$. The update u is of the form $\wedge_{i=1}^{n} x_i := C_i$ with:

- *if $k' < k$, i.e. the transition decreases the level, then C_i is of the form $\sum_{j=1}^{i-1} a_j x_j + b$ or $C_i = x_i$ for $1 \leq i \leq k'$ and $C_i = x_i$ otherwise;*
- *if $k' \geq k$ then C_i is of the form $\sum_{j=1}^{i-1} a_j x_j + b$ or $C_i = x_i$ for $1 \leq i \leq k$, $C_i = 0$ if $k < i \leq k'$ and $C_i = x_i$ if $i > k'$.*

Thus, clocks from levels higher than the target state are ignored, and when newly relevant clocks appear upon increasing the level, they are reset.

Definition 2 (Semantics of an ITA). *The semantics of an ITA $\mathcal{A}$ is defined by the transition system $\mathcal{T}_{\mathcal{A}} = (S, s_0, \rightarrow)$. The set S of configurations is $\{(q, v) \mid q \in Q,\ v \in \mathbb{R}^X\}$, with initial configuration $(q_0, \mathbf{0})$. An accepting configuration of $\mathcal{T}_{\mathcal{A}}$ is a pair (q, v) with q in F. The relation $\rightarrow$ on S consists of two types of steps:*

Time steps: *Only the active clock in a state can evolve, all other clocks are suspended. For a state q with active clock $x_{\lambda(q)}$, a time step of duration d is defined by $(q, v) \xrightarrow{d} (q, v')$ with $v'(x_{\lambda(q)}) = v(x_{\lambda(q)}) + d$ and $v'(x) = v(x)$ for any other clock x.*

Discrete steps: *A discrete step $(q, v) \xrightarrow{a} (q', v')$ occurs if there exists a transition $q \xrightarrow{\varphi, a, u} q'$ in Δ such that $v \models \varphi$ and $v' = v[u]$.*

Remarks. Observe that in state q the only relevant clocks are $\{x_k\}_{k \le \lambda(q)}$ since any other clock will be reset before being tested for the first time in the future. We have not stated this feature more explicitely in the definition for the sake of simplicity.

Concerning updates, if we allow a slight generalization, substituting $x_i := \sum_{j=1}^{i-1} a_j x_j + b$ by $x_i := \sum_{j=1}^{i} a_j x_j + b$, it is easy to simulate a two-counter machine with a three clocks-ITA, thus implying undecidability of reachability for the model.

A timed word is a finite sequence $(a_1, t_1) \dots (a_p, t_p) \in (\Sigma \times \mathbb{R}_{\ge 0})^*$, where the t_i's form a non decreasing sequence. A timed language is a set of timed words. For a timed language L, the corresponding untimed language, written $Untime(L)$, is the projection of L on Σ^*. For an ITA $\mathcal{A}$, a run is a path in $\mathcal{T}_{\mathcal{A}}$ from the initial to an accepting configuration such that time steps alternate with discrete steps: $(q_0, v_0) \xrightarrow{d_1} (q_0, v_0') \xrightarrow{a_1} (q_1, v_1) \cdots \xrightarrow{d_n} (q_{n-1}, v_{n-1}') \xrightarrow{a_n} (q_n, v_n)$, with $v_0 = \mathbf{0}$. The sequence $t_1, \dots, t_n$ of absolute dates associated with this run is $t_i = \sum_{j=1}^{i} d_j$ and a timed word accepted by $\mathcal{A}$ is obtained by removing from the

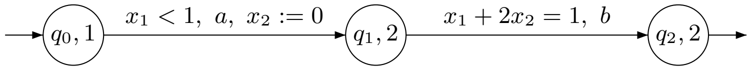


Fig. 1. An ITA $\mathcal{A}_1$ with two interrupt levels

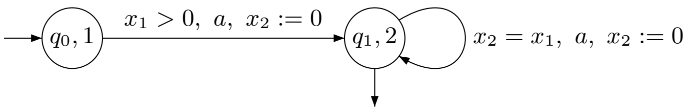


Fig. 2. An ITA $\mathcal{A}_2$ for L_2

sequence $(a_1, t_1) \ldots (a_n, t_n)$ the pairs such that $a_i = \varepsilon$. We denote by $\mathcal{L}(\mathcal{A})$ the set of timed words accepted by $\mathcal{A}$. ITL denotes the family of timed languages accepted by an ITA.

We end this paragraph with two examples of ITA. In the figures, the level of a state is indicated beside its name. For the automaton $\mathcal{A}_1$ in Fig. 1, state q_0 is the initial state with level 1. States q_1 and q_2 are on level 2, and q_2 is the final state. There are two interrupt clocks x_1 and x_2.Entering state q_1 at time $1-\tau$ for some τ, clock x_1 is suspended and state q_2 is reached at time $1-\tau+t$ with $1-\tau+2t = 1$. The language accepted by $\mathcal{A}_1$ is thus $L_1 = \{(a, 1-\tau)(b, 1-\tau/2) \mid 0 < \tau \leq 1\}$. The ITA in Fig. 2 also has two levels and two interrupt clocks x_1 and x_2. It accepts $L_2 = \{(a, \tau)(a, 2\tau) \ldots (a, n\tau) \mid n \in \mathbb{N}, \tau > 0\}$.

2.2 Expressive Power of ITA

We now compare the expressive power of ITA with classical Timed Automata (TA) and Controlled Real-Time Automata (CRTA) [9].

Recall that a Timed Automaton is a tuple $\mathcal{A} = (\Sigma, Q, q_0, F, X, \Delta)$, where Σ is a finite alphabet, Q is a finite set of states, q_0 is the initial state, $F \subseteq Q$ is the set of final states, X is a set of clocks and $\Delta \subseteq Q \times [\mathcal{C}(X) \times (\Sigma \cup \{\varepsilon\}) \times \mathcal{U}(X)] \times Q$ is the set of transitions.

Since all clocks evolve with rate 1, the only difference from ITA in the definition of semantics concerns a time step of duration d, which is defined by $(q, v) \xrightarrow{d} (q, v+d)$.

CRTA extend TA with the following features: the clocks and the states are partionned according to colors belonging to a set Ω and with every state is associated a rational velocity. When time elapses in a state, the set of active clocks (i.e. with the color of the state) evolve with rate equal to the velocity of the state while other clocks remain unchanged. For sake of simplicity, we now propose a slightly simplified version of CRTA.

Definition 3. *A CRTA $\mathcal{A} = (\Sigma, Q, q_0, F, X, up, low, vel, \lambda, \Delta)$ on a finite set Ω of colors is defined by:*

- Σ is the alphabet of actions,
- Q is a set of states, $q_0 \in Q$ is the initial state, F is the set of final states,
- X is a set of clocks, up and low are mappings which associate with each clock respectively an upper and a lower bound, $vel : Q \mapsto \mathbb{Q}$ is the velocity mapping,
- $\lambda : X \cup Q \mapsto \Omega$ is the coloring mapping and
- Δ is the set of transitions. A transition in Δ has guards in $\mathcal{C}(X)$ with constants in $\mathbb{Q}$ and updates in $\mathcal{U}(X)$ (i.e. only reset). The lower and upper bound mappings satisfy $low(x) \leq 0 \leq up(x)$ for each clock $x \in X$, and $low(x) \leq b \leq up(x)$ for each constant such that $x \bowtie b$ is a constraint in $\mathcal{A}$.

The original semantics of CRTA is rather involved in order to obtain decidability of the reachability problem. It ensures that entering a state q in which clock x

is active, the following conditions on the clock bounds hold : if $vel(q) > 0$ then $x \geq low(x)$ and if $vel(q) < 0$ then $x \leq up(x)$. Instead (and equivalently) we add a syntactical restriction which ensures this behaviour. For instance, if a transition with guard φ and reset u enters state q with $vel(q) < 0$ and if x is the only clock such that $\omega(x) = \omega(q)$, then we replace this transition by two other transitions: the first one has guard $\varphi \wedge x > up(x)$ and adds $x := 0$ to the reset condition u, the other has guard $\varphi \wedge x \leq up(x)$ and reset u. In the general case where k clocks have color $\omega(q)$, this leads to 2^k transitions. With this syntactical condition, again the only difference from ITA concerns a time step of duration d, defined by $(q, v) \xrightarrow{d} (q, v')$, with $v'(x) = v(x) + vel(q)d$ if $\omega(x) = \omega(q)$ and $v'(x) = v(x)$ otherwise.

We denote by TL (resp. CRTL) the family of timed languages accepted by TA (resp. CRTA), with TL strictly contained in CRTL.

Proposition 1

1. *There exists a language in ITL which is not in TL.*
2. *There exists a language in ITL whichis not in CRTL.*

Proof. To prove the first point, consider the ITA $\mathcal{A}_1$ in Fig. 1. Suppose, by contradiction, that L_1 is accepted by some timed automaton $\mathcal{B}$ in TA (possibly with ε-transitions) and let d be the granularity of $\mathcal{B}$, *i.e.* the gcd of all rational constants appearing in the constraints of $\mathcal{B}$ (thus each such constant can be written k/d for some integer k). Then the word $w = (a, 1 - 1/d)(b, 1 - 1/2d)$ is accepted by $\mathcal{B}$ through a finite path. Consider now the automaton $\mathcal{B}'$ in TA, consisting of this single path (where states may have been renamed). We have $w \in \mathcal{L}(\mathcal{B}') \subseteq \mathcal{L}(\mathcal{B}) = L$ and $\mathcal{B}'$ contains no cycle. Using the result in [5], we can build a timed automaton $\mathcal{B}''$ without ε-transition and with same granularity d such that $\mathcal{L}(\mathcal{B}'') = \mathcal{L}(\mathcal{B}')$, so that $w \in \mathcal{L}(\mathcal{B}'')$. The accepting path for w in $\mathcal{B}''$ contains two transitions : $p_0 \xrightarrow{\varphi_1, a, r_1} p_1 \xrightarrow{\varphi_2, b, r_2} p_2$. After firing the a-transition, all clock values are $1 - 1/d$ or 0, thus all clock values are $1 - 1/2d$ or $1/2d$ when the b-transition is fired. Let $x \bowtie c$ be an atomic proposition appearing in φ_2. Since the granularity of $\mathcal{B}''$ is d, the $\bowtie$ operator cannot be $=$ otherwise the constraint would be $x = 1/2d$ or $x = 1 - 1/2d$. If the constraint is $x < c$, $x \leq c$, $x > c$, or $x \geq c$, the path will also accept some word $(a, 1 - 1/d)(b, t)$ for some $t \neq 1 - 1/2d$. This is also the case if the constraint φ_2 is true. We thus obtain a contradiction with $\mathcal{L}(\mathcal{B}'') \subseteq L$, which ends the proof.

To prove the second point, consider the language $L_2 = \{(a, \tau)(a, 2\tau) \ldots (a, n\tau) \mid n \in \mathbb{N}, \tau > 0\}$ defined above, accepted by the ITA $\mathcal{A}_2$ in Fig. 2. This language cannot be accepted by a CRTA (see [9]).

Note that we do not yet know of a language accepted by an automaton in TA (or CRTA) but not by an automaton in ITA. However, we conjecture that these classes are incomparable.

3 Reachability Is Decidable in ITA

3.1 General Case

Similarly to the decision algorithm for reachability in TA (and in CRTA), the procedure for an ITA $\mathcal{A}$ is based on the construction of a (finite) class graph which is time abstract bisimilar to the transition system $\mathcal{T}_{\mathcal{A}}$. However the construction of classes is much more involved than in the case of TA. More precisely, it depends on the expressions occurring in the guards and updates of the automaton (while in TA it depends only on the maximal constant occurring in the guards). We associate with each state q a set of expressions $Exp(q)$ with the following meaning. The values of clocks giving the same ordering of these expressions correspond to a class. In order to define $Exp(q)$, we first build a family of sets $\{E_i\}_{1\leq i\leq n}$. Then $Exp(q) = \bigcup_{i\leq\lambda(q)} E_i$. Finally in proposition 3 we show how to build the class graph which decides the reachability problem.

We first introduce an operation, called *normalization*, on expressions relative to some level. As explained in the construction below, this operation will be used to order the respective values of expressions at a given level.

Definition 4 (Normalization). *Let $C = \sum_{i\leq k} a_i x_i + b$ be an expression over $X_k = \{x_j \mid j \leq k\}$, the k-*normalization *of C, $\texttt{norm}(C,k)$, is defined by:*

- *if $a_k \neq 0$ then $\texttt{norm}(C,k) = x_k + (1/a_k)(\sum_{i<k} a_i x_i + b)$;*
- *else $\texttt{norm}(C,k) = C$.*

Since guards are linear expressions with rational constants, we can assume that in a guard $C \bowtie 0$ occurring in a transition outgoing from a state q with level k, the expression C is either $x_k + \sum_{i<k} a_i x_i + b$ (by k-normalizing the expression and if necessary changing the comparison operator) or $\sum_{i<k} a_i x_i + b$.

Construction of $\{E_k\}_{k\leq n}$. The construction proceeds top down from level n to level 1 after initializing $E_k = \{x_k, 0\}$ for all k. As we shall see below, when handling the level k, we add new terms to $\{E_i\}_{1\leq i\leq k}$.

- At level k, first for every expression $\alpha x_k + \sum_{i<k} a_i x_i + b$ (with $\alpha \in \{0,1\}$) occurring in a guard of an edge leaving a state of level k, we add $-\sum_{i<k} a_i x_i - b$ to E_k.
- Then we iterate the following procedure until no new term is added to any E_i for $1 \leq i \leq k$.
  1. Let $q \xrightarrow{\varphi,a,u} q'$ with $\lambda(q') \geq k$ and $\lambda(q) \geq k$. Let $C \in E_k$, then we add $C[u]$ to E_k.
  2. Let $q \xrightarrow{\varphi,a,u} q'$ with $\lambda(q') \geq k$ and $\lambda(q) < k$. Let $C, C' \in E_k$, then we compute $C'' = norm(C[u] - C'[u], \lambda(q))$. Let us write C'' as $\alpha x_{\lambda(q)} + \sum_{i<\lambda(q)} a_i x_i + b$ with $\alpha \in \{0,1\}$. Then we add $-\sum_{i<\lambda(q)} a_i x_i - b$ to $E_{\lambda(q)}$.

Proposition 2. *The construction procedure of $\{E_k\}_{k\leq n}$ terminates and the size of every E_k is bounded by $B^{2^{n(n-k+1)}+1}$ where B is the maximum between 2 and the number of edges of the ITA.*

Proof. Given some k, we prove the termination of the stage relative to k. Observe that the second step only adds new expressions to $E_{k'}$ for $k' < k$. Thus the two steps can be ordered. Let us prove the termination of the first step of the saturation procedure. We denote $E_k^0 \equiv E_k$ at the beginning of this stage and $E_k^i \equiv E_k$ after the insertion of the i^{th} item in it. With each added item $C[u]$ can be associated its *father* C. Thus we can view E_k as an increasing forest with finite degree (due to the finitess of the edges). Assume that this step does not terminate. Then we have an infinite forest and by König lemma, it has an infinite branch $C_0, C_1, \ldots$ where $C_{i+1} = C_i[u_i]$ for some update u_i such that $C_{i+1} \neq C_i$. Observe that the number of updates that change the variable x_k is either 0 or 1 since once x_k disappears it cannot appear again. We split the branch into two parts before and after this update or we still consider the whole branch if there is no such update. In these (sub)branches, we conclude with the same reasonning that there is at most one update that change the variable x_{k-1}. Iterating this process, we conclude that the number of updates is at most $2^k - 1$ and the length of the branch is at most 2^k. Thus the final size of E_k is at most $E_k^0 \times B^{2^k}$ since the width of the forest is bounded by B.

In the second step, we add at most $B \times (|E_k| \times (|E_k| - 1))/2$ to E_i for every $i < k$. This concludes the proof of termination.

We now prove by a painful backward induction that as soon as $n \geq 2$, $|E_k| \leq B^{2^{n(n-k+1)}+1}$. We define $p_k \equiv |E_k|$.

Basis case $k = n$

$p_n \leq p_n^0 \times B^{2^n}$ where p_n^0 is the number of guards of the outgoing edges from states of level n. Thus:

$p_n \leq B \times B^{2^n} = B^{2^n+1} = B^{2^{n(n-n+1)}+1}$

which is the claimed bound.

Inductive case

Assume that the bound holds for $k < j \leq n$. Due to the second step of the procedure, we have:

$p_k^0 \leq B + B \times ((p_{k+1} \times (p_{k+1} - 1))/2 + \cdots + (p_n \times (p_n - 1))/2)$

$p_k^0 \leq B + B \times (B^{2^{n(n-k)+1}+2} + \cdots + B^{2^{n+1}+2})$

$p_k^0 \leq B \times (n - k + 1) \times B^{2^{n(n-k)+1}+2}$

$p_k^0 \leq B \times B^n \times B^{2^{n(n-k)+1}+2}$ *(here we use* $B \geq 2$*)*

$p_k^0 \leq B^{2^{n(n-k)+1}+n+3}$

$p_k \leq B^{2^{n(n-k)+1}+2^k+n+3}$

Let us consider the term $\delta = 2^{n(n-k+1)} + 1 - 2^{n(n-k)+1} - 2^k - n - 3$

$\delta \geq (2^{n-1} - 1)2^{n(n-k)+1} - (2^k + n + 2)$

$\delta \geq (2^{n-1} - 1)2^{n(n-k)+1} - (2^{n-1} + 2^n)$

$\delta \geq (2^{n-1} - 1)2^{n(n-k)+1} - 2^{n+1} \geq 0$

Thus: $p_k \leq B^{2^{n(n-k)+1}+2^k+n+3} \leq B^{2^{n(n-k+1)}+1}$

which is the claimed bound.

Proposition 3. *The reachability problem for ITA is decidable and belongs to* 2-EXPSPACE *and to* PSPACE *when the number of clocks is fixed.*

Proof.
Class definition. Let $\mathcal{A}$ be an ITA, the decision algorithm is based on the construction of a (finite) class graph which is time abstract bisimilar to the transition system $\mathcal{T}_{\mathcal{A}}$. A class is a syntactical representation of a subset of reachable configurations. More precisely, it is defined as a pair $R = (q, \{\preceq_k\}_{1\leq k\leq\lambda(q)})$ where q is a state and $\preceq_k$ is a total preorder over E_k.

The class R describes the set of valuations:

$$[\![R]\!]= \{(q,v) \mid \forall k \leq \lambda(q)\ \forall (g,h) \in E_k,\ g[v] \leq h[v] \text{ iff } g \preceq_k h\}$$

Observe that the number of classes is bounded by:

$$|Q| \cdot 3^{B^{2^{(n^2)}+1}}$$

where n is the number of clocks of $\mathcal{A}$ and B is defined in proposition 2.

As usual, there are two kinds of transitions in the graph, corresponding to discrete steps and time steps.

Discrete step. Let $R = (q, \{\preceq_k\}_{1\leq k\leq\lambda(q)})$ and $R' = (q', \{\preceq'_k\}_{1\leq k\leq\lambda(q')})$ be two classes. There is a transition $R \xrightarrow{e} R'$ for a transition $e : q \xrightarrow{\varphi,a,u} q'$ if there is some $(q,v) \in[\![R]\!]$ and $(q',v') \in[\![R']\!]$ such that $(q,v) \xrightarrow{e} (q',v')$. In this case, for all $(q,v) \in[\![R]\!]$ there is a $(q',v') \in[\![R']\!]$ such that $(q,v) \xrightarrow{e} (q',v')$. This can be decided as follows.

Firability condition. Write $\varphi = \bigwedge_{1\leq j\leq J'} C_j \leq 0 \wedge \bigwedge_{J'+1\leq j\leq J} \neg(C_j \leq 0)$. By definition of an ITA, for every j, $C_j = \alpha x_{\lambda(q)} + \sum_{i<\lambda(q)} a_i x_i + b$ (with $\alpha \in \{0,1\}$). By construction $C'_j = -\sum_{i<\lambda(q)} a_i x_i - b \in E_{\lambda(q)}$. If $j \leq J'$ then we require that $\alpha x_{\lambda(q)} \preceq_k C'_j$. If $j > J'$ then we require that $\neg(\alpha x_{\lambda(q)} \preceq_k C'_j)$.

Successor definition. R' is defined as follows. Let $k \leq \lambda(q')$ and $g', h' \in E_k$.

1. Either $k \leq \lambda(q)$, by construction, $g'[u], h'[u] \in E_k$ then $g' \preceq'_k h'$ iff $g'[u] \preceq_k h'[u]$.
2. Or $k > \lambda(q)$, let $D = g'[u] - h'[u] = \sum_{i\leq\lambda(q)} c_i x_i + d$, and $C = norm(D, \lambda(q))$, and write $C = \alpha x_{\lambda(q)} + \sum_{i<\lambda(q)} a_i x_i + b$ (with $\alpha \in \{0,1\}$). By construction $C' = -\sum_{i<\lambda(q)} a_i x_i - b \in E_{\lambda(q)}$.
   When $c_{\lambda(q)} \geq 0$ then $g' \preceq'_k h'$ iff $C' \preceq_{\lambda(q)} \alpha x_{\lambda(q)}$.
   When $c_{\lambda(q)} < 0$ then $g' \preceq'_k h'$ iff $\alpha x_{\lambda(q)} \preceq_{\lambda(q)} C'$.

By definition of $[\![\ ]\!]$,

- $\forall (q,v) \in[\![R]\!]$, if there exists $(q,v) \xrightarrow{e} (q',v')$ then the firability condition is fulfilled and (q',v') belongs to $[\![R']\!]$.
- If the firability condition is fulfilled then $\forall (q,v) \in[\![R]\!]$ there exists $(q',v') \in [\![R']\!]$ such that $(q,v) \xrightarrow{e} (q',v')$.

Time step. Let $R = (q, \{\preceq_k\}_{1\leq k\leq\lambda(q)})$.
The time successor $Post(R) = (q, \{\preceq'_k\}_{1\leq k\leq\lambda(q)})$ of R is defined as follows.

For every $k' < \lambda(q)$ $\preceq'_k = \preceq_k$. Let $\sim = \preceq_{\lambda(q)} \cap \preceq^{-1}_{\lambda(q)}$ be the equivalence relation induced by the preorder. On equivalence classes, this (total) preorder becomes a (total) order. Let V be the equivalence class containing $x_{\lambda(q)}$.

1. Either $V = \{x_{\lambda(q)}\}$ and it is the greatest equivalence class. Then $\preceq'_{\lambda(q)} = \preceq_{\lambda(q)}$ (thus $Post(R) = R$).
2. Either $V = \{x_{\lambda(q)}\}$ and it is not the greatest equivalence class. Let V' be the next equivalence class. Then $\preceq'_{\lambda(q)}$ is obtained by merging V and V', and preserving $\preceq_{\lambda(q)}$ elsewhere.
3. Either V is not a singleton. Then we split V into $V \setminus \{x_{\lambda(q)}\}$ and $\{x_{\lambda(q)}\}$ and "extend" $\preceq_{\lambda(q)}$ by $V \setminus \{x_{\lambda(q)}\} \preceq'_{\lambda(q)} \{x_{\lambda(q)}\}$.

By definition of $[\![\,]\!]$, $\forall (q, v) \in [\![R]\!]$, there exists $d > 0$ such that $(q, v+d) \in Post(R)$ and $\forall 0 \leq d' \leq d,\ (q, v + d') \in R \cup Post(R)$.

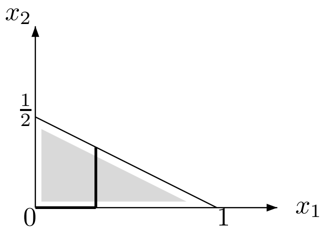
The initial state of this graph is defined by the class R_0 with $[\![R_0]\!]$ containing $(q_0, \mathbf{0})$ which can be straightforwardly determined. The reachability problem is then solved by a non deterministic search of a path in this graph (without building it) leading to the complexity stated in the proposition. When the number of clocks is fixed the length of this path is at most exponential w.r.t. the size of the problem leading to a PSPACE procedure.

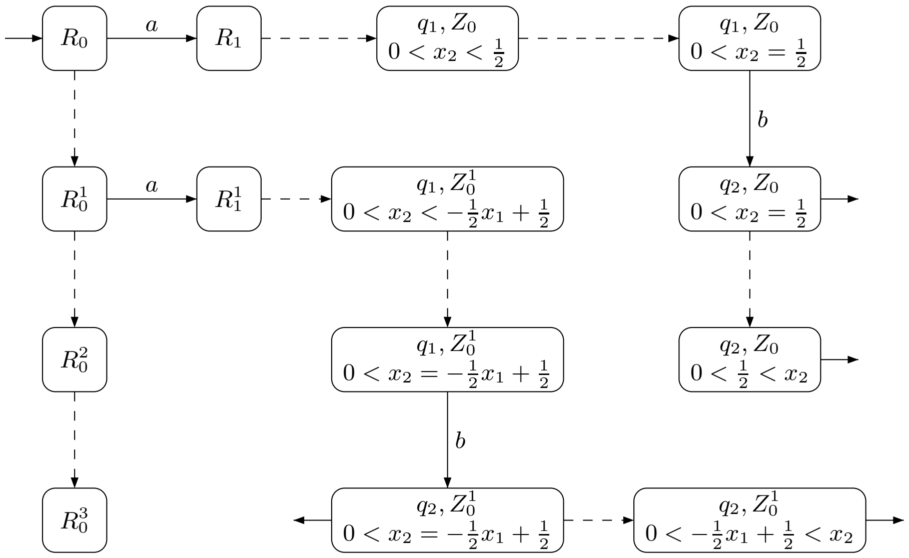
Example. We illustrate this construction of a class automaton for the automaton $\mathcal{A}_1$ from section 2 (see figure 3, where dashed lines indicate time successors).

In this case, we obtain $E_1 = \{x_1, 0, 1\}$ and $E_2 = \{x_2, 0, -\frac{1}{2}x_1 + \frac{1}{2}\}$. In state q_0, the only relevant clock is x_1 and the initial class is $R_0 = (q_0, Z_0)$ with $Z_0 : x_1 = 0 < 1$. Its time successor is $R_0^1 = (q_0, Z_0^1)$ with $Z_0^1 : 0 < x_1 < 1$. Transition a leading to q_1 can be taken from both classes, but not from the next time successors $R_0^2 = (q_0, 0 < x_1 = 1)$ and $R_0^3 = (q_0, 0 < 1 < x_1)$.

Transition a switches from R_0 to $R_1 = (q_1, Z_0, x_2 = 0 < \frac{1}{2})$, because $x_1 = 0$, and from R_0^1 to $R_1^1 = (q_1, Z_0^1, x_2 = 0 < -\frac{1}{2}x_1 + \frac{1}{2})$. Transition b is fired from those time successors for which $x_2 = -\frac{1}{2}x_1 + \frac{1}{2}$.

A geometric view is given below, with a possible trajectory: first the value of x_1 increases from 0 in state q_0 (horizontal line) and, after transition a occurs, its value is frozen in state q_1 while x_2 increases (vertical line) until reaching the line $x_2 = -\frac{1}{2}x_1 + \frac{1}{2}$. The light gray zone is $(0 < x_1 < 1,\ 0 < x_2 < -\frac{1}{2}x_1 + \frac{1}{2})$, associated with q_1.





R_0
a
R_1
q_1, Z_0 $0 < x_2 < \frac{1}{2}$
q_1, Z_0 $0 < x_2 = \frac{1}{2}$
b
R_0^1
R_1^1
q_1, Z_0^1 $0 < x_2 < -\frac{1}{2}x_1 + \frac{1}{2}$
q_2, Z_0 $0 < x_2 = \frac{1}{2}$
R_0^2
q_1, Z_0^1 $0 < x_2 = -\frac{1}{2}x_1 + \frac{1}{2}$
q_2, Z_0 $0 < \frac{1}{2} < x_2$
R_0^3
q_2, Z_0^1 $0 < x_2 = -\frac{1}{2}x_1 + \frac{1}{2}$
q_2, Z_0^1 $0 < -\frac{1}{2}x_1 + \frac{1}{2} < x_2$

Fig. 3. The class automaton for $\mathcal{A}_1$

3.2 A Simpler Model

In practice, the clock associated with some level measures the time spent in this level or more generally the time spent by some tasks at this level. Thus when going to a higher level, this clock is "frozen" until returning to this level. The following restriction of the ITA model takes this feature into account.

Definition 5. *The subclass ITA$_-$ of ITA is defined by the following restriction on updates. For a transition $q \xrightarrow{\varphi,a,u} q'$ of an automaton $\mathcal{A}$ in ITA$_-$ (with $k = \lambda(q)$ and $k' = \lambda(q')$), the update u is of the form $\wedge_{i=1}^{n} x_i := C_i$ with:*

- *if $k' < k$, $u = \wedge_{i=1}^{n} x_i := x_i$ i.e. no updates;*
- *if $k' \geq k$ then C_k is of the form $\sum_{j=1}^{k-1} a_j x_j + b$ or $C_k = x_k$, $C_i = 0$ if $k < i \leq k'$ and $C_i = x_i$ otherwise.*

Observe that the automata of figures 1 and 2 belong to ITA$_-$. So the expressiveness results of proposition 1 still hold for ITA$_-$.

It turns out that the reachability problem for ITA$_-$ can be solved more efficiently.

Proposition 4. *The reachability problem for ITA$_-$ belongs to* NEXPTIME *and to* NP *when the number of clocks is fixed.*

Proof. Let $\mathcal{A} = (\Sigma, Q, q_0, F, X, \lambda, \Delta)$ be an ITA$_-$. In the sequel, the level of a transition is the level of its source state. Let $E = |\Delta|$ be the number of transitions and given a fixed run, let m_k be the number of occurrences of transitions of level k.

Assume that there is a run ρ from (q_0, v_0) to some configuration (q_f, v_f). We build a run ρ' from (q_0, v_0) to (q_f, v_f) which fulfills:

- $m'_1 \leq (E+1)^2$
- $\forall k \; m'_{k+1} \leq (E+1)^2(m'_k + 1)$

Thus $\sum_{k=1}^{n} m'_k = O(E^{2n})$.

We iteratively modify the run ρ by considering the transitions of level k from 1 to n. For the basis case $k = 1$, we consider in the run ρ the subsequence $(e_1, \cdots, e_p)$ of transitions in Δ of level 1 which update x_1. Observe that if $e_i = e_j$ for some $i < j$, we can remove the subrun between these two transitions, because x_1 is the only relevant clock before the firing of e_i (or e_j). Thus we obtain a run with at most E such transitions. Now we consider a subsequence $(e'_1, \cdots, e'_r)$ of transitions of level 1 occurring between two of these transitions (or before the first or after the last). Observe that if $e'_i = e'_j$ for some $i < j$, we can replace the subrun between these two transitions by a time step corresponding to the difference of values of x_1. Indeed, since there is no update, the clock value after the second transition is greater than or equal to the value after the first transition. Thus we obtain a run with at most $(E+1)^2$ transitions of level 1 (including at most $E(E+1)$ transitions without update).

Assume that the bound holds at levels less than $k+1$ and consider the subrun between two consecutive transitions of level less than $k+1$ (or before the first or after the last). By definition of ITA$_-$, the values of clocks $x_1, \ldots, x_k$ are unchanged during this subrun. Thus for two occurrences of the same transition of level $k+1$, the update of x_{k+1} is the same. So we can apply the same reasoning as for the basis case, thus leading to the claimed bound.

The decision procedure is as follows. It non deterministically guesses a path in the ITA$_-$ whose length is less than or equal to the bound. In order to check that this path yields a run, it builds a linear program whose variables are $\{x_i^j\}$, where x_i^j is the value of clock x_i after the jth step, and $\{d_j\}$ where d_j is the amount of time elapsed during the jth step, when j corresponds to a time step. The equations and inequations are deduced from the guards and updates of discrete transitions in the path and the delay of the time steps. The size of this linear program is exponential w.r.t. the size of the ITA$_-$. As a linear program can be solved in polynomial time [15], we obtain a procedure in NEXPTIME. If the number of clocks is fixed the number of variables is now polynomial w.r.t. the size of the problem.

4 Combining ITA and CRTA

We finally define an extended class denoted by ITA$^+$, including a set of clocks at an implicit additional level 0, corresponding to a basic task described as in a CRTA.

Definition 6 (ITA$^+$). *An* extended interrupt timed automaton *is a tuple* $\mathcal{A} = (Q, q_0, F, X \uplus Y, \Sigma, \Omega, \lambda, up, low, vel, \Delta)$*, where:*

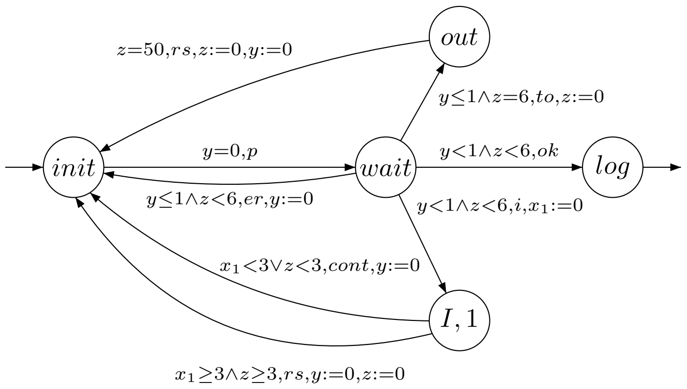
- *Q is a finite set of states, q_0 is the initial state and $F \subseteq Q$ is the set of final states.*
- *$X = \{x_1, \ldots, x_n\}$ consists of n interrupt clocks and Y is a set of basic clocks,*
- *Σ is a finite alphabet,*
- *Ω is a set of colours, the mapping $\lambda : Q \uplus Y \mapsto \{1, \ldots, n\} \uplus \Omega$ associates with each state its level or its colour, with $x_{\lambda(q)}$ the active clock in state q for $\lambda(q) \in \mathbb{N}$ and $\lambda(y) \in \Omega$ for $y \in Y$,*
- *up and low are mappings from Y to $\mathbb{Q}$ with the same constraints of CRTA (see definition 3), and $vel : Q \mapsto \mathbb{Q}$ is the clock rate with $\lambda(q) \notin \Omega \Rightarrow vel(q) = 1$*
- *$\Delta \subseteq Q \times [\mathcal{C}^+(X \cup Y) \times (\Sigma \cup \{\varepsilon\}) \times \mathcal{U}^+(X \cup Y)] \times Q$ is the set of transitions. Let $q \xrightarrow{\varphi,a,u} q'$ in Δ be a transition.*
  1. *The guard φ is of the form $\varphi_1 \wedge \varphi_2$ with the following conditions. If $\lambda(q) \in \mathbb{N}$, φ_1 is an ITA guard on X and otherwise $\varphi_1 = true$. Constraint φ_2 is a CRTA guard on Y (also possibly equals to true).*
  2. *The update u is of the form $u_1 \wedge u_2$ fullfilling the following conditions. Assignments from u_1 update the clocks in X with the constraints of ITA when $\lambda(q)$ and $\lambda(q')$ belong to $\mathbb{N}$. Otherwise it is a global reset of clocks in X. Assignments from u_2 update clocks from Y, like in CRTA.*

Any ITA can be viewed as an ITA$^+$ with Y empty and $\lambda(Q) \subseteq \{1, \ldots, n\}$, and any CRTA can be viewed as an ITA$^+$ with X empty and $\lambda(Q) \subseteq \Omega$. Class ITA$^+$ combines both models in the following sense. When the current state q is such that $\lambda(q) \in \Omega$, the ITA part is inactive. Otherwise, it behaves as an ITA but with additional constraints about clocks of the CRTA involved by the extended guards and updates. The semantics of ITA$^+$ is defined as usual but now takes into account the velocity of CRTA clocks.

Definition 7 (Semantics of ITA$^+$). *The semantics of an automaton $\mathcal{A}$ in ITA$^+$ is defined by the transition system $\mathcal{T}_{\mathcal{A}} = (S, s_0, \rightarrow)$. The set S of configurations is $\{(q, v) \mid q \in Q,\ v \in \mathbb{R}^{X \cup Y}\}$, with initial configuration $(q_0, \mathbf{0})$. An accepting configuration of $\mathcal{T}_{\mathcal{A}}$ is a pair (q, v) with q in F. The relation $\rightarrow$ on S consists of time steps and discrete steps, the definition of the latter being the same as before:*

Time steps: *Only the active clocks in a state can evolve, all other clocks are suspended. For a state q with $\lambda(q) \in \mathbb{N}$ (the active clock is $x_{\lambda(q)}$), a time step of duration d is defined by $(q, v) \xrightarrow{d} (q, v')$ with $v'(x_{\lambda(q)}) = v(x_{\lambda(q)}) + d$ and $v'(x) = v(x)$ for any other clock x. For a state q with $\lambda(q) \in \Omega$ (the active clocks are $Y' = Y \cap \lambda^{-1}(\lambda(q))$), a time step of duration d is defined by $(q, v) \xrightarrow{d} (q, v')$ with $v'(y) = v(y) + vel(q)d$ for $y \in Y'$ and $v'(x) = v(x)$ for any other clock x.*

Discrete steps: *A discrete step $(q, v) \xrightarrow{a} (q', v')$ occurs if there exists a transition $q \xrightarrow{\varphi,a,u} q'$ in Δ such that $v \models \varphi$ and $v' = v[u]$.*



In order to illustrate the interest of the combined models, an example of a (simple) login procedure is described in the figure above as a TA with interruptions at a single level. First it immediately displays a prompt and arms a time-out of 1 t.u. handled by clock y (transition $init \xrightarrow{p} wait$). Then either the user answers correctly within this delay (transition $wait \xrightarrow{ok} log$) or he/she answers incorrectly or let time elapse, both cases with transition $wait \xrightarrow{er} init$, and the system prompts again. The whole process is controlled by a global time-out of 6 t.u. (transition $wait \xrightarrow{to} out$) followed by a long suspension (50 t.u.) before reinitializing the process (transition $out \xrightarrow{rs} init$). Both delays are handled by clock z. At any time during the process (in fact in state *wait*), a system interrupt may occur (transition $wait \xrightarrow{i} I$). If the time spent (measured by clock x_1) during the interrupt is less than 3 t.u. or the time already spent by the user is less than 3, the login process resumes (transition $I \xrightarrow{cont} init$). Otherwise the login process is reinitialized allowing again the 6 t.u. (transition $I \xrightarrow{rs} init$). In both cases, the prompt will be displayed again. Since invariants are irrelevant for the reachability problem we did not include them in the models. Of course, in this example state *wait* should have invariant $y \leq 1 \wedge z \leq 6$ and state *out* should have invariant $z \leq 50$.

We extend the decidability and complexity results of the previous models when combining them with CRTA. Class ITA_-^+ is obtained in a similar way by combining ITA_- with CRTA. Proofs are omitted here.

Proposition 5

1. The reachability problem for ITA$^+$ is decidable and belongs to 2-EXPSPACE *and is* PSPACE*-complete when the number of interrupt clocks is fixed.*
2. The reachability problem for ITA$_-^+$ belongs to NEXPTIME *and is* PSPACE*-complete when the number of interrupt clocks is fixed.*

5 Conclusion

We have proposed a subclass of hybrid automata, called ITA. An ITA describes a set of tasks, executing at interrupt levels, with exactly one active clock at each level. We prove that the reachability problem is decidable in this class, with a procedure in 2-EXPSPACE. We also consider restrictions on this class, that make the complexity of decision lower (in NEXPTIME). We show that these results still hold for a combination of ITA with the class CRTA. When the number of clocks is fixed, the complexity bound is the same as the one of TA and even better in case of ITA_{-}. Whether the classes TA or CRTA are contained in ITA and whether ITA_{-} is a strict subclass of ITA are open questions.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126, 183–235 (1994)
2. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theoretical Computer Science 138, 3–34 (1995)
3. Asarin, E., Maler, O., Pnueli, A.: Reachability Analysis of Dynamical Systems having Piecewise-Constant Derivatives. Theoretical Computer Science 138, 35–66 (1995)
4. Asarin, E., Schneider, G., Yovine, S.: Algorithmic Analysis of Polygonal Hybrid Systems, Part I: Reachability. Theoretical Computer Science 379(1-2), 231–265 (2007)
5. Bérard, B., Diekert, V., Gastin, P., Petit, A.: Characterization of the expressive power of silent transitions in timed automata. Fundamenta Informaticae 36, 145–182 (1998)
6. Bouyer, P.: Forward analysis of updatable timed automata. Formal Methods in System Design 24(3), 281–320 (2004)
7. Brihaye, T., Bruyère, V., Raskin, J.-F.: On Model-Checking Timed Automata with Stopwatch Observers. Information and Computatiion 2004(3), 408–433 (2006)
8. Cassez, F., Larsen, K.G.: The impressive power of stopwatches. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 138–152. Springer, Heidelberg (2000)
9. Demichelis, F., Zielonka, W.: Controlled timed automata. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 455–469. Springer, Heidelberg (1998)
10. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? Journal of Computer and System Sciences 57, 94–124 (1998)
11. Kesten, Y., Pnueli, A., Sifakis, J., Yovine, S.: Decidable Integration Graphs. Information and Computation 150(2), 209–243 (1999)
12. Lafferriere, G., Pappas, G.J., Yovine, S.: A new class of decidable hybrid systems. In: Vaandrager, F.W., van Schuppen, J.H. (eds.) HSCC 1999. LNCS, vol. 1569, pp. 137–151. Springer, Heidelberg (1999)
13. Lafferriere, G., Pappas, G.J., Yovine, S.: Symbolic reachability computations for families of linear vector fields. Journal of Symbolic Computation 32(3), 231–253 (2001)
14. Maler, O., Manna, Z., Pnueli, A.: From Timed to Hybrid Systems. In: Huizing, C., de Bakker, J.W., Rozenberg, G., de Roever, W.-P. (eds.) REX 1991. LNCS, vol. 600, pp. 447–484. Springer, Heidelberg (1992)
15. Roos, C., Terlaky, T., Vial, J.-P.: Theory and Algorithms for Linear Optimization. An Interior Point Approach. Wiley-Interscience, John Wiley & Sons Ltd., West Sussex (1997)

Parameter Reduction in Grammar-Compressed Trees

Markus Lohrey[1], Sebastian Maneth[2], and Manfred Schmidt-Schauß[3]

[1] Universität Leipzig, Institut für Informatik, Germany
[2] NICTA and University of New South Wales, Australia
[3] Johann Wolfgang Goethe-Universität Frankfurt, Institut für Informatik, Germany
lohrey@informatik.uni-leipzig.de, sebastian.maneth@nicta.com.au, schauss@cs.uni-frankfurt.de

Abstract. Trees can be conveniently compressed with linear straight-line context-free tree grammars. Such grammars generalize straight-line context-free string grammars which are widely used in the development of algorithms that execute directly on compressed structures (without prior decompression). It is shown that every linear straight-line context-free tree grammar can be transformed in polynomial time into a monadic (and linear) one. A tree grammar is monadic if each nonterminal uses at most one context parameter. Based on this result, a polynomial time algorithm is presented for testing whether a given nondeterministic tree automaton with sibling constraints accepts a tree given by a linear straight-line context-free tree grammar. It is shown that if tree grammars are nondeterministic or non-linear, then reducing their numbers of parameters cannot be done without an exponential blow-up in grammar size.

1 Introduction

The current massive increase in data volumes motivates the development of algorithms on *compressed data*, like for instance compressed strings, trees, and graphs. The general goal is to construct algorithms that work directly on compressed data, without prior decompression. Considerable amount of work has been done concerning algorithms that execute on compressed strings, see [13] for a survey. In this field, a popular succinct string representation are context-free grammars which generate exactly one string. It can be statically guaranteed that only one string is generated, by restricting to acyclic grammars with exactly one production per nonterminal. Such grammars are known as straight-line programs, briefly SLPs. Since an SLP with n productions may generate a string of length 2^n, an SLP can be seen as a compressed representation of the generated string. Some of the nice features of SLPs are:

- Many dictionary based compression schemes, like for instance LZ78 and LZ77 can be converted efficiently into SLPs, see, e.g., [13] for further details.
- SLPs are based on context-free grammars and are apt for concise and clean mathematical proofs.
- For many algorithmic problems, SLPs allow efficient algorithms that avoid prior decompression. The most studied example in this context is the pattern matching problem for compressed strings, see the references in [13].

Due to these appealing properties, it is natural to generalize SLPs to other more complex data structures. For trees, this is done in [3,11]. In this context, a tree is represented by a *context-free tree grammar* that generates exactly one tree. Such grammars are called straight-line context-free tree grammars, briefly SLCF tree grammars in [3,11]. They generalize the sharing of repeated subtrees in a tree as it is well-known from DAGs (directed acyclic graphs) to the sharing of repeated subpatterns in a tree (a subpattern is a connected subgraph of the tree). In the context of commonly used XML documents, experiments show that SLCF tree grammars can give approximately 2-3 times higher compression ratios [3] than DAGs [2] when compressing document tree structures. Since sharing of patterns in an SLCF tree grammar can occur along the paths of a tree, it is possible to have a grammar of size $O(n)$[1] that generates a tree of height 2^n; this is not possible with a DAG (the DAG has the same height as its represented tree). More dramatically, an SLCF tree grammar of size $O(n)$ can even generate a full binary tree of height 2^n, which has 2^{2^n} many nodes. Hence, double exponential compression rates can be achieved.

The downside of such extreme compression capabilities is that arbitrary SLCF tree grammars do not inherit some of the nice algorithmic properties of (string) SLPs. For instance, whereas evaluating a given automaton on an SLP representation of a string can be done in polynomial time [13], this problem becomes PSPACE-complete for tree automata and SLCF tree grammars [11]. This motivates the investigation of restricted classes of SLCF tree grammars. Linearity is one of these restrictions: a context-free tree grammar is *linear* if every context parameter occurs at most once in every right-hand side. Note that our tree compression algorithm BPLEX [3] generates a small *linear* SLCF tree grammar for a given input tree. It can be checked in polynomial time whether two linear SLCF tree grammars generate the same tree [3,14]. This result generalizes a corresponding result for (string) SLPs of Plandowski [12]. It remains open whether polynomial time equality testing is also possible for non-linear SLCF tree grammars.

Another useful restriction on SLCF tree grammars is *k-boundedness* (for some fixed k): a context-free tree grammar is k-bounded if every nonterminal uses at most k context parameters; 1-bounded grammars are also called *monadic*. In this paper we study the impact of the various restrictions on SLCF tree grammars with respect to compression. Our main result is the following: a given linear SLCF tree grammar can be transformed in polynomial time into an equivalent linear and monadic SLCF tree grammar (Theorem 7). In other words, for the purpose of compression by linear grammars, one parameter is already enough; the corresponding linear monadic grammars offer the same kind of compression as linear SLCF tree grammars. Linear monadic SLCF tree grammars are also used in [9,10,14], where they are called *singleton tree grammars*. We present two algorithmic applications of Theorem 7: it can be tested in polynomial time whether a given tree automaton accepts the tree given by a linear SLCF tree grammar (Corollary 9). This solves our main open problem from [11], where we could only present a polynomial time algorithm for linear k-bounded SLCF tree grammars (when k is a fixed constant). Our second application generalizes Corollary 9 to tree automata with equality and disequality constraints between sibling nodes [1,4]. These are bottom-up tree automata which can test whether the subtrees rooted at children of

[1] The size of a grammar is defined as the sum of the sizes of all right-hand sides of productions.

the current node are equal or not equal. Their recognized languages are closed under Boolean operations and are strictly more general than regular tree languages (for a recent generalization see [5]). The running time of this second polynomial time algorithm is much worse than the running time stated in Corollary 9 for ordinary tree automata; therefore we state the two results separately.

In Section 7 we show that Theorem 7 does not extend to larger classes of grammars. First, we consider nondeterministic linear SLCF tree grammars, i.e., acyclic grammars (no recursion) which may have several productions for each nonterminal. Such grammars represent finite sets of trees. We give an example of a linear and n-bounded nondeterministic SLCF tree grammar for which every equivalent k-bounded such grammar ($k < n$) must be exponentially larger. Using a straightforward extension of our proof of Theorem 7, we show that this exponential blow-up is also the worst case. Next, we consider non-linear SLCF tree grammars. We present an example of a non-linear n-bounded SLCF tree grammar of size $O(n)$ for which every equivalent k-bounded SLCF tree grammar ($k < n$) has size at least 2^{n-k}.

A full version of this paper including all proofs will appear.

2 Trees and SLCF Tree Grammars

A *ranked alphabet* is a pair $(\mathbb{F}, \mathrm{rank})$, where $\mathbb{F}$ is a finite set of function symbols and $\mathrm{rank} : \mathbb{F} \to \mathbb{N}$ assigns to each $\alpha \in \mathbb{F}$ its rank. Let $\mathbb{F}_i = \{\alpha \in \mathbb{F} \mid \mathrm{rank}(\alpha) = i\}$ and $\mathbb{F}_{\geq i} = \bigcup_{j \geq i} \mathbb{F}_j$. Symbols in $\mathbb{F}_0$ are called *constants*. We fix a ranked alphabet $(\mathbb{F}, \mathrm{rank})$ in the following. An $\mathbb{F}$*-labeled ordered tree* t (or *ground term* over $\mathbb{F}$) is a pair $t = (\mathrm{dom}_t, \lambda_t)$, where (i) $\mathrm{dom}_t \subseteq \mathbb{N}^*$ is finite, (ii) $\lambda_t : \mathrm{dom}_t \to \mathbb{F}$, (iii) if $w = vv' \in \mathrm{dom}_t$, then also $v \in \mathrm{dom}_t$, and (iv) if $v \in \mathrm{dom}_t$ and $\lambda_t(v) \in \mathbb{F}_n$, then $vi \in \mathrm{dom}_t$ if and only if $1 \leq i \leq n$. The edge relation of t is implicitly given as $\{(v, vi) \in \mathrm{dom}_t \times \mathrm{dom}_t \mid v \in \mathbb{N}^*, i \in \mathbb{N}\}$. The size of t is $|t| = |\mathrm{dom}_t|$. We identify an $\mathbb{F}$-labeled tree t with a term in the usual way: if $\lambda_t(\varepsilon) = \alpha \in \mathbb{F}_i$, then this term is $\alpha(t_1, \ldots, t_i)$, where t_j is the term associated with the subtree of t rooted at node j. The set of all $\mathbb{F}$-labeled trees is $T(\mathbb{F})$. Let us fix a countable set $\mathbb{Y} = \{y_1, y_2, \ldots\}$ of *(formal context-) parameters* (below we also use a distinguished parameter $z \notin \mathbb{Y}$). The set of all $\mathbb{F}$-labeled trees with parameters from $Y \subseteq \mathbb{Y}$ is $T(\mathbb{F}, Y)$. Formally, we consider parameters as new constants and define $T(\mathbb{F}, Y) = T(\mathbb{F} \cup Y)$. The tree $t \in T(\mathbb{F}, Y)$ is *linear*, if every parameter $y \in Y$ occurs at most once in t. For trees $t \in T(\mathbb{F}, \{y_1, \ldots, y_n\}), t_1, \ldots, t_n \in T(\mathbb{F}, Y)$, by $t[y_1/t_1 \cdots y_n/t_n]$ we denote the tree that is obtained by replacing in t every y_i-labeled leaf with t_i ($1 \leq i \leq n$). A *context* is a tree $C \in T(\mathbb{F}, \mathbb{Y} \cup \{z\})$, in which the distinguished parameter z appears exactly once. Instead of $C[z/t]$ we write briefly $C[t]$. When talking about algorithms on trees, we assume the RAM model of computation, and we assume that trees are given by the standard pointer representation.

For further consideration, let us fix a countable infinite set $\mathbb{N}_i$ of symbols of rank i with $\mathbb{F}_i \cap \mathbb{N}_i = \emptyset$. Hence, every finite subset $N \subseteq \bigcup_{i \geq 0} \mathbb{N}_i$ is a ranked alphabet. A *context-free tree grammar* (*over* $\mathbb{F}$) is a triple $\mathcal{G} = (N, P, S)$, where (i) $N \subseteq \bigcup_{i \geq 0} \mathbb{N}_i$ is a finite set of *nonterminals*, (ii) P (the set of *productions*) is a finite set of pairs of the form $(A \to t)$, where $A \in N$ and $t \in T(\mathbb{F} \cup N, \{y_1, \ldots, y_{\mathrm{rank}(A)}\})$, and (iii)

$S \in N \cap \mathbb{N}_0$ is the *start nonterminal* of rank 0. We assume that every nonterminal $B \in N \setminus \{S\}$ as well as every terminal symbol from $\mathbb{F}$ occurs in the right-hand side t of some production $(A \to t) \in P$. For a production $(A \to t) \in P$ with $A \in N \cap \mathbb{N}_n$, we also write $A(y_1 \ldots, y_n) \to t(y_1, \ldots, y_n)$ in order to emphasize that $\mathrm{rank}(A) = n$. The *size* $|\mathcal{G}|$ of $\mathcal{G}$ is $|\mathcal{G}| = \sum_{(A \to t) \in P} |t|$. Let us define the derivation relation $\Rightarrow_{\mathcal{G}}$ on $T(\mathbb{F} \cup N, \mathbb{Y})$ as follows: $s \Rightarrow_{\mathcal{G}} s'$ if there exist a production $(A \to t) \in P$ with $\mathrm{rank}(A) = n$, a context $C \in T(\mathbb{F} \cup N, \mathbb{Y} \cup \{z\})$, and trees $t_1, \ldots, t_n \in T(\mathbb{F} \cup N, \mathbb{Y})$ such that $s = C[A(t_1, \ldots, t_n)]$ and $s' = C[t[y_1/t_1 \cdots y_n/t_n]]$. Let $L(\mathcal{G}) = \{t \in T(\mathbb{F}) \mid S \Rightarrow_{\mathcal{G}}^* t\} \subseteq T(\mathbb{F})$. We consider several subclasses of context-free tree grammars:

- $\mathcal{G}$ is *linear*, if for every production $(A \to t) \in P$ the term t is linear.
- $\mathcal{G}$ is *non-erasing*, if $t \notin \mathbb{Y}$ for every production $(A \to t) \in P$.
- $\mathcal{G}$ is *non-deleting*, if for every production $(A \to t) \in P$, each of the parameters $y_1, \ldots, y_{\mathrm{rank}(A)}$ appears in t.
- $\mathcal{G}$ is *productive*, if it is non-erasing and non-deleting.
- $\mathcal{G}$ is *k-bounded* (for $k \in \mathbb{N}$), if $\mathrm{rank}(A) \leq k$ for every $A \in N$.
- $\mathcal{G}$ is *monadic* if it is 1-bounded.

Finally, a *straight-line context-free tree grammar* (*SLCF tree grammar*) is a context-free tree grammar $\mathcal{G} = (N, P, S)$, where (i) for every $A \in N$ there is *exactly one* production $(A \to t_A) \in P$ with left-hand side A and (ii) the relation $\{(A, B) \in N \times N \mid B \text{ occurs in } t_A\}$ is acyclic; we call the reflexive transitive closure of this relation the *hierarchical order* of $\mathcal{G}$. Conditions (i) and (ii) ensure that $L(\mathcal{G})$ contains exactly one tree, which we denote with $\mathrm{val}(\mathcal{G})$. Alternatively, for every term $t \in T(\mathbb{F} \cup N, \{y_1, \ldots, y_n\})$ we can define a term $\mathrm{val}_{\mathcal{G}}(t) \in T(\mathbb{F}, \{y_1, \ldots, y_n\})$ by induction on the hierarchical order as follows, where $1 \leq i \leq n$, $f \in \mathbb{F}_m$, and $A \in N \cap \mathbb{N}_m$:

- $\mathrm{val}_{\mathcal{G}}(y_i) = y_i$
- $\mathrm{val}_{\mathcal{G}}(f(t_1, \ldots, t_m)) = f(\mathrm{val}_{\mathcal{G}}(t_1), \ldots, \mathrm{val}_{\mathcal{G}}(t_m))$
- $\mathrm{val}_{\mathcal{G}}(A(t_1, \ldots, t_m)) = \mathrm{val}_{\mathcal{G}}(t_A)[y_1/\mathrm{val}_{\mathcal{G}}(t_1) \cdots y_m/\mathrm{val}_{\mathcal{G}}(t_m)]$

Finally, let $\mathrm{val}_{\mathcal{G}}(A) = \mathrm{val}_{\mathcal{G}}(A(y_1, \ldots, y_{\mathrm{rank}(A)}))$ and $\mathrm{val}(\mathcal{G}) = \mathrm{val}_{\mathcal{G}}(S)$. SLCF tree grammars generalize *string* generating straight-line programs [13] in a natural way to trees. The following example shows that SLCF tree grammars may lead to doubly exponential compression ratios; thus, they can be exponentially more succinct than DAGs.

Example 1. Let the (non-linear) monadic SLCF tree grammar $\mathcal{G}_n$ consist of the productions $S \to A_0(a)$, $A_i(y_1) \to A_{i+1}(A_{i+1}(y_1))$ for $0 \leq i < n$, and $A_n(y_1) \to f(y_1, y_1)$. Then $\mathrm{val}(\mathcal{G}_n)$ is a complete binary tree of height $2^n + 1$. Thus, $|\mathrm{val}(\mathcal{G}_n)| = 2 \cdot 2^{2^n} - 1$.

On the other hand, it is not difficult to show that for a *linear* SLCF tree grammar $\mathcal{G}$ one has $|\mathrm{val}(\mathcal{G})| \leq 2^{O(|\mathcal{G}|)}$. Thus, linear SLCF tree grammars have at most exponential compression ratios, just like DAGs, which can be seen as 0-bounded SLCF tree grammars. But even linear SLCF tree grammars can be exponentially more succinct than DAGs: the linear SLCF tree grammar $\mathcal{G}_n$ with the productions $S \to A_0(a)$, $A_i(y_1) \to A_{i+1}(A_{i+1}(y_1))$ for $0 \leq i < n$, and $A_n(y_1) \to f(y_1)$ generates a monadic tree of height $2^n + 1$. The minimal DAG for this tree is the tree itself and thus has size $2^n + 1$. The following result was shown in [3].

Theorem 2. *There exists a polynomial time algorithm that tests for two given linear SLCF tree grammars $\mathcal{G}$ and $\mathcal{H}$, whether* $\mathrm{val}(\mathcal{G}) = \mathrm{val}(\mathcal{H})$.

It is open whether Theorem 2 can be generalized to non-linear SLCF tree grammars. In [3] we could only prove a PSPACE upper bound for the equality problem for non-linear SLCF tree grammars.

3 Tree Automata

Let $\mathbb{F}$ be a ranked alphabet. A *nondeterministic tree automaton* (*over* $\mathbb{F}$), NTA for short, is a tuple $\mathcal{A} = (Q, \Delta, F)$, where (i) Q is a finite set of *states*, (ii) $F \subseteq Q$ is the set of *final states*, and (iii) Δ is a set of *transitions* of the form $(q_1, \dots, q_{\mathrm{rank}(f)}, f, q)$, where $f \in \mathbb{F}$ and $q_1, \dots, q_{\mathrm{rank}(f)}, q \in Q$. We define the mapping $\widetilde{\Delta} : T(\mathbb{F}) \to 2^Q$ inductively as follows, where $n \geq 0$, $f \in \mathbb{F}_n$, and $t_1, \dots, t_n \in T(\mathbb{F})$:

$$\widetilde{\Delta}(f(t_1, \dots, t_n)) = \{q \in Q \mid \exists (q_1, \dots, q_n, f, q) \in \Delta : q_1 \in \widetilde{\Delta}(t_1), \dots, q_n \in \widetilde{\Delta}(t_n)\}$$

The *language* defined by $\mathcal{A}$ is $L(\mathcal{A}) = \{t \in T(\mathbb{F}) \mid \widetilde{\Delta}(t) \cap F \neq \emptyset\}$. The *size* of the NTA $\mathcal{A} = (Q, \Delta, F)$ is defined as $|\mathcal{A}| = \sum_{(q_1,\dots,q_n,f,q)\in\Delta}(n \cdot \log|Q| + \log|\mathbb{F}|)$.

A *nondeterministic tree automaton with sibling-constraints* (*over* $\mathbb{F}$), NTAC for short, is a tuple $\mathcal{A} = (Q, \Delta, F)$, where Q and F are as for NTAs and Δ is a set of *transitions* of the form $(E, D, q_1, \dots, q_{\mathrm{rank}(f)}, f, q)$, where $E, D \subseteq \{1, \dots, \mathrm{rank}(f)\}^2$ are disjoint relations such that D is irreflexive, $f \in \mathbb{F}$, and $q_1, \dots, q_{\mathrm{rank}(f)}, q \in Q$. The relation E (resp. D) is a set of *equality* (resp. *disequality*) *constraints between siblings*. We define the mapping $\widetilde{\Delta} : T(\mathbb{F}) \to 2^Q$ inductively as follows, where $n \geq 0$, $f \in \mathbb{F}_n$, and $t_1, \dots, t_n \in T(\mathbb{F})$:

$$\begin{aligned}\widetilde{\Delta}(f(t_1, \dots, t_n)) = \{q \in Q \mid \exists (E, D, q_1, \dots, q_n, f, q) \in \Delta :\\ q_1 \in \widetilde{\Delta}(t_1), \dots, q_n \in \widetilde{\Delta}(t_n), \forall (i,j) \in E : t_i = t_j, \forall (i,j) \in D : t_i \neq t_j\}\end{aligned}$$

Again, the language defined by $\mathcal{A}$ is $L(\mathcal{A}) = \{t \in T(\mathbb{F}) \mid \widetilde{\Delta}(t) \cap F \neq \emptyset\}$. The size of the NTAC $\mathcal{A}$ is $|\mathcal{A}| = \sum_{(E,D,q_1,\dots,q_n,f)\in\Delta}(n^2 + n \cdot \log|Q| + \log|\mathbb{F}|)$.

4 Normal Forms for Linear SLCF Tree Grammars

In this section, we only deal with *linear* SLCF tree grammars. It is easy to see that a linear SLCF tree grammar $\mathcal{G} = (N, P, S)$ can be transformed in linear time into an equivalent linear and *non-deleting* SLCF tree grammar: if for a production $A \to t_A$ (with $\mathrm{rank}(A) = n$) the parameters $y_{i_1}, \dots, y_{i_k} \in \{y_1, \dots, y_n\}$ do not occur in t_A, then we can reduce the rank of A to $n - k$. Moreover, if A occurs in a right-hand side t_B at position $v \in \mathrm{dom}_{t_B}$, then we remove from t_B the subtrees rooted at positions $vi_1, \dots, vi_k$. We now produce an equivalent non-deleting grammar in one pass through $\mathcal{G}$: starting from the leaves of the hierarchical order of $\mathcal{G}$, we reduce the rank of each nonterminal A and store with it the indices of removed parameters (so that in later occurrences of A we know which subtrees to remove). Note that the size of the new grammar is at most $|\mathcal{G}|$.

Now, let $\mathcal{G}$ be a linear and non-deleting SLCF tree grammar. Again it is easy to see that $\mathcal{G}$ can be transformed in linear time into an equivalent linear and *productive* SLCF tree grammar: we remove each production with right hand side y_1, and apply the removed productions in all remaining right-hand sides. As before, this can be done in one pass through the grammar $\mathcal{G}$, and the resulting grammar has size at most $|\mathcal{G}|$.

A linear SLCF tree grammar $\mathcal{G} = (N, P, S)$ is in *Chomsky normal form* (CNF) if it is productive, and for every production $(A \to t_A) \in P$ with $\mathrm{rank}(A) = n$, the term t_A has one of the following two forms:

(a) $f(y_1, \dots, y_n)$ with $f \in \mathbb{F}_n$
(b) $B(y_1, \dots, y_{i-1}, C(y_i, \dots, y_{j-1}), y_j, \dots, y_n)$ with $B, C \in N$, $1 \le i \le j \le n+1$.

The proof of the following proposition is a straightforward extension of the corresponding construction for context-free string grammars.

Proposition 3. *Let $\mathcal{G} = (N, P, S)$ be a linear and productive SLCF tree grammar over $\mathbb{F}$ and let r be the maximal rank in $N \cup \mathbb{F}$. We can construct in time $O(r \cdot |\mathcal{G}|)$ a linear SLCF tree grammar $\mathcal{G}' = (N', P', S)$ in CNF such that $N' \supseteq N$, $|N'| \le 2 \cdot |\mathcal{G}|$, $\mathcal{G}'$ is k'-bounded, $k' \le 2r - 1$, and $\mathrm{val}_{\mathcal{G}'}(A) = \mathrm{val}_{\mathcal{G}}(A)$ for all $A \in N$.*

For macro grammars, a normal form similar to CNF exists (called IO standard form in [7, Definition 3.1.7]), where the nonterminal C in the second type (b) can even be assumed to be the first argument of B (for us this does not work, because in CNF the parameters have to occur in the order $y_1, \dots, y_{\mathrm{rank}(A)}$ in the right-hand side for A). Macro grammars are similar to context-free tree grammars except that they generate strings. Since in an SLCF tree grammar, every nonterminal has exactly one production, it is not difficult to see that the derivation order (IO or OI, see e.g. [4] for a definition) does not matter for SLCF tree grammars. It is also known that for arbitrary linear and non-deleting context-free tree grammars the derivation order again does not matter [8].

Example 4. Consider the linear and productive SLCF tree grammar with the two productions $S \to X(X(a, b), X(b, a))$ and $X(y_1, y_2) \to h(i(y_1), i(y_2))$. An equivalent linear SLCF tree grammar in CNF consists of the following productions:

$$\begin{aligned}
S &\to X_0(X_1) & X(y_1, y_2) &\to Y(I(y_1), y_2)\\
X_0(y_1) &\to X(y_1, X_2) & Y(y_1, y_2) &\to H(y_1, I(y_2))\\
X_1 &\to X_3(A) & A &\to a\\
X_2 &\to X_4(B) & B &\to b\\
X_3(y_1) &\to X(y_1, B) & I(y_1) &\to i(y_1)\\
X_4(y_1) &\to X(y_1, A) & H(y_1, y_2) &\to h(y_1, y_2).
\end{aligned}$$

Linear SLCF tree grammars in CNF can be stored more efficiently than ordinary SLCF tree grammars: if we know the rank of each (non)terminal, then for a right-hand side $B(y_1, \dots, y_i, C(y_{i+1}, \dots, y_j), y_{j+1}, \dots, y_m)$ (resp. $f(y_1, \dots, y_n)$) we only need to store the triple (B, C, i) (resp. the symbol f) which has size $O(\log k)$ if the grammar is k-bounded. We call this new representation of a CNF grammar its *triple notation*. From a given linear SLCF tree grammar $\mathcal{G}$, we can construct an equivalent linear SLCF tree grammar in CNF in time $O(r \cdot |\mathcal{G}|)$ (where r is again the maximal rank of (non)terminals) which needs only space $O(\log(r) \cdot |\mathcal{G}|)$ in triple notation.

5 Parameter Reduction in Linear SLCF Tree Grammars

In this section our main result is proved. We show that a given linear SLCF tree grammar can be made monadic in polynomial time.

A *skeleton tree* of rank $n \geq 0$ is a linear tree $s \in T(\mathbb{N}_0 \cup \mathbb{N}_1 \cup \mathbb{F}_{\geq 2}, \{y_1, \dots, y_n\})$, such that every parameter y_i $(1 \leq i \leq n)$ occurs in s and the following additional properties are satisfied.

(a) The tree s does not contain a subtree of the form $X(Y(t))$ for $X, Y \in \mathbb{N}_1$.
(b) For every subtree $f(t_1, \dots, t_m)$ of s with $f \in \mathbb{F}_{\geq 2}$ there exist at least two distinct $i \in \{1, \dots, m\}$ such that t_i contains a parameter from $\{y_1, \dots, y_n\}$.

In our construction, a skeleton tree will store the branching structure (with respect to those leaf nodes that are parameters) of the tree generated by a certain nonterminal, i.e., the information on how the paths from the root to parameters branch. Nonterminals of rank 1 in a skeleton tree represent those tree parts that are in between two branching nodes in this branching structure. The crucial point about skeleton trees is that their size can be bounded polynomially. For the following lemma, it is important that a skeleton tree only contains function symbols of rank ≥ 2.

Lemma 5. *Let r be the maximal rank of a symbol from $\mathbb{F}$. A skeleton tree s of rank $n \geq 1$ contains at most $2(r \cdot n - r + 1)$ many nodes.*

Let $\mathcal{G} = (N, P, S)$ be a linear SLCF tree grammar. By Proposition 3 we may assume that $\mathcal{G}$ is in CNF. The set of nonterminals N is a finite subset of $\bigcup_{i \geq 0} \mathbb{N}_i$. We now define in a bottom-up process, for every nonterminal A of rank $n \geq 1$, a skeleton tree sk_A of rank n. Simultaneously, we construct a new linear and monadic SLCF tree grammar $\mathcal{G}' = (N', P', S)$. Consider a production $A \to t_A$ from P and let $n = \mathrm{rank}(A)$.

Case 1. $t_A = f(y_1, \dots, y_n)$, where $f \in \mathbb{F}_n$: if $n \leq 1$, then we add the production $A(y_1, \dots, y_n) \to t_A$ to P' and set $\mathrm{sk}_A = A(y_1, \dots, y_n)$. If $n \geq 2$, then we set $\mathrm{sk}_A = t_A$ and do not add any new productions to P'.

Case 2. $t_A = B(y_1, \dots, y_{i-1}, C(y_i, \dots, y_{j-1}), y_j, \dots, y_n)$, where $i \leq j$ and the trees sk_B, sk_C are already constructed. In a first step we define the tree

$$s = \mathrm{sk}_B[y_i/\mathrm{sk}_C[y_1/y_i, y_2/y_{i+1}, \dots, y_{j-i}/y_{j-1}], \\ y_{i+1}/y_j, y_{i+2}/y_{j+1}, \dots, y_{n+i-j+1}/y_n]. \quad (1)$$

But this tree is not necessarily a skeleton tree; it may locally violate the conditions (a) and (b) on skeleton trees. Hence, we apply a contract-operation to s which yields the

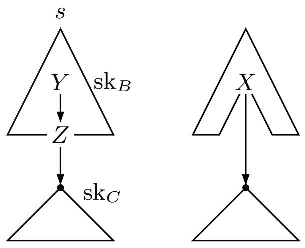


Fig. 1. Contract-1

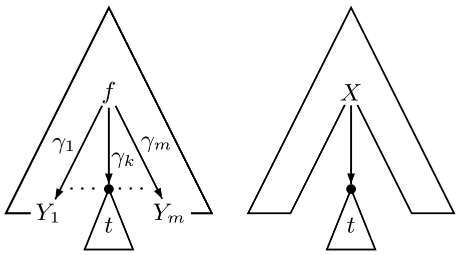


Fig. 2. Contract-2

skeleton tree sk_A. Moreover, as a side effect, the contract-operation adds new productions and nonterminals to $\mathcal{G}'$. The contract-operation works in two steps:

Contract-1. Assume that s contains a subtree of the form $Y(Z(t))$. There can be only one subtree of this form in s, see the left tree in Figure 1. We now do the following:

1. Add a fresh nonterminal $X \in \mathbb{N}_1$ of rank 1 to N'.
2. Add the production $X(y_1) \to Y(Z(y_1))$ to P'.
3. Replace the subtree $Y(Z(t))$ by $X(t)$.

Contract-2. After contract-1, s can only violate condition (b) for skeleton trees. Hence, assume that s contains a subtree of the form $f(t_1, \ldots, t_m)$ such that $f \in \mathbb{F}_{\geq 2}$ and there is exactly one $k \in \{1, \ldots, m\}$ such that t_k contains a parameter from $\{y_1, \ldots, y_n\}$, say y_p. Again there can be only one subtree of this form in s. Moreover, this case may only occur, if C has rank 0. In the following consideration, it is useful to set $\varepsilon(t) = t$ for an arbitrary term. Hence, ε is just the identity function on all terms.

Since condition (a) is already satisfied, every subtree t_ℓ ($\ell \neq k$) is of the form $\gamma_\ell(Y_\ell)$ with $Y_\ell \in \mathbb{N}_0$ and $\gamma_\ell \in \{\varepsilon\} \cup \mathbb{N}_1$, whereas t_k can be written as $\gamma_k(t)$, where $\gamma_k \in \{\varepsilon\} \cup \mathbb{N}_1$ and t is a tree that does not start with a non-terminal of rank 1. We now do the following:

1. Add a fresh nonterminal $X \in \mathbb{N}_1$ of rank 1 to N'.
2. Add to P' the production
$$X(y_1) \to f(\gamma_1(Y_1), \ldots, \gamma_{k-1}(Y_{k-1}), \gamma_k(y_1), \gamma_{k+1}(Y_{k+1}), \ldots, \gamma_m(Y_m)).$$
3. Replace the subtree
$$f(\gamma_1(Y_1), \ldots, \gamma_{k-1}(Y_{k-1}), \gamma_k(t), \gamma_{k+1}(Y_{k+1}), \ldots, \gamma_m(Y_m))$$
of s by $X(t)$.

After this operation, another contract-1 operation might be necessary (if the new subtree $X(t)$ is below an $\mathbb{N}_1$-labeled node). The resulting tree is the skeleton tree sk_A.

Note that the SLCF tree grammar $\mathcal{G}'$ is linear, productive, and monadic. The following lemma can be shown by induction on the hierarchical order of $\mathcal{G}$.

Lemma 6. *For every nonterminal A of $\mathcal{G}$ we have* $\mathrm{val}_{\mathcal{G}}(A) = \mathrm{val}_{\mathcal{G}'}(\mathrm{sk}_A)$.

Theorem 7. *Let r be the maximal rank of a symbol from $\mathbb{F}$. From a given linear and k-bounded SLCF tree grammar $\mathcal{G} = (N, P, S)$ we can construct in time $O(k \cdot r \cdot |\mathcal{G}|)$ a linear, productive, and monadic SLCF tree grammar $\mathcal{G}' = (N', P', S)$ of size $O(r \cdot |\mathcal{G}|)$ such that $N \cap (\mathbb{N}_0 \cup \mathbb{N}_1) \subseteq N'$ and $\mathrm{val}_{\mathcal{G}'}(A) = \mathrm{val}_{\mathcal{G}}(A)$ for every $A \in N \cap (\mathbb{N}_0 \cup \mathbb{N}_1)$.*

Proof. Using the constructions from Section 4, we first transform $\mathcal{G}$ into a linear CNF grammar $\mathcal{H}$ with $O(|\mathcal{G}|)$ many nonterminals. This needs time $O(\max\{k, r\} \cdot |\mathcal{G}|)$. Now we construct for every nonterminal A of $\mathcal{H}$ the skeleton tree sk_A and simultaneously the linear and monadic SLCF tree grammar $\mathcal{H}'$. In order to construct the tree s in Equation (1), we have to copy the already constructed skeleton trees sk_B and sk_C (since we may need these trees in later steps), which by Lemma 5 needs time $O(k \cdot r)$. The construction of sk_A from s needs at most three contraction steps, each of which requires $O(1)$ many pointer operations. Moreover, in every contraction step we add to $\mathcal{H}'$ a production of size at most $O(r)$. Hence, the total size of $\mathcal{H}'$ is $O(r \cdot |\mathcal{G}|)$ and the construction takes time $O(k \cdot r \cdot |\mathcal{G}|)$. We obtain the final grammar $\mathcal{G}'$ by adding to $\mathcal{H}'$ every nonterminal $A \in N \cap (\mathbb{N}_0 \cup \mathbb{N}_1)$, which does not already belong to $\mathcal{H}'$, together with the production $A \to \mathrm{sk}_A$. By Lemma 6 we have $\mathrm{val}_{\mathcal{G}'}(A) = \mathrm{val}_{\mathcal{G}}(A)$. Note that in general $\mathcal{G}'$ is not in CNF, and that it might contain useless productions. □

Finite unions of linear monadic SLCF tree grammars are studied e.g. in [10] under the name *singleton tree grammar* (STG). They are, by Theorem 7, polynomially equivalent to finite unions of linear SLCF grammars and hence their results can be applied for linear grammars.

Example 8. We transform the linear CNF grammar constructed in Example 4 into an equivalent linear monadic SLCF tree grammar. We start with the set of productions $P' = \{A \to a, B \to b, I(y_1) \to i(y_1)\}$ (see case 1) and the following skeleton trees:

$$\mathrm{sk}_A = A, \qquad \mathrm{sk}_B = B, \qquad sk_I = I(y_1), \qquad sk_H = h(y_1, y_2).$$

Next, for X and Y we obtain without contract operations:

$$\mathrm{sk}_Y = h(y_1, I(y_2)), \qquad \mathrm{sk}_X = h(I(y_1), I(y_2))$$

Let us now construct sk_{X_4}, sk_{X_3}, sk_{X_2}, sk_{X_1}, sk_{X_0}, and sk_S in this order:

- construction of sk_{X_4}: For the tree s in (1) we obtain $s = h(I(y_1), I(A))$. With contract-2, we obtain the new production $C(y_1) \to h(I(y_1), I(A))$ and the skeleton tree $\mathrm{sk}_{X_4} = C(y_1)$.
- Construction of sk_{X_3}: we get $s = h(I(y_1), I(B))$. With contract-2, we obtain the new production $D(y_1) \to h(I(y_1), I(B))$ and the skeleton tree $\mathrm{sk}_{X_3} = D(y_1)$.
- Construction of sk_{X_2}: we get $s = C(B)$. Thus, we do not add a new production to P' and set $\mathrm{sk}_{X_2} = C(B)$.
- Construction of sk_{X_1}: we get $s = D(A)$. Again, we do not add a new production to P' and set $\mathrm{sk}_{X_1} = D(A)$.
- Construction of sk_{X_0}: we get $s = h(I(y_1), I(C(B)))$. A first contract-1 operation adds the production $E(y_1) \to I(C(y_1))$ to P' and updates s to $s = h(I(y_1), E(B))$. Now, we have to apply another contract-2 operation, which adds the production $F(y_1) \to h(I(y_1), E(B))$ to P'. We set $\mathrm{sk}_{X_0} = F(y_1)$.

– Construction of sk_S. We set $s = F(D(A))$. Hence, we add to P' the production $G(y_1) \to F(D(y_1))$ and set $\mathrm{sk}_S = G(A)$.

Thus, an equivalent linear and monadic SLCF tree grammar contains the following productions:

$$\begin{array}{lll} S \to G(A) & C(y_1) \to h(I(y_1), I(A)) & F(y_1) \to h(I(y_1), E(B)) \\ A \to a & D(y_1) \to h(I(y_1), I(B)) & G(y_1) \to F(D(y_1)) \\ B \to b & E(y_1) \to I(C(y_1)) & I(y_1) \to i(y_1) \end{array}$$

6 Applications to Tree Automata Evaluation

In [11], we have shown how to check for (i) a given NTA $\mathcal{A}$ with n states and (ii) a given linear and k-bounded SLCF tree grammar $\mathcal{G}$ in time $O(|\mathcal{G}| \cdot |\mathcal{A}| \cdot n^{k+1})$, whether $\mathrm{val}(\mathcal{G}) \in L(\mathcal{A})$. If the automaton is a deterministic bottom-up tree automaton then time $O(|\mathcal{G}| \cdot |\mathcal{A}| \cdot n^k)$ suffices. Together with Theorem 7 we obtain the following.

Corollary 9. *For a given NTA $\mathcal{A}$ with n states and a given linear and k-bounded SLCF tree grammar $\mathcal{G}$ such that r is the maximal rank of a terminal symbol from $\mathbb{F}$, we can check in time $O(r \cdot |\mathcal{G}| \cdot (k + |\mathcal{A}| \cdot n^2))$, whether* $\mathrm{val}(\mathcal{G}) \in L(\mathcal{A})$.

We may assume that $r, k \leq |\mathcal{G}|$ in Corollary 9, since we assume for context-free tree grammars that every (non)terminal occurs in a right-hand side. Moreover, we can eliminate states from an NTA that do not occur in transition tuples. Hence, $n \leq |\mathcal{A}|$. Thus, the time bound in Corollary 9 can be replaced by $O(|\mathcal{G}|^3 + |\mathcal{G}|^2 \cdot |\mathcal{A}|^3)$. Hence, $\mathrm{val}(\mathcal{G}) \in L(\mathcal{A})$ can be checked in polynomial time. In the rest of this section, we extend this result to tree automata with sibling-constraints.

Theorem 10. *The problem of checking* $\mathrm{val}(\mathcal{G}) \in L(\mathcal{A})$ *for a given linear SLCF tree grammar $\mathcal{G}$ and a given NTAC $\mathcal{A}$ can be solved in polynomial time.*

Proof. By Theorem 7 we can assume that $\mathcal{G} = (N, P, S)$ is linear and monadic. Moreover, we can assume that all productions in P are of one of the following 4 types:

– $A \to f(A_1, \ldots, A_n)$ for $A, A_1 \ldots, A_n \in \mathbb{N}_0$ and $f \in \mathbb{F}_n$
– $A \to B(C)$ for $A, C \in \mathbb{N}_0$ and $B \in \mathbb{N}_1$
– $A(y) \to f(A_1, \ldots, A_{i-1}, y, A_i, \ldots, A_n)$ for $A \in \mathbb{N}_1$, $A_1, \ldots, A_n \in \mathbb{N}_0$, $f \in \mathbb{F}_{n+1}$
– $A(y) \to B(C(y))$ for $A, B, C \in \mathbb{N}_1$

Let $\mathcal{A} = (Q, \Delta, F)$ be an NTAC. We will compute for every $A \in \mathbb{N}_0 \cap N$ the set of states $\widetilde{\Delta}(\mathrm{val}_{\mathcal{G}}(A))$. Consider such a nonterminal $A \in \mathbb{N}_0 \cap N$.

Case 1. The production for A is of the form $A \to f(A_1, \ldots, A_n)$. Assume that for every $1 \leq i \leq n$, the set of states $\widetilde{\Delta}(\mathrm{val}_{\mathcal{G}}(A_i))$ is already computed. Using Theorem 2, we can find out in polynomial time which of the trees $\mathrm{val}_{\mathcal{G}}(A_i)$ $(1 \leq i \leq n)$ are equal or disequal. Using this information, it is straightforward to compute the set $\widetilde{\Delta}(\mathrm{val}_{\mathcal{G}}(A))$.

Case 2. The production for A is of the form $A \to B(C)$. This case requires more work. Assume that the set of states $\widetilde{\Delta}(\mathrm{val}_{\mathcal{G}}(C))$ is already computed. Define a straight-line context-free *string* grammar $\mathcal{G}_B$ as follows:

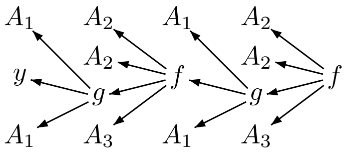
– The set of nonterminals is $\mathbb{N}_1 \cap N$, i.e., the nonterminals of $\mathcal{G}$ of rank 1.
– The set of terminal symbols is $\Sigma = \{[A_1, \ldots, A_{i-1}, y, A_i, \ldots, A_n, f] \mid f \in \mathbb{F}_{n+1}, A_1, \ldots, A_n \in \mathbb{N}_0 \cap N, 1 \le i \le n+1\}$.
– If $(X(y) \to Y(Z(y))) \in P$, then $\mathcal{G}_B$ contains the production $X \to ZY$; if $(X(y) \to f(A_1, \ldots, A_{i-1}, y, A_i, \ldots, A_n)) \in P$, then $\mathcal{G}_B$ contains the production $X \to [A_1 \ldots, A_{i-1}, y, A_i, \ldots, A_n, f]$. These are all productions of $\mathcal{G}_B$.
– The start nonterminal of $\mathcal{G}_B$ is B.

The string generated by $\mathcal{G}_B$ represents the outcome of a partial derivation from the nonterminal B in the tree grammar $\mathcal{G}$, where the derivation process is stopped as soon as a nonterminal of rank 0 is reached.

Example 11. Let $\mathcal{G}$ contain the following four productions for nonterminals of rank one: $B(y) \to B_1(B_1(y))$, $B_2(y) \to f(A_2, A_2, y, A_3)$, $B_1(y) \to B_2(B_3(y))$, $B_3(y) \to g(A_1, y, A_1)$. Here A_1, A_2, A_3 are nonterminals of rank 0. Then, the SLCF string grammar $\mathcal{G}_B$ consists of the productions $B \to B_1B_1$, $B_2 \to [A_2, A_2, y, A_3, f]$, $B_1 \to B_3B_2$, and $B_3 \to [A_1, y, A_1, g]$ and generates the string

$$\mathrm{val}(\mathcal{G}_B) = [A_1, y, A_1, g]\,[A_2, A_2, y, A_3, f]\,[A_1, y, A_1, g]\,[A_2, A_2, y, A_3, f].$$

This string represents the following tree:



For a nonterminal $X \in \mathbb{N}_0 \cap N$ of rank 0, let $s(X) = |\mathrm{val}_{\mathcal{G}}(X)|$ be the number of nodes of the generated tree; this number can be computed in polynomial time. For a terminal symbol $[A_1, \ldots, A_{i-1}, y, A_i, \ldots, A_n, f] \in \Sigma$ of the string grammar $\mathcal{G}_B$ let $s([A_1, \ldots, A_{i-1}, y, A_i, \ldots, A_n, f]) = 1 + s(A_1) + \cdots + s(A_n)$. The mapping $s : \Sigma \to \mathbb{N}$ is extended to Σ^* in the natural way: $s(a_1 \cdots a_n) = s(a_1) + \cdots + s(a_n)$ for $a_1, \ldots, a_n \in \Sigma$. Finally, for a position $0 \le p \le |\mathrm{val}(\mathcal{G}_B)|$ let $s(p) = s(C) + s(\mathrm{val}(\mathcal{G}_B)[:\, p])$, where $w[:\, k]$ is the prefix of length k of the string w. Also the value $s(p)$ can be computed for a given position p in polynomial time by first constructing in polynomial time an SLCF string grammar for the prefix $\mathrm{val}(\mathcal{G}_B)[:\, p]$. Then the number $s(\mathrm{val}(\mathcal{G}_B)[:\, p])$ can be easily computed bottom-up. The value $s(p)$ is the size of a certain subtree of $\mathrm{val}_{\mathcal{G}}(A) = \mathrm{val}_{\mathcal{G}}(B)[y/\mathrm{val}_{\mathcal{G}}(C)]$, namely the subtree that is obtained by going p steps up (towards the root) from the unique occurrence of y in $\mathrm{val}_{\mathcal{G}}(B)(y)$.

Let us next determine the set $N_{B,0} \subseteq \mathbb{N}_0 \cap N$ of all nonterminals of rank 0 that appear in terminal symbols of $\mathrm{val}(\mathcal{G}_B)$: If $X \to [A_1, \ldots, A_{i-1}, y, A_i, \ldots, A_n, f]$ is a production of $\mathcal{G}_B$, then set $N_{X,0} = \{A_1, \ldots, A_n\}$. If $X \to YZ$ is a production of $\mathcal{G}_B$, then set $N_{X,0} = N_{Y,0} \cup N_{Z,0}$. In this way we can compute the set $N_{B,0}$ in polynomial time. Let $\{k_1, k_2, \ldots, k_m\} = \{s(X) \mid X \in N_{B,0}\}$, where $k_1 < k_2 < \cdots < k_m$. Also this enumeration can be computed in polynomial time. We now compute a certain splitting of the string $\mathrm{val}(\mathcal{G}_B)$. More precisely, for every $1 \le i \le m$ we compute the largest position (i.e. highest position in the tree) $0 \le p_i \le |\mathrm{val}(\mathcal{G}_B)|$ such that

$s(p_i) \leq k_i$. This position p_i can be computed in polynomial time with binary search (using the fact that $s(p)$ can be computed in polynomial time for a given p).

Example 11 (continued). Assume that $s(C) = s(A_1) = 2$, $s(A_2) = 7$ and $s(A_3) = 9$. Then, we obtain $k_1 = 2$, $k_2 = 7$, and $k_3 = 9$, as well as $s(0) = 2$, $s(1) = 7$, $s(2) = 31$, $s(3) = 36$, and $s(4) = 60$. Thus, $p_1 = 0$, $p_2 = p_3 = 1$.

From the list $0 \leq p_1 \leq p_2 \leq \cdots \leq p_m \leq |\mathrm{val}(\mathcal{G}_B)|$, we remove every position p_i such that $s(p_i) \neq k_i$ or $p_i = |\mathrm{val}(\mathcal{G}_B)|$. Let $0 \leq p'_1 < p'_2 < \cdots < p'_\ell < |\mathrm{val}(\mathcal{G}_B)|$ be the resulting list. In our example, we only keep $p'_1 = 0$ and $p'_2 = 1$. This list defines our splitting of $\mathrm{val}(\mathcal{G}_B)$. More precisely, we compute in polynomial time the symbols $a_i = \mathrm{val}(\mathcal{G}_B)[p'_i + 1] \in \Sigma$ ($w[p]$ is the p-th symbol of the string w) and SLCF string grammars $\mathcal{G}_0, \ldots, \mathcal{G}_\ell$ such that

$$\mathrm{val}(\mathcal{G}_B) \;=\; \mathrm{val}(\mathcal{G}_0)\, a_1\, \mathrm{val}(\mathcal{G}_1)\, a_2 \cdots \mathrm{val}(\mathcal{G}_{\ell-1})\, a_\ell\, \mathrm{val}(\mathcal{G}_\ell). \tag{2}$$

Recall that every prefix of $\mathrm{val}(\mathcal{G}_B)$ represents a tree with a unique occurrence of the parameter y (if this prefix is the empty string then the tree is just y). For $0 \leq i \leq \ell$ let $t_i(y)$ be the tree represented by the prefix $\mathrm{val}(\mathcal{G}_0)\, a_1 \cdots \mathrm{val}(\mathcal{G}_{i-1})\, a_i$ (thus $t_0(y) = y$) and let $u_i(y)$ be the tree represented by the prefix $\mathrm{val}(\mathcal{G}_0)\, a_1 \cdots \mathrm{val}(\mathcal{G}_{i-1})\, a_i \mathrm{val}(\mathcal{G}_i)$. We compute the set of states $P_i = \widetilde{\Delta}(t_i[y/\mathrm{val}_\mathcal{G}(C)])$ and $Q_i = \widetilde{\Delta}(u_i[y/\mathrm{val}_\mathcal{G}(C)])$ successively. We start with $P_0 = \widetilde{\Delta}(\mathrm{val}_\mathcal{G}(C))$; recall that this set is already computed.

Computing the set P_i from Q_{i-1} ($i > 0$) is straightforward: assume that $a_i = [A_1, \ldots, A_{j-1}, y, A_j, \ldots, A_n, f]$. From (2) we can easily compute a monadic SLCF-tree grammar for the tree $u_{i-1}[y/\mathrm{val}_\mathcal{G}(C)]$. Hence, using Theorem 2, we can check in polynomial time, whether the tree $u_{i-1}[y/\mathrm{val}_\mathcal{G}(C)]$ equals some $\mathrm{val}_\mathcal{G}(A_j)$. Using this information, we can compute in polynomial time the set of states P_i from Q_{i-1}.

In order to compute Q_i from P_i, one has to note that when walking down from the root of $u_i(y)$ to the unique occurrence of y for $|\mathrm{val}(\mathcal{G}_i)|$ steps, then the current subtree is never equal to one of its sibling nodes. Hence, for every terminal symbol $a = [A_1, \ldots, A_{j-1}, y, A_{j+1}, \ldots, A_n, f]$ that occurs in the grammar $\mathcal{G}_i$ we can compute a transition mapping $\delta_a : Q \to 2^Q$ as follows, where $q \in Q$ (recall that the sets $\widetilde{\Delta}(\mathrm{val}_\mathcal{G}(A_k))$ for $k \in \{1, \ldots, n\} \setminus \{j\}$ are already computed):

$$\begin{aligned}
\delta_a(q) = \{q' \in Q \mid\ & \exists (E, D, q_1, \ldots, q_{j-1}, q, q_{j+1}, \ldots, q_n, f, q') \in \Delta : \\
& \forall k \in \{1, \ldots, n\} \setminus \{j\} : q_k \in \widetilde{\Delta}(\mathrm{val}_\mathcal{G}(A_k)), \\
& \forall (k, m) \in E : k = m \vee (k \neq j \neq m \wedge \mathrm{val}_\mathcal{G}(A_k) = \mathrm{val}_\mathcal{G}(A_m)), \\
& \forall (k, m) \in D : k = j \vee m = j \vee (k \neq j \neq m \wedge \mathrm{val}_\mathcal{G}(A_k) \neq \mathrm{val}_\mathcal{G}(A_m))\}.
\end{aligned}$$

Using the mappings δ_a and the SLCF string grammar $\mathcal{G}_i$, we can compute Q_i from P_i easily in polynomial time. □

7 Adding Nondeterminism or Non-linearity

If we relax condition (i) of the definition of SLCF tree grammars to (i') P contains for every $A \in N$ *at least one* production with left-hand side A (but keep the acyclicity condition (ii)) then we obtain *nondeterministic* SLCF tree grammars (NSLCF tree grammars). Such grammars generate finite sets of trees, which by the following example may contain double-exponentially many trees.

Example 12. For $n \geq 1$, let the linear, productive, and monadic NSLCF tree grammar $\mathcal{G}_n$ consist of the productions $S \to A_0(a)$, $A_i(y_1) \to A_{i+1}(A_{i+1}(y_1))$ for $0 \leq i < n$, $A_n(y_1) \to f(y_1)$, and $A_n(y_1) \to g(y_1)$. Then $L(\mathcal{G}_n)$ consists of all monadic trees with 2^n many internal nodes, each of which is labeled f or g. Thus $|L(\mathcal{G}_n)| = 2^{2^n}$.

We now want to show that given a linear and productive NSLCF tree grammar $\mathcal{G}$, we can, in general, *not* obtain an equivalent *monadic* grammar of size $|\mathcal{G}|^{O(1)}$. In fact, there is a family $\mathcal{G}_n$ ($n \geq 1$) of linear and productive NSLCF tree grammars such that any monadic, linear, and productive NSLCF tree grammar that generates $L(\mathcal{G}_n)$ is of size $2^{O(|\mathcal{G}_n|^{1/2})}$. Thus, for nondeterministic grammars an exponential blow-up cannot be avoided when going to monadic grammars. Later we show that this is the worst case blow-up and that in fact any linear and non-deleting NSLCF tree grammar can be transformed into an equivalent monadic one which is at most exponentially larger.

Example 13. For $n \geq 1$, let the symbol f_n be of rank n and define the linear and productive NSLCF tree grammar $\mathcal{G}_n$ (of size $O(n^2)$) with the following productions:

$$\begin{aligned} S &\to A_0(a, \ldots, a) \\ A_i(y_1, \ldots, y_n) &\to A_{i+1}(f(y_1), \ldots, f(y_n)) \text{ for } 0 \leq i < n \\ A_i(y_1, \ldots, y_n) &\to A_{i+1}(g(y_1), \ldots, g(y_n)) \text{ for } 0 \leq i < n \\ A_n(y_1, \ldots, y_n) &\to f_n(y_1, \ldots, y_n) \end{aligned}$$

Then $L_n = L(\mathcal{G}_n)$ consists of all trees $f_n(t, t, \ldots, t)$ where t is a monadic tree with n many internal nodes, each of which is labeled f or g.

Lemma 14. *Let $n \geq 1$, $k < n$, and let $\mathcal{G}$ be a linear, non-deleting, and k-bounded NSLCF grammar such that $L(\mathcal{G}) = L_n$ is the set from Example 13. Then $|\mathcal{G}| \geq 2^n$.*

Proof. Assume that $\mathcal{G}$ is a linear, non-deleting, and k-bounded NSLCF tree grammar such that $k < n$ and $L(\mathcal{G}) = L_n$. W.l.o.g. we can assume that every nonterminal of $\mathcal{G}$ appears in a successful derivation of $\mathcal{G}$. Let $P(f_n)$ be the set of all productions of the form $A \to t$, where t contains a subtree of the form $f_n(t_1, \ldots, t_n)$. Clearly, since $\mathcal{G}$ is non-deleting, every right-hand side of a production from $P(f_n)$ contains a unique such subtree. Moreover, in every successful derivation of $\mathcal{G}$, a production from $P(f_n)$ has to be applied exactly once. We claim that $|P(f_n)| \geq 2^n$. Consider a production $(A \to t) \in P(f_n)$ and consider the unique subtree in t of the form $f_n(t_1, \ldots, t_n)$. Since $\mathrm{rank}(A) \leq k < n$ and $\mathcal{G}$ is linear, there exists an $i \in \{1, \ldots, n\}$ such that t_i does not contain a parameter, i.e., $t_i \in T(\mathbb{F} \cup N)$. Assume that two different terminal trees can be derived from t_i. Then we can derive with $\mathcal{G}$ a tree, where the root has two different subtrees, a contradiction. Hence, from t_i we can generate exactly one tree. We denote this tree by $\tau[A \to t]$, since it can be associated with the production $(A \to t) \in P(f_n)$. Hence, for every successful derivation $S \Rightarrow_{\mathcal{G}}^{*} s$, where the production $(A \to t) \in P(f_n)$ is applied (exactly once), we must have $s = f_n(\tau[A \to t], \ldots, \tau[A \to t])$. Since we can generate 2^n many terminal trees from S and in each derivation exactly one production from $P(f_n)$ is applied, it follows that $|P(f_n)| \geq 2^n$. □

For arbitrary linear context-free tree grammars (thus, with recursion and nondeterminism), the number of parameters gives rise to a hierarchy of languages which is strict at each level. In fact, the family of languages that can be used to prove the strictness of this hierarchy is similar to the one of Example 13.

Example 15. For $n \geq 1$, let f_n be a symbol of rank n and A be a nonterminal of rank n. Define the linear and productive context-free tree grammar $\mathcal{G}_n$ with the productions $S \to A(a, \ldots, a)$, $A(y_1, \ldots, y_n) \to A(f(y_1), \ldots, f(y_n))$, and $A(y_1, \ldots, y_n) \to f_n(y_1, \ldots, y_n)$. Then $L'_n = L(\mathcal{G}_n)$ consists of all trees $f_n(t, t, \ldots, t)$ where t is a monadic tree of the form $f^m(a)$ for some $m \geq 0$.

The proof of the following lemma is similar to the one of Lemma 14.

Lemma 16. *Let $n \geq 1$ and $k < n$. The set L'_n from Example 15 cannot be generated by a linear, non-deleting, and k-bounded context-free tree grammar.*

By the following theorem, the lower bound from Lemma 14 can be matched by an upper bound. The proof of this result is similar to the proof of Theorem 7.

Theorem 17. *For a given linear NSLCF tree grammar $\mathcal{G} = (N, P, S)$ we can construct in time $2^{O(|\mathcal{G}|)}$ a linear and monadic NSLCF tree grammar $\mathcal{G}' = (N', P', S)$ of size $2^{O(|\mathcal{G}|)}$ such that $L(\mathcal{G}') = L(\mathcal{G})$.*

One might also think about extending Theorem 7 to *non-linear* SLCF tree grammars. But results from [11] make such an extension quite unlikely: it is PSPACE-complete to check whether a deterministic bottom-up tree automaton accepts $\mathrm{val}(\mathcal{G})$, where $\mathcal{G}$ is a given (non-linear) SLCF tree grammar. If we restrict this problem by requiring $\mathcal{G}$ to be k-bounded for a fixed constant k, then it becomes P-complete. Here is an explicit example showing that Theorem 7 cannot be extended to non-linear SLCF tree grammars.

Example 18. For $n \geq 1$, let the symbol f_n be of rank n, let g have rank 2, and let 0 and 1 have rank 0. Define the productive (but non-linear) SLCF tree grammar $\mathcal{G}_n$ with the following productions, where A_i is a nonterminal of rank i ($1 \leq i \leq n$):

$$\begin{aligned} S &\to g(A_1(0), A_1(1)) \\ A_i(y_1, \ldots, y_i) &\to g(A_{i+1}(y_1, \ldots, y_i, 0), A_{i+1}(y_1, \ldots, y_i, 1)) \text{ for } 1 \leq i < n \\ A_n(y_1, \ldots, y_n) &\to f_n(y_1, \ldots, y_n) \end{aligned}$$

Then $\mathrm{val}(\mathcal{G}_n)$ results from a complete binary g-tree of height n by replacing the k-th leaf ($0 \leq k \leq 2^n - 1$) by the tree $f_n(b_1, \ldots, b_n)$, where $b_1 b_2 \cdots b_n$ is the binary representation of k. The size of $\mathcal{G}_n$ is $O(n^2)$.

Lemma 19. *Let $n \geq 1$, $k < n$, and let $\mathcal{G}$ be a k-bounded SLCF tree grammar such that $\mathrm{val}(\mathcal{G}) = \mathrm{val}(\mathcal{G}_n)$, where $\mathcal{G}_n$ is the SLCF tree grammar of Example 18. Then $|\mathcal{G}| \geq 2^{n-k}$.*

Proof. Let T_n be the set of all occurrences of subterms of the form $f_n(t_1, \ldots, t_n)$ that occur in right-hand sides of $\mathcal{G}$. We claim that $|T_n| \geq 2^{n-k}$. Consider a term $f_n(t_1, \ldots, t_n) \in T_n$. Since $\mathcal{G}$ is k-bounded, at most k parameters can occur among the terms $t_1, \ldots, t_n$. During the derivation, each of these parameters may be either substituted by the constant 0 or 1. Hence, from each $f_n(t_1, \ldots, t_n) \in T_n$, we can obtain during the derivation at most 2^k different trees of the form $f(b_1, \ldots, b_n)$ with $b_1, \ldots, b_n \in \{0, 1\}$. Since $\mathrm{val}(\mathcal{G}_n)$ contains 2^n such subtrees, we get $|T_n| \geq 2^{n-k}$. □

Clearly, Lemma 19 implies that without an exponential blow-up, we cannot reduce the number of parameters in any non-linear SLCF tree grammar to a constant. But we cannot even reduce the number of parameters from n to $\varepsilon \cdot n$ (where $\varepsilon < 1$ is a constant) without an exponential blowup. For arbitrary context-free tree grammars with OI derivation order it is proved in Theorem 6.5 of [6] that the number of parameters gives rise to a hierarchy that is proper at each step (even for the string yield languages).

Acknowledgments. The first author is supported by the DFG research project *Algorithms on compressed data* (ALKODA). We would like thank Christian Mathissen for pointing out a mistake in a previous version of the contract-2 operation.

References

1. Bogaert, B., Tison, S.: Equality and disequality constraints on direct subterms in tree automata. In: Finkel, A., Jantzen, M. (eds.) STACS 1992. LNCS, vol. 577, pp. 161–171. Springer, Heidelberg (1992)
2. Buneman, P., Grohe, M., Koch, C.: Path queries on compressed XML. In: VLDB 2003, pp. 141–152. Morgan Kaufmann, San Francisco (2003)
3. Busatto, G., Lohrey, M., Maneth, S.: Efficient memory representation of XML document trees. Information Systems 33(4–5), 456–474 (2008)
4. Comon-Lundh, H., Dauchet, M., Gilleron, R., Jacquemard, F., Löding, C., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications (2007), http://www.grappa.univ-lille3.fr/tata
5. Comon-Lundh, H., Jacquemard, F., Perrin, N.: Tree automata with memory, visibility and structural constraints. In: Seidl, H. (ed.) FOSSACS 2007. LNCS, vol. 4423, pp. 168–182. Springer, Heidelberg (2007)
6. Engelfriet, J., Rozenberg, G., Slutzki, G.: Tree transducers, L systems, and two-way machines. J. Comp. Syst. Sci. 20, 150–202 (1980)
7. Fischer, M.: Grammars with macro-like productions. PhD thesis, Harvard University, Massachusetts (May 1968)
8. Fujiyoshi, A., Kasai, T.: Spinal-formed context-free tree grammars. Theory Comput. Syst. 33(1), 59–83 (2000)
9. Gascón, A., Godoy, G., Schmidt-Schauß, M.: Context matching for compressed terms. In: LICS 2008, pp. 93–102. IEEE Computer Society Press, Los Alamitos (2008)
10. Levy, J., Schmidt-Schauß, M., Villaret, M.: Bounded second-order unification is NP-complete. In: Pfenning, F. (ed.) RTA 2006. LNCS, vol. 4098, pp. 400–414. Springer, Heidelberg (2006)
11. Lohrey, M., Maneth, S.: The complexity of tree automata and XPath on grammar-compressed trees. Theor. Comput. Sci. 363(2), 196–210 (2006)
12. Plandowski, W.: Testing equivalence of morphisms on context-free languages. In: van Leeuwen, J. (ed.) ESA 1994. LNCS, vol. 855, pp. 460–470. Springer, Heidelberg (1994)
13. Rytter, W.: Grammar compression, LZ-encodings, and string algorithms with implicit input. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 15–27. Springer, Heidelberg (2004)
14. Schmidt-Schauß, M.: Polynomial equality testing for terms with shared substructures. Technical Report 21, Institut für Informatik, J. W. Goethe-Universität Frankfurt am Main (2005)

The Calculus of Handshake Configurations

Luca Fossati[1,2,⋆] and Daniele Varacca[2]

[1] Dip. di Informatica - Università di Torino, Italia
fossati@di.unito.it
[2] PPS - CNRS & Univ. Paris Diderot, France

Abstract. Handshake protocols are asynchronous protocols that enforce several properties such as absence of transmission interference and insensitivity from delays of propagation on wires. We propose a concurrent process calculus for handshake protocols . This calculus uses two mechanisms of synchronization: rendez-vous communication à la CCS, and shared resource usage. To enforce the handshake discipline, the calculus is endowed with a typing system.

We provide an LTS semantics of the calculus and show that typed processes denote handshake protocols. We give the calculus another semantics in terms of a special kind of Petri nets called handshake Petri nets. We show that this semantics is complete and fully abstract with respect to weak bisimilarity.

Keywords: Handshake protocols, Petri nets, process calculus, types.

1 Introduction

Asynchronous circuits are used to design systems where the local activity of each subunit is not restrained by some global condition, like the long time intervals imposed by a system clock. When designing such systems, one has to face several questions. How do we know when a message we sent has reached its destination so that we can use the same channel again, i.e. how can we avoid *transmission interference*? How can we ensure the correct behavior regardless of computational speeds of single modules and propagation delays over wires, i.e. how can we enforce *delay-insensitivity*?

Handshake protocols try to answer these questions by imposing an interactive communication discipline. In particular, the protocols require that after a circuit has sent a message on a channel, it has to wait for a confirmation that the message was received before sending again on the same channel. This requirement alone is enough to rule out transmission interference. For their simplicity and efficiency, handshake protocols have been employed by enterprises like Philips and Sun in the development of a series of VLSI chips [9].

The first attempts to formalize delay-insensitive protocols and their properties employed trace sets [17]. In particular, the first model to specifically address the handshake case was given in [18]. Trace models have been able to neatly formalize the properties of handshake protocols which ensure delay-insensitivity, but so far they have failed in representing correctly their composition [4].

⋆ Corresponding author.

To overcome this limitation, we propose an alternative approach to modeling handshake protocols: we propose a process calculus inspired by Robin Milner's Calculus of Communicating Systems (CCS) [13]. Similarly to CCS, our calculus defines concurrent processes that communicate via rendez-vous channels. However, in order to ensure the handshake discipline, the calculus features another synchronization mechanism, by means of shared resources, reminiscent of coordination languages like Linda [7]. Also, the calculus is endowed with a linear typing system, inspired by [11,19]. These design choices allow to express the external behavior of a handshake protocol, along with a more complex internal behavior.

We say that this is the first independent syntactical description of the handshake behavior, as our *handshake configurations* are independent from any semantical interpretation while the handshake behavior is ensured by the typing system. This was not the case in previous process algebras for handshake protocols [18,10], where only processes whose trace semantics satified the handshake behavior were considered, thus processes were trace sets and inevitably suffered from the compositionality problems observed in the underlying trace model.

We then compare our calculus and the trace model by defining, for each configuration, the corresponding set of *quiescent* traces, i.e. the traces corresponding to computations that may not be extended with an output event. We show that this quiescent trace semantics is sound w.r.t. Van Berkel's definition [18]. By means of an example, we show that quiescent trace equivalence is not a congruence w.r.t. parallel composition. This confirms the intuition that trace models of handshake protocols are not informative enough, and their branching structure needs to be taken into account. Indeed weak bisimilarity is a congruence for our calculus.

To show the expressive power of the calculus, we give it a Petri net semantics, where the handshake discipline is imposed by restricting a net's external structure. We studied this model in details in a previous work [6], where we showed that it captures precisely the behavior of a protocol, in the sense that there is a protocol for each net and a net (possibly with an infinite number of places and/or transitions) for each protocol. In this work we show that there is a correspondence between handshake processes and *finite* Petri nets, in the sense that for each finite handshake Petri net, there is a weakly bisimilar process.

The graphical approach to the formal analysis of asynchrony is not a new one ([1],[12],...). In particular Dan Ghica developed a language for asynchronous hardware design by taking inspiration from the Geometry of Interaction and handshake circuits [8]. However his goal was to improve previous hardware design languages [18,3] and not to capture all handshake behaviors.

The paper is structured as follows. In Section 2 we recall the first formalization of handshake protocols as we introduce the notion of handshake language. In Section 3, we present syntax, operational semantics and type system of our calculus. We show that the set of quiescent traces of a typed configuration is a handshake language. We show that weak bisimilarity is a congruence, while quiescent trace semantics is not. In Section 4, we present handshake Petri nets, with some examples to show how they work. In Section 5, we present the interpretation of the calculus into handshake nets, and we show that it is fully abstract with respect to weak bisimilarity. To conclude, we show the universality of the semantics, by showing that every bisimilarity class of finite handshake nets is denoted by a process.

2 Handshake Protocols

In this section we recall the background properties of handshake protocols and introduce the notion of *handshake language*.

Definition 1. *A* handshake structure *is a pair $\langle Ports, d\rangle$, where Ports is a finite set of* ports *and the function $d : Ports \to \{!, ?\}$ determines a direction for each port,* active *or* passive.

As we shall see, active ports are allowed to start a communication, while passive ports are initially waiting. For the rest of this section let $\langle Ports, d\rangle$ be a handshake structure. For each port a, there are two possible messages: a (input message), and $\bar{a}$ (output message). Let t be a finite trace on the alphabet of messages $\cup_{a\in Ports}\{a, \bar{a}\}$. t is a *handshake trace* on $\langle Ports, d\rangle$ if for all $a \in Ports$:

- $t \upharpoonright \{a, \bar{a}\} = \bar{a}a\bar{a}a\ldots$ when $d(a) = !$;
- $t \upharpoonright \{a, \bar{a}\} = a\bar{a}a\bar{a}\ldots$ when $d(a) = ?$;

Given a set of traces S we write $S^{\leq}$ for its prefix-closure. Let σ be a set of handshake traces, $s \in \sigma^{\leq}$ is *passive* in σ if and only if there is no extension of s in σ obtained by appending an output message: $\forall s \cdot m \in \sigma^{\leq}$, m is an input message. We write $Pas(\sigma)$ for the set of passive traces in $\sigma^{\leq}$.

We define $\mathbf{r}_{Ports}$ as the smallest binary relation which is closed by reflexivity, transitivity and concatenation and such that for any distinct ports $a, b \in Ports$:

1. $a\bar{b}\ \mathbf{r}_{Ports}\ \bar{b}a$;
2. $\bar{a}\bar{b}\ \mathbf{r}_{Ports}\ \bar{b}\bar{a}$;
3. $ab\ \mathbf{r}_{Ports}\ ba$

We say that *s reorders t* in *Ports* if $s\ \mathbf{r}_{Ports}\ t$.

Let s be a handshake trace and $a \in Ports$. We write $a\ \mathbf{x}_{Ports}\ s$ if sa is still a handshake trace. Finally:

Definition 2. *A* handshake language *σ on $\langle Ports, d\rangle$ is a non-empty set of finite handshake traces on $\langle Ports, d\rangle$ such that:*

1. *$Pas(\sigma) \subseteq \sigma$ (closed under passive prefixes);*
2. *$(t \in \sigma \wedge s\ \mathbf{r}_{Ports}\ t) \Rightarrow s \in \sigma$ (reorder closed);*
3. *$(s \in \sigma^{\leq} \wedge a\ \mathbf{x}_{Ports}\ s) \Rightarrow s \cdot a \in \sigma^{\leq}$ (receptive).*

Closedness under passive prefixes, rather than under any prefix as it is usually the case for trace semantics, allows us to represent *may&must nondeterminism*. We want to represent systems which are able not only to make an exclusive choice between two outputs, but also to choose between sending an output and not doing anything.

The intuition is thus that a trace in a handshake language represents a *quiescent* execution of a protocol, that is an execution that ends in a state in which the system may decide to wait for more inputs before sending any output. By definition, after a passive trace the system cannot do anything but receiving. Then all passive traces correspond to quiescent executions.

Reorder-closedness says that a message m' cannot "block" a message m'' on a different channel unless m' is an input and m'' an output. The intuition is that inputs may carry necessary information and thus may block, while the transmission of an output may require informations and can thus be blocked.

Finally, receptiveness means that whenever it is the environment's turn to send a message, the system must be ready to accept it.

Definition 2 is like VanBerkel's original definition of handshake process [18] without data-passing. No satisfactory definition of composition for handshake languages exists. In particular the definition given by VanBerkel is not associative, as shown by the first author [4] using a counter-example by Roscoe [16].

In the following sections, handshake languages will be used as a yardstick against which to measure the correctness of other descriptions of handshake protocols.

3 The Calculus

In this section we provide the formal definition for our *Calculus of Handshake Configurations (CHC)*. We stress that we do not model data-passing as we are only interested in the communication protocol. The calculus is endowed with two communication mechanisms. Besides the external communication via rendez-vous channels, there is also a form of internal communication, invisible to the outside, where actions may require resources in order to be performed and may release resources for other actions to use. This is necessary to model internal synchronizations between different ports of the same system. However, different systems shall communicate only through channels.

3.1 Syntax and Operational Semantics

We consider a set A of *channels* denoted by a, b[1], and a set R of *resources* denoted by r, k. The syntax of the calculus uses three syntactic categories: *threads*, *processes* and *handshake configurations*. Threads are purely sequential and allow prefixing while processes are parallel compositions of threads. The prefixes are *input* and *output* actions on a finite set of resources. As we will see later, input actions *release* resources and output actions *use* resources. Let $\Delta \subseteq A$:

$$\begin{array}{rcll} act & ::= & a^{\{r_1,\ldots,r_n\}} \;\mid\; \bar{a}_{\{r_1,\ldots,r_n\}} & \text{Actions} \\ T & ::= & \mathbf{0} \;\mid\; act.T \;\mid\; \mathbf{Rec}\, T & \text{Threads} \\ P, Q & ::= & T \;\mid\; P \mid Q \;\mid\; P \setminus \Delta & \text{Processes} \end{array}$$

A handshake configuration is composed of a process P along with a *multiset* of resources S for internal synchronization. A configuration can be *open* or *closed*:

$$M ::= \wr P, S \wr \;\mid\; \langle P, S \rangle \quad \text{Open and closed configurations}$$

[1] We will use channels to model ports, but we prefer to keep the conceptual difference between the two notions.

$$\frac{}{\lbag a^{\{r_1,\ldots,r_n\}}.T, S \rbag \xrightarrow{a} \lbag T, S + \{r_1,\ldots,r_n\} \rbag}\ \text{(inev)} \qquad \frac{\lbag T \cdot \mathbf{Rec}\, T, S \rbag \xrightarrow{e} \lbag T', S' \rbag}{\lbag \mathbf{Rec}\, T, S \rbag \xrightarrow{e} \lbag T', S' \rbag}\ \text{(rec)}$$

$$\frac{}{\lbag \bar{a}_{\{r_1,\ldots,r_n\}}.T, S + \{r_1,\ldots,r_n\} \rbag \xrightarrow{\bar{a}} \lbag T, S \rbag}\ \text{(outev)} \qquad \frac{M \xrightarrow{e} M' \quad ch(e) \notin \Delta}{M \setminus \Delta \xrightarrow{e} M' \setminus \Delta}\ \text{(res)}$$

$$\frac{M \xrightarrow{e} M'}{M \mid N \xrightarrow{e} M' \mid N}\ \text{(par1)} \qquad \frac{M \xrightarrow{\bar{a}} M' \qquad N \xrightarrow{a} N'}{M \mid N \xrightarrow{\tau} M' \mid N'}\ \text{(par2)}$$

$$\frac{P \equiv P' \quad \lbag P', S \rbag \xrightarrow{e} \lbag Q', S' \rbag \quad Q \equiv Q'}{\lbag P, S \rbag \xrightarrow{e} \lbag Q, S' \rbag}\ \text{(struct)} \qquad \frac{\lbag P, S \rbag \xrightarrow{e} \lbag Q, S' \rbag}{\langle P, S \rangle \xrightarrow{e} \langle Q, S' \rangle}\ \text{(closure)}$$

Fig. 1. Labeled transition semantics

Intuitively, open configurations represent systems under construction, whose resources are still accessible to the environment. Closed configurations represent completed systems and can only communicate via handshake channels.

The operational semantics is given in terms of an LTS over handshake configurations. Labels are channels with their polarity, plus the unobservable label:

$$e ::= \bar{a} \mid a \mid \tau \qquad a \in A$$

Given an observable label, the function *ch* returns the channel on which it occurred. Formally: $ch(\bar{a}) = ch(a) = a$ for any channel a. The definition of the operational semantics is simplified thanks to the congruence ($\equiv$) between processes:

$$P \mid Q \equiv Q \mid P \qquad \mathbf{Rec}\,\mathbf{0} \equiv \mathbf{0}$$

Let $res(P)$ be the set of resources of a process P. As meta-notation, we define sequential composition of threads $T \cdot T'$:

$$(act.T) \cdot T' = act.(T \cdot T') \qquad T \cdot T' = T' \ \ (T \equiv \mathbf{0}) \qquad (\mathbf{Rec}\, T) \cdot T' = \mathbf{Rec}\, T \ \ (T \not\equiv \mathbf{0})$$

and we extend process operators to configurations:

$$\lbag P_1, S_1 \rbag \mid \lbag P_2, S_2 \rbag = \lbag P_1 \mid P_2, S_1 + S_2 \rbag \qquad \lbag P, S \rbag \setminus \Delta = \lbag P \setminus \Delta, S \rbag$$

$$\langle P_1, S_1 \rangle \mid \langle P_2, S_2 \rangle = \langle \mu_1(P_1) \mid \mu_2(P_2), \mu_1(S_1) + \mu_2(S_2) \rangle \qquad \langle P, S \rangle \setminus \Delta = \langle P \setminus \Delta, S \rangle$$

where $+$ denotes the union of multisets and $\mu_1 : res(P_1) \cup S_1 \to R_1$ and $\mu_2 : res(P_2) \cup S_2 \to R_2$ are injective functions between resources such that R_1 and R_2 are disjoint and all the resources they contain are fresh. Moreover $\mu(S)$ is the point-to-point application of the function μ to the multiset S, while $\mu(P)$ is the process obtained from P by renaming any label occurrence according to μ. This guarantees that two closed configurations can only communicate via channels .

$$\frac{a \in A}{\mathbf{0} \rhd !a}\ \text{(ax)} \qquad \frac{T \rhd !a}{\mathbf{Rec}T \rhd !a}\ \text{(rec)}$$

$$\frac{T \rhd ?a}{\bar{a}_{\{r_1,\dots r_n\}}.T \rhd !a}\ \text{(outpref)} \qquad \frac{T \rhd !a}{a^{\{r_1,\dots r_n\}}.T \rhd ?a}\ \text{(inpref)}$$

$$\frac{P \rhd \Gamma' \qquad Q \rhd \Gamma'' \qquad \forall a \in Dom(\Gamma') \cap Dom(\Gamma''),\ \Gamma'(a) \neq \Gamma''(a)}{(P \mid Q) \setminus (Dom(\Gamma') \cap Dom(\Gamma'')) \rhd \Gamma' \odot \Gamma''}\ \text{(par)}$$

$$\frac{P \rhd \Gamma}{\wr P, S \wr \rhd \Gamma}\ \text{(oconf)} \qquad \frac{P \rhd \Gamma}{\langle P, S \rangle \rhd \Gamma}\ \text{(cconf)}$$

Fig. 2. Handshake types

Note that we do not define the parallel composition of an open and a closed configuration. The idea is that we may combine two different parts of a system (in the construction stage) or two completed systems (for interaction), but we may not combine a system under construction with a completed one.

The derivation rules for the operational semantics are shown in Figure 1. When an input occurs, a set of resources becomes available; while an output requires a set of resources in order to occur, then the used resources disappear. The other rules are quite standard. Note however that the operational distinction between open and closed configurations comes from the two distinct cases of composition given above. In the parallel composition of open configurations, one side may influence the other by modifying a shared resource, as no renaming takes place. This is not possible for closed configurations, as renaming prevents the sharing of resources.

A sequence of transitions $M_0 \xrightarrow{e_0} M_1 \dots M_n \xrightarrow{e_n} M$ is denoted $M_0 \overset{t}{\twoheadrightarrow} M$, where $t = e_0 \dots e_n$. The string t is called the *strong* trace of the sequence, while the *weak* trace is the restriction of t to the labels other than τ. Strong ($\sim$) and weak ($\approx$) bisimilarity are also defined as usual [13] on the labeled transition system for CHC.

3.2 Typing System

A *type* Γ is a partial function from channel names to $\{!, ?\}$. We will use the shorthand notation $!a$ or $?a$ to describe a type defined on channel a, and commas to join types. We say that a is *active* in Γ when $\Gamma(a) = !$ and we say it is *passive* when $\Gamma(a) = ?$.

Let Γ' and Γ'' be two types and let a be a channel. Let us define the function $\Gamma' \odot \Gamma'' : (Dom(\Gamma') \setminus Dom(\Gamma'')) \cup (Dom(\Gamma'') \setminus Dom(\Gamma')) \to \{!, ?\}$ such that:

– $\Gamma' \odot \Gamma''(a) = \Gamma'(a)$, when $a \in Dom(\Gamma') \setminus Dom(\Gamma'')$;
– $\Gamma' \odot \Gamma''(a) = \Gamma''(a)$, when $a \in Dom(\Gamma'') \setminus Dom(\Gamma')$.

Typing judgements are of the form $T \rhd \Gamma$, $P \rhd \Gamma$, $M \rhd \Gamma$, where T is a thread, P a process, M a configuration and Γ a type. The typing rules are shown in Figure 2. The empty thread is active: this models receptiveness, because a thread of passive type must always be able to perform another input. The following three rules guarantee that threads are alternating on each channel. The parallel composition of two processes is allowed only if threads on the same channel have dual types. These channels must then be restricted so that no other process can communicate on them. This models the point-to-point communication discipline of handshake protocols. Note that resources do not play any role in the typing.

The following results show the intuition behind the typing system.

Lemma 3 (Reduction). *Let M be a configuration such that $M \rhd \Gamma$. Then:*

- $M \xrightarrow{a} \quad \Longleftrightarrow \quad \Gamma(a) = ?$;
- $M \xrightarrow{\bar{a}} \quad \Rightarrow \quad \Gamma(a) = !$;
- $M \xrightarrow{e} M' \wedge e \neq \tau \quad \Rightarrow \quad M' \rhd \Gamma'$ *s.t.* $Dom(\Gamma') = Dom(\Gamma) \wedge$
  $\forall b \in Dom(\Gamma), b \neq ch(e) \leftrightarrow \Gamma(b) = \Gamma'(b)$;
- $M \xrightarrow{\tau} M' \quad \Rightarrow \quad M' \rhd \Gamma$;

Corollary 4 (Subject Reduction). *Let $M \rhd \Gamma$ and $M \overset{s}{\twoheadrightarrow} M'$ then there is a type Γ' such that $M' \rhd \Gamma'$.*

3.3 Examples

As a first example, we show a configuration representing the *OR* handshake protocol.

$$OR = \langle a^{\{r_1\}}.\mathbf{Rec}\ \bar{a}_{\{r_2\}}.a^{\{r_1\}}.\mathbf{0} \mid \mathbf{Rec}\ \bar{b}_{\{r_1\}}.b^{\{r_2\}}.\mathbf{0} \mid \mathbf{Rec}\ \bar{c}_{\{r_1\}}.c^{\{r_2\}}.\mathbf{0},\ \emptyset\rangle$$

We have that $OR \rhd ?a, !b, !c$. When a request on the passive port a arrives, the resource r_1 becomes available and this enables *OR* to send a request on either active port b, c. An acknowledge to this last request enables an acknowledge to the first one. The second configuration represents the *MIX* protocol.

$$MIX = \langle b^{\{k_1\}}.\mathbf{Rec}\ \bar{b}_{\{k_2\}}.b^{\{k_1\}}.\mathbf{0} \mid c^{\{k_1\}}.\mathbf{Rec}\ \bar{c}_{\{k_2\}}.c^{\{k_1\}}.\mathbf{0} \mid \mathbf{Rec}\ \bar{d}_{\{k_1\}}.d^{\{k_2\}}.\mathbf{0},\ \emptyset\rangle$$

We have that $MIX \rhd ?b, ?c, !d$. Each time an environment request arrives (on either passive port b, c), the component *MIX* handshakes on its active port d and after completion it acknowledges to the first request. If, by the time the handshake on the active port is complete, the environment has sent a request on the other port, *MIX* chooses nondeterministically which request to acknowledge first.

The two protocols can be composed in parallel, communicating on the common ports. We have $(OR \mid MIX) \setminus \{b, c\} \rhd ?a, !d$.

3.4 Soundness

In this section we show that typed CHC configurations indeed define handshake languages, using the weak traces of the labeled transition semantics.

Each feature of the calculus plays a role in modeling the handshake discipline. Let us see informally how. First of all, handshake languages alternate input and output on the same port. This is enforced by the typing system. The reorder closure is guaranteed by the fact that different ports are on different parallel threads. The only reordering that is in general not allowed is when an input blocks an output. An input can block an output because an output may need resources that will only be provided by the input. Finally, receptiveness is guaranteed by the fact that inputs do not need resources, and can always occur, provided that the alternation with the corresponding outputs is respected.

In order to denote handshake languages we consider the weak traces of the transition sequences of a configuration. If we considered the traces of all transition sequences, the denoted languages would always be prefix closed and some handshake languages would excape us. To characterize the larger class of languages closed under passive prefixes, we consider only the traces of the *quiescent* transition sequences.

A configuration M is *quiescent* if it cannot (weakly) perform an output, i.e. if there is no transition sequence of the form $M \overset{(\tau)^*}{\twoheadrightarrow}\xrightarrow{\bar{a}}$, for any channel a. A transition sequence $M \overset{t}{\twoheadrightarrow} M'$ is *quiescent* if M' is.

Definition 5. *Let M be a handshake configuration. We define $HL(M)$ to be the set of weak traces of all the quiescent transition sequences which start from M.*

Let M be a configuration and Γ a type such that $M \rhd \Gamma$. The handshake structure $HS(\Gamma) = \langle Ports_\Gamma, d_\Gamma \rangle$ is defined by setting $Ports_\Gamma = Dom(\Gamma)$ and $d_\Gamma = \Gamma$.

Proposition 6 (Soundness). *Let M be a handshake configuration, such that $M \rhd \Gamma$. Then $HL(M)$ is a handshake language on the handshake structure $HS(\Gamma)$.*

We observe, however, that the other direction, the fact that each language is the denotation of some configuration, cannot be established. This is due to the presence of non recursive handshake languages which could never be captured by finite configurations. It would still be interesting to characterize the class of handshake languages that correspond to CHC configurations. We leave this as future work.

3.5 Compositionality

Open configurations can communicate via shared resources, but this is not directly observable in the labeled transition semantics. Thus we cannot expect a labeled equivalence to be fully congruent for them. However weak bisimilarity is a congruence with respect to composition of closed configurations:

Proposition 7. *Let M_1, M_2, N be closed handshake configurations such that $M_1, M_2 \rhd \Gamma$, $N \rhd \Gamma'$ and $(M_1 \mid N) \setminus \Delta \rhd \Gamma \odot \Gamma'$, $(M_2 \mid N) \setminus \Delta \rhd \Gamma \odot \Gamma'$, where $\Delta = Dom(\Gamma) \cap Dom(\Gamma')$. Then $M_1 \approx M_2$ implies $(M_1 \mid N) \setminus \Delta \approx (M_2 \mid N) \setminus \Delta$.*

This is consistent with our intepretation of resources as *internal* means of communication. Our main goal was to describe the *externally observable* behavior of a system and we do so by considering only those configurations whose resources cannot be accessed by the environment.

In Section 2 we talked about the difficulty of finding a good definition of composition for handshake languages. This intuition is confirmed as "quiescent trace equivalence" is not a congruence. Consider the following processes:

$$P_1 = \bar{c}_{\{r_1,r_2\}}.c^{\{\}}.\mathbf{0} \mid \bar{b}_{\{r_3,r_1\}}.b^{\{r_1\}}.\mathbf{Rec}\ \bar{b}_{\{r_1\}}.b^{\{r_1\}}.\mathbf{0} \mid (\bar{d}_{\{r_3,r_2\}}.d^{\{r_3\}}.\mathbf{0} \mid d^{\{\}}.\bar{d}_{\{\}}.d^{\{\}}.\mathbf{0}) \setminus \{d\}$$

$$P_2 = \bar{c}_{\{r_1\}}.c^{\{\}}.\mathbf{0} \mid \mathbf{Rec}\ \bar{b}_{\{r_1\}}.b^{\{r_1\}}.\mathbf{0}\ .$$

Consider the closed configurations $M_1 = \langle P_1, \{r_1, r_2, r_3\}\rangle$ and $M_2 = \langle P_2, \{r_1, r_2, r_3\}\rangle$. They are both interpreted as the same handshake language:

$$HL(M_1) = \{\bar{c}, \bar{c}c, \bar{b}, \bar{b}b\bar{c}, \bar{b}b\bar{c}c, \bar{b}b\bar{b}, \ldots\} = HL(M_2)$$

however, if we compose them with $N = \langle b^{\{\}}.\mathbf{Rec}\,\bar{b}_{\{\}}.b^{\{\}}.\mathbf{0} \mid c^{\{k\}}.\mathbf{0} \mid \bar{a}_{\{k\}}.a^{\{\}}.\mathbf{0},\ \emptyset\rangle$ we obtain two configurations with different interpretations:

$$HL((M_1 \mid N) \setminus \{b, c\}) = \{\varepsilon, \bar{a}, \bar{a}a\} \qquad HL((M_2 \mid N) \setminus \{b, c\}) = \{\bar{a}, \bar{a}a\}$$

Therefore the parallel composition of CHC configurations cannot be used to define the composition of handshake languages. In order to compose handshake protocols, some more knowledge on the branching structure is needed. CHC provides a suitable formalism to study this structure.

4 Handshake Petri Nets

We argued that not all handshake languages can be represented by CHC configurations, as, for instance, there are non recursive handshake languages. To show the expressive power of our calculus, we provide an alternative semantics of CHC based on Petri nets. In [6], we studied a Petri net representation of handshake protocols, called handshake Petri net, and we showed that all handshake languages can have a, possibly infinite, handshake Petri net representation. In this paper we show that every *finite* handshake Petri net is weakly bisimilar to a CHC configuration.

In this section we introduce handshake Petri nets. The present definition is slightly different from the one in [6], but the results of that paper carry over. We assume some basic knowledge on Petri nets, which we will use in their standard graphical representation [15]. Throughout the paper we will consider Petri nets in their *unsafe* version, where places are allowed to contain several tokens at the same time. This is not just for convenience. Unsafe nets are necessary to carry out our construction.

4.1 Definition

Handshake Petri nets are Petri nets with a special “external interface”, reflecting the structure of handshake ports. We define handshake ports in two phases. We first define the static structure of ports, and then we specify the markings.

Let G be a Petri net and let N_O and N_I be a partition of its nodes (places and transitions). The elements of N_O will be called *output transitions / places* while the elements of N_I will be called *input transitions / places*. We give an inductive definition of a *static handshake port* $a = \langle G, N_O, N_I\rangle$ as follows:

– (Basic cases) N_O and N_I contain no transition;
– (Inductive cases) let $a' = \langle G', N'_O, N'_I\rangle$ be a static port:
  • (input prefixing) given a place $p \in N'_I$ with no outgoing arcs, a is obtained from a' by adding an input transition t and an arc from p to t;

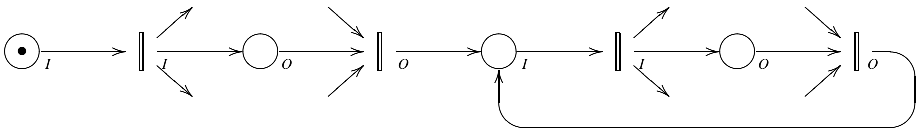


Fig. 3. A passive port (I is for input and O is for output)

- (output prefixing) given a place $p \in N'_O$ with no outgoing arcs and a place $p' \in N'_I$ with exactly one outgoing arc, a is obtained from a' by adding an output transition t, an arc from p to t and an arc from t to p';
- (alternation) given a place $p \in N'_O$ and a transition $t \in N'_I$ with no outgoing arcs, a is obtained from a' by adding an arc from t to p.

Let a' be a static port and let p be a place of (the Petri net of) a' such that if p is an input place, p has an outgoing arc. Let a be the net obtained from a' either by adding one token to p or by keeping a' with no tokens, then a is a *handshake port*. Moreover, if a is as a' (no tokens) or if p is an output place we say that a is an *active port*, otherwise we say that a is a *passive port*.

Let G be a Petri net, G' a subgraph of G and $a = \langle G', N_O, N_I \rangle$ a port s.t. :

– a place of G' may only be connected to transitions of G';
– a transition $t \in N_O$ of G' may only have outgoing arcs to places of G';
– a transition $t \in N_I$ of G' may only have incoming arcs from places of G'.

then a is a *handshake port of* G. Figure 3 shows an example of a passive handshake port of some net. The arrows without source or target indicate the way the port may connect to the rest of the net. The statical structure imposes alternation between the firings of input and output transitions. It is also ensured that, if an input place may ever contain a token, then it must have an outgoing arc to an input transition (receptiveness). Finally, by allowing a port to contain several input and output transitions we are able to model each event separately. For instance, two distinct input transitions may connect differently to the rest of the net, thus providing different resources.

Definition 8. *The pair $H = \langle G_H, Ports_H \rangle$ is a* handshake Petri net (hpn) *just when G_H is a Petri net and $Ports_H$ a set of disjoint handshake ports of G_H.*

Let $H = \langle G_H, Ports_H \rangle$ and let t (p) be a transition (place) of G_H. Then t (p) is *internal* of H if it is neither of input nor of output (in some port a of H).

4.2 Composition

A *linkage* between an active port $a \in Ports_H$ and a passive port $b \in Ports_H$ of an hpn $H = \langle G_H, Ports_H \rangle$ is the hpn $L(H, a, b) = \langle G_{H,a,b}, Ports_H \backslash \{a, b\} \rangle$, where $G_{H,a,b}$ is the net obtained by adding two fresh places p_1 and p_2 to G_H and arcs from each output transition of a to p_1, from p_1 to each input transition of b, from each output transition of b to p_2 and from p_2 to each input transition of a.

We call *link* of $L(H, a, b)$, denoted $link(H, a, b)$, the graph consisting of the graphs of a and b plus p_1, p_2 and any arc connecting them to transitions of a or b. Figure 4 shows an example of link between an active port (left) and a passive port (right).

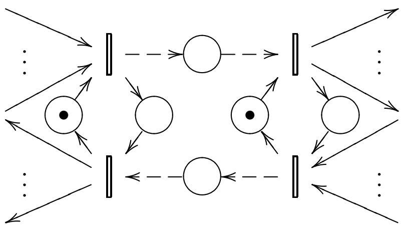


Fig. 4. Example of composition of ports

Definition 9. *Let $H_1 = \langle G_1, Ports_1 \rangle$ and $H_2 = \langle G_2, Ports_2 \rangle$ be two handshake Petri nets. Let $\{a_1, \ldots a_n\} \subseteq Ports_1 \cup Ports_2$ be a set of active handshake ports and let $\{b_1, \ldots b_n\} \subseteq Ports_1 \cup Ports_2$ be a set of passive handshake ports, such that for $1 \leq i \leq n$, $a_i \in Ports_1$ if and only if $b_i \in Ports_2$. Then:*

$$H_1 \parallel_{\{(a_1,b_1),\ldots(a_n,b_n)\}} H_2$$
$$=$$
$$L(\ldots L(\langle G_1 + G_2, Ports_1 \cup Ports_2 \rangle, a_1, b_1), \ldots a_n, b_n)$$

is the composition of H_1 with H_2 by linking the pairs $(a_1, b_1), \ldots (a_n, b_n)$.

It is easy to see that the composition of two hpns is well-defined and associative.

4.3 Observational Properties of hpns

Let us label a ($\bar{a}$) each input (output) transition of port a, for any $a \in Ports_H$, and τ each internal transition of H. We write $H \xrightarrow{l} H'$ (which reads H l-reduces to H') if a transition labeled l is enabled in G_H and its firing leads to the hpn H'. Seen as labeled transition systems, hpns naturally inherit many definitions that we gave for CHC. Two of these are strong ($\sim$) and weak ($\approx$) bisimilarity. The generality of these definitions allow us to go as far as saying that an hpn is (weakly or strongly) bisimilar to a handshake configuration.

Another inherited definition is that of the function HL, which is identical to the one we gave in Section 3.4 for a handshake configuration. However, we still need to adapt HS. Let $H = \langle G_H, Ports_H \rangle$ be an hpn and let $d_H : Ports_H \to \{!, ?\}$ be the function which maps each port a to !, when a is active in H, or to ?, when it is passive. We define $HS(H) = \langle Ports_H, d_H \rangle$.

Proposition 10. **[6]** *Let H be a handshake Petri net, then $HL(H)$ is a handshake language on the handshake structure $HS(H)$.*

Proposition 11. **[6]** *Let σ be a handshake language on a handshake structure $\langle P, d \rangle$. Then there is an hpn H_σ such that $HS(H_\sigma) = \langle P, d \rangle$ and $HL(H_\sigma) = \sigma$.*

Note that, for the last result, the net H_σ may contain an infinite number of internal places and transitions. This is unavoidable as languages are in general not recursive and thus not finitely representable. Figure 5 shows two simple examples of handshake protocols: *OR* and *MIX* described in Section 3.3.

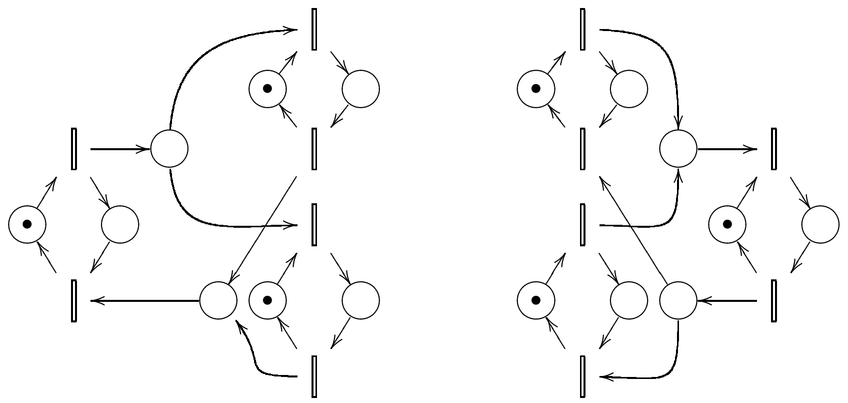


Fig. 5. *OR* (left) and *MIX* (right) handshake components

5 Full Abstraction and Definability

In this section we relate the calculus CHC with its Petri nets model . We only sketch the constructions, the detailed proofs are available on the extended version [5].

5.1 Full Abstraction

Let M be a handshake configuration, such that $M \rhd \Gamma$. We can assume $M = \langle P, S\rangle$ and define the hpn $[\![M]\!]_\Gamma$ by cases of P. The definitions for open configurations are identical, $[\![\wr P, S \wr]\!]_\Gamma = [\![\langle P, S\rangle]\!]_\Gamma$.

Let $P = \mathbf{0}$. Then $\Gamma = !a$ for some channel a. Now, define a port which contains a single output place p holding a token and call it a as the channel. Let G be the Petri net which contains p plus an internal place q, labeled r, for each $r \in S$, where q contains as many tokens as there are occurrences of r in S. Then $[\![M]\!]_\Gamma$ is the hpn $\langle G, \{a\}\rangle$.

Let $P = a^{\{r_1,\dots r_n\}}.P'$. Then the last applied typing rule is (inpref), then $\Gamma = ?a$ and $P' \rhd !a$. Let $[\![\langle P', S\rangle]\!]_{!a} = \langle G', Ports'\rangle$. By construction, $Ports' = \{a\}$ where a is an active port. Let p' be the place of a with a token. Let's extend a by adding a fresh input place p, a fresh input transition t labeled a and arcs from p to t and from t to p', then by removing the token from p' and putting a token into p. Finally let's add arcs from t to any place labeled r_i, for $1 \le i \le n$. If any of these places does not exist yet, add it anew. We thus obtain a graph G. Then $[\![M]\!]_\Gamma = \langle G, \{a\}\rangle$. The case of the output prefix is dual.

Let $P = \mathbf{Rec}P'$. Then $\Gamma = !a$ and $P' \rhd !a$, for some channel a. Let $[\![\langle P', S\rangle]\!]_{!a} = \langle G', Ports'\rangle$. By construction, $Ports' = \{a\}$ where a is also the active port associated to channel a. Let p be the place of a which holds a token. If p has an incoming arc or if any other place in a has two incoming arcs, $[\![\langle P, S\rangle]\!]_\Gamma = [\![\langle P', S\rangle]\!]_{!a}$. Otherwise a must contain a place p' with no outgoing arcs, by construction. Note that both p and p' must be output places, also by construction. Then replace p and p' by a place q obtained by "joining" them. In particular, q must be the new source of p's outgoing arc and the new target of p''s incoming arc. Call G the graph so obtained. Then $[\![\langle P, S\rangle]\!]_\Gamma = \langle G, \{a\}\rangle$.

Let $P = (P' \mid P'') \setminus \Delta$. We construct $[\![\langle P, S\rangle]\!]_\Gamma$ in three steps. First let $[\![\langle P', S\rangle]\!]^{\Delta}_{\Gamma_1}$ be obtained from $[\![\langle P', S\rangle]\!]_{\Gamma_1}$ by renaming each port a such that $(a, a) \in \Delta$, as $a^!$ when $\Gamma_1(a) =!$ and as $a^?$ when $\Gamma_1(a) =?$. Define analogously $[\![\langle P'', S\rangle]\!]^{\Delta}_{\Gamma_2}$. Then let $\langle G', Ports\rangle = [\![\langle P', S\rangle]\!]^{\Delta}_{\Gamma_1} \parallel_{\{(a^!,a^?)\mid a\in\Delta\}} [\![\langle P'', S\rangle]\!]^{\Delta}_{\Gamma_2}$. Then, for any two distinct places p and p' of G' labeled by the same resource r do the following: substitute p and p' by a single place also labeled by r, having all the arcs of both p and p'; note also that by construction, p and p' contained the same number of tokens k, then put k tokens in the new place as well. Let G be the net so obtained. Then $[\![M]\!]_\Gamma = \langle G, Ports\rangle$.

The semantics is well defined and fully abstract with respect to weak bisimilarity:

Lemma 12. *Let M be a configuration such that $M \rhd \Gamma$. Then $[\![M]\!]_\Gamma$ as defined above is a handshake Petri net and $M \approx [\![M]\!]_\Gamma$.*

Theorem 13 (Full Abstraction). *Let M and M' be two configurations such that $M \rhd \Gamma$ and $M' \rhd \Gamma'$. Then $M \approx M' \iff [\![M]\!]_\Gamma \approx [\![M']\!]_{\Gamma'}$.*

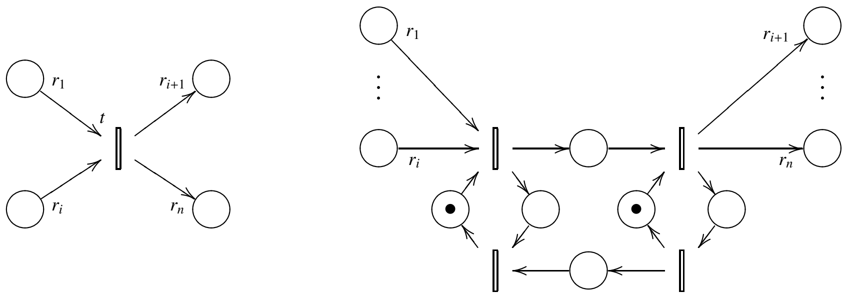
5.2 Definability

For each finite hpn there is a weakly bisimilar handshake configuration:

Theorem 14 (Definability). *Let $H = \langle G, Ports\rangle$ be a handshake Petri net. Then there are a closed handshake configuration M and a handshake type Γ, such that $M \rhd \Gamma$ and $[\![M]\!]_\Gamma \approx H$.*

We present here a simplified construction of the configuration associated to H. The idea is that each port a of the net H can be modeled by a thread $Proc(a, H)$, inductively on the structure of the port.

Each internal transition t is first *unfolded* as a link between two ports and then associated to a process. Let t have incoming arcs from internal places labeled $r_1, \dots r_i$ and outgoing arcs to internal places labeled $r_{i+1}, \dots r_n$. Then t is unfolded as follows:



where $r_1, \dots r_n$ are place labels. . Then let H be a hpn, $u(H)$ is the hpn obtained from H by unfolding each of its internal transitions. It can be shown that $H \approx u(H)$.

For each internal transition t, let l_t be a fresh label associated to it. Consider the following process.

$$Proc(t, H) = (\mathbf{Rec}\ \bar{l}_{t\{r_1,\dots r_i\}}.l_t.\mathbf{0} \mid l_t^{\{r_{i+1},\dots r_n\}}.\mathbf{Rec}\ \bar{l}_t.l_t^{\{r_{i+1},\dots r_n\}}.\mathbf{0}) \setminus \{l_t\}$$

Then we define

$$Proc(H) = Proc(a_1, H) \mid \ldots \mid Proc(a_n, H) \mid Proc(t_1, H) \mid \ldots \mid Proc(t_m, H)$$

where $a_1, \ldots a_n$ are the ports of H and $t_1, \ldots t_m$ are the internal transitions of H. Then let $Conf(H) = \langle Proc(H), S_H \rangle$, where S_H is the multiset of labels of internal places of H with a token and a label appears in S_H as many times as the number of tokens in the corresponding place. Finally let $ch(Ports)$ be the set of names of ports in $Ports$, then $\Gamma_H : ch(Ports) \to \{!, ?\}$ is the function which associates ! to its active ports' names and ? to its passive ports' names. It can be shown that $[\![Conf(H)]\!]_{\Gamma_H} \approx u(H)$. Thus $[\![Conf(H)]\!]_{\Gamma_H} \approx H$.

6 Conclusions

We presented the calculus CHC which describes handshake protocols of communication. We have given it an lts semantics and a Petri nets semantics in terms of handshake Petri nets. We have shown that every finite handshake Petri net corresponds to a closed configuration of the calculus.

We have argued that a branching semantics is necessary to understand handshake protocols, as the trace model cannot properly define composition. The calculus and the two semantics provide the necessary framework to formally study handshake protocols.

Our original aim had been to devise a typing system for CCS, that would ensure the handshake discipline. After many attempts, we came to believe that such typing system would be cumbersome, if at all possible. Consider in particular the MIX component defined in Section 4.3. It is essentially characterized by a form of *inclusively disjunctive* causality: The request sent on the active port causally depends on either of the requests received on the passive ports. However, if both are received, it cannot be established which of the two is actually the cause. This is in contrast with the fact that CCS can be modeled using safe occurrence nets (which correspond to stable event structures), where this kind of causality cannot be represented. Therefore we decided to use a second, different form of communication, in the form of shared resources.

As usual, a result opens new directions to inspect. We would like to characterize the handshake languages that are described by CHC configurations (and thus by finite nets). We would also like to study restrictions on the language or the types to characterize behavioral classes of protocols: deterministic, positional, free choice, etc. It would be interesting to extend the calculus with mobility primitives, like the ones of the π-calculus, and study its expressive power. We also would like to use the calculus to specify and prove the correctness of specific protocols.

Acknowledgments. We thank Roberto Amadio, Catuscia Palamidessi and Frank Valencia for their useful comments. We thank an anonymous referee for spotting a mistake in a previous version of our proofs. The first author would like to thank Simona Ronchi della Rocca, Felice Cardone, Pierre-Louis Curien for their valuable inputs and support. The second author acknowledges the support of the ANR project ParSec: ANR-06-SETI-010-02.

References

1. Abramsky, S., Gay, S., Nagarajan, R.: Interaction categories and the foundations of types concurrent programming. In: Proc. of the 1994 Marktoberdorf Summer School on Deductive Program Design, pp. 35–113. Springer, Heidelberg (1996)
2. Arun-Kumar, S., Hennessy, M.: An efficiency preorder for processes. Acta Informatica 29(9), 737–760 (1992)
3. Bardsley, A.: Balsa: an asynchronous circuit synthesis system. Master's thesis. Department of Computer Science, University of Manchester (1998)
4. Fossati, L.: Modeling the Handshake Protocol for Asynchrony. PhD thesis, Dip. di Informatica, Univ. di Torino & Lab. Preuves Programmes et Systèmes (PPS), Univ. Paris 7 (2009)
5. Fossati, L., Varacca, D.: The calculus of handshake configurations (extended version) (2008), http://www.di.unito.it/~fossati/
6. Fossati, L., Varacca, D.: A Petri net model of handshake circuits. In: Proc. of First International Workshop on Interactive Concurrency Experience, ICE 2008. ENTCS. Elsevier, Amsterdam (to be published, 2008), http://www.di.unito.it/~fossati/
7. Gelernter, D.: Generative communication in Linda. ACM Transactions on Programming Languages and Systems 7(1), 80–112 (1985)
8. Ghica, D.R.: Geometry of synthesis: a structured approach to VLSI design. In: Proc. of POPL 2007, pp. 363–375. ACM Press, New York (2007)
9. http://www.handshakesolutions.com/
10. Josephs, M., Udding, J., Yantchev, Y.: Handshake algebra. Technical Report SBU-CISM-93-1, School of Computing, Information Systems and Mathematics, South Bank University, London (1993)
11. Kobayashi, N., Pierce, B., Turner, D.: Linearity and the π-calculus. ACM Transactions on Programming Languages and Systems 21(5), 914–947 (1999)
12. Mackie, I.: The geometry of interaction machine. In: Proc. of POPL 1995, pp. 198–208. ACM Press, New York (1995)
13. Milner, R.: Communication and Concurrency, 2nd edn. Prentice-Hall, Englewood Cliffs (1991)
14. Milner, R., Sangiorgi, D.: Techniques for "weak bisimulation up-to". Revised version of a paper that appeared in: Cleaveland, W.R. (ed.) CONCUR 1992. LNCS, vol. 630. Springer, Heidelberg (1992); available on Sangiorgi's webpage
15. Reisig, W.: Petri Nets: An Introduction. Monographs in Theoretical Computer Science. An EATCS Series, vol. 4. Springer, Heidelberg (1985)
16. Roscoe, A.W.: Unbounded nondeterminism in CSP. Journal of Logic and Computation 3(2), 131–172 (1993); Previously appeared in Two Papers on CSP, tech. monograph PRG-67, Oxford University Computing Laboratory(July 1988)
17. Udding, J.: Classification and Composition of Delay-Insensitive Circuits. PhD thesis, Department of Math. and C.S., Eindhoven University of Technology, Eindhoven (1984)
18. Van Berkel, K.: Handshake Circuits: an Asynchronous Architecture for VLSI Design. Cambridge International Series on Parallel Computation, vol. 5. Cambridge University Press, Cambridge (1993)
19. Yoshida, N., Honda, K., Berger, M.: Linearity and bisimulation. In: Nielsen, M., Engberg, U. (eds.) FOSSACS 2002. LNCS, vol. 2303, pp. 417–433. Springer, Heidelberg (2002)

On the Expressive Power of Restriction and Priorities in CCS with Replication

Jesús Aranda[1,⋆], Frank D. Valencia[2], and Cristian Versari[3]

[1] LIX École Polytechnique and Universidad del Valle Colombia
[2] CNRS and LIX École Polytechnique
[3] Università di Bologna

Abstract. We study the expressive power of restriction and its interplay with replication. We do this by considering several syntactic variants of $\mathrm{CCS}_!$ (CCS with replication instead of recursion) which differ from each other in the use of restriction with respect to replication. We consider three syntactic variations of $\mathrm{CCS}_!$ which do not allow the use of an unbounded number of restrictions: $\mathrm{CCS}_!^{-!\nu}$ is the fragment of $\mathrm{CCS}_!$ not allowing restrictions under the scope of a replication. $\mathrm{CCS}_!^{-\nu}$ is the restriction-free fragment of $\mathrm{CCS}_!$. The third variant is $\mathrm{CCS}_{!+pr}^{-!\nu}$ which extends $\mathrm{CCS}_!^{-!\nu}$ with Phillips' priority guards.

We show that the use of unboundedly many restrictions in $\mathrm{CCS}_!$ is necessary for obtaining Turing expressiveness in the sense of Busi et al [8]. We do this by showing that there is no encoding of RAMs into $\mathrm{CCS}_!^{-!\nu}$ which preserves and reflects convergence. We also prove that up to failures equivalence, there is no encoding from $\mathrm{CCS}_!$ into $\mathrm{CCS}_!^{-!\nu}$ nor from $\mathrm{CCS}_!^{-!\nu}$ into $\mathrm{CCS}_!^{-\nu}$. As lemmata for the above results we prove that convergence is decidable for $\mathrm{CCS}_!^{-!\nu}$ and that language equivalence is decidable for $\mathrm{CCS}_!^{-\nu}$. As corollary it follows that convergence is decidable for restriction-free CCS. Finally, we show the expressive power of priorities by providing an encoding of RAMs in $\mathrm{CCS}_{!+pr}^{-!\nu}$.

1 Introduction

As for other language-based formalisms (e.g., logic, formal grammars, λ-calculus, etc) a fundamental part of the research in *process calculi* involves the study of the expressiveness of *syntactic* fragments or variants of a given process calculus.

Process calculi provide a *language* in which the structure of *terms* represents the structure of processes together with a *transition* relation to represent computational steps. Consider for example CCS [15]. The parallel composition term $P|Q$, which is built from the terms P and Q represents the process that results from the parallel execution of the processes P and Q. The *restriction* $(\nu x)P$ represents a process P with a private/local/restricted/bound resource x–e.g., a location, a link, or a name. Processes with infinite behaviour are often specified with *recursive expressions* of the form $\mu X.P$ which behaves as $P[\mu X.P/X]$, i.e., P with the (free) occurrences of X replaced by $\mu X.P$. A transition semantics dictates that if P may have a transition into P' by performing an action α, written $P \xrightarrow{\alpha} P'$, then $P \mid Q \xrightarrow{\alpha} P' \mid Q$ and if α does not involve x also $(\nu x)P \xrightarrow{\alpha} (\nu x)P'$.

⋆ The work of Jesús Aranda has been supported by COLCIENCIAS (Instituto Colombiano para el Desarrollo de la Ciencia y la Tecnología "Francisco José de Caldas") and INRIA Futurs.

Classifying Criteria. One natural approach to comparing expressiveness of two given process calculus variants is by comparing them wrt some standard process equivalence $\asymp$. If there exists a computable function (*encoding*) $[\![\cdot]\!]$ from the terms of a variant $\mathcal{C}$ into the terms of another variant $\mathcal{C}'$ such that for every P in $\mathcal{C}$ we have $P \asymp [\![P]\!]$, we say that $\mathcal{C}'$ is *at least as expressive as* $\mathcal{C}$ up to $\asymp$.

Another way of classifying the variants of a given calculus is by considering the complexity or decidability of a fundamental property of processes. For example, the decidability of *convergence* (the existence of a terminating computation) or *divergence* (the existence of a non-terminating computation).

The CCS$_!$ Calculus. The CCS$_!$ calculus [7] is a variant of CCS which instead of using recursive expressions to specify infinite behaviour uses processes of form $!P$. The replicated process $!P$ can be thought of as abbreviating the parallel composition $P \mid P \mid \ldots$ of an unbounded number of P processes. In [8] it is shown that CCS$_!$ is less expressive than CCS wrt (weak) bisimilarity. It is also shown that convergence is undecidable for CCS$_!$ while divergence, unlike for CCS, is decidable.

Turing Expressiveness and Convergence in CCS$_!$. A remarkable expressiveness result in [8] states that, in spite of its being less expressive than CCS, CCS$_!$ is in fact Turing powerful. This is done by encoding Random Access Machines (RAMs) [16]. The fundamental property of the encoding is that it *preserves (and reflects) convergence*; i.e., the RAM converges if and only if its encoding converges.

The CCS$_!$ encoding of RAMs in [8] uses an unbounded number of restrictions arising from having restriction operators *under the scope* of a replication operator as for example in $!(\nu x)P$. Similarly, the CCS encoding of RAMs in [7] involves also an unbounded number of restrictions arising from having restrictions under the scope of recursive expressions as for example in $\mu X.(\nu x)(P \mid X)$. One then may wonder if the generation of unboundedly many names is necessary for Turing Expressiveness.

This Work. In this paper we study the expressiveness of restriction and its interplay with replication. We do this by considering two syntactic fragments of CCS$_!$, namely CCS$_!^{-!\nu}$ and CCS$_!^{-\nu}$ which differ from CCS$_!$ in the occurrences of restriction under the scope of replication. These fragments and a variant of CCS$_!$, CCS$_{!+pr}^{-!\nu}$, as well as our classification criteria are described and motivated below.

Although different in nature, our work was inspired by the study of decidable classes (wrt satisfiability) of formulae involving the occurrence of existential quantifiers *under the scope* of universal quantification. E.g., Skolem showed that the class of formulae of the form $\forall y_1 \ldots y_n \exists z_1 \ldots z_m F$, where F is quantifier-free formula, is undecidable while from Gödel we know that its subclass $\forall y_1 y_2 \exists z_1 \ldots z_m F$ is decidable [5].

The CCS$_!$ Variants. As explained above CCS$_!$ allows processes with restriction under the scope of replication and hence they can generate an unbounded number of restricted names. In order to allow only processes with a number of restricted names bounded by their size, we consider CCS$_!^{-!\nu}$ which represents the CCS$_!$ fragment which does not allow restrictions under the scope of replications. To illustrate the expressiveness of CCS$_!^{-!\nu}$ take for example $P = (\nu k)(\nu u)(\bar{k} \mid !(k.a.(\bar{k} \mid \bar{u})) \mid k.!(u.b))$ which uses only two restricted names. The reader familiar with CCS can verify that the set of (maximal) finite sequences of visible actions performed by P corresponds to the *context-free* language $a^n b^n$. A similar but slightly more complex example involves a $\mathrm{CCS}_!^{-!\nu}$ process with only five restricted names whose set of (maximal) finite sequences of visible actions corresponds to the *context-sensitive* language $a^n b^n c^n$–see [2].

Now, one may wonder whether a process that uses only a number of restricted names bounded by its size, can be encoded, perhaps by introducing some additional non-restricted names, into one which uses none. For this purpose we shall also consider the *restriction-free* fragment of $\mathrm{CCS}_!$, which shall denote as $\mathrm{CCS}_!^{-\nu}$.

Finally, we may also wonder whether some other natural process construct can replace the use in $\mathrm{CCS}_!$ of unboundedly many restrictions in achieving Turing expressiveness. For this purpose we shall consider $\mathrm{CCS}_{!+pr}^{-!\nu}$ which is $\mathrm{CCS}_!^{-!\nu}$ extended with Phillips' *priority guards* construct [17].

Classifying Criteria. Our main comparison criteria for the above variants are the decidability of *convergence* and their relative expressiveness wrt *failures equivalence* [6,15].

As mentioned before, convergence is a fundamental property of processes and its preservation and reflection are also fundamental properties of the encoding of RAMs in $\mathrm{CCS}_!$. Furthermore, we choose it over divergence because the former is undecidable for $\mathrm{CCS}_!$ while the latter is already known to be decidable for $\mathrm{CCS}_!$.

Failures equivalence is a well-established notion of process equivalence and we choose it over other equivalences because of its sensitivity to convergence. In fact unlike failures equivalence, other standard equivalences for observable behaviour such as weak-bisimilarity, must testing, trace equivalence and language equivalence may actually equate a convergent process with a non-convergent one. This claim about sensitivity to convergence will be shown later on in the paper (Section 3) once we fix our notation.

Contributions. Our main contributions are the following:

– We show that convergence is decidable for $\mathrm{CCS}_!^{-!\nu}$ and thus that there is no (computable) encoding, which preserves and reflects convergence, of RAMs using only a bounded number of restricted names. We do this by encoding $\mathrm{CCS}_!^{-!\nu}$ into Petri Nets. Thus convergence is also decidable for the fragment of CCS with no restrictions within recursive expressions, here refered to as $\mathrm{CCS}^{-\mu\nu}$, because of the convergence preserving and reflecting encoding into $\mathrm{CCS}_!^{-!\nu}$ given in [12].
– We show that, up to failures equivalence, $\mathrm{CCS}_!$ is strictly more expressive than $\mathrm{CCS}_!^{-!\nu}$ and, similarly, that $\mathrm{CCS}_!^{-!\nu}$ is strictly more expressive than $\mathrm{CCS}_!^{-\nu}$. Thus up to failures equivalence, we cannot encode a process with an unbounded number of restrictions into one with a bounded number of restrictions, nor one with a bounded number of restrictions into a restriction-free process.
– We show that adding Phillips' priority guards to $\mathrm{CCS}_!^{-!\nu}$ renders the resulting calculus capable of encoding RAMs. Furthermore, unlike the encoding into $\mathrm{CCS}_!$ and just like the encoding into CCS, the encoding of RAMs into $\mathrm{CCS}_{!+pr}^{-!\nu}$ preserves and reflects both convergence and divergence. This bears witness to the expressive power of Phillips' priority guards.

The classification of the various fragments mentioned above are summarized in Figure 1. The undecidability of convergence and decidability of divergence for $\mathrm{CCS}_!$ as well as the undecidability of both divergence and convergence for CCS were shown in [7,8]. The other results are derived from the work here presented.

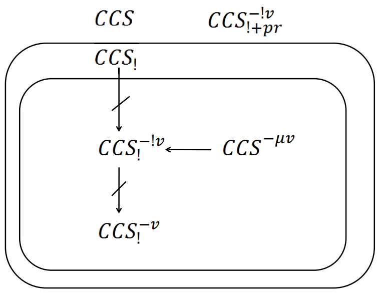


Fig. 1. A (crossed) arrow from $\mathcal{C}$ to $\mathcal{C}'$ represents the (non) existence of an encoding from $\mathcal{C}$ into $\mathcal{C}'$ preserving and reflecting failures equivalence. Convergence is/isn't decidable for each $\mathcal{C}$ in/outside the inner rectangle. Divergence is/isn't decidable for each $\mathcal{C}$ in/outside the outer rectangle.

2 The Calculi

CCS processes can perform actions or synchronize on them. These actions can be either offering port *names* for communication, or the so-called *silent* action τ. We presuppose a countable set $\mathcal{N}$ of port *names*, ranged over by $a, b, x, y \ldots$ and their primed versions. We then introduce a set of *co-names* $\overline{\mathcal{N}} = \{\overline{a} \mid a \in \mathcal{N}\}$ disjoint from $\mathcal{N}$. The set of *labels*, ranged over by l and l', is $\mathcal{L} = \mathcal{N} \cup \overline{\mathcal{N}}$. The set of *actions* Act, ranged over by α and β, extends $\mathcal{L}$ with a new symbol τ. Actions a and $\overline{a}$ are thought of as *complementary*, so we decree that $\overline{\overline{a}} = a$. We also decree that $\overline{\tau} = \tau$.

The CCS$_!$ processes [7] are defined as in CCS except that recursive expressions are replaced by replication expression of the form $!P$.

Definition 1. *Processes in CCS$_!$ are built from names by the following syntax:*

$$P, Q \ldots := 0 \mid \alpha.P \;\big|\; P + Q \;\big|\; P \mid Q \;\big|\; (\nu a)P \;\big|\; !P \tag{1}$$

Convention 1. *We use $\Sigma_{i \in I} P_i$ where $I = \{i_1 \ldots i_n\}$, to denote $P_{i_1} + \ldots + P_{i_n}$., the order of the summands being insignificant. We use $\Pi_{i \in I} P_i$, where $I = \{i_1 \ldots i_n\}$, to denote $P_{i_1} \mid \ldots \mid P_{i_n}$. Both $\Sigma_{i \in I} P_i$ and $\Pi_{i \in I} P_i$ are assumed to be 0 if $I = \emptyset$. The names a and $\bar{a}$ in P are said to be* bound *in $(\nu a)P$. The* bound names *of P, $bn(P)$, are those with a bound occurrence in P, and the* free names *of P, $fn(P)$, are those with an not bound occurrence in P. The set of names of P, $n(P)$, is then given by $fn(P) \cup bn(P)$. We use $(\nu a_1 \ldots a_n)P$ as an abbreviation of $(\nu a_1)(\nu a_2) \ldots (\nu a_n)P$.*

Process expressions are endowed with meaning by using the labeled transitions of the form $P \xrightarrow{\alpha} Q$ which intuitively says that P may perform α and evolve into Q. Similarly, $P \overset{s}{\Longrightarrow} Q$, where $s \in \mathcal{L}^*$, means that P can evolve into Q after zero or more transitions labeled with the elements of s without considering τ moves. Formally,

Definition 2. *The labeled transition relation $\xrightarrow{\cdot}$ is given by the rules in Table 1. Define $\overset{s}{\Longrightarrow}$, with $s = \alpha_1 \ldots . \alpha_n \in \mathcal{L}^*$, as $(\xrightarrow{\tau})^* \xrightarrow{\alpha_1} (\xrightarrow{\tau})^* \ldots (\xrightarrow{\tau})^* \xrightarrow{\alpha_n} (\xrightarrow{\tau})^*$. For the empty sequence $s = \epsilon$, $\overset{s}{\Longrightarrow}$ is defined as $(\xrightarrow{\tau})^*$.*

Table 1. The labeled semantics of CCS$_!$

PREF $\dfrac{}{\alpha.P \xrightarrow{\alpha} P}$	RES $\dfrac{P \xrightarrow{\alpha} P'}{(\nu\, a)P \xrightarrow{\alpha} (\nu\, a)P'}$ if $\alpha \notin \{a, \overline{a}\}$
SUM$_1$ $\dfrac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$	SUM$_2$ $\dfrac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$
PAR$_1$ $\dfrac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$	PAR$_2$ $\dfrac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'}$
COM $\dfrac{P \xrightarrow{l} P' \quad Q \xrightarrow{\overline{l}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$	REP $\dfrac{P \mid !P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'}$

Intuition and Basic Ideas. We shall now give some intuition and state some conventions on process expressions. We shall concentrate on the expressions $!P$ and $(\nu x)R$ and their interplay because they are central to our work.

The process $P \mid Q$ represents the parallel execution of P and Q. Intuitively, $P \mid Q$ may perform either an action performed by P, an action performed by Q, or if P and Q perform complementary actions (and thus synchronize), a τ action. Thus, $a.P \mid \bar{a}.Q \xrightarrow{a} P \mid \bar{a}.Q$, $a.P \mid \bar{a}.Q \xrightarrow{\bar{a}} a.P \mid Q$, and $a.P \mid \bar{a}.Q \xrightarrow{\tau} P \mid Q$.

The restriction process $(\nu a)P$ behaves as P except that it can offer neither a nor $\bar{a}$ to its environment. One may think of $(\nu a)P$ as a *local declaration of name* a in P and thus can be used to restrict the possible synchronization (interactions) of a process. For example, $(\nu a)(a.P \mid \bar{a}.Q)$ may not perfom a or $\bar{a}$ though –crucially– it may perform a τ action resulting from the synchronization over the actions a and $\bar{a}$. Thus $(\nu a)(a.P \mid \bar{a}.Q) \not\xrightarrow{\alpha}$ for $\alpha \in \{a, \bar{a}\}$ but $(\nu a)(a.P \mid \bar{a}.Q) \xrightarrow{\tau} (\nu a)(P \mid Q)$.

Replication is the only means to specify infinite behaviour in CCS$_!$. The replication $!P$ behaves as $P \mid P \mid \ldots \mid !P$; unboundedly many $P's$ in parallel.

We can use restriction under the scope of replication to specify an unbounded number of local names. It should be clear from the intuition above that $(\nu a)P$ should behave exactly as $(\nu b)(P[b/a])$ where b is not free in P and $P[b/a]$ is the process that results from replacing in P every free occurrence of a with b renaming bound names wherever needed to avoid capture (α-conversion). Thus, for example $!(\nu a)P$ can be viewed as

$$(\nu a_1)P[a_1/a] \mid (\nu a_2)P[a_2/a] \mid \ldots \mid !(\nu a)P$$

thus allowing the declaration of unboundedly many different restricted names $a_1, a_2, \ldots$. As previously mentioned in the introduction, this sort of unbounded generation of restricted names appears to be crucial in the encodings of Turing-powerful formalisms.

The Sub-calculi. To understand the above-mentioned interplay between restriction and replication we shall consider two calculi which arise from syntactic constraints over CCS$_!$. Namely, CCS$_!^{-!\nu}$ and CCS$_!^{-\nu}$.

Definition 3 (CCS$_!^{-!\nu}$ and CCS$_!^{-\nu}$). *The processes of CCS$_!^{-!\nu}$ are those CCS$_!$ processes which do not have occurrences of a process of the form $(\nu x)P$ within a process* *of the form* $!R$*. The processes of* $CCS_!^{-\nu}$ *are those* $CCS_!$ *processes with no occurrences of processes of the form* $(\nu x)P$*.*

3 Convergence, Failures and Related Notions

In this section we shall introduce the notions we shall consider as classifying criteria, namely *convergence* and *failures equivalence*, as well as some other related notions.

Following [4], we say that a process generates a sequence of non-silent actions s if it can perform the actions of s in a finite maximal sequence of transitions. More precisely:

Definition 4 (Sequence and language generation). *The process* P generates *a sequence* $s \in \mathcal{L}^*$ *if and only if there exists* Q *such that* $P \overset{s}{\Longrightarrow} Q$ *and* $Q \not\xrightarrow{\alpha}$ *for any* $\alpha \in Act$*. Define the language of (or generated by) a process* P*,* $L(P)$*, as the set of all sequences* P *generates. We say that* P *and* Q *are language equivalent, written* $P \sim_L Q$*, iff* $L(P) = L(Q)$*.*

We recall the notion of *failure* following [15]. We need the following notion:

Definition 5. *We say that* P *is stable iff* $P \not\xrightarrow{\tau}$*.*

Intuitively we say that a pair $\langle e, L\rangle$, with $e \in \mathcal{L}^*$ and $L \subseteq \mathcal{L}$, is failure of P if P can perform e and thereby reach a state in which no further action (including τ) is possible if the environment will only allow actions in L.

Definition 6 (Failures). *A pair* $\langle e, L\rangle$*, where* $e \in \mathcal{L}^*$ *and* $L \subseteq \mathcal{L}$*, is a failure of* P *iff there is* P' *such that: (1)* $P \overset{e}{\Longrightarrow} P'$*, (2)* $P' \not\xrightarrow{l}$ *for all* $l \in L$*, and (3)* P' *is stable.*

Define $Failures(P)$ *as the set of failures of a process* P*. We say that* P *and* Q *are* failures equivalent*, written* $P \sim_F Q$ *iff* $Failures(P) = Failures(Q)$*.*

We recall the notions of convergence and divergence following [7,8]. Intuitively, a process converges if it can reach a stable process after a sequence of τ moves. A process is deemed divergent iff it can perform an infinite sequence of τ moves.

Definition 7 (Convergence and Divergence). *We say that* P *is convergent iff there is a stable process* Q *such that* $P(\xrightarrow{\tau})^*Q$*. We say that* P *is divergent iff* $P(\xrightarrow{\tau})^\omega$*, i.e., there exists an infinite sequence* $P = P_0 \xrightarrow{\tau} P_1 \xrightarrow{\tau} \ldots$*.*

We conclude this section by stating relations between the above notions which we shall use in the following sections.

3.1 Some Basic Properties of Failures

We claimed in the introduction that unlike other standard notions such as weak bisimilarity, must testing and trace equivalence, failures equivalence never equates a convergent process with a non-convergent one. In fact,

Proposition 1. *Suppose that* $P \sim_F Q$*. Then* P *is convergent iff* Q *is convergent.*

To justify the rest of the above claim, take $P = \tau.!a.0$ and $P' = !\tau.0$. Clearly P converges but P' does not, however they are both language equivalent. Now take $Q = \tau.!\tau.0 + \tau.0$ and $Q' = !\tau.0$. Thus Q converges but Q' does not. It can be verified that Q and Q' are equated by these standard equivalences.

We shall use the fact that failures equivalence implies language equivalence.

Proposition 2. $\sim_F \subseteq \sim_L$.

4 Decidability of Convergence in $\mathrm{CCS}_!^{-!\nu}$

In this section we show the decidability of convergence for $\mathrm{CCS}_!^{-!\nu}$ by a reduction to the same problem for a fragment of Petri Nets.

4.1 Convergence-Invariant Properties in Fragments of $\mathrm{CCS}_!^{-!\nu}$

Notice that decidability of convergence in $\mathrm{CCS}_!^{-!\nu}$ can be reduced to the decidability of convergence in $\mathrm{CCS}_!^{-\nu}$.

Proposition 3. *For every P in $\mathrm{CCS}_!^{-!\nu}$ one can effectively construct a $\mathrm{CCS}_!^{-\nu}$ process P', such that P converges if only if P' converges.*

Proof. (Outline) First α-convert P so that each bound name in P is replaced with a unique bound name. Then remove from the resulting process each occurrence of a "(νx)". Let P' be the resulting restriction-free process. One can verify that P' converges iff P converges. □

Consequently, in what follows we reduce the convergence problem for $\mathrm{CCS}_!^{-\nu}$ to convergence problem in Petri nets.

In order to simplify the reduction to Petri Nets, we shall consider the fragment $\mathrm{CCS}_{s!}^{-\nu}$ of those $\mathrm{CCS}_!^{-\nu}$ processes in which replication can only be applied to prefix or summation processses.

Proposition 4. *For every $\mathrm{CCS}_!^{-\nu}$ process P, one can effectively construct a $\mathrm{CCS}_{s!}^{-\nu}$ process Q such that P converges iff Q converges.*

Proof. (Outline) Systematically replace in P every occurrence of the form $!!R$ with $!R$, $!(Q|Q')$ with $!Q \mid !Q'$, and $!0$ with 0. The resulting process converges iff P converges.

Finitely Branching Transition System. In order to prove the decidability of convergence, we shall make use of an alternative but equivalent definition (up to failures equivalence) of the transition relation for $\mathrm{CCS}_!$. The equivalent definition can be obtained by replacing Rule Rep in Table 1 with the rules in Table 2.

One can verify that the resulting transition relation is finitely-branching. This is essential for being able to provide an effective Petri net construction for any given $\mathrm{CCS}_{s!}^{-\nu}$ process.

Table 2.

REPL$_1$	$\dfrac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P}$	REPL$_2$	$\dfrac{P \xrightarrow{\alpha} P' \quad P \xrightarrow{\overline{\alpha}} P''}{!P \xrightarrow{\tau} P' \mid P'' \mid !P}$

4.2 The Reduction to Petri Nets

Here we shall provide a (Unlabelled Place/Transition) Petri Net semantics for CCS$_{s!}^{-\nu}$ which considers only the τ moves. For these Petri Nets convergence is decidable [11].

Definition 8 (Petri Nets). *A Petri net is a 3-tuple (S, T, m_0), where S is a set of places, T is a set of transitions $\mathcal{M}_{fin}(S) \times \mathcal{M}_{fin}(S)$ with $\mathcal{M}_{fin}(S)$ being a finite multiset of S called a marking. The (non-empty) multiset m_0 is the initial marking; for each place $s \in S$, there are $m_0(s)$ tokens.*

A transition (c, p) is written in the form $c \Longrightarrow p$. A transition is enabled at a marking m if $c \subseteq m$. The execution of the transition produces the marking $m' = (\ m \setminus c\ ) \oplus p$ (where $\setminus$ and $\oplus$ are the difference and the union operators on multisets). This is written as $m \rhd m'$. If no transition is enable at m we say that m is a dead marking.

We say that the Petri net (S, T, m_0) converges *iff there exists a dead marking m' such that $m_0(\rhd)^* m'$.*

Intuitively, we will associate to each CCS$_{s!}^{-\nu}$ a Petri net so that:

- Places are identified as syntactic components reachable from P,
- Markings are descriptions of processes reachable from P through τ-actions. The places and tokens in the marking represent different syntactic components and their number of occurrences in the process described.
- Transitions represent the τ-actions enabled to be performed at certain process. Input places correspond to the components in the process involved in the τ-action and Output places are the components to be enabled once the τ-action has been executed.

Given a Petri net for P the elements of $Sub(P)$ below will be the syntactic components represented by places in the Petri net.

Definition 9. *Define $Sub(P)$, where $P \in$ CCS$_{s!}^{-\nu}$, as $Sub(0) = \{0\}$, $Sub(\Sigma_{i\in I} P_i) = \{\Sigma_{i\in I} P_i\} \cup (\bigcup_{i\in I} Sub(P_i))$, $Sub(\alpha.P) = \{\alpha.P\} \ \cup\ Sub(P)$, $Sub(!P) = \{!P\} \ \cup\ Sub(P)$, $Sub(P \mid Q) = Sub(P) \ \cup\ Sub(Q)$.*

$Sub(P)$ denotes the set all null, replicated, summation, prefix processes occurring in P. Since a process P may have several parallel occurrences of an element in $Sub(P)$ we use a multi-set $Occur(P)$ take into account its number of occurrences.

Definition 10 (Occurrence). *Let $P \in$ CCS$_{s!}^{-\nu}$. The multiset of processes which occur in P, $Occur(P)$, is given by the following rule: $Occur(P) = Occur(Q) \ \oplus\ Occur(R)$ if $P = Q \ \mid\ R$ else $Occur(P) = \{P\}$. Furthermore, we say that Q is an occurrence of a process P if and only if $Q \in Occur(P)$.*

$Occur(P)$ associates to a $\text{CCS}_{s!}^{-\nu}$ process P the multiset of its immediate parallel components (occurrences) and will be identified as the marking of P in the Petri net.

We are now ready to define our Petri net encoding of $\text{CCS}_{s!}^{-\nu}$ processes.

Definition 11 (Nets for $\text{CCS}_{s!}^{-\nu}$). *Given a $\text{CCS}_{s!}^{-\nu}$ process P, we define its Petri net $N_P = (S, T, m_0)$ where $S = \{Q \mid Q \in Sub(P)\}$, $m_0 = Occur(P)$ and $T = T_1 \cup T_2$ where: $T_1 = \{\{P\} \Longrightarrow Occur(P') | \ P \xrightarrow{\tau} P'\}$ and $T_2 = \{\{P, Q\} \Longrightarrow Occur(P') \oplus Occur(Q') | P \xrightarrow{\alpha} P' \text{ and } Q \xrightarrow{\overline{\alpha}} Q'\}$.*

Clearly, given P, N_P can be effectively constructed—here we use the finite-branching nature of the alternative transition semantics in Section 4.1.

Roughly speaking, the set of transitions T represents the possible τ moves to be performed and the initial marking $Occur(P)$ is the one which identifies the process P. In particular:

- T_1 : this type of transition reflects a τ move coming from one of the components, it is referred as P, going to the process P'. Notice as a token representing P is consumed and the tokens representing P', there might be more than one component, are added, in this way the transition reflects the evolution from the component P into the process P' .
- T_2 : this type of transition reflects the τ-actions resulting from the synchronisation of two components P and Q, as a result of the synchronisation the processes P' and Q' are reached, in this case, a token associated to both P and another one associated to Q are consumed, the tokens representing P' and Q' are added.

Wee can now state the correctness of the encoding of $\text{CCS}_{!}^{-\nu}$ into Petri nets.

Lemma 1 (Convergence-invariance property between $\text{CCS}_{!}^{-\nu}$ and Petri nets) *For any $\text{CCS}_{s!}^{-\nu}$ process P, P converges if and only if the Petri net N_P converges.*

Since convergence is decidable for Petri nets [11], we conclude from the above lemma and our effective construction of Petri Nets that convergence is also decidable for $\text{CCS}_{s!}^{-\nu}$. Thus, from Propositions 3 and 4, we obtain the following corollary.

Theorem 2. *Convergence is a decidable property for $\text{CCS}_{!}^{-!\nu}$ processes.*

5 Decidability of Language Equivalence in $\text{CCS}_{!}^{-\nu}$

We now prove that decidability of language equivalence for $\text{CCS}_{!}^{-\nu}$. The crucial observation is that up to language equivalence every occurrence of a replicated process $!R$ in a $\text{CCS}_{!}^{-\nu}$ process can be replaced with $!\tau.0$ if R can perform at least an action, otherwise it can be replaced with 0. More precisely, let $P[Q/R]$ the process that results from replacing in P every occurrence of R with Q.

Proposition 5. *Let P be a $\text{CCS}_{!}^{-\nu}$ process and suppose that $!R$ occurs in P. Then $L(P) = L(P[Q/!R])$ where $Q = !\tau.0$ if there exists α s.t., $R \xrightarrow{\alpha}$ else $Q = 0$.*

Given any R in CCS$_!$ one can effectively decide whether there exists α such that $R \xrightarrow{\alpha}$ (This can be proven using the alternative finitely-branching presentation of the transition relation in Section 4.1). We can then use the above proposition for proving the following statement.

Lemma 2. *Let P be a CCS$_!^{-\nu}$ process. One can effectively construct a process P' such that $L(P) = L(P')$ and P' is either $!\tau.0$ or a replication-free CCS$_!^{-\nu}$ process.*

Proof. (Sketch.) Notice that we can use systematically Proposition 5 to transform any CCS$_!^{-\nu}$ process P into an language equivalent process Q whose replicated occurrences are all of the form $!\tau.0$. Now a $!\tau.0$ can occur either in a parallel composition, a summation or prefix process. Observe that (1) $P \mid !\tau.0 \sim_L !\tau.0$, (2) $!\tau.0 \mid P \sim_L !\tau.0$, (3) $\alpha.!\tau.0 \sim_L !\tau.0$, (4) $P + !\tau.0 \sim_L P$, (5) $!\tau.0 + P \sim_L P$. One can apply (1-5) from left to right to systematically transform Q into the process P' as required in the lemma. □

From the above lemma, we conclude that every CCS$_!^{-\nu}$ process can be effectively transformed into a language equivalent finite-state process. Hence,

Theorem 3. *Given P and Q in CCS$_!^{-\nu}$, the question of whether $L(P) = L(Q)$ is decidable.*

6 Impossibility Results for Failure-Preserving Encodings in CCS$_!$, CCS$_!^{-!\nu}$ and CCS$_!^{-\nu}$

In this section, we shall state the impossibility results about the existence of computable encodings from CCS$_!$ into CCS$_!^{-!\nu}$ and from CCS$_!^{-!\nu}$ into CCS$_!^{-\nu}$ which preserve and reflect failures equivalence. The separation results follow from our previous decidability results and the undecidability results in the literature.

The non-existence of failure-preserving encoding from CCS$_!$ into CCS$_!^{-!\nu}$ follows from Proposition 1, Theorem 2 and the undecidability of convergence for CCS$_!$ [8].

Theorem 4. *There is no computable function $[\![\cdot]\!]$: CCS$_!$ $\rightarrow$ CCS$_!^{-!\nu}$ s.t $[\![P]\!] \sim_F P$.*

To state the non-existence of failure-preserving encoding from CCS$_!^{-!\nu}$ into CCS$_!^{-\nu}$ we appeal to the undecidability of language equivalence for BPP processes [9,14]. BPP processes form a subset of restriction-free CCS processes. Now we can use the encoding of [12] to transform a restriction-free CCS processes into CCS$_!^{-!\nu}$—the encoding is correct up to failures equivalence (see [3]). We can therefore conclude, with the help of Proposition 2, that language-equivalence for CCS$_!^{-!\nu}$ processes is undecidable.

Proposition 6. *Given P and Q in CCS$_!^{-!\nu}$, the problem of whether $P \sim_L Q$ is undecidable.*

From the above proposition, the decidability of language equivalence for CCS$_!^{-\nu}$ (Theorem 3) and Propositions 1 and 2 we can conclude the following.

Theorem 5. *There is no computable function $[\![\cdot]\!]$: CCS$_!^{-!\nu}$ $\rightarrow$ CCS$_!^{-\nu}$ s.t. $[\![P]\!] \sim_F P$.*

Remark 1. We can use the encoding of [12] to transform any CCS process which uses no restriction within recursive expression into a failures equivalent CCS$_!^{-!\nu}$ process [3]. Thus, from Proposition 1 and Theorem 2 we can conclude that convergence is also decidable for CCS with no restriction within recursive expressions.

7 Expressiveness of Priorities

In this section we add Phillips' priority guards [17] to $\mathrm{CCS}_!^{-!\nu}$. We shall refer to the resulting calculus as $\mathrm{CCS}_{!+pr}^{-!\nu}$. This calculus corresponds to Phillips' Calculus of Priority Guards (CPG) with replication rather than recursion and no restrictions within the scope of replication—hence it cannot use an unbounded number of restrictions.

We show that $\mathrm{CCS}_{!+pr}^{-!\nu}$ turns out to be Turing powerful in the sense of Busi et al [8] (i.e., preserving and reflecting convergence), thus bearing witness to computational expressiveness of priority guards. Recall that from the previous sections $\mathrm{CCS}_!$, and even CCS, cannot encode Turing machines, in the sense above, without using an unbounded number of restrictions (Theorem 2 and Remark 1).

7.1 $\mathrm{CCS}_{!+pr}^{-!\nu}$

In CPG there are two sets of names: N which corresponds to the set of names used to represent the visible actions in $\mathrm{CCS}_!^{-!\nu}$ and a set of priority names U. Each set has a set of complementary actions : $\bar{N}$ and $\bar{U}$, where $\mathit{Std} = N \cup \bar{N}$ (the standard visible actions), $\mathit{Pri} = U \cup \bar{U}$ (the priority actions), $\mathit{Vis} = \mathit{Std} \cup \mathit{Pri}$ (the visible actions), and $\mathit{Act} = \mathit{Vis} \cup \tau$ (all actions). We let $a, b, \ldots$ range over $N \cup U$; u, v, $\ldots$ over Pri; $\lambda, \ldots$ over Vis; and α, β, $\ldots$ over Act. Also S, T, $\ldots$ range over finite subsets of Vis, and U, V, $\ldots$ over finite subsets of Pri.

The syntax of processes in $\mathrm{CCS}_{!+pr}^{-!\nu}$ is like that of $\mathrm{CCS}_!^{-\nu}$, except for the summations which now take the form of *priority-guarded* summations: $\Sigma_{i\in I} S_i : \alpha_i.P_i$ where I and each S_i are finite. The meaning of the priority guard $S : \alpha$ is that α can only be performed if the environment does not offer any action in $\bar{S} \bigcap \mathit{Pri}$ (see [17] for details).

Labelled Transition and Offers

We recall the set $\mathit{off}(P)$ of "higher priority" actions "offered" by P.

Definition 12 (Offers). *Let P be a $\mathrm{CCS}_{!+pr}^{-!\nu}$ process and $u \in \mathit{Pri}$. The relation P off u (P offers u) is given by the rules in Table 3. We define $\mathit{off}(P) = \{u \in \mathit{Pri} : P \ \mathit{off} \ u\}$. Finally, we say that P* eschews *U iff $\mathit{off}(P) \cap \bar{U} = \emptyset$.*

The transitions are conditional on offers from the environment. Intuitively, a transition of the form $P \xrightarrow{\alpha}_U P'$ means that P may perform α as long as the environment does not offer $\bar{u}$ for any $u \in U$ (i.e., the environment "eschews" U). E.g. $a : b.P \xrightarrow{b}_{\{a\}} P$ means that $a : b.P$ may perform b as long as the environment does not offer $\overline{a}$. Thus,

Table 3.

$$M + S : u.P + N \ \mathit{off} \ u \quad \mathit{if} \quad u \notin S$$

$$\frac{P \ \mathit{off} \ u}{P \mid Q \ \mathit{off} \ u} \qquad \frac{Q \ \mathit{off} \ u}{P \mid Q \ \mathit{off} \ u}$$

$$\frac{P \ \mathit{off} \ u}{(\nu a) \ P \ \mathit{off} \ u} \ \mathit{if} \ a \neq \mathit{name}(u) \qquad \frac{P \ \mathit{off} \ u}{!P \ \mathit{off} \ u}$$

Table 4. An operational semantics for CCS$^{-!\nu}_{!+pr}$

$$\text{SUM } M + S : \alpha.P + N \xrightarrow{\alpha}_{S\cap Pri} P \ \ \mathit{if} \ \ \alpha \in S \cap Pri$$

$$\text{PAR}_1 \ \frac{P \xrightarrow{\alpha}_U P' \ \ Q \ \ \mathit{eschews} \ \ U}{P \mid Q \xrightarrow{\alpha}_U P' \mid Q} \qquad \text{PAR}_2 \ \frac{Q \xrightarrow{\alpha}_U Q' \ \ P \ \ \mathit{eschews} \ \ U}{P \mid Q \xrightarrow{\alpha}_U P \mid Q'}$$

$$\text{REACT} \ \frac{P \xrightarrow{\lambda}_{U_1} P' \quad Q \xrightarrow{\overline{\lambda}}_{U_2} Q' \ \ P \ \ \mathit{eschews} \ \ U_2 \ \ Q \ \ \mathit{eschews} \ \ U_1}{P \mid Q \xrightarrow{\tau}_{U_1 \cup U_2} P' \mid Q'}$$

$$\text{REP} \ \frac{P \mid !P \xrightarrow{\alpha}_U P'}{!P \xrightarrow{\alpha}_U P'}$$

$$\text{RES} \ \frac{P \xrightarrow{\alpha}_U P' \ \ \mathit{if} \ \ \alpha \ \ \notin \{a, \overline{a}\}}{(\nu\, a)P \xrightarrow{\alpha}_{U - \{a,\overline{a}\}} (\nu\, a)P'}$$

$$\text{SUM} \frac{}{\Sigma_{i\in I}\alpha_i.P_i \xrightarrow{\alpha_j} P_j} \mathit{if}\, j \ \in \ I$$

$a : b.P \mid \overline{b}.Q$ could evolve into $P \mid Q$ however the system $a : b.P \mid \overline{b}.Q \mid \overline{a}$ could not evolve into $P \mid Q \mid \overline{a}$ as the presence of $\overline{a}$ prevents the execution of b and thus the τ-action resulting from $(b, \overline{b})$ communication. This capability of processes to test the presence or the absence of a channel ready to be performed will be fundamental to represent the test for $zero$ in the encoding of RAMs in CCS$^{-!\nu}_{!+pr}$ presented in the next subsection. Transitions are determined by the rules in Table 4.

Convention 6. *We write $P \xrightarrow{\alpha}_{\emptyset} P'$ as $P \xrightarrow{\alpha} P'$ (i.e., α is not constrained on offers from the environment thus corresponding to a standard CCS$_!$ transition). Thus, the notions of divergence and convergence for CCS$^{-!\nu}_{!+pr}$ are obtained as in Definition 7 by replacing $\xrightarrow{\tau}$ with $\xrightarrow{\tau}_{\emptyset}$.*

7.2 Encoding RAMs in CCS$^{-!\nu}_{!+pr}$

A RAM can be seen as a program consisting of a finite sequence of instructions labeled with numbers $(1 : I_1), (2 : I_2) \ldots, (m : I_m)$ which modify the values of a finite set of non-negative registers r_1, ..., r_n. The instructions are either $Incr(r_j)$ which adds 1 to the contents of register r_j and goes to the next instruction, or $DecJump(r_j, l)$ which tests the register r_j value, if it is not zero then decreases it by 1 and goes to the next instruction, otherwise jumps to instruction l.

A state of a RAM is given by $(i, c_1, \ldots, c_n)$ where i is the program counter indicating the next instruction to be executed, and $c_1, \ldots, c_n$ are the current values of the registers $r_1, \ldots, r_n$ (resp.). Given a program its computation proceeds by executing the instructions as indicated by the program counter. The execution stops when the program counter reaches the value $m + 1$ where m is the label of the last instruction; in this case we say that the program terminates.

The Encoding. A register r_j with value c_j (written $r_j : c_j$) is modeled by a corresponding number of processes of the form $\overline{u}_j$.

$$[\![(r_j : c_j)]\!] \;=\; \prod_{1}^{c_j} \overline{u}_j$$

The program counter is modeled with the *absence* of $\overline{p}_i$ (i.e., the action p_i is eschwed by the encoding) indicating that the i-th instruction is the next to be executed. The initial value of the program counter is 1 so by using $\prod_{i=2}^{m+1} \overline{p}_i$ we indicate the absence of $\overline{p}_1$.

The increasing instruction is modelled with a process $[\![(i : Incr(r_j))]\!]$ which is guarded by a τ-action which is only performed when there is an absence of $\overline{p}_i$.

$$[\![(i : Incr(r_j))]\!] \;= !(\{p_i\} : \tau.(\overline{p_i} \mid p_{i+1} \mid \overline{u}_j))$$

Once activated, the instruction increases the register r_j by offering $\overline{u}_j$, and goes to the next instruction by both disallowing the current one by offering $\overline{p}_i$ and allowing the next one by performing p_{i+1} so that $\overline{p}_{i+1}$ can be consumed.

The decreasing instruction is defined similarly. In addition we consider the absence of $\overline{u}_j$ to test for zero.

$$[\![(i : DecJump(r_j, l))]\!] = !(\{p_i\} : u_j.(\overline{p_i} \mid p_{i+1}))|!(\{p_i, u_j\} : \tau.(\overline{p_i} \mid p_l))$$

The encoding of a RAM is given below. Without loss of generality we assume that initially the RAM has all its registers set to zero and its program counter is 1.

Definition 13. *Let R be a RAM with program instructions* $(1 : I_1), \ldots, (m : I_m)$ *and registers* $r_1, \ldots, r_n$. *We define its encoding into CCS*$^{-!\nu}_{!+pr}$ *as:*

$$[\![R]\!] = (\nu p_1, \ldots, p_{m+1}, u_1, \ldots, u_n)(\textstyle\prod_{i=1}^{m} [\![(i : I_i)]\!] \mid \prod_{i=1}^{n} [\![(r_i : 0)]\!] \mid \prod_{i=2}^{m} \overline{p}_i)$$

The correctness of the encoding is stated as follows (see the extended version [3]).

Theorem 7. *Let R be a RAM with program instructions* $(1 : I_1), \ldots, (m : I_m)$ *and registers* $r_1, \ldots, r_n$. *Then, R terminates if and only if $[\![R]\!]$ converges. Furthermore, R does not terminate if and only if $[\![R]\!]$ diverges.*

As corollary we obtain that convergence and divergence are *undecidable* for CCS$^{-!\nu}_{!+pr}$.

8 Concluding Remarks and Related Work

The most closely related works are [7,8] and they were already discussed in the introduction. In [12] the authors study replication and recursion in CCS focusing on the role of restriction and name scoping. In particular they show that CCS$_!$ is equivalent to CCS with recursion with *static scoping*. The standard CCS is shown to have *dynamic* scoping precisely because the use of restriction within recursive definitions. However, if no restriction appears within recursive expressions then there is no distinction between static and dynamic scoping. Hence, if no restriction is allowed within recursive expressions then we know from [12] that CCS can be encoded in CCS$_!$, without restriction under replication, while preserving and reflecting convergence. As for the other direction, clearly $\nu X.(P|X)$ behaves as $!P$. Nevertheless, if recursion is required to be *prefix guarded*, it is not clear how to produce an encoding which preserves and reflects convergence—without appealing to the decidability results for CCS$_!$ here presented.

Consider e.g., $E = \nu X.(P|\alpha.X)$ and $!P$. If $\alpha = \tau$ then E does not converge and $!P$ may—take $P = a.0$. If $\alpha \neq \tau$ then E may converge and $!P$ may not—take $P = \tau.0$.

The authors in [10] also pointed out the role of restriction in the expressiveness of CCS. They showed that strong bisimilarity is decidable for restriction-free CCS, in contrast with the undecidability result for CCS [18]. It is not clear to us how to relate strong bisimilarity with convergence or failures equivalence.

The authors of [1] studied a fragment of the asynchronous π-calculus with restricted forms of bound name generation. A closely related result in of [1] is the decidability of the control reachability problem for restriction-free asynchronous π-calculus. This implies the decidability of the same problem for the restriction-free fragment of asynchronous $\mathrm{CCS}_!$ (i.e., only 0 can be prefixed with an output action). It is not obvious how to relate control reachability to failures equivalence or convergence. Also it is not clear how to encode our $\mathrm{CCS}_!$ fragment into restriction-free asynchronous $\mathrm{CCS}_!$.

In [13] a Petri net semantics is proposed for a subset of CCS without restriction and with guarded choice. Also in [18] it was shown that the subset studied in [13] can not be extended significantly. These works also presuppose guarded recursion in their fragments which seem crucial for their Petri net constructions. We do not restrict our Petri net construction to guarded sums. Furthermore, as explained above, it is not clear how to translate $\mathrm{CCS}_!$ into CCS with guarded recursion while preserving convergence.

In [20] the authors show the decidability of convergence for a restriction-free calculus for the compositional description of chemical systems, called *CFG* which seems closely related to CCS. The calculus, however, presupposes guarded summation and guarded recursion and thus, as argued before, it is not clear how to encode $\mathrm{CCS}_!$ into such a calculus while preserving convergence.

In [17] it was shown that priorities add expressive power to CCS by modelling electoral systems that cannot be modelled in CCS. Also [19] studies two process algebras enriched with different priority mechanisms. The work reveals the gap between the two prioritised calculi and the two non prioritised ones by modeling electoral systems. Both works state the impossibility of the existence of an encoding subject to certain structural requirements such as homomorphism wrt parallel composition and name invariance. Our derived impossibility result about the non-existence of convergent preserving encodings makes no structural assumptions on the encodings. Finally, we claim that our expressivity results involving priorities are also held by using other priority approaches as they provide the capability of processes to know if another process is ready to perform a synchronisation on some channel or not.

Acknowledgments. We would like to thank Ian Phillips and Uwe Nestmann for insightful discussions on the topics here studied. We are also grateful to Jorge A. Pérez and the anonymous reviewers for their remarks and suggestions.

References

1. Amadio, R., Meyssonnier, C.: On decidability of the control reachability problem in the asynchronous $\pi-$calculus. Nordic Journal of Computing 9(2) (2002)
2. Aranda, J., Giusto, C.D., Nielsen, M., Valencia, F.D.: CCS with replication in the chomsky hierarchy: The expressive power of divergence. In: Shao, Z. (ed.) APLAS 2007. LNCS, vol. 4807, pp. 383–398. Springer, Heidelberg (2007)

3. Aranda, J., Valencia, F., Versari, C.: On the expressive power of restriction and priorities in ccs with replication. Technical report, l'École Polytechnique (2008), http://www.lix.polytechnique.fr/Labo/Jesus.Aranda/publications/trccs.pdf
4. Baeten, J.C.M., Bergstra, J.A., Klop, J.W.: Decidability of bisimulation equivalence for processes generating context-free languages. J. ACM 40(3), 653–682 (1993)
5. Borger, E., Gradel, E., Gurevich, Y.: The Classical Decision Problem. Springer, Heidelberg (1994)
6. Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A theory of communicating sequential processes. Journal of the ACM 31(3), 560–599 (1984)
7. Busi, N., Gabbrielli, M., Zavattaro, G.: Replication vs. recursive definitions in channel based calculi. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 133–144. Springer, Heidelberg (2003)
8. Busi, N., Gabbrielli, M., Zavattaro, G.: Comparing recursion, replication, and iteration in process calculi. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 307–319. Springer, Heidelberg (2004)
9. Christensen, S.: Decidability and Decomposition in Process Algebras. PhD thesis, Edinburgh University (1993)
10. Christensen, S., Hirshfeld, Y., Moller, F.: Decidable subsets of ccs. Comput. J. 37(4), 233–242 (1994)
11. Esparza, J., Nielsen, M.: Decidability issues for petri nets. Technical report, BRICS RS-94-8 (1994)
12. Giambiagi, P., Schneider, G., Valencia, F.D.: On the expressiveness of infinite behavior and name scoping in process calculi. In: Walukiewicz, I. (ed.) FOSSACS 2004. LNCS, vol. 2987, pp. 226–240. Springer, Heidelberg (2004)
13. Goltz, U.: Ccs and petri nets. In: Guessarian, I. (ed.) LITP 1990. LNCS, vol. 469, pp. 334–357. Springer, Heidelberg (1990)
14. Hirshfeld, Y.: Petri nets and the equivalence problem. In: Meinke, K., Börger, E., Gurevich, Y. (eds.) CSL 1993. LNCS, vol. 832, pp. 165–174. Springer, Heidelberg (1994)
15. Milner, R.: Communication and Concurrency. Prentice-Hall, Englewood Cliffs (1989)
16. Minsky, M.: Computation: finite and infinite machines. Prentice-Hall, Englewood Cliffs (1967)
17. Phillips, I.: Ccs with priority guards. J. Log. Algebr. Program. 75(1), 139–165 (2008)
18. Taubner, D.: Finite representation of CCS and TCSP programs by automata and Petri nets. In: Taubner, D.A. (ed.) Finite Representations of CCS and TCSP Programs by Automata and Petri Nets. LNCS, vol. 369. Springer, Heidelberg (1989)
19. Versari, C., Busi, N., Gorrieri, R.: On the expressive power of global and local priority in process calculi. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 241–255. Springer, Heidelberg (2007)
20. Zavattaro, G., Cardelli, L.: Termination problems in chemical kinetics. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 477–491. Springer, Heidelberg (2008)

Normal Bisimulations in Calculi with Passivation

Sergueï Lenglet[1], Alan Schmitt[2], and Jean-Bernard Stefani[2]

[1] Université Joseph Fourier, Grenoble, France
[2] INRIA Grenoble-Rhône-Alpes, France

Abstract. Behavioral theory for higher-order process calculi is less well developed than for first-order ones such as the π-calculus. In particular, effective coinductive characterizations of barbed congruence, such as the notion of *normal bisimulation* developed by Sangiorgi for the higher-order π-calculus, are difficult to obtain. In this paper, we study bisimulations in two simple higher-order calculi with a passivation operator, that allows the interruption and thunkification of a running process. We develop a normal bisimulation that characterizes barbed congruence, in the strong and weak cases, for the first calculus which has no name restriction operator. We then show that this result does not hold in the calculus extended with name restriction.

1 Introduction

Motivation. A natural notion of behavioral equivalence for process calculi is *barbed congruence*. Informally, two processes are barbed-congruent if they behave in the same way (i.e., have the same reductions and the same observables) when placed in similar, but arbitrary, contexts. Due to this quantification on contexts, barbed congruence is unwieldy to use in proofs of equivalence, or to serve as a basis for automated verification tools. One is thus lead to study coinductive characterizations of barbed congruence, typically in the form of bisimilarity relations. For first-order process calculi, such as the π-calculus and its variants, the resulting behavioral theory is well developed, and one can in general readily define bisimilarity relations that characterize barbed congruence.

For higher-order process calculi, the situation is less satisfactory. Simple higher-order calculi, such as HOπ [10,11], have a well-studied behavioral theory. For HOπ, Sangiorgi has defined *context* and *normal* bisimilarity relations, which both are sound with respect to barbed congruence (i.e. are included in barbed congruence) and sometimes complete (i.e. they contain barbed congruence), leading to a full characterization. However, context bisimilarity still involves some quantification over test contexts. For instance, when assessing the equivalence of two processes which consist only of the output of a message on a communication channel a, context bisimilarity needs to consider every interacting system that is capable of doing an input on channel a. Normal bisimilarity improves context bisimilarity by requiring only a single test context. E.g., in the case of two emitting processes, as above, normal bisimilarity only requires to compare the behavior of the two processes when placed in parallel with a single particular receiving process. Furthermore, context and normal bisimilarities characterize barbed congruence both in the strong case (where internal steps are observable), and in the weak case (where internal steps are not observable).

Unfortunately, HOπ is not expressive enough to faithfully model concurrent systems with dynamic reconfiguration or strong mobility capabilities. For instance, a running HOπ process cannot be stopped, which prevents the faithful modeling of process failures, of online process replacement, or of strong process mobility. It is for this reason that we have seen the emergence of process calculi with (forms of) *process passivation*. Process passivation allows a named process to be stopped and its state captured at any time during its execution. The Kell calculus [13] and Homer [5] are examples of higher-order process calculi with passivation. The behavioral theory of these calculi is less understood than the one for HOπ, whose proof techniques and relations do not carry over. No sound and complete characterization of barbed congruence has been found in the weak case for these calculi. Importantly, no relation akin to normal bisimilarity has been developed.

Contributions. To pinpoint issues that arise in the development of a behavioral theory for higher-order calculi with passivation, and to show that they arise from the interplay between passivation and restriction, we consider in this paper two calculi with passivation, which are simpler than both Homer and the Kell calculus, and which differ merely in the presence of restriction. The first one, called HOP, extends HOcore with passivation and sum. HOcore is a minimal higher-order concurrent calculus without restriction that has recently been studied in [7]. As a first contribution, we show that HOP admits a sound and complete form of normal bisimulation, in both the strong and weak cases. The second calculus, called HOπP, extends HOπ with passivation. As a second contribution, we show that with HOπP a large class of tests does not suffice to build a sound normal bisimulation. This casts some doubt as to whether a suitable notion of normal bisimilarity, that is with finite testing, can be found for HOπP, and therefore for Homer and the Kell calculus.

Summary. In Section 2, we define HOπP and recall the previous works on behavioral equivalences in the Kell calculus and Homer. We define in Section 3 a sound and complete normal bisimilarity for HOP. We show in Section 4 that this relation is not suitable for HOπP. We discuss related work in Section 5, and Section 6 concludes the paper. The paper only contains proof sketches for some results. Complete proofs can be found in [8].

2 Bisimulations in HOπP

Studying proof techniques for establishing contextual equivalence in calculi such as Homer and the Kell calculus has been the main motivation for this work. Instead of working directly in one of these calculi, we consider a simpler calculus, HOπP (for Higher-Order π with Passivation), which extends the HOπ calculus studied in [11] with a passivation operator, and which exhibits the same technical difficulties encountered in Homer and Kell.

Variables, names:	**Agents:**
$m, n, \overline{m}, \overline{n}, \ldots$: first-order names and co-names	P, Q, R, S: Processes
$a, b, \overline{a}, \overline{b}, \ldots$: higher-order names and co-names	F, G: Abstractions
x, y: first or higher-order names	C, D: Concretions
X, Y: process variables	A, B: Agents

Actions:
τ: Internal action
$l \in \{m, \overline{m}, \ldots\} \cup \tau$: first-order actions
$\alpha \in \{m, \overline{m}, \ldots\} \cup \tau \cup \{a, \overline{a}, \ldots\}$: first or higher-order actions
Syntax:
$P ::= \mathbf{0} \mid X \mid P \,|\, P \mid l.P \mid a(X)P \mid \overline{a}\langle P\rangle P \mid \nu x.P \mid\ !P \mid a[P]$

Fig. 1. Meta-Variables and Syntax of HOπP

2.1 Syntax and Transition Semantics

Meta-variables and syntax of HOπP are given Figure 1. We add localities $a[P]$ to the HOπ constructs. These are passivation units. As long as no passivation occurs, a locality $a[P]$ is a transparent evaluation context: the process P may evolve and communicate freely with processes outside of a, independently of their position in the locality tree. At any time, passivation may be triggered and the process $a[P]$ becomes a concretion $\langle P\rangle\mathbf{0}$. Passivation may thus occur as an internal τ step only if there is a receiver on a ready to receive the contents of the locality. The receiver may then choose to spawn, forward, or discard the process.

Name restriction $\nu x.P$ makes the name x private to process P. We write $\text{bn}(P)$ (resp. $\text{fn}(P)$) for the bound names (resp. free names) of P. Message input $a(X)P$ binds the variable X in P. We write $\text{fv}(P)$ for the free process variables of a process P. A process P is said to be closed if $\text{fv}(P) = \emptyset$. We identify processes up to α-conversion of names and variables. Structural congruence $\equiv$ is the smallest congruence verifying the following laws.

$$P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad P \mid Q \equiv Q \mid P \qquad P \mid \mathbf{0} \equiv P \qquad \nu x.\nu y.P \equiv \nu y.\nu x.P$$

$$\nu x.\mathbf{0} \equiv \mathbf{0} \qquad !P \equiv P \mid !P \qquad \nu x.(P \mid Q) \equiv P \mid \nu x.Q \ (x \notin \text{fn}(P))$$

We now give an informal account of the labeled transition semantics (LTS) $\xrightarrow{\alpha}$ of the calculus. There are three kinds of transitions: first-order transition, higher-order input, and higher-order output. In a first-order transition $P \xrightarrow{l} Q$, processes may evolve towards processes by an internal action τ, or by a first-order input or output (labeled by the corresponding name or co-name). In the higher-order input $P \xrightarrow{a} F = (X)Q$, P evolves towards an *abstraction* F, which states that it may receive a process R on name a to continue as $Q\{R/X\}$. In the higher-order output $P \xrightarrow{\overline{a}} C = \nu\widetilde{x}.\langle R\rangle S$, P evolves towards a *concretion* C, which states that it may send process R on name a and continue as S, and the scope of names $\widetilde{x}$ (such that $\widetilde{x} \subseteq \text{fn}(R)$) has to be expanded to encompass the recipient of R. We call the set $\widetilde{x}$ the bound names of C, written $\text{bn}(C)$. A higher-order communication takes place when a concretion interacts with an abstraction. We define a pseudo-application operator $\bullet$ between F and C above by $F \bullet C \stackrel{\Delta}{=} \nu\widetilde{x}.(Q\{R/X\} \mid S)$ (with $\mathrm{fn}(Q) \cap \widetilde{x} = \emptyset$).

Let the set of *agents*, written A, be the set of all processes, abstractions, and concretions. We extend restriction, parallel composition, and locality to all agents. Let $F = (X)P$ be an abstraction, we then have $\nu x.F = (X)\nu x.P$ and $a[F] = (X)a[P]$. If $X \notin \mathrm{fv}(Q)$, then $F \mid Q = (X)(P \mid Q)$ and $Q \mid F = (X)(Q \mid P)$. Let $C = \nu\widetilde{y}.\langle Q\rangle R$ be a concretion and $x \notin \widetilde{y}$. If $x \in \mathrm{fn}(Q)$, then $\nu x.C = \nu x, \widetilde{y}.\langle Q\rangle R$, otherwise $\nu x.C = \nu\widetilde{y}.\langle Q\rangle \nu x.R$. If $\widetilde{y} \cap \mathrm{fn}(P) = \emptyset$, then $C \mid P = \nu\widetilde{y}.\langle Q\rangle(R \mid P)$ and $P \mid C = \nu\widetilde{y}.\langle Q\rangle(P \mid R)$. If $a \notin \widetilde{y}$, then $a[C] = \nu\widetilde{y}.\langle Q\rangle a[R]$.

The LTS rules are given in Figure 2, with the exception of the symmetric rules for LTS-Par, LTS-FO, and LTS-HO.

According to rule LTS-Loc, a locality $a[P]$ becomes a concretion when P outputs a message and becomes a concretion. Since the bound names of a concretion are extruded "by need" to encompass the receiving process, their scope may thus cross locality boundaries.

Remark 1. Passivation in HOπP can be seen as objective, as it requires a receiver on the locality's name to result in a silent τ step.

$$l.P \xrightarrow{l} P \quad \text{LTS-Prefix} \qquad\qquad a(X)P \xrightarrow{a} (X)P \quad \text{LTS-Abstr}$$

$$\overline{a}\langle Q\rangle P \xrightarrow{\overline{a}} \langle Q\rangle P \quad \text{LTS-Concr} \qquad\qquad \frac{P \xrightarrow{\alpha} A}{P \mid Q \xrightarrow{\alpha} A \mid Q} \quad \text{LTS-Par}$$

$$\frac{P \xrightarrow{\alpha} A \qquad \alpha \notin \{x, \overline{x}\}}{\nu x.P \xrightarrow{\alpha} \nu x.A} \quad \text{LTS-Restr} \qquad\qquad \frac{P \mid !P \xrightarrow{\alpha} A}{!P \xrightarrow{\alpha} A} \quad \text{LTS-Replic}$$

$$\frac{P \xrightarrow{m} P' \qquad Q \xrightarrow{\overline{m}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \quad \text{LTS-FO} \qquad\qquad \frac{P \xrightarrow{a} F \qquad Q \xrightarrow{\overline{a}} C}{P \mid Q \xrightarrow{\tau} F \bullet C} \quad \text{LTS-HO}$$

$$\frac{P \xrightarrow{\alpha} A}{a[P] \xrightarrow{\alpha} a[A]} \quad \text{LTS-Loc} \qquad\qquad a[P] \xrightarrow{\overline{a}} \langle P\rangle \mathbf{0} \quad \text{LTS-Passiv}$$

Fig. 2. Labeled Transition System for HOπP

2.2 Strong Behavioral Equivalences

Barbed congruence is a uniform definition of process equivalence among process calculi based on the reduction relation $\longrightarrow$ (defined as $\longrightarrow \stackrel{\Delta}{=} \equiv\xrightarrow{\tau}\equiv$), the observable actions of a process, called *barbs*, and contexts. In HOπP, a process P has a barb $\mu = x \mid \overline{x}$, written $P \downarrow_\mu$, iff we have $P \xrightarrow{\mu}$. Contexts are processes with a hole $\Box$; filling a context $\mathbb{C}$ with a process P gives a process written $\mathbb{C}\{P\}$.

Definition 1. *A relation $\mathcal{R}$ on closed processes is a* strong barbed bisimulation *iff $\mathcal{R}$ is symmetric, and $P \mathrel{\mathcal{R}} Q$ implies:*

- *If $P \downarrow_\mu$ then $Q \downarrow_\mu$*
- *If $P \longrightarrow P'$, then there exists Q' such that $Q \longrightarrow Q'$ and $P' \mathrel{\mathcal{R}} Q'$.*

Processes P and Q are strongly barbed congruent*, written $P \sim_b Q$, iff for all contexts $\mathbb{C}$, there exists a strong barbed bisimulation $\mathcal{R}$ such that $\mathbb{C}\{P\} \mathrel{\mathcal{R}} \mathbb{C}\{Q\}$.*

The universal quantification over contexts makes barbed congruence difficult to use in practice. Sangiorgi introduced *context* bisimilarity for HOπ [11] as an LTS-based alternative to barbed congruence. Context bisimilarity is *sound*, i.e. is included in barbed congruence. In the weak case, there exists a version ("early non delay") of the bisimilarity which is also *complete*, i.e. contains barbed congruence, and therefore is a characterization of weak barbed congruence (see [10] for further details). We write $\mathcal{B}$ for the strong context bisimilarity of HOπ (see [11] for the definition). Using this bisimilarity with HOπP leads to a relation which is not sound: there exist HOπP processes related by $\mathcal{B}$ which are not strong barbed congruent. Consider the following processes:

$$P_0 = \overline{a}\langle \mathbf{0} \rangle !m.\mathbf{0} \qquad Q_0 = \overline{a}\langle m.\mathbf{0} \rangle !m.\mathbf{0}$$

Processes P_0 and Q_0 are related by $\mathcal{B}$: the difference between the emitted messages is shadowed by the continuation $!m.\mathbf{0}$. They cannot be distinguished by a HOπ context, but are distinguished by an HOπP context which may discard the message continuations: $\mathbb{C} = b[\square] \mid a(X)X \mid b(X)\mathbf{0}$. With a communication on a followed by passivation/communication on b, we have $\mathbb{C}\{P_0\} \longrightarrow b[!m.\mathbf{0}] \mid \mathbf{0} \mid b(X)\mathbf{0} \longrightarrow \mathbf{0}$. It can only be matched by $\mathbb{C}\{Q_0\} \longrightarrow b[!m.\mathbf{0}] \mid m.\mathbf{0} \mid b(X)\mathbf{0} \longrightarrow m.\mathbf{0}$. The two resulting processes have different barbs, therefore P_0 and Q_0 are not barbed congruent. Hence relation $\mathcal{B}$ is not sound with HOπP.

In a concretion $\nu\widetilde{x}.\langle R \rangle S$, the emitted process R may be sent outside a locality b while the continuation S stays in b. If the passivation on b is triggered, S may be destroyed (as with P_0 and Q_0) or put in a different context. Hence the passivation may separate the processes R and S and put them in totally different contexts, which is not possible in a calculus without passivation. As in the Kell calculus and Homer, we address this issue by testing messages and continuations in different *evaluation contexts* $\mathbb{E}$. These contexts, when applied to concretions, take into account the fact that a message and its continuation are separated: in the definition of $a[C]$ for some concretion C, the message part of C is put outside the locality whereas the continuation part remains inside. The grammar of HOπP evaluation contexts is:

$$\mathbb{E} ::= \square \mid \nu x.\mathbb{E} \mid \mathbb{E} \mid P \mid P \mid \mathbb{E} \mid a[\mathbb{E}]$$

Early strong context bisimulation for HOπP is defined as follows:

Definition 2. *A relation $\mathcal{R}$ on closed processes is an early strong context bisimulation iff $\mathcal{R}$ is symmetric and $P \mathrel{\mathcal{R}} Q$ implies $fn(P) = fn(Q)$ and:*

- *For all $P \xrightarrow{l} P'$, there exists Q' such that $Q \xrightarrow{l} Q'$ and $P' \mathcal{R} Q'$.*
- *For all $P \xrightarrow{a} F$, for all closed concretions C, there exists G such that $Q \xrightarrow{a} G$ and $F \bullet C \mathcal{R} G \bullet C$.*
- *For all $P \xrightarrow{\overline{a}} C$, for all closed abstractions F, there exists D such that $Q \xrightarrow{\overline{a}} D$ and for all closed evaluation contexts $\mathbb{E}$, we have $F \bullet \mathbb{E}\{C\} \mathcal{R} F \bullet \mathbb{E}\{D\}$.*

Early strong context bisimilarity, written $\sim$, is the largest early strong context bisimulation.

Example 1. The two processes $m.\mathbf{0} \mid !a[m.\mathbf{0}] \mid !a[\mathbf{0}]$ and $!a[m.\mathbf{0}] \mid !a[\mathbf{0}]$ are strong early context bisimilar.

The main difference with $\mathcal{B}$ is the additional evaluation context $\mathbb{E}$ in the concretion case, that is similar to the Homer path contexts [5] or Kell calculus applicative contexts [13]. We also add the condition $\text{fn}(P) = \text{fn}(Q)$ since two equivalent processes with different free names may be distinguished with scope extrusion outside localities, as is illustrated in Section 4 and further developed in [8]. Early strong context bisimilarity is a suitable relation, since we have the following characterization result, which we prove with the technique used for the Kell calculus, namely proving directly a substitution lemma.

Theorem 1. *We have $P \sim Q$ iff $P \sim_b Q$.*

2.3 Weak Behavioral Equivalences

We now give results for the weak case, where we abstract from internal actions. We write $\Longrightarrow$ the reflexive and transitive closure of $\longrightarrow$. The definition of (weak) barbed congruence, written $\approx_b$, is given by changing the two clauses of Definition 1 to:

- If $P \downarrow_\mu$ then $Q \Longrightarrow \downarrow_\mu$
- If $P \longrightarrow P'$, then there exists Q' such that $Q \Longrightarrow Q'$ and $P' \mathcal{R} Q'$.

The soundness proof method used for Kell (and Theorem 1) does not work with weak relations (see [8] for details). As in Homer [4], we can use *Howe's method* [6], a systematic soundness proof technique, to show that *input-early* weak delay bisimulation, an early relation with a late condition in the output case, is sound. The use of such a delay relation is required to apply Howe's method. Let $\Rightarrow$ be the reflexive and transitive closure of $\xrightarrow{\tau}$ and define weak delay transitions by $\overset{\tau}{\Rightarrow} \overset{\Delta}{=} \Rightarrow$ and $\overset{\alpha}{\Rightarrow} \overset{\Delta}{=} \Rightarrow \xrightarrow{\alpha}$ for $\alpha \neq \tau$.

Definition 3. *A relation $\mathcal{R}$ on closed processes is an input-early weak (delay) bisimulation iff $\mathcal{R}$ is symmetric and $P \mathcal{R} Q$ implies $fn(P) = fn(Q)$ and:*

- *For all $P \xrightarrow{l} P'$, there exists Q' such that $Q \overset{l}{\Rightarrow} Q'$ and $P' \mathcal{R} Q'$.*
- *For all $P \xrightarrow{a} F$, for all closed concretions C and all closed evaluation contexts $\mathbb{E}$, there exists G such that $Q \overset{a}{\Rightarrow} G$ and $\mathbb{E}\{F\} \bullet C \mathcal{R} \mathbb{E}\{G\} \bullet C$.*
- *For all $P \xrightarrow{\overline{a}} C$, there exists D such that $Q \overset{\overline{a}}{\Rightarrow} D$ and for all closed abstractions F and evaluation contexts $\mathbb{E}$, we have $F \bullet \mathbb{E}\{C\} \mathcal{R} F \bullet \mathbb{E}\{D\}$.*

Input-early weak delay bisimilarity, written $\approx_{ie}$, is the largest input-early weak delay context bisimulation.

The additional context in the abstraction case is required for technical reasons, see [8] for details. Notice that input-early bisimilarity is a *delay* relation since silent steps are not allowed after a visible action. Consequently, input-early bisimilarity is sound but likely not complete.

Theorem 2. *If $P \approx_{ie} Q$, then $P \approx_b Q$.*

For the time being, the characterization of weak barbed congruence in HOπP remains an open problem. In the next section, we show that this is due to the interaction between passivation and name restriction.

3 Normal Bisimilarity in HOP

In this section, we develop a full behavioral theory for HOP, a calculus with passivation but without restriction: we define context and normal bisimilarities which characterize barbed congruence in both strong and weak cases. HOP (for Higher Order with Passivation) is the calculus obtained by removing restriction from HOπP (Figure 1) and adding a sum operator (to obtain the characterization result, since $+$ is needed to show the completeness of HO bisimilarity and requires restriction to be faithfully encoded). The LTS rules for HOP are as in Figure 2, with the addition of the rule

$$\frac{P \xrightarrow{\alpha} A}{P + Q \xrightarrow{\alpha} A}\ \text{LTS-SUM}$$

and of its symmetric rule. The structural congruence rules for HOP, also written $\equiv$, is the smallest congruence that verifies the following laws.

$$P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad P \mid Q \equiv Q \mid P \qquad P \mid \mathbf{0} \equiv P$$

$$P + (Q + R) \equiv (P + Q) + R \qquad P + Q \equiv Q + P \qquad P + \mathbf{0} \equiv P \qquad !P \equiv P \mid !P$$

Even without restriction, HOP remains quite expressive since it is an extension of the Turing-complete HOcore calculus defined in [7].

3.1 HO Bisimulation

The definition of strong barbed congruence is identical to Definition 1. We now give an LTS-based characterization of strong barbed congruence.

As pointed out in Section 2.2, a message and its continuation may be put in different contexts because of passivation. Moreover, they are completely independent since they no longer share private names, as there is no restriction. Instead of keeping them together, we can now study them separately and still have a sound and complete bisimilarity. We propose the following bisimulation, called HO bisimulation, similar to the higher-order bisimulation given by Thomsen for Plain CHOCS [14]. For an abstraction $F = (X)Q$ and a process P, we write $F \circ P$ for the process $Q\{P/X\}$.

Definition 4. *A relation $\mathcal{R}$ on closed processes is an early strong HO bisimulation iff $\mathcal{R}$ is symmetric and $P \mathrel{\mathcal{R}} Q$ implies:*

- *For all $P \xrightarrow{l} P'$, there exists Q' such that $Q \xrightarrow{l} Q'$ and $P' \mathrel{\mathcal{R}} Q'$.*
- *For all $P \xrightarrow{a} F$, for all closed processes R, there exists G such that $Q \xrightarrow{a} G$ and $F \circ R \mathrel{\mathcal{R}} G \circ R$.*
- *For all $P \xrightarrow{\overline{a}} \langle R \rangle S$, there exists R', S' such that $Q \xrightarrow{\overline{a}} \langle R' \rangle S'$, $R \mathrel{\mathcal{R}} R'$, $S \mathrel{\mathcal{R}} S'$.*

Early strong HO bisimilarity, written $\dot{\sim}$, is the largest early strong HO bisimulation.

In the following we also use the late counterpart of HO bisimilarity, written $\dot{\sim}_l$, which is obtained by replacing the input case by:

- For all $P \xrightarrow{a} F$, there exists G such that $Q \xrightarrow{a} G$ and for all closed processes R, $F \circ R \mathrel{\mathcal{R}} G \circ R$.

We show later that early and late HO bisimilarities coincide (as in HOπ). Using the same proof technique as for HOπP, we prove that $\dot{\sim}_l$ is sound and complete.

Theorem 3. *We have $P \mathrel{\dot{\sim}_l} Q$ iff P and Q are strong barbed congruent.*

Unlike HOπP, we are able to characterize barbed congruence also in the weak case. We define early weak (non-delay) HO bisimulation as:

Definition 5. *A relation $\mathcal{R}$ on closed processes is an early weak HO bisimulation iff $\mathcal{R}$ is symmetric and $P \mathrel{\mathcal{R}} Q$ implies:*

- *For all $P \xrightarrow{l} P'$, there exists Q' such that $Q \overset{l}{\Rightarrow}\overset{\tau}{\Rightarrow} Q'$ and $P' \mathrel{\mathcal{R}} Q'$.*
- *For all $P \xrightarrow{a} F$, for all closed processes R, there exist G, Q' such that $Q \overset{a}{\Rightarrow} G$, $G \circ R \overset{\tau}{\Rightarrow} Q'$, and $F \circ R \mathrel{\mathcal{R}} Q'$.*
- *For all $P \xrightarrow{\overline{a}} \langle R \rangle S$, there exist R', S'', S' such that $Q \overset{\overline{a}}{\Rightarrow} \langle R' \rangle S''$, $S'' \overset{\tau}{\Rightarrow} S'$, $R \mathrel{\mathcal{R}} R'$, and $S \mathrel{\mathcal{R}} S'$.*

Early weak HO bisimilarity, written $\dot{\approx}$, is the largest early weak HO bisimulation.

We define late weak HO bisimilarity, written $\dot{\approx}_l$, by replacing the input clause by:

- For all $P \xrightarrow{a} F$, there exists G such that $Q \overset{a}{\Rightarrow} G$ and for all closed processes R, there exists Q' such that $G \circ R \overset{\tau}{\Rightarrow} Q'$ and $F \circ R \mathrel{\mathcal{R}} Q'$.

Since there is no universal quantification in the concretion case, early and input-early versions of the bisimulation coincide. Besides, the bisimilarity condition on messages makes Howe's method work with this bisimulation:

Theorem 4. *If $P \mathrel{\dot{\approx}} Q$, then P and Q are weak barbed congruent.*

As in π-calculus [12], we prove completeness on *image-finite* processes. A process P is image finite iff for all l and α, the set $\{P' | P \overset{l}{\Rightarrow}\overset{\tau}{\Rightarrow} P'\} \cup \{A | P \overset{\alpha}{\Rightarrow} A\}$ is finite.

Theorem 5. *Let P, Q be image finite processes. If P, Q are weak barbed congruent, then they are early weak HO bisimilar.*

We note that the definitions of higher-order bisimulations are easier to use since there is no universal quantification in the concretion case. In the following subsection, we show that the one in the abstraction case is not necessary.

3.2 Normal Bisimulation

In this section, we define a sound and complete bisimulation for the strong and weak cases without any universal quantification, similar to HOπ normal bisimulation [11]. Sangiorgi first defined it in the weak case, and then Cao extended it to the strong case [1].

In the message input case, HOπ normal bisimulation tests abstractions with only one trigger $m.\mathbf{0}$, where m is a fresh name. This testing is not sufficient in HOP. Consider the following processes:

$$P_1 \stackrel{\Delta}{=} !a[X] \mid !a[\mathbf{0}] \qquad Q_1 \stackrel{\Delta}{=} X \mid P_1$$

Let $P_m \stackrel{\Delta}{=} P_1\{m.\mathbf{0}/X\}$, $Q_m \stackrel{\Delta}{=} Q_1\{m.\mathbf{0}/X\}$, $P_{m,n} \stackrel{\Delta}{=} P_1\{m.n.\mathbf{0}/X\}$, and $Q_{m,n} \stackrel{\Delta}{=} Q_1\{m.n.\mathbf{0}/X\}$, where m, n do not occur in P_1, Q_1.

We first prove that $P_m \dot{\sim}_l Q_m$. Since the other transitions are easily matched, we consider only the move $Q_m \xrightarrow{m} \mathbf{0} \mid P_m$. It can only be matched by a replicated locality $a[m.\mathbf{0}]$; we have $P_m \xrightarrow{m} a[\mathbf{0}] \mid P_m$. The two resulting processes $\mathbf{0} \mid P_m$ and $a[\mathbf{0}] \mid P_m$ are immediately bisimilar, due to the presence of $!a[\mathbf{0}]$ in P_m. Consequently we have $P_m \dot{\sim}_l Q_m$.

However we have $P_{m,n} \dot{\not\sim}_l Q_{m,n}$. Indeed, the transition $Q_{m,n} \xrightarrow{m} n.\mathbf{0} \mid P_{m,n} \stackrel{\Delta}{=} Q'_{m,n}$ can only be matched by $P_{m,n} \xrightarrow{m} a[n.\mathbf{0}] \mid P_{m,n} \stackrel{\Delta}{=} P'_{m,n}$. Processes $P'_{m,n}$ and $Q'_{m,n}$ are not HO bisimilar: by passivation of locality $a[n.\mathbf{0}]$, we have $P'_{m,n} \xrightarrow{\overline{a}} \langle n.\mathbf{0}\rangle P_{m,n}$, which can only be matched by $Q'_{m,n} \xrightarrow{\overline{a}} \langle m.n.\mathbf{0}\rangle Q'_{m,n}$ or $Q'_{m,n} \xrightarrow{\overline{a}} \langle \mathbf{0}\rangle Q'_{m,n}$. The emitted processes are not pairwise HO bisimilar, consequently we have $P'_{m,n} \dot{\not\sim}_l Q'_{m,n}$.

One could argue that the weakness of the distinguishing power of the trigger $m.\mathbf{0}$ is due to the fact that localities are completely transparent, thus the provenance of a message may not be directly observed. However, the existence of localities around a message has indirect effects, when passivation transforms an evaluation context (the locality) into a message that may be discarded. Triggers of the form $m.n.\mathbf{0}$ allow the observation of an evaluation context (there is an emission on m) that disappears (there is no further emission on n), thus the presence of enclosing localities.

We now generalize this idea to show that it may be used to pinpoint the position of a process variable in the locality tree. Suppose we have $P\{m.n.\mathbf{0}/X\}$ bisimilar to $Q\{m.n.\mathbf{0}/X\}$, with m, n not occurring in P, Q. Suppose further that $P \xrightarrow{m} P'$ is matched by $Q \xrightarrow{m} Q'$. The processes P', Q' may now perform one and only one $\xrightarrow{n}$ transition from the single process $n.\mathbf{0}$. Now suppose that $n.\mathbf{0}$

is in a locality a in P'. Passivation of this locality results in a concretion whose message R is such that $R \xrightarrow{n}$. The process Q' has to match these transitions with $Q' \xrightarrow{\overline{a}} \langle R' \rangle S'$ such that $R \dot{\sim}_l R'$. Since $R \xrightarrow{n}$, we have $R' \xrightarrow{n}$; it is possible if and only if the single occurrence of $n.\mathbf{0}$ in Q' was in a locality a. With the same argument on R, R', we prove that the locality hierarchies around $n.\mathbf{0}$ in P' and Q' are the same. This result is formalized by the following lemma:

Lemma 1. *Let P, Q such that $fv(P, Q) \subseteq \{X\}$ and m, n two names which do not occur in P, Q. Suppose we have $P\{m.n.\mathbf{0}/X\} \dot{\sim}_l Q\{m.n.\mathbf{0}/X\}$ and $P\{m.n.\mathbf{0}/X\} \xrightarrow{m} P'\{m.n.\mathbf{0}/X\}\{n.\mathbf{0}/Y\} \stackrel{\Delta}{=} P_n$ matched by $Q\{m.n.\mathbf{0}/X\} \xrightarrow{m} Q'\{m.n.\mathbf{0}/X\}\{n.\mathbf{0}/Y\} \stackrel{\Delta}{=} Q_n$ with $P_n \dot{\sim}_l Q_n$.*

There exists $k \geq 0$, $a_1, \dots a_k$, $P_1 \dots P_{k+1}$, $Q_1 \dots Q_{k+1}$ such that either $P_n \equiv n.\mathbf{0} \mid P_1$ and $Q_n \equiv n.\mathbf{0} \mid Q_1$ or

$$P_n \equiv a_1[\dots a_{k-1}[a_k[n.\mathbf{0} \mid P_{k+1}] \mid P_k] \mid P_{k-1} \dots] \mid P_1$$
$$Q_n \equiv a_1[\dots a_{k-1}[a_k[n.\mathbf{0} \mid Q_{k+1}] \mid Q_k] \mid Q_{k-1} \dots] \mid Q_1$$

and for all $1 \leq j \leq k+1$, $P_j \dot{\sim}_l Q_j$.

The lemma allows us to decompose P_n, Q_n in bisimilar sub-processes. For instance, if we have $P_n \equiv a[b[n.\mathbf{0} \mid P_3] \mid P_2] \mid P_1$ with $P_n \dot{\sim}_l Q_n$, then $Q_n \equiv a[b[n.\mathbf{0} \mid Q_3] \mid Q_2] \mid Q_1$ with $P_1 \dot{\sim}_l Q_1$, $P_2 \dot{\sim}_l Q_2$, and $P_3 \dot{\sim}_l Q_3$. Notice that we do not decompose the initial processes P and Q themselves, but this result is enough to prove the following theorem:

Theorem 6. *Let P, Q two processes such that $fv(P, Q) \subseteq \{X\}$ and m, n two names which do not occur in P, Q. If $P\{m.n.\mathbf{0}/X\} \dot{\sim}_l Q\{m.n.\mathbf{0}/X\}$, then for all closed processes R, we have $P\{R/X\} \dot{\sim}_l Q\{R/X\}$*

We sketch the proof of Theorem 6 to explain how Lemma 1 is used.

Proof (Sketch). We show that the symmetric closure of relation

$$\mathcal{R} \stackrel{\Delta}{=} \{(P\{R/X\}, Q\{R/X\}) \mid P\{m.n.\mathbf{0}/X\} \dot{\sim}_l Q\{m.n.\mathbf{0}/X\}, m, n \text{ not in } P, Q\}$$

is a late HO bisimulation. It is done by case analysis on the transition performed by $P\{R/X\}$. Suppose we have $P\{R/X\} \xrightarrow{l} P'\{R'/X_i\}\{R/X\}$, i.e. a copy of R (at position X_i) performs a transition $R \xrightarrow{l} R'$. Occurence X_i is in an evaluation context, so we have $P\{m.n.\mathbf{0}/X\} \xrightarrow{m} P'\{n.\mathbf{0}/X_i\}\{m.n.\mathbf{0}/X\} = P'_n$, matched by $Q\{m.n.\mathbf{0}/X\} \xrightarrow{m} Q'\{n.\mathbf{0}/X_j\}\{m.n.\mathbf{0}/X\} = Q'_n$ with $P'_n \dot{\sim}_l Q'_n$. As X_j is also in an evaluation context, we have $Q\{R/X\} \xrightarrow{l} Q'\{R'/X_j\}\{R/X\}$. We now have to prove that $P'\{R'/X_i\}\{m.n.\mathbf{0}/X\} \dot{\sim}_l Q'\{R'/X_j\}\{m.n.\mathbf{0}/X\}$.

Lemma 1 allows us to write $P'_n \equiv a_1[\dots a_k[n.\mathbf{0} \mid P_{k+1}] \mid P_k \dots] \mid P_1$ and $Q'_n \equiv a_1[\dots a_k[n.\mathbf{0} \mid Q_{k+1}] \mid Q_k \dots] \mid Q_1$ with $(P_r), (Q_r)$ pairwise bisimilar processes for $r \in \{1 \dots k+1\}$. Since $P_{k+1} \dot{\sim}_l Q_{k+1}$ and $\dot{\sim}_l$ is sound, we have $a_k[R' \mid P_{k+1}] \dot{\sim}_l a_k[R' \mid Q_{k+1}]$. By induction on $r \in \{k \dots 1\}$, we prove that $a_r[\dots a_k[R' \mid P_{k+1}] \mid P_k \dots] \mid P_j \dot{\sim}_l a_r[\dots a_k[R' \mid Q_{k+1}] \mid Q_k \dots] \mid Q_j$, obtaining $P'\{R'/X_i\}\{m.n.\mathbf{0}/X\} \dot{\sim}_l Q'\{R'/X_j\}\{m.n.\mathbf{0}/X\}$ (for $r = 1$) as needed. □

Using this result we define a normal bisimulation for HOP:

Definition 6. *A relation $\mathcal{R}$ on closed processes is a strong normal bisimulation iff $\mathcal{R}$ is symmetric and $P \mathcal{R} Q$ implies :*

– *For all $P \xrightarrow{l} P'$, there exists Q' such that $Q \xrightarrow{l} Q'$ and $P' \mathcal{R} Q'$.*
– *For all $P \xrightarrow{a} F$, there exists G such that $Q \xrightarrow{a} G$ and for two names m, n which do not occur in processes P, Q, we have $F \circ m.n.\mathbf{0} \mathcal{R} G \circ m.n.\mathbf{0}$.*
– *For all $P \xrightarrow{\overline{a}} \langle R \rangle S$, there exists R', S' such that $Q \xrightarrow{\overline{a}} \langle R' \rangle S'$, $R \mathcal{R} R'$ and $S \mathcal{R} S'$.*

Strong normal bisimilarity, written $\dot{\sim}_n$, is the largest strong normal bisimulation.

As a corollary of Theorem 6, we have

Corollary 1. $\dot{\sim}_l = \dot{\sim}_n = \dot{\sim}$.

By definition, we have $\dot{\sim}_l \subseteq \dot{\sim} \subseteq \dot{\sim}_n$. The inclusion $\dot{\sim}_n \subseteq \dot{\sim}_l$ is a consequence of Theorem 6.

Weak normal bisimilarity that coincides with weak HO bisimilarity may also be defined.

Definition 7. *A relation $\mathcal{R}$ on closed processes is a weak normal simulation iff $\mathcal{R}$ is symmetric and $P \mathcal{R} Q$ implies:*

– *For all $P \xrightarrow{l} P'$, there exists Q' such that $Q \overset{l}{\Rightarrow} \overset{\tau}{\Rightarrow} Q'$ and $P' \mathcal{R} Q'$.*
– *For all $P \xrightarrow{a} F$, there exists G such that $Q \overset{a}{\Rightarrow} G$ and for two names m, n which do not occur in processes P, Q, there exists Q' such that $G \circ m.n.\mathbf{0} \overset{\tau}{\Rightarrow} Q'$ and $F \circ m.n.\mathbf{0} \mathcal{R} Q'$.*
– *For all $P \xrightarrow{\overline{a}} \langle R \rangle S$, there exists R', S'', S' such that $Q \overset{\overline{a}}{\Rightarrow} \langle R' \rangle S''$, $S'' \overset{\tau}{\Rightarrow} S'$, $R \mathcal{R} R'$ and $S \mathcal{R} S'$.*

Weak normal bisimilarity, written $\dot{\approx}_n$, is the largest weak normal bisimulation.

Theorem 7. $\dot{\approx}_n = \dot{\approx} = \dot{\approx}_l$

The proof technique is similar to the strong case one and relies on weak versions of Theorem 6 and Lemma 1. Hence in a calculus with passivation and without restriction, we can define a suitable bisimulation without any universal quantification in the strong and weak cases. We show in the next section that the result on abstractions does not hold in HOπP.

4 Abstraction Equivalence in HOπP

In this section, we present a counter-example to show that a simplification similar to the one of Section 3.2 is not possible in HOπP. We prove that testing a large sub-class of HOπP processes (the *abstraction-free* processes) is not enough to guarantee bisimilarity of abstraction. Note that these counter-examples only depend on the interaction between the scope extrusion of restriction and process duplication, and not on whether passivation or message provenance are directly observable. More complex counter-examples, where scope extrusion is not needed, are presented in [8].

In the following, we omit the trailing zeros to improve readability; in an agent definition, m stands for $m.\mathbf{0}$. We also write $\nu ab.P$ for $\nu a.\nu b.P$. Let $\mathbf{0}_m \stackrel{\Delta}{=} \nu x.x.m$. Process $\mathbf{0}_m$ cannot perform any transition, like $\mathbf{0}$, but it has a free name m. We define the following abstractions:

$$(X)P \stackrel{\Delta}{=} (X)\nu nb.(b[X \mid \nu m.\overline{a}\langle \mathbf{0}_m\rangle(m \mid n \mid \overline{m}.\overline{m}.p)] \mid \overline{n}.b(Y)(Y \mid Y))$$
$$(X)Q \stackrel{\Delta}{=} (X)\nu mnb.(b[X \mid \overline{a}\langle \mathbf{0}\rangle(m \mid n \mid \overline{m}.\overline{m}.p)] \mid \overline{n}.b(Y)(Y \mid Y))$$

The two abstractions differ in the process emitted on a and in the position of name restriction on m (inside or outside hidden locality b). An abstraction-free process is a process built with the regular HOπP syntax (Figure 1) but without message input $a(X)P$.

We recall that $\sim$ is the early strong context bisimilarity (Definition 2).

Lemma 2. *Let R be an abstraction-free process. We have $(X)P \circ R \sim (X)Q \circ R$.*

Since R is abstraction-free, it cannot receive the message emitted on a; consequently R cannot interact with P or Q. Passivation of locality b and transitions from R in $(X)P \circ R$ are easily matched by the same transitions in $(X)Q \circ R$.

Let $P_{m,R} = \nu nb.(b[R \mid m \mid n \mid \overline{m}.\overline{m}.p] \mid \overline{n}.b(Y)(Y \mid Y))$, F be an abstraction, and $\mathbb{E}$ be an evaluation context such that $m \notin \text{fn}(\mathbb{E}, F)$. We now prove that $(X)P \circ R \xrightarrow{\overline{a}} \nu m.\langle \mathbf{0}_m\rangle P_{m,R}$ is matched by $(X)Q \circ R \xrightarrow{\overline{a}} \langle \mathbf{0}\rangle \nu m.P_{m,R}$, i.e. that we have $\nu m.(F \circ \mathbf{0}_m \mid \mathbb{E}\{P_{m,R}\}) \sim F \circ \mathbf{0} \mid \mathbb{E}\{\nu m.P_{m,R}\}$. Since $m \notin \text{fn}(\mathbb{E}, F)$, there is no interaction between $F, \mathbb{E}$ and $P_{m,R}$, and the inert process $\mathbf{0}_m$ does not interfere either. Hence the possible transitions from $\nu m.(F \circ \mathbf{0}_m \mid \mathbb{E}\{P_{m,R}\})$ are only from $F, \mathbb{E}, R$, and internal actions in $P_{m,R}$, and are matched by the same transitions in $F \circ \mathbf{0} \mid \mathbb{E}\{\nu m.P_{m,R}\}$.

Abstractions $(X)P$ and $(X)Q$ may have different behaviors with an argument which may receive on a, like $a(Z)q$, where q is a first-order name such that $p \neq q$. By communication on a, we have $(X)Q \circ a(Z)q \xrightarrow{\tau} \nu mnb.(b[q \mid m \mid n \mid \overline{m}.\overline{m}.p] \mid \overline{n}.b(Y)(Y \mid Y)) \stackrel{\Delta}{=} Q_1$. Since Q_1 may perform a $\xrightarrow{q}$ transition, it can only be matched by $(X)P \circ a(Z)q \xrightarrow{\tau} \nu nb.(b[\nu m.(q \mid m \mid n \mid \overline{m}.\overline{m}.p)] \mid \overline{n}.b(Y)(Y \mid Y)) \stackrel{\Delta}{=} P_1$. Notice that in P_1, the restriction on m remains inside hidden locality b.

After synchronization on n and passivation/communication on b, we have $Q_1(\xrightarrow{\tau})^2\nu mnb.(q \mid q \mid m \mid m \mid \overline{m}.\overline{m}.p \mid \overline{m}.\overline{m}.p) \stackrel{\Delta}{=} Q_2$ (the process inside b in Q_1 is duplicated). After two synchronizations on m, we have $Q_2(\xrightarrow{\tau})^2\nu mnb.(q \mid q \mid p \mid \overline{m}.\overline{m}.p) \stackrel{\Delta}{=} Q_3$, and Q_3 may perform a $\xrightarrow{p}$ transition. These transitions cannot be matched by P_1. Performing the duplication, we have $P_1(\xrightarrow{\tau})^2\nu nb.(\nu m.(q \mid m \mid \overline{m}.\overline{m}.p) \mid \nu m.(q \mid m \mid \overline{m}.\overline{m}.p)) \stackrel{\Delta}{=} P_2$. Each copied sub-process $q \mid m \mid \overline{m}.\overline{m}.p$ of P_2 has its own private copy of m, and we can no longer perform any transition to have the observable p. More generally, the sequence of transitions $Q_1(\xrightarrow{\tau})^4 \xrightarrow{p}$ cannot be matched by P_1, consequently Q_1 and P_1 (and therefore $(X)Q \circ a(Z)q$ and $(X)P \circ a(Z)q$) are not bisimilar.

The previous example shows that testing abstractions with abstraction-free processes (such as $m.n.\mathbf{0}$) is not enough to distinguish them. This example relies heavily on the chosen "by need" scope extrusion (restrictions are extruded outside localities along with messages), which is also used in Homer or Kell. Such scope extrusion has unusual consequences: the example can be adapted to show that $\mathbf{0}$ and $\mathbf{0}_m$ are not equivalent. Using a different definition of scope extrusion, for instance by considering name restriction to be a fresh name generator, is unfortunately not a solution: we present in [8] other counter-examples which do not rely on scope extrusion yet show that testing a large class of finite processes is not sufficient to derive abstractions equivalence. Whether one can define a normal-like bisimilarity in HOπP that only uses a finite number of tests remains an open issue.

5 Related Work

Sangiorgi studies behavioral equivalences for HOπ in [11]. We reviewed his work earlier in the paper.

The Kell calculus [13] and Homer [5] are two higher-order calculi with passivation in which bisimulations have been defined and which share common concepts, like hierarchical localities, local names, objective passive and active process mobility. The calculi differ in how they handle communication. In the Kell calculus, communications are only local: processes may communicate only if they are in the same locality or in direct parent-child localities. In the strong case, a sound and complete early context bisimulation has been defined. In Homer, a process may passivate or send a message to an arbitrary nested sub-locality, but the interactions are not allowed in the other way: a process in a sub-locality cannot send a message to a process in a parent one. In [4], the authors define an input early context bisimulation which is late in the message output case and early in the message input case. The relation is shown to be sound in the weak (delay) case, and sound and complete in the strong case. The definition is similar to the HOπP one except it features an additional quantification on so-called path contexts.

The Seal calculus [3] allows a process mobility similar to the passivation feature: localities may be stopped, duplicated, and moved up and down in the locality hierarchy. Mobility is less flexible than in Homer or Kell since a process inside a locality cannot be dissociated from the locality boundary. The authors define a bisimilarity, called *Hoe bisimilarity*, for the Seal calculus, which is similar to the normal bisimulation for HOπ in the message output case. However, Hoe bisimilarity is sound in the strong and weak cases but not complete.

Mobile Ambients [2] is also a higher-order calculus with hierarchical localities. Unlike previous calculi, mobility in Mobile Ambients is subjective: localities move by themselves, without any acknowledgment from their environment. In

[9], Merro and Zappa Nardelli define a context bisimilarity which characterizes barbed congruence in the weak case. A normal bisimulation without universal quantification has yet to be found.

6 Conclusion

Behavioral theory in calculi with passivation (like the Kell calculus or Homer) is less developed than the HOπ one. They are equipped with a sound and complete context bisimulation in the strong case only, which features additional tests on contexts in the message output case. This additional complexity comes from the interference between name restriction and passivation.

In HOP, a calculus with passivation but without name restriction, we have similar results on bisimulations as in Sangiorgi's HOπ. First, we have a simple higher-order bisimulation which characterizes barbed congruence. In a message output, the message and the continuation are considered separately, since they do not share private names and passivation may put them in different contexts. Early and late higher-order bisimulations coincide.

We also have a normal bisimulation without any universal quantification which coincides with higher-order bisimulation. In the case of HOπ, normal bisimulation comes from an encoding of higher-process in a first-order, which is not possible in HOP. Instead, normal bisimulation in HOP relies on some means (a process $m.n.\mathbf{0}$) to observe locality hierarchies and to decompose abstractions in bisimilar sub-processes. Both higher-order and normal bisimilarities are defined in the weak and strong cases.

We have shown that we cannot adapt this proof technique to the calculus with restriction. As proved in Section 4, testing any abstraction-free processes is not enough to establish abstractions equivalence. We conjecture that in a calculus featuring passivation and name restriction, we cannot define a sound and complete strong bisimilarity with fewer tests than in Definition 2.

Acknowledgments

We are grateful to Samuel Hym, Jorge A. Pérez, and Davide Sangiorgi for their helpful comments on an earlier draft of the paper.

References

1. Cao, Z.: More on bisimulations for higher order π-calculus. In: Aceto, L., Ingólfsdóttir, A. (eds.) FOSSACS 2006. LNCS, vol. 3921, pp. 63–78. Springer, Heidelberg (2006)
2. Cardelli, L., Gordon, A.D.: Mobile ambients. In: Nivat, M. (ed.) FOSSACS 1998. LNCS, vol. 1378, p. 140. Springer, Heidelberg (1998)
3. Castagna, G., Vitek, J., Zappa Nardelli, F.: The Seal Calculus. Information and Computation 201(1) (2005)
4. Godskesen, J.C., Hildebrandt, T.: Extending howe's method to early bisimulations for typed mobile embedded resources with local names. In: Ramanujam, R., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 140–151. Springer, Heidelberg (2005)

5. Hildebrandt, T., Godskesen, J.C., Bundgaard, M.: Bisimulation congruences for Homer — a calculus of higher order mobile embedded resources. Technical Report ITU-TR-2004-52, IT University of Copenhagen (2004)
6. Howe, D.J.: Proving congruence of bisimulation in functional programming languages. Information and Computation 124(2) (1996)
7. Lanese, I., Pérez, J.A., Sangiorgi, D., Schmitt, A.: On the expressiveness and decidability of higher-order process calculi. In: 23rd Annual IEEE Symposium on Logic in Computer Science (LICS). IEEE Computer Society, Los Alamitos (2008)
8. Lenglet, S., Schmitt, A., Stefani, J.B.: Normal bisimulations in process calculi with passivation. Technical Report RR 6664, INRIA (2008), http://sardes.inrialpes.fr/papers/files/RR-6664.pdf
9. Merro, M., Zappa Nardelli, F.: Behavioral theory for mobile ambients. Journal of the ACM 52(6) (2005)
10. Sangiorgi, D.: Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms. PhD thesis, Department of Computer Science, University of Edinburgh (1992)
11. Sangiorgi, D.: Bisimulation for higher-order process calculi. Information and Computation 131(2) (1996)
12. Sangiorgi, D., Walker, D.: The Pi-Calculus: A Theory of Mobile Processes. Cambridge University Press, Cambridge (2001)
13. Schmitt, A., Stefani, J.B.: The Kell Calculus: A Family of Higher-Order Distributed Process Calculi. In: Priami, C., Quaglia, P. (eds.) GC 2004. LNCS, vol. 3267, pp. 146–178. Springer, Heidelberg (2005)
14. Thomsen, B.: Plain chocs: A second generation calculus for higher order processes. Acta Informatica 30(1) (1993)

Reactive Systems, Barbed Semantics, and the Mobile Ambients[⋆]

Filippo Bonchi[1,2], Fabio Gadducci[1], and Giacoma Valentina Monreale[1]

[1] Dipartimento di Informatica, Università di Pisa, Italy
[2] Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands

Abstract. Reactive systems, proposed by Leifer and Milner, represent a meta-framework aimed at deriving behavioral congruences for those specification formalisms whose operational semantics is provided by rewriting rules. Despite its applicability, reactive systems suffered so far from two main drawbacks. First of all, no technique was found for recovering a set of inference rules, e.g. in the so-called SOS style, for describing the distilled observational semantics. Most importantly, the efforts focused on strong bisimilarity, tackling neither weak nor barbed semantics.

Our paper addresses both issues, instantiating them on a calculus whose semantics is still in a flux: Cardelli and Gordon's mobile ambients.

While the solution to the first issue is tailored over our case study, we provide a general framework for recasting (weak) barbed equivalence in the reactive systems formalism. Moreover, we prove that our proposal captures the behavioural semantics for mobile ambients proposed by Rathke and Sobociński and by Merro and Zappa Nardelli.

1 Introduction

Reactive systems [1] were proposed by Leifer and Milner as an abstract formalism for specifying the dynamics of a computational device. Indeed, the usual specification technique is based on a reduction system: a set, representing the possible states of the device; and a relation among these states, usually inductively defined, representing the possible evolutions of the device. Despite the advantage of conveying the semantics with relatively few compact rewriting rules, freely instantiated and contextualized, the main drawback of reduction-based solutions is poor compositionality, since the dynamic behaviour of arbitrary standalone terms can be interpreted only by inserting them in appropriate contexts, where a reduction may take place. The theoretical appeal of reactive systems is their ability to distill labelled transition systems (LTSs), hence, behavioural equivalences, for devices specified by a reduction system. The idea is simple: whenever a device specified by a term $C[P]$, i.e., by a subterm P inserted into a (unary) context $C[-]$, may evolve to a state Q, the associated labelled transition system has a transition $P \xrightarrow{C[-]} Q$, i.e., the state P evolves into Q with a label $C[-]$.

[⋆] Partly supported by the EU FP6-IST IP 16004 SEnSOria and carried out during the first author's tenure of an ERCIM "Alain Bensoussa" Fellowship Programme.

If all contexts are admitted, the resulting semantics is called saturated semantics, and the resulting strong bisimilarity on the derived LTS is a congruence. Clearly, it becomes unfeasible to check the bisimulation game. Hence, it is necessary to consider a subset of contexts guaranteeing that the distilled behavioural semantics is a congruence. Such a set, the "minimal" contexts allowing a reduction to occur, was identified in [1] by the categorical notion of *relative pushout*: the resulting strong bisimilarity is a congruence, even if it often does not coincide with the saturated one. Semi-saturated equivalences [2,3] were introduced for recovering saturated semantics. The bisimulation game becomes asymmetric, and a minimal context may be matched by any context. They capture saturated semantics, yet slashing the number of transitions that need to be checked.

Several attempts were made to encode specification formalisms (Petri nets [4,5], logic programming [2], etc.) as reactive systems, either hoping to recover the standard observational equivalence, whenever such a behavioural semantics exists (CCS [6], pi-calculus [7], etc.), or trying to distill a meaningful semantics. The results are not yet fully satisfactory. On the one-side, bisimilarity via minimal contexts is usually too fine-grained. On the other side, saturated semantics are often too coarse, and for e.g. CCS the standard strong bisimilarity is strictly included in the saturated one. The situation is potentially worse for weak saturated semantics. Intuitively, since in weak semantics the observer can not check for the occurrence of reductions, all systems are observationally equivalent.

Barbs were introduced by Milner and Sangiorgi [8] for addressing this kind of problem. Intuitively, a barb is just a predicate on the states of a system, and barbed equivalences add the check of such predicates in the bisimulation game. The flexibility of the definition allows for recasting a wide variety of observational, bisimulation-based equivalences. Theoretically, the main contribution of our paper is the introduction of suitable notions of barbed and weak barbed saturated semantics for reactive systems, and their characterization via transition systems labelled with minimal contexts, by exploiting the semi-saturated game.

The results above may have potentially far reaching consequences over the usability for the reactive system formalism. However, their adequacy has to be properly established, by checking it against suitable case studies. To this end, we instantiate our proposal over a calculus whose observational semantics is still in a flux, namely, the calculus of mobile ambients, proposed by Cardelli and Gordon [9]. We proceed as follows. First of all, we consider a minimal context semantics for ambients, as distilled in [10] by means of a graphical encoding. Another drawback of reactive systems is that such distilled semantics are usually not in a standard form. So, we propose an alternative, yet equivalent presentation of that LTS, by means of a set of SOS rules. This is a first step toward a possible overcoming of the problem, but here we use such a characterization to establish that the resulting LTS is the same as the one previously proposed by Rathke and Sobociński [11]. This is pivotal in proving our main practical result, namely, that barbed and weak barbed semi-saturated semantics for mobile ambients do capture the strong and weak barbed congruences for the calculus, as proposed by Rathke and Sobociński [11] and by Merro and Zappa Nardelli [12], respectively.

The paper is organized as follows. Section 2 recalls the basic notions of reactive systems, while Section 3 similarly introduces the main definitions concerning mobile ambients. Section 4 presents the LTS for ambients that we synthesized in [10], introduces a novel characterization of it by means of SOS rules, and finally proves its equivalence with the LTS proposed by Rathke and Sobociński in [11]. Section 5 presents the technical core of the paper, the introduction of barbed and weak barbed saturated semantics for reactive systems, and offers a labelled characterization by means of their semi-saturated counterparts. Finally, Section 6 proves that the two barbed semi-saturated bisimilarities we introduced capture the barbed congruences proposed so far for mobile ambients.

2 Reactive Systems

This section summarizes the main results concerning (the theory of) reactive systems (RSs) [1]. The formalism aims at deriving labelled transition systems (LTSs) and bisimulation congruences for a system specified by a reduction semantics, and it is centered on the concepts of *term*, *context* and *reduction rules*: contexts are arrows of a category, terms are arrows having as domain 0, a special object that denotes groundness, and reduction rules are pairs of (ground) terms.

Definition 1 (Reactive System). *A* reactive system $\mathbb{C}$ *consists of*

1. *a category* $\mathbf{C}$*;*
2. *a distinguished object* $0 \in |\mathbf{C}|$*;*
3. *a composition-reflecting subcategory* $\mathbf{D}$ *of* reactive contexts*;*
4. *a set of pairs* $\mathfrak{R} \subseteq \bigcup_{I \in |\mathbf{C}|} \mathbf{C}(0, I) \times \mathbf{C}(0, I)$ *of* reduction rules.

Intuitively, reactive contexts are those in which a reduction can occur. By composition-reflecting we mean that $d' \circ d \in \mathbf{D}$ implies $d, d' \in \mathbf{D}$. Note that the rules have to be ground, i.e., left-hand and right-hand sides have to be terms without holes and, moreover, with the same codomain.

The reduction relation is generated from the reduction rules by closing them under all reactive contexts. Formally, the *reduction relation* is defined by taking $P \rightsquigarrow Q$ if there is $\langle l, r \rangle \in \mathfrak{R}$ and $d \in \mathbf{D}$ such that $P = d \circ l$ and $Q = d \circ r$.

Thus the behaviour of a RS is expressed as an unlabelled transition system. In order to obtain a LTS, we can plug a term P into some context $C[-]$ and observe if a reduction occurs. In this case we have that $P \xrightarrow{C[-]}$. Categorically speaking, this means that $C[-] \circ P$ matches $d \circ l$ for some rule $\langle l, r \rangle \in \mathfrak{R}$ and some reactive context d. This situation is formally depicted by diagram (i) in Fig. 1: a commuting diagram like this is called a *redex square*.

Definition 2 (Saturated Transition System). *The* saturated transition system *(*STS*) is defined as follows*

- *states: arrows* $P : 0 \to I$ *in* $\mathbf{C}$*, for arbitrary* I*;*
- *transitions:* $P \xrightarrow{C[-]}_{SAT} Q$ *if* $C[P] \rightsquigarrow Q$*.*

$$\begin{tikzcd}
 & I_4 & \\
I_2 \arrow[ur, "C[-]"] & & I_3 \arrow[ul, "d"'] \\
 & 0 \arrow[ul, "P"] \arrow[ur, "l"'] &
\end{tikzcd}$$

(i)

$$\begin{tikzcd}
 & I_4 & \\
I_2 \arrow[ur, "C[-]"] \arrow[r, "e"'] & I_5 \arrow[u, "g"'] & I_3 \arrow[ul, "d"'] \arrow[l, "f"] \\
 & 0 \arrow[ul, "P"] \arrow[ur, "l"'] &
\end{tikzcd}$$

(ii)

$$\begin{tikzcd}
 & I_6 & \\
I_2 \arrow[ur, "e'"] \arrow[r, "e"'] & I_5 \arrow[u, "h"'] & I_3 \arrow[ul, "f'"'] \arrow[l, "f"]
\end{tikzcd}$$

(iii)

$$\begin{tikzcd}
 & I_4 \\
I_6 \arrow[ur, "g'"] & I_5 \arrow[u, "g"'] \arrow[l, "h"]
\end{tikzcd}$$

(iv)

Fig. 1. Redex Square and RPO

Note that $C[P]$ is a stand-in for $C[-] \circ P$: the same notation is chosen in Definitions 3 and 6 below, in order to allows for an easier comparison with the process calculi notation, to be adopted in the following sections.

Definition 3 (Saturated Bisimulation). Saturated bisimilarity $\sim^S$ *is the largest symmetric relation* $\mathcal{R}$ *such that whenever* $P \, \mathcal{R} \, Q$ *then* $\forall C[-]$

– *if* $C[P] \rightsquigarrow P'$ *then* $C[Q] \rightsquigarrow Q'$ *and* $P' \, \mathcal{R} \, Q'$.

It is obvious that $\sim^S$ is a congruence. Indeed, it is the coarsest symmetric relation satisfying the bisimulation game on $\rightsquigarrow$ that is also a congruence.

Note that STS is often infinite-branching since all contexts allowing reductions may occur as labels. Moreover, it has redundant transitions. For example, consider the term $a.0$ of CCS. We have both the transitions $a.0 \overset{\overline{a}.0|-}{\rightarrow}_{SAT} 0|0$ and $a.0 \overset{P|\overline{a}.0|-}{\rightarrow}_{SAT} P \mid 0 \mid 0$, yet P does not "concur" to the reduction. We thus need a notion of "minimal context allowing a reduction", captured by *idem pushouts.*

Definition 4 (RPO, IPO). *Let the diagrams in Fig. 1 be in a category* $\mathbf{C}$*, and let (i) be a commuting diagram. A* candidate *for (i) is any tuple* $\langle I_5, e, f, g\rangle$ *making (ii) commute. A* relative pushout (RPO) *is the smallest such candidate, i.e., such that for any other candidate* $\langle I_6, e', f', g'\rangle$ *there exists a unique morphism* $h : I_5 \rightarrow I_6$ *making (iii) and (iv) commute. A commuting square such as diagram (i) of Fig. 1 is called* idem pushout (IPO) *if* $\langle I_4, c, d, id_{I_4}\rangle$ *is its RPO.*

Hereafter, we say that a RS *has redex RPOs* (*IPOs*) if every redex square has an RPO (IPO) as candidate. For a better understanding of these two notions, we refer the reader to [2]. For the aim of this paper it is enough to know that the former notion is more restrictive than the latter.

Definition 5 (IPO-Labelled Transition System). *The* IPO-labelled transition system *(*ITS*) is defined as follows*

– *states:* $P : 0 \rightarrow I$ *in* $\mathbf{C}$*, for arbitrary* I*;*
– *transitions:* $P \overset{C[-]}{\rightarrow}_{IPO} d \circ r$ *if* $d \in \mathbf{D}$*,* $\langle l, r\rangle \in \mathfrak{R}$*, and (i) in Fig. 1 is an IPO.*

In other words, if inserting P into the context $C[-]$ matches $d \circ l$, and $C[-]$ is the "smallest" such context, then P transforms to $d \circ r$ with label $C[-]$.

Bisimilarity on ITS is referred to as *IPO-bisimilarity* ($\sim^I$). Leifer and Milner have shown that if the RS has redex RPOs, then it is a congruence.

Proposition 1. *In a reactive system having redex-RPOs, $\sim^I$ is a congruence.*

Clearly, $\sim^I \subseteq \sim^S$. In [2,3] the first author, with König and Montanari, shows that this inclusion is strict for many formalisms and introduces *semi-saturation.*

Definition 6 (Semi-Saturated Bisimulation). Semi-saturated bisimilarity $\sim^{SS}$ *is the largest symmetric relation $\mathcal{R}$ such that whenever $P \, \mathcal{R} \, Q$ then*

– *if $P \xrightarrow{C[-]}_{IPO} P'$ then $C[Q] \rightsquigarrow Q'$ and $P' \, \mathcal{R} \, Q'$.*

Proposition 2. *In a reactive system having redex-IPOs, $\sim^{SS} = \sim^S$.*

3 Mobile Ambients

This section shortly recalls the finite, communication-free fragment of mobile ambients (MAs) [9], its reduction semantics and behavioural equivalences.

Fig. 2 shows the syntax of the calculus. We assume a set $\mathcal{N}$ of *names* ranged over by $m, n, u, \ldots$. Besides the standard constructors, we included a set of *process variables* $\mathcal{X} = \{X, Y, \ldots\}$, and a set of *name variables* $\mathcal{V} = \{x, y, \ldots\}$. Intuitively, an extended process $x[P]|X$ represents an underspecified process, where either the process X or the name of the ambient $x[-]$ can be further instantiated. These are needed for the presentation of the LTSs in Section 4.

We let $P, Q, R, \ldots$ range over the set $\mathcal{P}$ of *pure* processes, containing neither process nor name variables; while $P_\epsilon, Q_\epsilon, R_\epsilon, \ldots$ range over the set $\mathcal{P}_\epsilon$ of *well-formed* processes, i.e., such that no process or ambient variable occurs twice.

We use the standard definitions for the set of free names of a pure process P, denoted by $fn(P)$, and for α-convertibility, with respect to the restriction operators (νn). We moreover assume that $fn(X) = fn(x[\mathbf{0}]) = \emptyset$. We also consider a family of *substitutions*, which may replace a process/name variable with a pure process/name, respectively. Substitutions avoid name capture: for a pure process P, the expression $(\nu n)(\nu m)(X|x[\mathbf{0}])\{^m/_x, ^{n[P]}/_X\}$ corresponds to the pure process $(\nu p)(\nu q)(n[P]|m[\mathbf{0}])$, for names $p, q \notin \{m\} \cup fn(n[P])$.

The semantics of the calculus exploits a *structural congruence*, denoted by $\equiv$, which is the least equivalence on pure processes that satisfies the axioms shown in Fig. 3. We assume that the structural congruence defined on processes over the extended syntax is induced by the same set of rules shown in Fig. 3.

The *reduction relation*, denoted by $\rightsquigarrow$, describes the evolution of pure processes over time. It is the smallest relation closed under the congruence $\equiv$ and inductively generated by the set of axioms and inference rules shown in Fig. 4.

A *strong barb* o is a predicate over the states of a system, with $P \downarrow_o$ denoting that P satisfies o. In MAs, $P \downarrow_n$ denotes the presence at top-level of a unrestricted ambient n. Formally, for a pure process P, $P \downarrow_n$ if $P \equiv (\nu A)(n[Q]|R)$ and $n \notin A$, for some processes Q and R and a set of restricted names A. A pure process P satisfies the *weak barb* n (denoted as $P \Downarrow_n$) if there exists a process P' such that $P \rightsquigarrow^* P'$ and $P' \downarrow_n$, where $\rightsquigarrow^*$ is the transitive and reflexive closure of $\rightsquigarrow$.

$P ::= \mathbf{0}, n[P], M.P, (\nu n)P, P_1 \vert P_2, X, x[P]$	$M ::= \mathit{in}\ n, \mathit{out}\ n, \mathit{open}\ n$

Fig. 2. (Extended) Syntax of mobile ambients

if $P \equiv Q$ then $P \vert R \equiv Q \vert R$	$P \vert \mathbf{0} \equiv P$	
if $P \equiv Q$ then $(\nu n)P \equiv (\nu n)Q$	$(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$	
if $P \equiv Q$ then $n[P] \equiv n[Q]$	$(\nu n)(P \vert Q) \equiv P \vert (\nu n)Q$	if $n \notin fn(P)$
if $P \equiv Q$ then $M.P \equiv M.Q$	$(\nu n)m[P] \equiv m[(\nu n)P]$	if $n \neq m$
$P \vert Q \equiv Q \vert P$	$(\nu n)M.P \equiv M.(\nu n)P$	if $n \notin fn(M)$
$(P \vert Q) \vert R \equiv P \vert (Q \vert R)$	$(\nu n)P \equiv (\nu m)(P\{^m/_n\})$	if $m \notin fn(P)$

Fig. 3. Structural congruence

$n[\mathit{in}\ m.P \vert Q] \vert m[R] \rightsquigarrow m[n[P \vert Q] \vert R]$	if $P \rightsquigarrow Q$ then $(\nu n)P \rightsquigarrow (\nu n)Q$
$m[n[\mathit{out}\ m.P \vert Q] \vert R] \rightsquigarrow n[P \vert Q] \vert m[R]$	if $P \rightsquigarrow Q$ then $n[P] \rightsquigarrow n[Q]$
$\mathit{open}\ n.P \vert n[Q] \rightsquigarrow P \vert Q$	if $P \rightsquigarrow Q$ then $P \vert R \rightsquigarrow Q \vert R$

Fig. 4. Reduction relation on pure processes

Definition 7 (Reduction Barbed Congruences [11,12]). Strong reduction barbed congruence $\cong$ *is the largest symmetric relation* $\mathcal{R}$ *such that whenever* $P \mathrel{\mathcal{R}} Q$ *then*

– *if* $P \downarrow_n$ *then* $Q \downarrow_n$*;*
– *if* $P \rightsquigarrow P'$ *then* $Q \rightsquigarrow Q'$ *and* $P' \mathrel{\mathcal{R}} Q'$*;*
– $\forall C[-], C[P] \mathrel{\mathcal{R}} C[Q]$.

Weak reduction barbed congruence $\cong^W$ *is the largest symmetric relation* $\mathcal{R}$ *such that whenever* $P \mathrel{\mathcal{R}} Q$ *then*

– *if* $P \downarrow_n$ *then* $Q \Downarrow_n$*;*
– *if* $P \rightsquigarrow P'$ *then* $Q \rightsquigarrow^* Q'$ *and* $P' \mathrel{\mathcal{R}} Q'$*;*
– $\forall C[-], C[P] \mathrel{\mathcal{R}} C[Q]$.

Labelled characterization of reduction barbed congruences over MAs processes are presented by Rathke and Sobociński for the strong case [11], and by Merro and Zappa Nardelli for the weak one [12].

The main result of this paper is the proposal of a novel notion of barbed saturated bisimilarity over reactive systems, both for the strong and weak case, that is able to capture the two behavioural semantics for MAs defined above.

4 Labelled Transition Systems for Mobile Ambients

In this section we first recall the IPO-transition system for MAs, previously proposed in [10], and then we introduce a SOS presentation for it. Finally, we

$$(\textsc{Tau})\ \frac{P \rightsquigarrow Q}{P \xrightarrow{-} Q} \qquad (\textsc{Out})\ \frac{P \equiv (\nu A)(out\ m.P_1|P_2) \quad m \notin A}{P \xrightarrow{m[x[-|X_1]|X_2]} (\nu A)(m[X_2]|x[P_1|P_2|X_1])}$$

$$(\textsc{In})\ \frac{P \equiv (\nu A)(in\ m.P_1|P_2) \quad m \notin A}{P \xrightarrow{x[-|X_1]|m[X_2]} (\nu A)m[x[P_1|P_2|X_1]|X_2]} \qquad (\textsc{OutAmb})\ \frac{P \equiv (\nu A)(n[out\ m.P_1|P_2]|P_3) \quad m \notin A}{P \xrightarrow{m[-|X_1]} (\nu A)(m[P_3|X_1]|n[P_1|P_2])}$$

$$(\textsc{InAmb})\ \frac{P \equiv (\nu A)(n[in\ m.P_1|P_2]|P_3) \quad m \notin A}{P \xrightarrow{-|m[X_1]} (\nu A)(m[n[P_1|P_2]|X_1]|P_3)} \qquad (\textsc{Open})\ \frac{P \equiv (\nu A)(open\ n.P_1|P_2) \quad n \notin A}{P \xrightarrow{-|n[X_1]} (\nu A)(P_1|P_2|X_1)}$$

$$(\textsc{CoIn})\ \frac{P \equiv (\nu A)(m[P_1]|P_2) \quad m \notin A}{P \xrightarrow{-|x[in\ m.X_1|X_2]} (\nu A)(m[x[X_1|X_2]|P_1]|P_2)} \qquad (\textsc{CoOpen})\ \frac{P \equiv (\nu A)(n[P_1]|P_2) \quad n \notin A}{P \xrightarrow{-|open\ n.X_1} (\nu A)(P_1|X_1|P_2)}$$

Fig. 5. The LTS $\mathcal{D}$

discuss the relationship between our SOS LTS and the LTS for MAs proposed by Rathke and Sobociński in [11]. Note that we implicitly assume that all the LTSs that we define are closed with respect to structural congruence.

4.1 An IPO-LTS for Mobile Ambients

This section presents the ITS $\mathcal{D}$ for MAs proposed in [10]. The inference rules describing this LTS are obtained from an analysis of a LTS over (processes as) graphs, derived by the borrowed context mechanism [13], which is an instance of the theory of RSs [14]. The labels of the transitions are unary contexts, i.e., terms of the extended syntax with a hole $-$. Note that they are minimal contexts, that is, they represent the exact amount of context needed by a system to react. We denote them by $C_\epsilon[-]$. The formal definition of the LTS is presented in Fig. 5.

The rule Tau represents the τ-actions modeling internal computations. Notice that the labels of the transitions are identity contexts composed of just a hole $-$, while the resulting states are processes over MAs standard syntax.

The other rules in Fig. 5 model the interactions of a process with its environment. Note that both labels and resulting states contain process and name variables. We define the LTS $\mathcal{D}_\mathcal{I}$ for processes over the standard syntax of MAs by instantiating all the variables of the labels and of the resulting states.

Definition 8. *Let P, Q be pure processes and let $C[-]$ be a pure context. Then, we have that $P \xrightarrow{C[-]}_{\mathcal{D}_\mathcal{I}} Q$ if there exists a transition $P \xrightarrow{C_\epsilon[-]}_{\mathcal{D}} Q_\epsilon$ and a substitution σ such that $Q_\epsilon\sigma = Q$ and $C_\epsilon[-]\sigma = C[-]$.*

In the above definition we implicitly consider only ground substitutions. Moreover, we recall that the substitutions do not capture bound names.

The rule Open models the opening of an ambient provided by the environment. In particular, it enables a process P with a capability *open* n at top level, for $n \in fn(P)$, to interact with a context providing an ambient n containing some process X_1. Note that the label $-|n[X_1]$ of the rule represents the minimal context needed by the process P for reacting. The resulting state is the process over the extended syntax $(\nu A)(P_1|X_1|P_2)$, where X_1 represents a process provided by the environment. Note that the instantiation of the process variable X_1

with a process containing a free name that belongs to the bound names in A is possible only α-converting the resulting process $(\nu A)(P_1|X_1|P_2)$ into a process that does not contain that name among its bound names at top level.

The rule CoOpen instead models an environment that opens an ambient of the process. The rule InAmb enables an ambient of the process to migrate into a sibling ambient provided by the environment, while in the rule In both the ambients are provided by the environment. In the rule CoIn an ambient provided by the environment enters an ambient of the process. The rule OutAmb models an ambient of the process exiting from an ambient provided by the environment, while in the rule Out both ambients are provided by the environment.

The LTS $\mathcal{D}$ does not conform to the so-called SOS style [15]: indeed, the premises of the inference rules are just constraints over the structure of the process, as typical of the LTSs distilled by IPOs. In the next section we propose an alternative, yet equivalent set of rules in SOS style, which allows an easier comparison between our proposal and the one by Rathke and Sobociński.

4.2 A SOS Presentation for the IPO-LTS

This section proposes a SOS presentation for the ITS $\mathcal{D}$ shown in the previous section. The SOS LTS $\mathcal{S}$ is directly obtained from the LTS $\mathcal{D}$ and it coincides with this last one. The rules of the LTS $\mathcal{S}$ are shown in Fig. 6.

The rules in the first two rows of Fig. 6 model internal computations. They are indeed obtained from the first rule in Fig. 5. In particular, since this last rule exactly derives the same transition relation of the reduction relation over MAs, we replace it with the reduction rules labelled with the identity context $-$. So, we obtain the axioms modelling the execution of the capabilities of the calculus, and a structural rule for each ambient, parallel and restriction operators.

Also the remaining rules in Fig. 6, modelling the interactions of a process with its environment, are obtained from the other rules in Fig. 5. In particular, for each of these rules we derive three rules. First, we determine the axiom by considering the minimal process needed by the reduction to occur. For e.g. the rule In of the LTS $\mathcal{D}$, the minimal process allowing the reduction is $in\, m.P_1$. Therefore, we determine the axiom $in\, m.P_1 \xrightarrow{x[-|X_1]|m[X_2]} m[x[P_1|X_1]|X_2]$. The next step consists in determining the relative structural rules in SOS style. So, as far as the rule In of the LTS $\mathcal{D}$ is concerned, we have that if $P \xrightarrow{x[-|X_1]|m[X_2]} P\epsilon$, then for the process $P|Q$ there is a transition labelled $x[-|X_1]|m[X_2]$ leading to the process P_ϵ with the process Q inside the ambient x, that is, $P|Q \xrightarrow{x[-|X_1]|m[X_2]} P_\epsilon\{^{Q|X_1}/_{X_1}\}$. Instead, if $P \xrightarrow{x[-|X_1]|m[X_2]} P\epsilon$ and $m \neq a$, then $(\nu a)P \xrightarrow{x[-|X_1]|m[X_2]} (\nu a)P_\epsilon$.

Theorem 1. *Let P be a pure process and let $C_\epsilon[-]$ be a context. Then, $P \xrightarrow{C_\epsilon[-]}_{\mathcal{D}} Q_\epsilon$ if and only if $P \xrightarrow{C_\epsilon[-]}_{\mathcal{S}} Q_\epsilon$.*

As for ITS $\mathcal{D}$, also for LTS $\mathcal{S}$ we define the LTS $\mathcal{S}_{\mathcal{I}}$ for pure processes by instantiating all the variables of the labels and of the resulting states.

(InTau) $\dfrac{}{n[in\ m.P\vert Q]\vert m[R]\xrightarrow{-}m[n[P\vert Q]\vert R]}$	(OutTau) $\dfrac{}{m[n[out\ m.P\vert Q]\vert R]\xrightarrow{-}n[P\vert Q]\vert m[R]}$	(OpenTau) $\dfrac{}{open\ n.P\vert n[Q]\xrightarrow{-}P\vert Q}$
(TauAmb) $\dfrac{P\xrightarrow{-}P'}{n[P]\xrightarrow{-}n[P']}$	(TauPar) $\dfrac{P\xrightarrow{-}P'}{P\vert Q\xrightarrow{-}P'\vert Q}$	(TauRes) $\dfrac{P\xrightarrow{-}P'}{(\nu a)P\xrightarrow{-}(\nu a)P'}$
(In) $\dfrac{}{in\ m.P_1\xrightarrow{x[-\vert X_1]\vert m[X_2]}m[x[P_1\vert X_1]\vert X_2]}$	(InPar) $\dfrac{P\xrightarrow{x[-\vert X_1]\vert m[X_2]}P_\epsilon}{P\vert Q\xrightarrow{x[-\vert X_1]\vert m[X_2]}P_\epsilon\{^{Q\vert X_1}/_{X_1}\}}$	(InRes) $\dfrac{P\xrightarrow{x[-\vert X_1]\vert m[X_2]}P_\epsilon \quad a\neq m}{(\nu a)P\xrightarrow{x[-\vert X_1]\vert m[X_2]}(\nu a)P_\epsilon}$
(InAmb) $\dfrac{P\xrightarrow{x[-\vert X_1]\vert m[X_2]}P_\epsilon}{n[P]\xrightarrow{-\vert m[X_2]}P_\epsilon\{^n/_x,^{\mathbf{0}}/_{X_1}\}}$	(InAmbPar) $\dfrac{P\xrightarrow{-\vert m[X_2]}P_\epsilon}{P\vert Q\xrightarrow{-\vert m[X_2]}P_\epsilon\vert Q}$	(InAmbRes) $\dfrac{P\xrightarrow{-\vert m[X_2]}P_\epsilon \quad a\neq m}{(\nu a)P\xrightarrow{-\vert m[X_2]}(\nu a)P_\epsilon}$
(CoIn) $\dfrac{}{m[P_1]\xrightarrow{-\vert x[in\ m.X_1\vert X_2]}m[x[X_1\vert X_2]\vert P_1]}$	(CoInPar) $\dfrac{P\xrightarrow{-\vert x[in\ m.X_1\vert X_2]}P_\epsilon}{P\vert Q\xrightarrow{-\vert x[in\ m.X_1\vert X_2]}P_\epsilon\vert Q}$	(CoInRes) $\dfrac{P\xrightarrow{-\vert x[in\ m.X_1\vert X_2]}P_\epsilon \quad a\neq m}{(\nu a)P\xrightarrow{-\vert x[in\ m.X_1\vert X_2]}(\nu a)P_\epsilon}$
(Out) $\dfrac{}{out\ m.P_1\xrightarrow{m[x[-\vert X_1]\vert X_2]}m[X_2]\vert x[P_1\vert X_1]}$	(OutPar) $\dfrac{P\xrightarrow{m[x[-\vert X_1]\vert X_2]}P_\epsilon}{P\vert Q\xrightarrow{m[x[-\vert X_1]\vert X_2]}P_\epsilon\{^{Q\vert X_1}/_{X_1}\}}$	(OutRes) $\dfrac{P\xrightarrow{m[x[-\vert X_1]\vert X_2]}P_\epsilon \quad a\neq m}{(\nu a)P\xrightarrow{m[x[-\vert X_1]\vert X_2]}(\nu a)P_\epsilon}$
(OutAmb) $\dfrac{P\xrightarrow{m[x[-\vert X_1]\vert X_2]}P_\epsilon}{n[P]\xrightarrow{m[-\vert X_2]}P_\epsilon\{^n/_x,^{\mathbf{0}}/_{X_1}\}}$	(OutAmbPar) $\dfrac{P\xrightarrow{m[-\vert X_2]}P_\epsilon}{P\vert Q\xrightarrow{m[-\vert X_2]}P_\epsilon\{^{Q\vert X_2}/_{X_2}\}}$	(OUtAmbRes) $\dfrac{P\xrightarrow{m[-\vert X_2]}P_\epsilon \quad a\neq m}{(\nu a)P\xrightarrow{m[-\vert X_2]}(\nu a)P_\epsilon}$
(Open) $\dfrac{}{open\ n.P_1\xrightarrow{-\vert n[X_1]}P_1\vert X_1}$	(OpenPar) $\dfrac{P\xrightarrow{-\vert n[X_1]}P_\epsilon}{P\vert Q\xrightarrow{-\vert n[X_1]}P_\epsilon\vert Q}$	(OpenRes) $\dfrac{P\xrightarrow{-\vert n[X_1]}P_\epsilon \quad a\neq n}{(\nu a)P\xrightarrow{-\vert n[X_1]}(\nu a)P_\epsilon}$
(CoOpen) $\dfrac{}{n[P_1]\xrightarrow{-\vert open\ n.X_1}P_1\vert X_1}$	(CoOpenPar) $\dfrac{P\xrightarrow{-\vert open\ n.X_1}P_\epsilon}{P\vert Q\xrightarrow{-\vert open\ n.X_1}P_\epsilon\vert Q}$	(CoOpenRes) $\dfrac{P\xrightarrow{-\vert open\ n.X_1}P_\epsilon \quad a\neq n}{(\nu a)P\xrightarrow{-\vert open\ n.X_1}(\nu a)P_\epsilon}$

Fig. 6. The LTS $\mathcal{S}$

4.3 Equivalence between LTSs

We now show that LTS $\mathcal{S}_\mathcal{I}$ defined on processes over the standard syntax of MAs coincides with the LTS for MAs proposed by Rathke and Sobociński in [11].

α	$C^{\alpha}_{\epsilon}[-]$	α	$C^{\alpha}_{\epsilon}[-]$
$in\ m$	$x[-\vert X_1]\vert m[X_2]$	$[out\ m]$	$m[-\vert X_2]$
$[in\ m]$	$-\vert m[X_2]$	$open\ n$	$-\vert n[X_1]$
$\overline{[in\ m]}$	$-\vert x[in\ m.X_1\vert X_2]$	$\overline{open\ n}$	$-\vert open\ n.X_1$
$out\ m$	$m[x[-\vert X_1]\vert X_2]$	τ	$-$

Fig. 7. Correspondence between the labels of the LTS $\mathcal{C}$ and the ones of the LTS $\mathcal{S}$

Their LTS is organized into three components: the process-view LTS $\mathcal{C}$, the context-view LTS $\mathcal{A}$, and the combined LTS $\mathcal{CA}$. The labels of the LTS $\mathcal{CA}$ have the shape $\alpha \downarrow \vec{M}$, where α is derived by LTS $\mathcal{C}$, and $\vec{M}$ by LTS $\mathcal{A}$. In a transition $P \xrightarrow{\alpha\downarrow\vec{M}}_{\mathcal{CA}} Q$, the label α identifies the minimal context needed by the pure process P to react, while $\vec{M}$ is a list of pure processes and ambient names, representing an instantiation of the context components. Therefore, since the labels of our LTS $\mathcal{S}$ are precisely the minimal contexts needed by a system to react, we establish a one-to-one correspondence between the labels α of LTS $\mathcal{C}$ and the labels $C_{\epsilon}[-]$ of our LTS $\mathcal{S}$. Fig. 7 shows this correspondence: $C^{\alpha}_{\epsilon}[-]$ denotes the label of our LTS $\mathcal{S}$ corresponding to the label α of their LTS $\mathcal{C}$.

In the label $\alpha \downarrow \vec{M}$ the list $\vec{M}$ represents an instantiation for the components of the context identified by α. Therefore, since the contexts identified by the α's correspond to the contexts representing our labels, the list $\vec{M}$ of $\alpha \downarrow \vec{M}$ contains as many elements as the variables of the label $C^{\alpha}[-]$. In particular, $\vec{M}$ contains k processes if and only if k process variables occur in $C^{\alpha}[-]$, and analogously, $\vec{M}$ contains h ambient names if and only if h name variables occur in $C^{\alpha}[-]$.

To better understand their relationship, we informally discuss an example showing how a transition in LTS $\mathcal{S}_{\mathcal{I}}$ corresponds to a transition in LTS $\mathcal{CA}$.

Example 1. Consider the pure process $P = in\ m.P_1$, for some process P_1. According to In rule in Fig. 6, $P \xrightarrow{x[-|X_1]|m[X_2]}_{\mathcal{S}} m[x[P_1|X_1]|X_2]$. If we consider a substitution $\sigma = \{{}^{P_2}/_{X_1}, {}^{n}/_{x}, {}^{P_3}/_{X_2}\}$, where P_2, P_3 are processes and n is an ambient name, we have $P \xrightarrow{n[-|P_2]|m[P_3]}_{\mathcal{S}_{\mathcal{I}}} m[n[P_1|P_2]|P_3]$.

Consider now $\mathcal{CA}$. According to In rule of [11], $P \xrightarrow{in\, m}_{\mathcal{C}} \lambda XxY.m[x[P_1|X]|Y]$, where X, Y and x are variables representing the components provided by the context. In particular, X and Y are process variables and x is a name variable. Consider the instantiation for context components induced by substitution σ, namely, $\vec{M} : P_2, n, P_3$. According to rule Inst of [11], we obtain $\lambda XxY.m[x[P_1|X]|Y] \xrightarrow{\vec{M}\downarrow}_{\mathcal{A}} m[n[P_1|P_2]|P_3]$ and, by applying Cλ rule of [11], $P \xrightarrow{in\, m\downarrow\vec{M}}_{\mathcal{CA}} m[n[P_1|P_2]|P_3]$. Therefore, both LTSs have a transition leading the process P to $m[n[P_1|P_2]|P_3]$.

Theorem 2. *Let P be a pure process. If $P \xrightarrow{\alpha\downarrow\vec{M}}_{\mathcal{CA}} Q$, then there is a unique (up-to $\equiv$) substitution σ s.t. $P \xrightarrow{C^{\alpha}_{\epsilon}[-]\sigma}_{\mathcal{S}_{\mathcal{I}}} Q$. Vice versa, if $P \xrightarrow{C[-]}_{\mathcal{S}_{\mathcal{I}}} Q$, then there are α and a unique (up-to $\equiv$) tuple $\vec{M}$ s.t. $C[-] = C^{\alpha}[-]$ and $P \xrightarrow{\alpha\downarrow\vec{M}}_{\mathcal{CA}} Q$.*

5 Barbed Semantics for Reactive Systems

Several attempts were made to encode specification formalisms (Petri nets [4,5], logic programming [2], CCS [16,17], λ-calculus [18,19], asynchronous π-calculus [20], fusion calculus [21], etc.) as RSs, either hoping to recover the standard observational equivalences, whenever such a behavioural semantics exists (CCS [6], π-calculus [7], etc.), or trying to distill a meaningful semantics. Unfortunately, IPO semantics is usually too fine-grained, and MAs are no exception.

On the other hand, saturated semantics are often too coarse. For example, the CCS processes $\omega = \tau.\Omega$ and $\Theta = \tau.\Omega + a.\Omega$ are saturated bisimilar [8], yet not strong bisimilar. This problem becomes potentially serious when considering *weak semantics*. Intuitively, two systems are saturated bisimilar if they cannot be distinguished by an external observer that, in any moment of their execution, can insert them into some context and observe a reduction. However, since in weak semantics reductions cannot be observed, all systems are equivalent.

Barbs were introduced for overcoming these problems [8]. This section proposes a notion of *barbed saturated bisimilarity* for RSs, showing that it is efficiently characterized through the IPO-transition systems by exploiting the semi-saturated game: Section 5.1 studies the strong case; Section 5.2, the weak one.

5.1 Barbed Saturated Bisimilarity

Barbed bisimilarity was introduced for CCS [8], using barbs $\downarrow_a$ and $\downarrow_{\bar{a}}$ that represented the ability of a process to perform an input, respectively an output, on channel a. As for MAs, $\downarrow_n$ means that an ambient n occurs at top level.

More generally, barbs are predicates over the states of a system. We then fix in the following a family O of barbs, and we write $P \downarrow_o$ if P satisfies $o \in O$.

Definition 9 (Barbed Bisimilarity, Barbed Congruence). *A symmetric relation $\mathcal{R}$ is a* barbed bisimulation *if whenever $P \,\mathcal{R}\, Q$ then*

– *if $P \downarrow_o$ then $Q \downarrow_o$;*
– *if $P \rightsquigarrow P'$ then $Q \rightsquigarrow Q'$ and $P' \,\mathcal{R}\, Q'$.*

Barbed bisimilarity $\sim^B$ *is the largest barbed bisimulation;* barbed congruence $\simeq^B$ *is the largest congruence contained in $\sim^B$.*

Barbed congruence is clearly a congruence, but there is no guarantee that it is also a bisimulation. In this paper, we consider a different notion of behavioural equivalence that is both a bisimulation and a congruence.

Definition 10 (Barbed Saturated Bisimulation). *A symmetric relation $\mathcal{R}$ is a* barbed saturated bisimulation *if whenever $P \,\mathcal{R}\, Q$ then $\forall C[-]$*

– *if $C[P] \downarrow_o$ then $C[Q] \downarrow_o$;*
– *if $C[P] \rightsquigarrow P'$ then $C[Q] \rightsquigarrow Q'$ and $P' \,\mathcal{R}\, Q'$.*

Barbed saturated bisimilarity $\sim^{BS}$ *is the largest barbed saturated bisimulation.*

It is easy to see that $\sim^{BS}$ is the largest barbed bisimulation that is also a congruence, and that it is finer than $\simeq^{B}$ (the largest congruence contained into barbed bisimilarity). Intuitively, in the former case the external observer can plug systems into contexts at any step of their execution, while in the latter the observer can contextualize systems only at the beginning. The former observer is more powerful than the latter, thus proving that $\sim^{BS}$ is indeed finer than $\simeq^{B}$.

It is our opinion that $\sim^{BS}$ is more appropriate, in order to model concurrent interactive systems embedded in an environment that continuously changes. And while in several formalisms the two notions coincide [22], for MAs the standard behavioural equivalence $\cong$ (Definition 7) is clearly an instance of $\sim^{BS}$.

Most importantly, though, barbed saturated bisimilarity can be efficiently characterized through the IPO-transition system via the semi-saturated game.

Definition 11 (Barbed Semi-Saturated Bisimulation). *A symmetric relation $\mathcal{R}$ is a* barbed semi-saturated bisimulation *if whenever $P \,\mathcal{R}\, Q$ then*

- *$\forall C[-]$, if $C[P] \downarrow_o$ then $C[Q] \downarrow_o$;*
- *if $P \xrightarrow{C[-]}_{IPO} P'$ then $C[Q] \rightsquigarrow Q'$ and $P' \,\mathcal{R}\, Q'$.*

Barbed semi-saturated bisimilarity $\sim^{BSS}$ *is the largest barbed semi-saturated bisimulation.*

Proposition 3. *In a reactive system having redex-IPOs, $\sim^{BSS} = \sim^{BS}$.*

Reasoning on $\sim^{BSS}$ is easier than on $\sim^{BS}$, because instead of looking at the reductions in all contexts, we consider only IPO-transitions. Even if barbs are still quantified over all contexts, for many formalisms (as for MAs) it is actually enough to check if $P \downarrow_o$ implies $Q \downarrow_o$, since this condition implies that $\forall C[-]$, if $C[P] \downarrow_o$ then $C[Q] \downarrow_o$. Barbs satisfying this property are called *contextual* barbs.

Definition 12 (Contextual Barbs). *A barb o is a* contextual barb *if whenever $P \downarrow_o$ implies $Q \downarrow_o$ then $\forall C[-]$, $C[P] \downarrow_o$ implies $C[Q] \downarrow_o$.*

5.2 Weak Barbed Saturated Bisimilarity

This section introduces weak barbed (semi-)saturated bisimilarity. We begin by recalling weak barbs. A state P satisfies the weak barb o (written $P \Downarrow_o$) if there exists a state P' such that $P \rightsquigarrow^* P'$ and $P' \downarrow_o$.

Definition 13 (Weak Barbed Saturated Bisimulation). *A symmetric relation $\mathcal{R}$ is a* weak barbed saturated bisimulation *if whenever $P \,\mathcal{R}\, Q$ then $\forall C[-]$*

- *if $C[P] \Downarrow_o$ then $C[Q] \Downarrow_o$;*
- *if $C[P] \rightsquigarrow^* P'$ then $C[Q] \rightsquigarrow^* Q'$ and $P' \,\mathcal{R}\, Q'$.*

Weak barbed saturated bisimilarity $\sim^{WBS}$ *is the largest weak barbed saturated bisimulation.*

Definition 14 (Weak Barbed Semi-Saturated Bisimulation). *A symmetric relation $\mathcal{R}$ is a* weak barbed semi-saturated bisimulation *if whenever $P\,\mathcal{R}\,Q$ then*

- *$\forall C[-]$, if $C[P] \downarrow_o$ then $C[Q] \Downarrow_o$;*
- *if $P \xrightarrow{C[-]}_{IPO} P'$ then $C[Q] \rightsquigarrow^* Q'$ and $P'\,\mathcal{R}\,Q'$.*

Weak barbed semi-saturated bisimilarity $\sim^{WBSS}$ *is the largest weak barbed semi-saturated bisimulation.*

Proposition 4. *In a reactive system having redex-IPOs, $\sim^{WBSS} = \sim^{WBS}$.*

Now we introduce weak contextual barbs. Analogously to the strong case, for those formalisms whose barbs are weakly contextual the first condition of Definition 14 becomes simpler: indeed, it suffices to check if $P \downarrow_o$ implies $Q \Downarrow_o$.

Definition 15 (Weak Contextual Barbs). *A barb o is a* weak contextual barb *if whenever $P \downarrow_o$ implies $Q \Downarrow_o$ then $\forall C[-]$, $C[P] \downarrow_o$ implies $C[Q] \Downarrow_o$.*

6 Labelled Characterizations of Barbed Congruences

This section proposes a labelled characterization of both strong and weak reduction barbed congruence for MAs, presented in Section 3. Indeed, MAs can be seen as a reactive system, with pure processes (up-to structural congruence) as ground terms: as shown in [10], pure processes must first be encoded into graphs, and the reduction semantics simulated by graph rewriting. We can then apply the *borrowed contexts* technique for distilling IPOs, which is proved to be an instance of the reactive system construction. The resulting ITS is the one that we presented in Section 4. Therefore, we can apply the notions of (weak) barbed saturated and semi-saturated bisimilarities, shown in the previous section, in order to capture those two behavioural semantics of MAs.

The first step is stated by the proposition below.

Proposition 5. *Strong reduction barbed congruence over MAs $\cong$ coincides with barbed saturated bisimilarity $\sim^{BS}$. Similarly, weak reduction barbed congruence over MAs $\cong^W$ coincides with weak barbed saturated bisimilarity $\sim^{WBS}$.*

As shown in Section 5, we can efficiently characterize (weak) barbed saturated bisimilarity through the IPO-transition system, and the semi-saturated game. We can then characterize strong and weak reduction barbed congruence over MAs by instantiating Definitions 10 and 14, respectively, with the ITS $\mathcal{S}_{\mathcal{I}}$.

Moreover, the quantification over all contexts can be removed from the first condition of the definition of (semi-)saturated bisimulation.

Proposition 6. *MAs barbs are both strong and weak contextual barbs.*

We then obtain a simpler definition of (weak) semi-saturated bisimilarity.

Definition 16 (Barbed Semi-Saturated Bisimulations for MAs). *A symmetric relation $\mathcal{R}$ is a* barbed semi-saturated bisimulation *for MAs if whenever $P \mathcal{R} Q$ then*

- *if $P \downarrow_n$ then $Q \downarrow_n$;*
- *if $P \xrightarrow{C[-]}_{\mathcal{S}_\mathcal{I}} P'$ then $C[Q] \rightsquigarrow Q'$ and $P' \mathcal{R} Q'$.*

Barbed semi-saturated bisimilarity $\sim^{BSS}_{MA}$ *is the largest barbed semi-saturated bisimulation.*

A symmetric relation $\mathcal{R}$ is a weak barbed semi-saturated bisimulation *for MAs if whenever $P \mathcal{R} Q$ then*

- *if $P \downarrow_n$ then $Q \Downarrow_n$;*
- *if $P \xrightarrow{C[-]}_{\mathcal{S}_\mathcal{I}} P'$ then $C[Q] \rightsquigarrow^* Q'$ and $P' \mathcal{R} Q'$.*

Weak barbed semi-saturated bisimilarity $\sim^{WBSS}_{MA}$ *is the largest weak barbed semi-saturated bisimulation.*

We finally introduce the main characterization theorem of the paper.

Theorem 3. *Barbed semi-saturated bisimilarity for MAs $\sim^{BSS}_{MA}$ coincides with strong reduction barbed congruence $\cong$. Similarly, weak barbed semi-saturated bisimilarity $\sim^{WBSS}_{MA}$ coincides with weak reduction barbed congruence $\cong^W$.*

It is easy to note that the two statements of the theorem above follow from Proposition 5, and from Proposition 3 and 4, respectively.

6.1 On Observing Ambient Migration

An alternative labelled characterization of weak reduction barbed congruence is presented in [12] by Merro and Zappa Nardelli. However, the bisimulation that they propose is not defined in the standard way. They indeed note that in MAs the ability of a (restricted) ambient to migrate is unobservable, therefore in order to take this phenomenon into account they propose a modification of the usual definition of bisimulation. On the contrary, Rathke and Sobociński use instead in [11] the ordinary bisimilarity for characterizing the strong reduction barbed congruence. However, they are forced to add a set of what they call Honda-Tokoro rules, in order to account for the same phenomenon about ambient migrations. We remark that in our proposal we are never able to observe migrations of private ambients, thanks to the use of semi-saturations: this is shown by the following example for the weak semi-saturated case.

Example 2. Let us consider the example below, originally proposed in [12], which illustrates two weak reduction barbed congruent processes

$$P = (\nu n)n[in\ k.\mathbf{0}] \qquad \text{and} \qquad Q = \mathbf{0}$$

The two processes P and Q are distinguished by the standard weak equivalence over our LTS $\mathcal{S}_{\mathcal{I}}$, since P can interact with a context $-|k[R]$, while $\mathbf{0}$ cannot. The weak barbed semi-saturated bisimulation instead does not observe the migration of the private ambient n. The transition $P \xrightarrow{-|k[R]}_{\mathcal{S}_{\mathcal{I}}} (\nu n)k[n[0]|R]$ is indeed matched by $0|k[R] \rightsquigarrow^* 0|k[R]$. Moreover, since $(\nu n)k[n[0]|R]$ and $0|k[R]$ are weak barbed semi-saturated equivalent, also P and Q are so.

7 Conclusions and Future Work

The main issues of this paper have been the introduction of barbed bisimilarities in reactive systems, and their exploitation for recasting the semantics of MAs.

In particular, we proposed the novel notions of barbed and weak barbed saturated bisimilarity over reactive systems, showing that they can be efficiently characterized through the IPO-transition systems by employing the semi-saturated game. We applied the framework to MAs, proving that it can capture the strong and the weak reduction barbed congruence for MAs, proposed by Rathke and Sobociński [11], and by Merro and Zappa Nardelli [12], respectively.

We thus obtained a labelled characterization for these barbed congruences, exploiting the ITS previously proposed in [10]. However, as it is typical of the LTSs distilled by IPOs, its presentation is far from standard. Therefore, we proposed an alternative, yet equivalent presentation for our ITS using SOS rules, which simplifies our proofs and furthermore allows for an easier comparison between our proposal and that one presented by Rathke and Sobociński in [11].

We consider such a presentation a first step towards solving the problem of synthesizing a set of SOS rules for describing the ITS distilled from a reactive system. Indeed, we are quite pleased by the parametric structure of the rules that we obtained, and we hope that we will be able to lift it to the instance of RPO adopted in graph rewriting, namely, borrowed contexts [13].

Finally, as discussed in Section 6, we recall that an alternative, labelled characterization of the strong reduction barbed congruence is presented in [11]. Rathke and Sobociński use there the standard bisimilarity to capture the congruence, but they are forced to add a set of Honda-Tokoro rules to deal with the unobservability of ambient migrations. Our solution instead accounts for this phenomenon by the use of the barbed semi-saturated bisimulation. It is true however that the proposal in [11] does not need any additional observation, while in our approach the choice of the right notion of barb is left to the ingenuity of the researcher. As a future work we would like to extend our solution by considering an automatically derived notion of barb, in the style of [23] and [24].

References

1. Leifer, J., Milner, R.: Deriving bisimulation congruences for reactive systems. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 243–258. Springer, Heidelberg (2000)
2. Bonchi, F., König, B., Montanari, U.: Saturated semantics for reactive systems. In: Logic in Computer Science, pp. 69–80. IEEE Computer Society, Los Alamitos (2006)

3. Bonchi, F.: Abstract Semantics by Observable Contexts. PhD thesis, Department of Informatics, University of Pisa (2008)
4. Milner, R.: Bigraphs for petri nets. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets. LNCS, vol. 3098, pp. 686–701. Springer, Heidelberg (2004)
5. Sassone, V., Sobociński, P.: A congruence for Petri nets. In: Petri Nets and Graph Transformation. ENTCS, vol. 127, pp. 107–120. Elsevier, Amsterdam (2005)
6. Milner, R.: Communication and Concurrency. Prentice-Hall, Englewood Cliffs (1989)
7. Milner, R.: Communicating and Mobile Systems: the π-Calculus. Cambridge University Press, Cambridge (1999)
8. Milner, R., Sangiorgi, D.: Barbed bisimulation. In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 685–695. Springer, Heidelberg (1992)
9. Cardelli, L., Gordon, A.: Mobile ambients. TCS 240(1), 177–213 (2000)
10. Bonchi, F., Gadducci, F., Monreale, G.V.: Labelled transitions for mobile ambients (as synthesized via a graphical encoding). In: Expressiveness in Concurrency. ENTCS. Elsevier, Amsterdam (forthcoming, 2008)
11. Rathke, J., Sobociński, P.: Deriving structural labelled transitions for mobile ambients. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 462–476. Springer, Heidelberg (2008)
12. Merro, M., Zappa Nardelli, F.: Behavioral theory for mobile ambients. Journal of the ACM 52(6), 961–1023 (2005)
13. Ehrig, H., König, B.: Deriving bisimulation congruences in the DPO approach to graph rewriting with borrowed contexts. Mathematical Structures in Computer Science 16(6), 1133–1163 (2006)
14. Sassone, V., Sobociński, P.: Reactive systems over cospans. In: Logic in Computer Science, pp. 311–320. IEEE Computer Society, Los Alamitos (2005)
15. Plotkin, G.D.: A structural approach to operational semantics. Journal of Logic and Algebraic Programming 60-61, 17–139 (2004)
16. Milner, R.: Pure bigraphs: Structure and dynamics. Information and Computation 204(1), 60–122 (2006)
17. Bonchi, F., Gadducci, F., König, B.: Process bisimulation *via* a graphical encoding. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 168–183. Springer, Heidelberg (2006)
18. Milner, R.: Local bigraphs and confluence: Two conjectures. In: Expressiveness in Concurrency. ENTCS, vol. 175, pp. 65–73. Elsevier, Amsterdam (2007)
19. Di Gianantonio, P., Honsel, F., Lenisa, M.: RPO, second-order contexts, and λ-calculus. In: Amadio, R. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 334–349. Springer, Heidelberg (2008)
20. Jensen, O., Milner, R.: Bigraphs and transitions. In: Principles of Programming Languages, pp. 38–49. ACM Press, New York (2003)
21. Grohmann, D., Miculan, M.: Reactive systems over directed bigraphs. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 380–394. Springer, Heidelberg (2007)
22. Fournet, C., Gonthier, G.: A hierarchy of equivalences for asynchronous calculi. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 844–855. Springer, Heidelberg (1998)
23. Honda, K., Yoshida, N.: On reduction-based process semantics. TCS 151(2), 437–486 (1995)
24. Rathke, J., Sassone, V., Sobocinski, P.: Semantic barbs and biorthogonality. In: Seidl, H. (ed.) FOSSACS 2007. LNCS, vol. 4423, pp. 302–316. Springer, Heidelberg (2007)

On the Foundations of Quantitative Information Flow

Geoffrey Smith

School of Computing and Information Sciences,
Florida International University, Miami, FL 33199, USA
smithg@cis.fiu.edu

Abstract. There is growing interest in quantitative theories of information flow in a variety of contexts, such as secure information flow, anonymity protocols, and side-channel analysis. Such theories offer an attractive way to relax the standard noninterference properties, letting us tolerate "small" leaks that are necessary in practice. The emerging consensus is that quantitative information flow should be founded on the concepts of *Shannon entropy* and *mutual information*. But a useful theory of quantitative information flow must provide appropriate security guarantees: if the theory says that an attack leaks x bits of secret information, then x should be useful in calculating bounds on the resulting threat. In this paper, we focus on the threat that an attack will allow the secret to be guessed correctly in one try. With respect to this threat model, we argue that the consensus definitions actually fail to give good security guarantees—the problem is that a random variable can have arbitrarily large Shannon entropy even if it is highly vulnerable to being guessed. We then explore an alternative foundation based on a concept of *vulnerability* (closely related to *Bayes risk*) and which measures uncertainty using Rényi's *min-entropy*, rather than Shannon entropy.

1 Introduction

Protecting the *confidentiality* of sensitive information is one of the most fundamental security issues:

– In *secure information flow analysis* [1] the question is whether a program could leak information about its *high* (i.e. secret) inputs into its *low* (i.e. public) outputs.
– In *anonymity protocols* [2] the question is whether network traffic could reveal information to an eavesdropper about *who* is communicating.
– In *side-channel analysis* [3] the question is whether the running time or power consumption of cryptographic operations could reveal information about the secret keys.

A classic approach is to try to enforce *noninterference*, which says that low outputs are independent of high inputs; this implies that an adversary can deduce nothing about the high inputs from the low outputs.

Unfortunately, achieving noninterference is often not possible, because sometimes we *want* or *need* to reveal information that depends on the high inputs. In an election protocol, for example, the individual votes should be secret, but of course we want to reveal the tally of votes publicly. And in a password checker, we need to reject an incorrect password, but this reveals information about what the secret password is *not*. A variety of approaches for dealing with these sorts of deliberate violations of noninterference are currently being explored; see Sabelfeld and Sands [4] for a survey.

One promising approach to relaxing noninterference is to develop a *quantitative* theory of information flow that lets us talk about "how much" information is being leaked, and perhaps allowing us to tolerate "small" leaks. Such a quantitative theory has long been recognized as an important generalization of noninterference (see for example Denning [5, Chapter 5] and Gray [6]) and there has been much recent work in this area, including the works of Clark, Hunt, Malacaria, and Chen [7,8,9,10,11] Clarkson, Myers, and Schneider [12], Köpf and Basin [3], Chatzikokolakis, Palamidessi, and Panangaden [2,13], Lowe [14], and Di Pierro, Hankin, and Wiklicky [15].

In this paper, we consider the foundations of quantitative information flow. The basic scenario that we imagine is a program (or protocol) that receives some high input $\mathtt{H}$ and produces some low output $\mathtt{L}$. An adversary $\mathcal{A}$, seeing $\mathtt{L}$, might be able to deduce something about $\mathtt{H}$. We would like to *quantify* the amount of information in $\mathtt{H}$ ($\mathcal{A}$'s initial uncertainty), the amount of information leaked to $\mathtt{L}$, and the amount of unleaked information about $\mathtt{H}$ ($\mathcal{A}$'s remaining uncertainty). These quantities intuitively ought to satisfy the following slogan:

"initial uncertainty = information leaked + remaining uncertainty".

Of course, for a definition of quantitative information flow to be *useful*, it must be possible to show that the numbers produced by the theory are *meaningful* with respect to security—if the theory says that an attack leaks x bits of secret information, then x should be useful in calculating bounds on the resulting threat. A natural first step along these lines is to show that a leakage of 0 corresponds to the case of noninterference. But this is just a first step—we also must be able to show that differences among nonzero leakage values have significance in terms of security. This is the key issue which we explore in this paper.

We begin in Section 2 by establishing our conceptual framework—we consider a deterministic or probabilistic program c that receives a high input $\mathtt{H}$, assumed to satisfy a publicly-known *a priori* distribution, and produces a low output $\mathtt{L}$.

In Section 3, we review definitions of quantitative information flow found in the literature; we note an emerging consensus toward a set of definitions based on information-theoretic measures such as Shannon entropy and mutual information.

Section 4 then explores the consensus definitions with respect to the security guarantees that they support. After reviewing a number of security guarantees that have appeared in the literature, we focus our attention on one specific threat model: the probability that adversary $\mathcal{A}$ can guess the value of $\mathtt{H}$ correctly in one try. With respect to this threat model, we argue that the consensus definitions actually do a poor job of measuring the threat; briefly, the problem is that the Shannon entropy $H(\mathtt{X})$ of a random variable $\mathtt{X}$ can be arbitrarily high, even if the value of $\mathtt{X}$ is highly vulnerable to being guessed.

Because of these limitations of the consensus definitions, in Section 5 we propose an alternative foundation for quantitative information flow. Our definitions are based directly on a concept of *vulnerability*, which is closely related to *Bayes risk*. The vulnerability $V(\mathtt{X})$ is simply the maximum of the probabilities of the values of $\mathtt{X}$; it is the worst-case probability that an adversary could correctly guess the value of $\mathtt{X}$ in one try.

Using vulnerability, we propose to use *min-entropy*, defined by $H_\infty(\mathtt{X}) = -\log V(\mathtt{X})$, as a better measure for quantitative information flow. (Min-entropy is an instance of *Rényi entropy* [16].)

It should be acknowledged that our observations about the limitations of Shannon entropy as a measure of uncertainty are not entirely new, having appeared in recent research on anonymity protocols, notably in papers by Tóth, Hornák, and Vajda [17] and by Shmatikov and Wang [18]; they also propose the use of min-entropy as an alternative. But these papers do not address *information flow* generally, considering only the narrower question of how to quantify an adversary's uncertainty about who is communicating in a mix network. In the literature on quantitative information flow, we believe that the ideas we propose have not previously appeared—see for instance the recent high-profile papers of Malacaria [10] and Köpf and Basin [3].

In Section 5, we also develop techniques (based on Bayesian inference) for calculating conditional vulnerability and conditional min-entropy, for both deterministic and probabilistic programs. And we illustrate the reasonableness of our definitions by considering a number of examples.

Finally, Section 6 discusses some future directions and concludes.

A preliminary discussion of the ideas in this paper—restricted to deterministic programs and not using min-entropy—was included as a part of an invited talk and subsequent invited tutorial paper at the *TGC 2007 Workshop on the Interplay of Programming Languages and Cryptography* [19].

2 Our Conceptual Framework

The framework that we consider in this paper aims for simplicity and clarity, rather than full generality. We therefore restrict our attention to total programs c with just one input $\mathtt{H}$, which is high, and one output $\mathtt{L}$, which is low. (Thus we do not consider the case of programs that receive both high and low inputs, or programs that might not terminate.) Our goal is to quantify how much, if any, information in $\mathtt{H}$ is leaked by program c to $\mathtt{L}$. More precisely, our question is how much information about $\mathtt{H}$ can be deduced by an *adversary* $\mathcal{A}$ who sees the output $\mathtt{L}$.

We further assume that there is an *a priori*, *publicly-known* probability distribution on $\mathtt{H}$. We therefore assume that $\mathtt{H}$ is a random variable with a finite space of possible values $\mathcal{H}$. We denote the *a priori* probability that $\mathtt{H}$ has value h by $P[\mathtt{H} = h]$, and we assume that each element h of $\mathcal{H}$ has nonzero probability. Similarly, we assume that $\mathtt{L}$ is a random variable with a finite space of possible values $\mathcal{L}$, and with probabilities $P[\mathtt{L} = \ell]$. We assume that each output $\ell \in \mathcal{L}$ is possible, in that it can be produced by some input $h \in \mathcal{H}$.

In general, program c could be *deterministic* or it could be *probabilistic*. We consider these two cases separately.

2.1 Deterministic Programs

If c is a deterministic program, then the output $\mathtt{L}$ is a *function* of the input $\mathtt{H}$; that is, c determines a function $f : \mathcal{H} \rightarrow \mathcal{L}$ such that $\mathtt{L} = f(\mathtt{H})$. Now, following Köpf and Basin [3], we note that f induces an *equivalence relation* $\sim$ on $\mathcal{H}$:

$$h \sim h' \text{ iff } f(h) = f(h').$$

(In set theory, $\sim$ is called the *kernel* of f.) Hence program c *partitions* $\mathcal{H}$ into the equivalence classes of $\sim$. We let $\mathcal{H}_\ell$ denote the equivalence class $f^{-1}(\ell)$:

$$\mathcal{H}_\ell = \{h \in \mathcal{H} \mid f(h) = \ell\}.$$

Notice that there are $|\mathcal{L}|$ distinct equivalence classes, since we are assuming that each output $\ell \in \mathcal{L}$ is possible. The importance of these equivalence classes is that they bound the knowledge of the adversary $\mathcal{A}$: if $\mathcal{A}$ sees that $\mathtt{L}$ has value ℓ, then $\mathcal{A}$ knows only that the value of $\mathtt{H}$ belongs to the equivalence class $\mathcal{H}_\ell$. How much does this equivalence class tell $\mathcal{A}$ about $\mathtt{H}$?

In one extreme, the function determined by c is a *constant* function, with just one possible value ℓ. In this case $\sim$ has just one equivalence class $\mathcal{H}_\ell$, which is equal to $\mathcal{H}$. Hence there is no leakage of $\mathtt{H}$ and *noninterference* holds.

In the other extreme, the function determined by c is *one-to-one*. Here the equivalence classes of $\sim$ are all singletons, and we have *total leakage* of $\mathtt{H}$. Note however that, given ℓ, adversary $\mathcal{A}$ might not be able to *compute* the equivalence class $\mathcal{H}_\ell$ efficiently; thus we are adopting a worst-case, *information theoretic* viewpoint, rather than a *computational* one.

As an intermediate example, suppose that $\mathtt{H}$ is a 32-bit unsigned integer, with range $0 \leq \mathtt{H} < 2^{32}$. The program

```
L := H & 037
```

copies the last 5 bits of $\mathtt{H}$ into $\mathtt{L}$. (Here `037` is an octal constant, and `&` denotes bitwise "and".) In this case, $\sim$ has $2^5 = 32$ equivalence classes, each of which contains 2^{27} elements. Intuitively, c leaks 5 bits (out of 32) of $\mathtt{H}$.

2.2 Probabilistic Programs

More generally, the program c might be probabilistic. In this case, each value of $\mathtt{H}$ could lead to more than one value of $\mathtt{L}$, which means that c may not give a partition of $\mathcal{H}$.

Following [2], we can model a probabilistic program c using a matrix whose rows are indexed by $\mathcal{H}$ and whose columns are indexed by $\mathcal{L}$, where the (h, ℓ) entry specifies the conditional probability $P[\mathtt{L} = \ell | \mathtt{H} = h]$. Notice that each row of this matrix must sum to 1. Also notice that in the special case where c is deterministic, each row will have one entry equal to 1 and all others equal to 0.

3 Existing Definitions of Quantitative Information Flow

Given (deterministic or probabilistic) program c, which may leak information from $\mathtt{H}$ to $\mathtt{L}$, we want to *define* how much information c leaks. In the literature, such definitions are usually based on information-theoretic measures, such as Shannon entropy [20,21,22].

First we briefly review some of these measures. Let $\mathtt{X}$ be a random variable whose set of possible values is $\mathcal{X}$. The *Shannon entropy* $H(\mathtt{X})$ is defined by

$$H(\mathtt{X}) = \sum_{x \in \mathcal{X}} P[\mathtt{X} = x] \log \frac{1}{P[\mathtt{X} = x]}.$$

(Throughout we assume that log denotes logarithm with base 2.) The Shannon entropy can be viewed intuitively as the "uncertainty about $\mathtt{X}$"; more precisely it can be understood as the expected number of bits required to transmit $\mathtt{X}$ optimally.

Given two (jointly distributed) random variables $\mathtt{X}$ and $\mathtt{Y}$, the *conditional entropy* $H(\mathtt{X}|\mathtt{Y})$, intuitively the "uncertainty about $\mathtt{X}$ given $\mathtt{Y}$", is

$$H(\mathtt{X}|\mathtt{Y}) = \sum_{y \in \mathcal{Y}} P[\mathtt{Y} = y] H(\mathtt{X}|\mathtt{Y} = y)$$

where

$$H(\mathtt{X}|\mathtt{Y} = y) = \sum_{x \in \mathcal{X}} P[\mathtt{X} = x|\mathtt{Y} = y] \log \frac{1}{P[\mathtt{X} = x|\mathtt{Y} = y]}.$$

Note that if $\mathtt{X}$ is determined by $\mathtt{Y}$, then $H(\mathtt{X}|\mathtt{Y}) = 0$.

The *mutual information* $I(\mathtt{X};\mathtt{Y})$, intuitively the "amount of information shared between $\mathtt{X}$ and $\mathtt{Y}$", is

$$I(\mathtt{X};\mathtt{Y}) = H(\mathtt{X}) - H(\mathtt{X}|\mathtt{Y}).$$

Mutual information turns out to be symmetric: $I(\mathtt{X};\mathtt{Y}) = I(\mathtt{Y};\mathtt{X})$.

The *guessing entropy* $G(\mathtt{X})$ is the expected number of guesses required to guess $\mathtt{X}$ optimally; of course the optimal strategy is to guess the values of $\mathtt{X}$ in nonincreasing order of probability. If we assume that $\mathtt{X}$'s probabilities are arranged in nonincreasing order $p_1 \geq p_2 \geq \ldots \geq p_n$, then we have

$$G(\mathtt{X}) = \sum_{i=1}^{n} i p_i.$$

Now we consider how these entropy concepts can be used to quantify information leakage. Recalling the slogan

initial uncertainty = information leaked + remaining uncertainty

we have three quantities to define. For the initial uncertainty about $\mathtt{H}$, the entropy $H(\mathtt{H})$ seems appropriate. For the remaining uncertainty about $\mathtt{H}$, the conditional entropy $H(\mathtt{H}|\mathtt{L})$ seems appropriate. Finally, for the information leaked to $\mathtt{L}$, the entropy $H(\mathtt{L})$ might appear appropriate as well, but this cannot be correct in the case where c is probabilistic. For in that case, $\mathtt{L}$ might get positive entropy simply from the probabilistic nature of c, even though there is no leakage from $\mathtt{H}$. So we need something different.

Rearranging the slogan above, we get

information leaked = initial uncertainty $-$ remaining uncertainty.

This suggests that the information leaked to $\mathtt{L}$ should be $H(\mathtt{H}) - H(\mathtt{H}|\mathtt{L})$, which is just the mutual information $I(\mathtt{H};\mathtt{L})$.

If, however, we restrict to the case where c is deterministic, then we know that $\mathtt{L}$ is determined by $\mathtt{H}$. In that case we have $H(\mathtt{L}|\mathtt{H}) = 0$ which implies that

$$I(\mathtt{H};\mathtt{L}) = I(\mathtt{L};\mathtt{H}) = H(\mathtt{L}) - H(\mathtt{L}|\mathtt{H}) = H(\mathtt{L}).$$

So, in the case of deterministic programs, the mutual information $I(\mathtt{H};\mathtt{L})$ can be simplified to the entropy $H(\mathtt{L})$.

We can apply these definitions to some example programs. If we assume that H is a uniformly-distributed 32-bit integer, with range $0 \leq \mathtt{H} < 2^{32}$, then we get the following results:

Program	$H(\mathtt{H})$	$I(\mathtt{H};\mathtt{L})$	$H(\mathtt{H}\|\mathtt{L})$
`L := 0`	32	0	32
`L := H`	32	32	0
`L := H & 037`	32	5	27

Turning to the research literature, the definitions we have described:

- initial uncertainty = $H(\mathtt{H})$
- information leaked = $I(\mathtt{H};\mathtt{L})$
- remaining uncertainty = $H(\mathtt{H}|\mathtt{L})$

seem to be the emerging consensus. Clarke, Hunt, and Malacaria [7,8,9,10] use these definitions, although they also address the more general case where the program c receives both high and low input. Köpf and Basin [3] use these definitions in their study of side-channel attacks, but they consider only the deterministic case. (They also consider guessing entropy and marginal guesswork in addition to Shannon entropy.) Chatzikokolakis, Palamidessi, and Panangaden [2] also use these definitions in their study of anonymity protocols. However, they are especially interested in situations where it is unreasonable to assume an *a priori* distribution on H; this leads them to emphasize the *channel capacity*, which is the *maximum* value of $I(\mathtt{H};\mathtt{L})$ over all distributions on H. Finally, the framework of Clarkson, Myers, and Schneider [12] is a significant extension of what we have described here, because they consider the case when the adversary $\mathcal{A}$ has (possibly mistaken) *beliefs* about the probability distribution on H. But in the special case when $\mathcal{A}$'s beliefs match the *a priori* distribution, and when the expected flow over all experiments is considered (see Section 4.4 of their paper), then their approach reduces to using the above definitions.

4 Security Guarantees with the Consensus Definitions

Given the consensus definitions of quantitative information flow described in Section 3, we now turn our attention to the question of what security guarantees these definitions support.

A first result along these lines is proved in [8]; they show, for deterministic programs, that $H(\mathtt{L})$ (the "information leaked") is 0 iff c satisfies noninterference. This is good, of course, but it is only a sanity check—it establishes that the zero/nonzero distinction is meaningful, but not that different nonzero values are meaningful.

Really the key question with respect to security is whether the value of $H(\mathtt{H}|\mathtt{L})$ (the "remaining uncertainty") accurately reflects the threat to H.

One bound that seems promising in justifying the significance of $H(\mathtt{H}|\mathtt{L})$ is given by Clark, Hunt, and Malacaria [7] based on work by Massey [23]. It states that the guessing entropy $G(\mathtt{H}|\mathtt{L})$, which is the expected number of guesses required to guess H given L, satisfies

$$G(\mathtt{H}|\mathtt{L}) \geq 2^{H(\mathtt{H}|\mathtt{L}) - 2} + 1 \tag{1}$$

provided that $H(\mathtt{H}|\mathtt{L}) \geq 2$. For example, consider the program discussed above,

```
L := H & 037
```

where H is uniformly distributed with range $0 \le \mathtt{H} < 2^{32}$. Here we have $H(\mathtt{H}|\mathtt{L}) = 27$, since each equivalence class contains 2^{27} elements, uniformly distributed. So by (1) we have

$$G(\mathtt{H}|\mathtt{L}) \ge 2^{25} + 1$$

which is quite an accurate bound, since the actual expected number of guesses is

$$\frac{2^{27}+1}{2}.$$

But note however that when we assess the threat to H, the adversary's *expected* number of guesses is probably not the key concern. The problem is that even if the expected number of guesses is huge, the adversary might nonetheless have a significant probability of guessing the value of H in just one try.

A result that addresses exactly this question is the classic *Fano inequality*, which gives lower bounds, in terms of $H(\mathtt{H}|\mathtt{L})$, on the probability that adversary $\mathcal{A}$ will *fail* to guess the value of H correctly in one try, given the value of L. Let P_e denote this probability. The Fano inequality is

$$P_e \ge \frac{H(\mathtt{H}|\mathtt{L}) - 1}{\log(|\mathcal{H}| - 1)}. \tag{2}$$

Unfortunately this bound is extremely weak in many cases. For example, on the program

```
L := H & 037
```

the Fano inequality gives

$$P_e \ge \frac{27-1}{\log(2^{32}-1)} \approx 0.8125$$

But this wildly understates the probability of error, since here the adversary has no knowledge of 27 of the bits of H, which implies that

$$P_e \ge \frac{2^{27}-1}{2^{27}} \approx 0.9999999925$$

One might wonder whether the Fano inequality could be strengthened, but (as we will illustrate below) this is not in general possible.

Fundamentally, the problem is that $H(\mathtt{H}|\mathtt{L})$ is of little value in characterizing the threat that the adversary, given L, could guess H. We demonstrate this claim through two key examples. Assume that H is a uniformly distributed $8k$-bit integer with range $0 \le \mathtt{H} < 2^{8k}$, where $k \ge 2$. Hence $H(\mathtt{H}) = 8k$.

The first example is the program

```
if H mod 8 = 0 then
  L := H
else
  L := 1
```
(3)

Since this program is deterministic, its information leakage is just $H(\mathtt{L})$. Notice that the `else` branch is taken on 7/8 of the values of H, namely those whose last 3 bits are not all 0. Hence

$$P[\mathtt{L} = 1] = \frac{7}{8}$$

and

$$P[\mathtt{L} = 8n] = 2^{-8k}$$

for each n with $0 \leq n < 2^{8k-3}$. Hence we have

$$H(\mathtt{L}) = \frac{7}{8}\log\frac{8}{7} + 2^{8k-3}2^{-8k}\log 2^{8k} \approx k + 0.169$$

This implies that

$$H(\mathtt{H}|\mathtt{L}) \approx 7k - 0.169$$

suggesting that about $7/8$ of the information in H remains unleaked. But, since the `then` branch is taken $1/8$ of the time, the adversary can guess the value of H at least $1/8$ of the time! (We remark that this example shows that the Fano inequality cannot in general be strengthened significantly—here the Fano inequality says that the probability of error is at least

$$\frac{7k - 1.169}{\log(2^{8k} - 1)}$$

which is close to $7/8$ for large k.)

The second example (using the same assumptions about H) is the program

$$\mathtt{L\ :=\ H\ \&\ 0}^{7k-1}\mathtt{1}^{k+1} \qquad (4)$$

where $\mathtt{0}^{7k-1}\mathtt{1}^{k+1}$ is a binary constant; this program copies the last $k+1$ bits of H into L. Hence we have

$$H(\mathtt{L}) = k + 1$$

and

$$H(\mathtt{H}|\mathtt{L}) = 7k - 1.$$

Here notice that, given L, the adversary's probability of guessing H is just $1/2^{7k-1}$.

The key point to emphasize here is that, under the consensus definitions, program (4) is actually *worse* than program (3), even though program (3) leaves H highly vulnerable to being guessed, while program (4) does not. The conclusion is that, with respect to this threat model, the consensus definitions do a poor job of measuring the threat: $H(\mathtt{H}|\mathtt{L})$ does not support good security guarantees about the probability that H could be guessed.

5 An Alternative Foundation: Vulnerability and Min-entropy

The limitations of the consensus definitions noted in Section 4 lead us now to explore alternative definitions of quantitative information flow, with the goal of finding a measure supporting better security guarantees with respect to the probability that the adversary could guess H in one try.

Rather than inventing a new measure and then trying to prove that it implies good security guarantees, why not define a measure of remaining uncertainty directly in terms of the desired security guarantees? To this end, we propose the concept of *vulnerability*:

Definition 1. *Given a random variable* $\mathtt{X}$ *with space of possible values* $\mathcal{X}$*, the* vulnerability of $\mathtt{X}$*, denoted* $V(\mathtt{X})$*, is given by*

$$V(\mathtt{X}) = \max_{x \in \mathcal{X}} P[\mathtt{X} = x].$$

The vulnerability $V(\mathtt{X})$ is thus the worst-case probability that an adversary $\mathcal{A}$ could guess the value of $\mathtt{X}$ correctly in one try. It is clearly a rather crude measure, because it depends only on the *maximum* probability in the distribution of $\mathtt{X}$, focusing on the single probability that brings the greatest risk. Limiting to a single guess might seem unreasonable, of course. But notice that with m guesses the adversary can succeed with probability at most $mV(\mathtt{X})$. This implies (very roughly speaking) that if the vulnerability with m guesses is "significant", where m is a "practical" number of tries, then $V(\mathtt{H})$ must itself be "significant".

Vulnerability is a probability, so its value is always between 0 and 1. But to quantify information flow, we would like to measure information in *bits*. We can convert to an entropy measure by mapping $V(\mathtt{X})$ to

$$\log \frac{1}{V(\mathtt{X})}.$$

This, it turns out, gives a measure known as *min-entropy*:

Definition 2. *The* min-entropy of $\mathtt{X}$*, denoted* $H_\infty(\mathtt{X})$*, is given by*

$$H_\infty(\mathtt{X}) = \log \frac{1}{V(\mathtt{X})}.$$

As far as we know, min-entropy has not previously been used in quantitative information flow. But, as noted in Section 1, it has been used to measure the anonymity provided by mix networks [17,18]. Also, Cachin [24] discusses its relevance in cryptographic guessing attacks.

Min-entropy is the instance of *Rényi entropy* [16]

$$H_\alpha(\mathtt{X}) = \frac{1}{1-\alpha} \log \left(\sum_{x \in \mathcal{X}} P[\mathtt{X} = x]^\alpha \right)$$

obtained when $\alpha = \infty$. Notice that if $\mathtt{X}$ is uniformly distributed among n values, then $V(\mathtt{X}) = 1/n$ and $H_\infty(\mathtt{X}) = \log n$. Hence Shannon entropy $H(\mathtt{X})$ and min-entropy $H_\infty(\mathtt{X})$ coincide on uniform distributions. But, in general, Shannon entropy can be arbitrarily greater than min-entropy, since $H(\mathtt{X})$ can be arbitrarily high even if $\mathtt{X}$ has a value with a probability close to 1.

We propose to use $H_\infty(\mathtt{H})$ as our measure of initial uncertainty. To measure the remaining uncertainty, we first consider *conditional vulnerability*, which gives the expected probability of guessing $\mathtt{X}$ in one try, given $\mathtt{Y}$:

Definition 3. *Given (jointly distributed) random variables* $\mathtt{X}$ *and* $\mathtt{Y}$*, the* conditional vulnerability $V(\mathtt{X}|\mathtt{Y})$ *is*

$$V(\mathtt{X}|\mathtt{Y}) = \sum_{y \in \mathcal{Y}} P[\mathtt{Y} = y] V(\mathtt{X}|\mathtt{Y} = y)$$

where

$$V(\mathtt{X}|\mathtt{Y}=y) = \max_{x\in\mathcal{X}} P[\mathtt{X}=x|\mathtt{Y}=y].$$

We now show that $V(\mathtt{H}|\mathtt{L})$ is easy to calculate for probabilistic programs c, given the *a priori* distribution on $\mathtt{H}$ and the matrix of conditional probabilities $P[\mathtt{L}=\ell|\mathtt{H}=h]$. In fact, $V(\mathtt{H}|\mathtt{L})$ is simply the complement of the *Bayes risk* P_e.

First we note that by Bayes' theorem we have

$$P[\mathtt{H}=h|\mathtt{L}=\ell]P[\mathtt{L}=\ell] = P[\mathtt{L}=\ell|\mathtt{H}=h]P[\mathtt{H}=h].$$

Now we have

$$\begin{aligned} V(\mathtt{H}|\mathtt{L}) &= \sum_{\ell\in\mathcal{L}} P[\mathtt{L}=\ell]V(\mathtt{H}|\mathtt{L}=\ell) \\ &= \sum_{\ell\in\mathcal{L}} P[\mathtt{L}=\ell]\max_{h\in\mathcal{H}} P[\mathtt{H}=h|\mathtt{L}=\ell] \\ &= \sum_{\ell\in\mathcal{L}} \max_{h\in\mathcal{H}} P[\mathtt{H}=h|\mathtt{L}=\ell]P[\mathtt{L}=\ell] \\ &= \sum_{\ell\in\mathcal{L}} \max_{h\in\mathcal{H}} P[\mathtt{L}=\ell|\mathtt{H}=h]P[\mathtt{H}=h]. \end{aligned}$$

It should be noted that [13] proposes Bayes risk as a measure of protection in anonymity protocols, and also includes essentially the same calculation as above.

We next observe that the calculation of $V(\mathtt{H}|\mathtt{L})$ becomes simpler in the special case where program c is deterministic. For in that case $\mathcal{H}$ is partitioned into $|\mathcal{L}|$ equivalence classes $\mathcal{H}_\ell$, where

$$\mathcal{H}_\ell = \{h\in\mathcal{H} \mid P[\mathtt{L}=\ell|\mathtt{H}=h]=1\}.$$

Hence we have

$$\begin{aligned} V(\mathtt{H}|\mathtt{L}) &= \sum_{\ell\in\mathcal{L}} \max_{h\in\mathcal{H}} P[\mathtt{L}=\ell|\mathtt{H}=h]P[\mathtt{H}=h] \\ &= \sum_{\ell\in\mathcal{L}} \max_{h\in\mathcal{H}_\ell} P[\mathtt{H}=h]. \end{aligned}$$

Finally, we note that in the special case where c is deterministic and $\mathtt{H}$ is uniformly distributed, the conditional vulnerability becomes very simple indeed:

$$\begin{aligned} V(\mathtt{H}|\mathtt{L}) &= \sum_{\ell\in\mathcal{L}} \max_{h\in\mathcal{H}_\ell} P[\mathtt{H}=h] \\ &= \sum_{\ell\in\mathcal{L}} (1/|\mathcal{H}|) \\ &= |\mathcal{L}|/|\mathcal{H}| \end{aligned}$$

Thus in this case all that matters is the *number* of equivalence classes. (We remark that Lowe [14] focuses on a quantity analogous to $|\mathcal{L}|$ in quantifying information flow in a process calculus, even though his approach is not probabilistic.)

We now define $H_\infty(\mathtt{H}|\mathtt{L})$, which will be our measure of remaining uncertainty:

Definition 4. *The* conditional min-entropy $H_\infty(\mathtt{X}|\mathtt{Y})$ *is*

$$H_\infty(\mathtt{X}|\mathtt{Y}) = \log \frac{1}{V(\mathtt{X}|\mathtt{Y})}.$$

Note that this definition of conditional min-entropy is *not* the same as the one given by Cachin [24, p. 16], but it is equivalent to the one proposed by Dodis et al. [25].

We now propose the following definitions:

– initial uncertainty = $H_\infty(\mathtt{H})$
– remaining uncertainty = $H_\infty(\mathtt{H}|\mathtt{L})$
– information leaked = $H_\infty(\mathtt{H}) - H_\infty(\mathtt{H}|\mathtt{L})$

Note that our measure of remaining uncertainty, $H_\infty(\mathtt{H}|\mathtt{L})$, gives an immediate security guarantee:

$$V(\mathtt{H}|\mathtt{L}) = 2^{-H_\infty(\mathtt{H}|\mathtt{L})}.$$

Thus the expected probability that the adversary could guess H given L decreases exponentially with $H_\infty(\mathtt{H}|\mathtt{L})$.

Also note that calculating the information leakage is easy in the case where c is deterministic and H is uniformly distributed:

Theorem 1. *If c is deterministic and* H *is uniformly distributed, then the information leaked is* $\log|\mathcal{L}|$.

Proof. Here we have $V(\mathtt{H}) = 1/|\mathcal{H}|$ and $V(\mathtt{H}|\mathtt{L}) = |\mathcal{L}|/|\mathcal{H}|$, so

$$H_\infty(\mathtt{H}) - H_\infty(\mathtt{H}|\mathtt{L}) = \log|\mathcal{H}| - \log(|\mathcal{H}|/|\mathcal{L}|) = \log|\mathcal{L}|. \qquad \square$$

Let us now revisit example programs (4) and (3) from Section 4 using our new definitions. Because these programs are deterministic and H is uniformly distributed, we only need to focus on $|\mathcal{H}|$ and $|\mathcal{L}|$. Note that $|\mathcal{H}| = 2^{8k}$, so the initial uncertainty $H_\infty(\mathtt{H})$ is $8k$, as before.

On program (4), we get the same values as before. We have $|\mathcal{L}| = 2^{k+1}$, which implies that the information leaked is $k+1$ and the remaining uncertainty is $7k-1$.

But on program (3), we have $|\mathcal{L}| = 2^{8k-3}+1$, which implies that the information leaked is about $8k-3$ and the remaining uncertainty is about 3. Thus our new measures hugely increase the leakage ascribed to this program.

It is interesting to compare program (3) with a program that always leaks all but the last 3 bits of H:

```
L := H | 07
```
(5)

(Here | denotes bitwise "or".) For this program, $|\mathcal{L}| = 2^{8k-3}$, so it is ascribed almost exactly the same leakage as program (3). Notice that while both of these programs make H highly vulnerable, the threats are different: with program (3), the adversary $\mathcal{A}$ learns H completely $1/8$ of the time, and learns very little $7/8$ of the time; with program (5), in contrast, $\mathcal{A}$ never learns H completely, but always learns it to within 8 possible values.

Is it reasonable to ascribe the same leakage to programs (3) and (5)? That seems hard to answer without further assumptions. For instance, if $\mathcal{A}$ is allowed *several* guesses, rather than just one, then program (5) is clearly worse. On the other hand, if a *wrong* guess would trigger an alert, then program (3) might be worse, since then $\mathcal{A}$ *knows* whether it knows H or not, and could choose to make a guess only when it knows. These examples suggest the difficulty of measuring a range of complex threat scenarios precisely using a single number; still, we feel that one-guess vulnerability is a sufficiently basic concern to serve as a generally useful foundation.

As another example, consider a password checker, which tests whether H (assumed uniformly distributed) is equal to some particular value and assigns the result to L. Since $|\mathcal{L}| = 2$, we get a leakage of 1 here.

We remark that Köpf and Basin [3] briefly consider *worst-case entropy measures* in addition to the "averaging" measures (like $G(\mathtt{H}|\mathtt{L})$) used in the rest of their paper. Specifically, they define the *minimum guessing entropy* by

$$\hat{G}(\mathtt{H}|\mathtt{L}) = \min_{\ell \in \mathcal{L}} G(\mathtt{H}|\mathtt{L} = \ell).$$

But this measure is not very useful, as shown by the password checker example—the password checker splits $\mathcal{H}$ into 2 equivalence classes, one of which is a singleton. Hence the minimum guessing entropy is 1. This measure thus judges a password checker to be as bad as a program that leaks H completely.

As a final example, consider an election system. Suppose that we have an election between candidates A and B with k voters, whose individual votes are represented by the k-bit random variable H. We (unrealistically) assume that each voter independently votes for either A or B with probability $1/2$, which implies that H is uniformly distributed over 2^k values. The election system reveals into L the tally of votes received by candidate A, which means that L ranges over $\{0, 1, 2, \ldots, k\}$. Here the initial uncertainty is k and the leakage is $\log(k+1)$. And the conditional vulnerability is

$$V(\mathtt{H}|\mathtt{L}) = \frac{k+1}{2^k}$$

some of whose values are shown in the following table:

k	1	2	3	4	5	6
$V(\mathtt{H}\vert\mathtt{L})$	1	3/4	1/2	5/16	3/16	7/64

So the adversary's ability to guess the individual votes decreases exponentially with k.

We conclude this section with a curious result, whose significance is unclear (to me, at least). In the case of a deterministic program c and uniformly-distributed H, it turns out that our new definition of information leakage exactly coincides with the classical notion of the *channel capacity* of c.

Theorem 2. *If c is deterministic and* H *is uniformly distributed, then the information leaked,* $\log |\mathcal{L}|$, *is equal to the channel capacity of c.*

Proof. In the deterministic case, the channel capacity is the maximum value of $H(\mathtt{L})$ over all distributions on H. This maximum is $\log |\mathcal{L}|$, since L has $|\mathcal{L}|$ possible values and we can put a distribution on H that makes them all equally likely. (This observation is also made in [11].) Curiously, this will typically *not* be a uniform distribution on H. $\square$

But perhaps this result is just a coincidence—it does not generalize to the case of probabilistic programs.

6 Conclusion

In this paper, we have focused on one specific, but natural, threat model: the expected probability that an adversary could guess $\mathtt{H}$ in one try, given $\mathtt{L}$. We have argued that the consensus definitions of quantitative information flow do poorly with respect to this threat model, and have proposed new definitions based on vulnerability and min-entropy.

We mention some important future directions. First, the reasonableness of our definitions should be further assessed, both in terms of their theoretical properties and also by applying them in various specific threat scenarios. Also the definitions need to be generalized to model explicitly allowed flows, such as from low inputs to low outputs or (perhaps) from the secret individual votes in an election to the public tally. It would seem that this could be handled through conditional min-entropy.

Second, the possibility of enforcing quantitative information flow policies through static analysis needs to be explored; in the case of the standard measures there has been progress [9], but it is unclear whether min-entropy can be handled similarly. The results presented in Section 5 on how to calculate vulnerability seem encouraging, especially in the important special case of a deterministic program c mapping a uniformly-distributed $\mathtt{H}$ to an output $\mathtt{L}$. For there we found that the leakage is simply $\log|\mathcal{L}|$. This fact seems to give insight into the difference between the cases of *password checking* and *binary search*. For in a password checker, we test whether a guess g is equal to the secret password. If the test comes out true, we know that the password is g; if it comes out false, we know only that the password is not g. Hence such a guess splits the space $\mathcal{H}$ into two equivalence classes, $\{g\}$ and $\mathcal{H} - \{g\}$. This implies that any *tree* of guesses of height k can give only $k + 1$ equivalence classes, which means that the vulnerability of the password increases slowly. In contrast, in binary search we compare the *size* of a guess g with the secret, discovering which is larger. Hence (with a well-chosen guess) we are able to split the space $\mathcal{H}$ into two equivalence classes of roughly equal size. This implies that a tree of guesses of height k can give 2^k equivalence classes, which means that the vulnerability of the secret increases very rapidly. These examples do raise concerns about *compositionality*, however, because the tests $g = \mathtt{H}$ and $g \leq \mathtt{H}$ both have a leakage of 1 under our definitions, even though *sequences* of these tests behave so differently.

Finally, it would be valuable (but challenging) to integrate the information-theoretic viewpoint used here with the computational complexity viewpoint used in cryptography.

Acknowledgments

I am grateful to Rafael Alpízar, Zesheng Chen, Pasquale Malacaria, Ziyuan Meng, Andrew Myers, Catuscia Palamidessi, and the anonymous reviewers for helpful discussions of this work. This work was partially supported by the National Science Foundation under grants HRD-0317692 and CNS-0831114.

References

1. Sabelfeld, A., Myers, A.C.: Language-based information flow security. IEEE Journal on Selected Areas in Communications 21(1), 5–19 (2003)
2. Chatzikokolakis, K., Palamidessi, C., Panangaden, P.: Anonymity protocols as noisy channels. Information and Computation 206, 378–401 (2008)
3. Köpf, B., Basin, D.: An information-theoretic model for adaptive side-channel attacks. In: Proceedings 14th ACM Conference on Computer and Communications Security, Alexandria, Virginia (2007)
4. Sabelfeld, A., Sands, D.: Dimensions and principles of declassification. In: Proceedings 18th IEEE Computer Security Foundations Workshop (June 2005)
5. Denning, D.: Cryptography and Data Security. Addison-Wesley, Reading (1982)
6. Gray III, J.W.: Probabilistic interference. In: Proceedings 1990 IEEE Symposium on Security and Privacy, Oakland, CA, pp. 170–179 (May 1990)
7. Clark, D., Hunt, S., Malacaria, P.: Quantitative analysis of the leakage of confidential data. Electronic Notes in Theoretical Computer Science 59(3) (2002)
8. Clark, D., Hunt, S., Malacaria, P.: Quantitative information flow, relations and polymorphic types. Journal of Logic and Computation 18(2), 181–199 (2005)
9. Clark, D., Hunt, S., Malacaria, P.: A static analysis for quantifying information flow in a simple imperative language. Journal of Computer Security 15, 321–371 (2007)
10. Malacaria, P.: Assessing security threats of looping constructs. In: Proceedings 34th Symposium on Principles of Programming Languages, Nice, France, pp. 225–235 (January 2007)
11. Malacaria, P., Chen, H.: Lagrange multipliers and maximum information leakage in different observational models. In: Proc. PLAS 2008: ACM SIGPLAN Workshop on Programming Languages and Analysis for Security, Tucson, Arizona, USA, pp. 135–146 (June 2008)
12. Clarkson, M., Myers, A., Schneider, F.: Belief in information flow. In: Proceedings 18th IEEE Computer Security Foundations Workshop, Aix-en-Provence, France, pp. 31–45 (June 2005)
13. Chatzikokolakis, K., Palamidessi, C., Panangaden, P.: Probability of error in information-hiding protocols. In: Proceedings 20th IEEE Computer Security Foundations Symposium, pp. 341–354 (2007)
14. Lowe, G.: Quantifying information flow. In: Proceedings 15th IEEE Computer Security Foundations Workshop, Cape Breton, Nova Scotia, Canada, pp. 18–31 (June 2002)
15. Di Pierro, A., Hankin, C., Wiklicky, H.: Approximate non-interference. In: Proceedings 15th IEEE Computer Security Foundations Workshop, Cape Breton, Nova Scotia, Canada, pp. 1–17 (June 2002)
16. Rényi, A.: On measures of entropy and information. In: Proceedings of the 4th Berkeley Symposium on Mathematics, Statistics and Probability 1960, pp. 547–561 (1961)
17. Tóth, G., Hornák, Z., Vajda, F.: Measuring anonymity revisited. In: Liimatainen, S., Virtanen, T. (eds.) Proceedings of the Ninth Nordic Workshop on Secure IT Systems, Espoo, Finland, pp. 85–90 (2004)
18. Shmatikov, V., Wang, M.H.: Measuring relationship anonymity in mix networks. In: WPES 2006: Proceedings of the 5th ACM workshop on Privacy in Electronic Society, Alexandria, Virginia, pp. 59–62 (2006)
19. Smith, G.: Adversaries and information leaks (Tutorial). In: Barthe, G., Fournet, C. (eds.) TGC 2007. LNCS, vol. 4912, pp. 383–400. Springer, Heidelberg (2008)
20. Shannon, C.E.: A mathematical theory of communication. Bell System Technical Journal 27, 379–423 (1948)

21. Gallager, R.G.: Information Theory and Reliable Communication. John Wiley and Sons, Inc., Chichester (1968)
22. Cover, T.M., Thomas, J.A.: Elements of Information Theory, 2nd edn. John Wiley & Sons, Inc., Chichester (2006)
23. Massey, J.L.: Guessing and entropy. In: Proceedings 1994 IEEE International Symposium on Information Theory, p. 204 (1994)
24. Cachin, C.: Entropy Measures and Unconditional Security in Cryptography. PhD thesis, Swiss Federal Institute of Technology (1997)
25. Dodis, Y., Ostrovsky, R., Reyzin, L., Smith, A.: Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. SIAM Journal of Computing 38(1), 97–139 (2008)

Cryptographic Protocol Composition via the Authentication Tests[⋆]

Joshua D. Guttman

The MITRE Corporation
guttman@mitre.org
http://eprint.iacr.org/2008/430

Abstract. Although cryptographic protocols are typically analyzed in isolation, they are used in combinations. If a protocol Π_1, when analyzed alone, was shown to meet some security goals, will it still meet those goals when executed together with a second protocol Π_2? Not necessarily: for every Π_1, some Π_2s undermine its goals. We use the strand space "authentication test" principles to suggest a criterion to ensure a Π_2 preserves Π_1's goals; this criterion strengthens previous proposals.

Security goals for Π_1 are expressed in a language $\mathcal{L}(\Pi_1)$ in classical logic. Strand spaces provide the models for $\mathcal{L}(\Pi_1)$. Certain homomorphisms among models for $\mathcal{L}(\Pi)$ preserve the truth of the security goals. This gives a way to extract—from a counterexample to a goal that uses both protocols—a counterexample using only the first protocol. This model-theoretic technique, using homomorphisms among models to prove results about a syntactically defined set of formulas, appears to be novel for protocol analysis.

Protocol analysis usually focuses on the secrecy and authentication properties of individual, finished protocols. There is a good reason for this: Each security goal then definitely either holds or does not hold. However, the analysis is more reusable if we know which results will remain true after combination with other protocols, and perhaps other kinds of elaborations to the protocol.

In practice, every protocol is used in combination with other protocols, often with the same long-term keys. Also, many protocols contain messages with "blank slots." Higher level protocols piggyback on them, filling the blank spots with their own messages. We want to find out when the goals that hold of a protocol on its own are preserved under combination with other protocols, and when these blanks are filled in.

Two Results on Composition. Two existing results, both within the Dolev-Yao model [12], are particularly relevant. We showed [17] that if two protocols manipulate disjoint sets of ciphertexts, then combining the protocols cannot undermine their security goals. A careful, asymmetric formulation of this "disjoint encryption" property allowed us to show that one protocol Π_1 may produce

[⋆] Supported by MITRE-Sponsored Research.

ciphertexts—in a broad sense including digital certificates as well as Kerberos-style tickets—consumed by another protocol Π_2, without Π_2 undermining any security goal of Π_1. The relation between Π_1 and Π_2 is *asymmetric* in that security goals of Π_2 could be affected by the behavior of Π_1, but not conversely.

Our result concerned only protocols that completely parse the messages they receive to atomic values, leaving no unstructured blank slots. A second limitation was to cover protocols using only atomic keys, not keys produced (e.g.) by hashing compound messages. A recent result by Delaune et al. [8] lifts these two limitations, but only in the *symmetric* case (akin to our Def. 9, clause 4). It applies only when neither protocol produces ciphertexts that may be consumed by the other, and hence when neither protocol could affect goals achieved by the other alone. Their method appears not to extend beyond this symmetric case.

One goal of this paper is an asymmetric result covering blank slots and compound keys.

Our Approach. Protocol executions—more specifically, the parts carried out by the rule-abiding, "regular" participants, but not the adversary—form objects we call *skeletons* [11]. A skeleton is *realized* if it contains enough protocol behavior so that, when combined with some adversary behavior, it can occur. If additional regular behavior is needed for a possible execution, then it is *unrealized.*

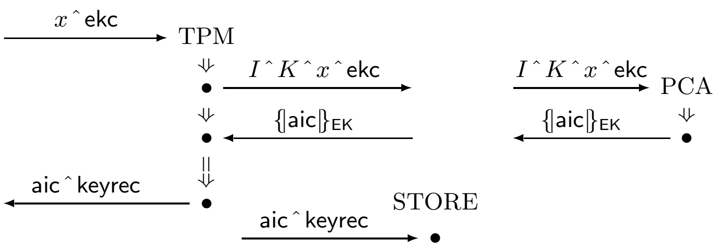
We introduce a new first order language $\mathcal{L}(\Pi)$ to describe skeletons of each protocol Π. Skeletons containing regular behaviors of Π provide a semantics, a set of models, for formulas of $\mathcal{L}(\Pi)$. Security goals are closed formulas $G \in \mathcal{L}(\Pi)$ of specific forms. A skeleton $\mathbb{A}$ is a *counterexample* to G when $\mathbb{A}$ is realized, but $\mathbb{A}$ satisfies G's negation, $\mathbb{A} \models \neg G$.

When Π_1 and Π_2 are protocols, $\mathcal{L}(\Pi_1)$ is a sublanguage of $\mathcal{L}(\Pi_1 \cup \Pi_2)$, and the security goals G_1 of $\mathcal{L}(\Pi_1)$ are some the goals of $\mathcal{L}(\Pi_1 \cup \Pi_2)$. The skeletons of Π_1 are those skeletons of $\Pi_1 \cup \Pi_2$ in which only Π_1 activity occurs.

We will define a syntactic relation between Π_1 and Π_2, called *strong disjoint encryption*, that ensures goals $G_1 \in \mathcal{L}(\Pi_1)$ are preserved. If any $\Pi_1 \cup \Pi_2$-skeleton is a counterexample to $G_1 \in \mathcal{L}(\Pi_1)$, we want to extract a Π_1-skeleton $\mathbb{A}_1$ which is a counterexample to G_1. Thus, Π_1 alone already undermines any goal that Π_1, Π_2 undermine jointly. The language $\mathcal{L}(\Pi)$, the definition of strong disjointness, and this result are the contributions of this paper.

The authentication test principles [10] suggest the definition of strong disjoint encryption, in two parts. First, Π_2 should not create encryptions of forms specified in Π_1; i.e. Π_2 should have no *creation conflicts.* Second, if a Π_2 execution receives a value only inside encryptions specified in Π_1, it should not re-transmit the value outside these encryptions; i.e. there should be no *extraction conflicts.*

We find Π_1-counterexamples from $\Pi_1 \cup \Pi_2$-counterexamples in two steps. First, we omit all non-Π_1 behavior. Second, we generalize: we remove all encrypted units not specified in Π_1 by inserting blank slots in their place. Each of these operations *preserves satisfaction* of $\neg G_1$. When Π_1, Π_2 has strong disjointness, they yield *realized* Π_1 skeletons from realized $\Pi_1 \cup \Pi_2$ skeletons. Hence, they preserve counterexamples.



$$\mathsf{ekc} = [\![\, ekc\ \mathsf{MF} \,\hat{}\, \mathsf{EK} \,]\!]_{\mathsf{sk(MF)}} \qquad \mathsf{aic} = [\![\, aic\ I \,\hat{}\, K \,\hat{}\, x \,]\!]_{\mathsf{sk(PCA)}}$$
$$\mathsf{keyrec} = \{\!|\, aikrec\ K, K^{-1} \,|\!\}_{\mathsf{SRK}}$$

Fig. 1. Modified Anonymous Identity Protocol

This reasoning is model-theoretic, characteristically combining two elements. One is the algebraic relations (embeddings and restrictions, homomorphisms, etc.) among the structures interpreting a logic. The second concerns syntax, often focusing on formulas of particular logical forms. A familiar example, combining these two ingredients, is the fact that a homomorphism between two structures for first order logic preserves satisfaction for atomic formulas.

A second goal of this paper is to illustrate this model-theoretic approach to security protocols.

Structure of This Paper. We first give an example certificate distribution protocol called MAIP. Section 1 gives background on strand spaces, including authentication tests. The goals of MAIP are formalized in $\mathcal{L}(\Pi)$, introduced (along with its semantics) in Section 2. Multiprotocols and strong disjointness are defined in Section 3. Section 4 gives the main results, and concludes.

Example: Anonymous Identities in Trusted Computing. A certificate distribution protocol (see Fig. 1) for "anonymous identity keys" is used with Trusted Platform Modules (TPMs). A Privacy Certificate Authority (PCA) creates a certificate aic binding a key K to a temporary name I. K is a public signature verification key. The certificate authorizes the signing key K^{-1} to sign requests on behalf of the holder of I [4].

Since the PCA knows nothing about the origin of a request, it transmits aic encrypted under key EK. The TPM manufacturer MF certifies in ekc that the matching private decryption key EK^{-1} resides within a TPM. If the request did not originate from this TPM, the certificate will never be decrypted, while otherwise that TPM will protect K^{-1} and use it according to certain rules. In particular, the TPM emits K, K^{-1} encrypted with a storage key SRK that only it possesses; this key record can be reloaded and used later.

We have omitted some details, and added a "blank slot" parameter x, which may be used to restrict the use of the aic. For instance, x can determine when the certificate expires, or it can limit its use to specific later protocols.

$[\![m]\!]_{\mathsf{sk}(A)}$ refers to m accompanied by a digitally signed hash, created with a signature key held by A. The tags *ekc*, *aic*, and *aikrec* are bitpatterns that distinguish units containing them from other cryptographically prepared values.

Security Goals of MAIP. MAIP has three main goals. They should hold whenever an $\mathsf{aic}, \mathsf{keyrec}$ pair is successfully stored for future use. First, the PCA should have produced aic, and transmitted it encrypted with *some* EK. Second, the TPM should have received aic encrypted with this EK, and retransmitted it in the clear. These goals are authentication goals, since they assert that uncompromised (regular) principals executed certain actions. The third is a confidentiality goal, stating that the private part K^{-1} of the AIK should never be observed, unprotected by encryption. Making our assumptions explicit, we get:

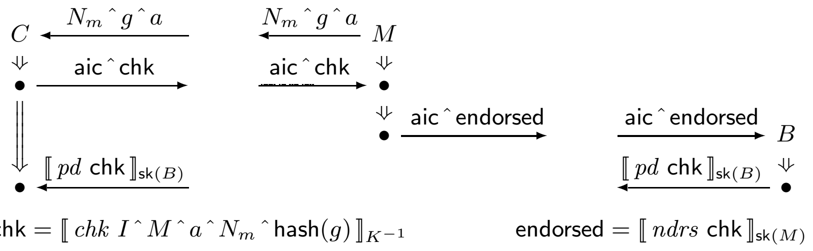
Whenever 1. STORE gets a message using parameters $I, K, x, \mathsf{PCA}, \mathsf{SRK}$;
2. $\mathsf{sk}(\mathsf{MF})$, $\mathsf{sk}(\mathsf{PCA})$, SRK^{-1} are used only in accordance with MAIP;
3. K, K^{-1} are generated only once,

then, for some public key EK,
1. A PCA run using the parameters $I, K, x, \mathsf{PCA}, \mathsf{EK}$ reached step 2;
2. A TPM run using the parameters $I, K, x, \mathsf{PCA}, \mathsf{EK}, \mathsf{SRK}$ reached step 3;
3. K^{-1} is never observed, unprotected by encryption.

Since the definitions of the protocol's roles are fixed, the goals do not need to say anything about the forms of the messages. They say only how far the role has progressed, and with what parameters. The unlinkability of K to a particular EK matches the classical existential quantifier used here.

MAIP and Other Protocols. A certificate distribution protocol like MAIP, on its own, is precisely useless.

MAIP becomes useful only if principals executing other protocols generate messages signed with K^{-1}, and accept messages verified with K, accompanied by a matching aic. For instance, a message signed with K^{-1} could be used to request access to network services, as is now widely done with Kerberos. More ambitiously, the aic could be regarded as a check-signing certificate from a bank's PCA. Then K^{-1} can be used to sign on-line checks that are anonymous to the payee, but guaranteed by the bank (Fig. 2). The blank slots g, a may be filled with formatted data representing the goods offered and amount to be paid.



C
$N_m \hat{} g \hat{} a$
$\mathsf{aic} \hat{} \mathsf{chk}$
$[\![pd\ \mathsf{chk}]\!]_{\mathsf{sk}(B)}$
M
$\mathsf{aic} \hat{} \mathsf{endorsed}$
B
$\mathsf{chk} = [\![chk\ I \hat{} M \hat{} a \hat{} N_m \hat{} \mathsf{hash}(g)]\!]_{K^{-1}}$
$\mathsf{endorsed} = [\![ndrs\ \mathsf{chk}]\!]_{\mathsf{sk}(M)}$

Fig. 2. AIC-based check cashing

Unfortunately, the symmetric criterion for combining protocols [8] says nothing about how to construct such secondary protocols, since aic is a cryptographic unit that must be shared between the protocols. Our asymmetric criterion [17] does not apply, since MAIP, like many protocols, contains a blank slot x.

A criterion for safe composition should guide protocol designers to construct suites that work together. Our criterion says to avoid encryption creation conflicts and extraction conflicts. The protocol must not *create* anything unifying with the aic, ekc, and keyrec message formats, or with an aic encrypted with a public key. It must not *extract* anything that unifies with an aic from an encrypted item such as $\{\!|\mathsf{aic}|\!\}_{\mathsf{EK}}$. Creating and extracting units of these forms must remain the exclusive right of the primary protocol MAIP.

Our criterion of strongly disjoint encryption (Def. 9, Thm. 2) indicates that these are the only constraints on secondary protocols to interoperate with MAIP.

1 Messages, Protocols, Skeletons

Let $\mathfrak{A}_0$ be an algebra equipped with some operators and a set of homomorphisms $\eta\colon \mathfrak{A}_0 \to \mathfrak{A}_0$. We call members of $\mathfrak{A}_0$ *atoms.*

For the sake of definiteness, we will assume here that $\mathfrak{A}_0$ is the disjoint union of infinite sets of *nonces*, *atomic keys*, *names*, and *texts.* The operator $\mathsf{sk}(a)$ maps names to (atomic) signature keys, and K^{-1} maps an asymmetric atomic key to its inverse, and a symmetric atomic key to itself. Homomorphisms η are maps that respect sorts, and act homomorphically on $\mathsf{sk}(a)$ and K^{-1}.

Let X is an infinite set disjoint from $\mathfrak{A}_0$; its members—called *indeterminates*—act like unsorted variables. $\mathfrak{A}$ is freely generated from $\mathfrak{A}_0 \cup X$ by two operations: encryption $\{\!|t_0|\!\}_{t_1}$ and tagged concatenation $\mathit{tag}\ t_0\,\hat{}\,t_1$, where the tags tag are drawn from some set TAG. For a distinguished tag nil, we write $\mathit{nil}\ \ t_0\,\hat{}\,t_1$ as $t_0\,\hat{}\,t_1$ with no tag. In $\{\!|t_0|\!\}_{t_1}$, a non-atomic key t_1 is a symmetric key. Members of $\mathfrak{A}$ are called *messages.*

A homomorphism $\alpha = (\eta, \chi)\colon \mathfrak{A} \to \mathfrak{A}$ consists of a homomorphism η on atoms and a function $\chi\colon X \to \mathfrak{A}$. It is defined for all $t \in \mathfrak{A}$ by the conditions:

$$\begin{array}{llr} \alpha(a) = \eta(a), & \text{if } a \in \mathfrak{A}_0 & \alpha(\{\!|t_0|\!\}_{t_1}) = \{\!|\alpha(t_0)|\!\}_{\alpha(t_1)} \\ \alpha(x) = \chi(x), & \text{if } x \in X & \alpha(\mathit{tag}\ t_0\,\hat{}\,t_1) = \mathit{tag}\ \alpha(t_0)\,\hat{}\,\alpha(t_1) \end{array}$$

Indeterminates x serve as blank slots, to be filled by any $\chi(x) \in \mathfrak{A}$. This $\mathfrak{A}$ has the most general unifier property, which we will rely on. That is, suppose that for $v, w \in \mathfrak{A}$, there exist α, β such that $\alpha(v) = \beta(w)$. Then there are α_0, β_0, such that $\alpha_0(v) = \beta_0(w)$, and whenever $\alpha(v) = \beta(w)$, then α and β are of the forms $\gamma \circ \alpha_0$ and $\gamma \circ \beta_0$. Messages are abstract syntax trees in the usual way:

1. Let ℓ and r be the partial functions such that for $t = \{\!|t_1|\!\}_{t_2}$ or $t = \mathit{tag}\ t_1\,\hat{}\,t_2$, $\ell(t) = t_1$ and $r(t) = t_2$; and for $t \in \mathfrak{A}_0$, ℓ and r are undefined.
2. A *path* p is a sequence in $\{\ell, r\}^*$. We regard p as a partial function, where $\langle\rangle = \mathsf{Id}$ and $\mathsf{cons}(f, p) = p \circ f$. When the rhs is defined, we have: 1. $\langle\rangle(t) = t$; 2. $\mathsf{cons}(\ell, p)(t) = p(\ell(t))$; and 3. $\mathsf{cons}(r, p)(t) = p(r(t))$.
3. *p traverses a key edge* in t if $p_1(t)$ is an encryption, where $p = p_1{}^\frown\langle r\rangle^\frown p_2$.
4. *p traverses a member of S* if $p_1(t) \in S$, where $p = p_1{}^\frown p_2$ and $p_2 \neq \langle\rangle$.

5. t_0 *is an ingredient of* t, written $t_0 \sqsubseteq t$, if $t_0 = p(t)$ for some p that does not traverse a key edge in t.[1]
6. t_0 *appears in* t, written $t_0 \ll t$, if $t_0 = p(t)$ for some p.

A single local session of a protocol at a single principal is a *strand*, containing a linearly ordered sequence of transmissions and receptions that we call *nodes*. In Fig. 1, the vertical columns of nodes connected by double arrows $\Rightarrow$ are strands.

A message t_0 *originates* at a node n_1 if (1) n_1 is a transmission node; (2) $t_0 \sqsubseteq \mathsf{msg}(n_1)$; and (3) whenever $n_0 \Rightarrow^+ n_1$, $t_0 \not\sqsubseteq \mathsf{msg}(n_0)$.

Thus, t_0 originates when it was transmitted without having been either received or transmitted previously on the same strand. Values assumed to originate only on one node in an execution—*uniquely originating* values—formalize the idea of freshly chosen, unguessable values. Values assumed to originate nowhere may be used to encrypt or decrypt, but are never sent as message ingredients. They are called *non-originating* values. For a non-originating value K, $K \not\sqsubseteq t$ for any transmitted message t. However, $K \ll \{|t_0|\}_K \sqsubseteq t$ possibly, which is why we distinguish $\sqsubseteq$ from $\ll$. See [18,11] for more details.

In the tree model of messages, to apply a homomorphism, we walk through, copying the tree, but inserting $\alpha(a)$ every time an atom a is encountered, and inserting $\alpha(x)$ every time that an indeterminate x is encountered.

Definition 1. *Let S be a set of encryptions. A message t_0* is found only within *S in t_1, written $t_0 \odot^S t_1$, iff for every path p such that $p(t_1) = t_0$, either (1) p traverses a key edge or else (2) p traverses a member of S before its end.*

Message t_0 is found outside *S in t_1, written $t_0 \dagger^S t_1$, iff not $(t_0 \odot^S t_1)$.* □

Equivalently, $t_0 \dagger^S t_1$ iff for some path p, (1) $p(t_1) = t_0$, (2) p traverses no key edge, and (3) p traverses no member of S before its end. Thus, $t_0 \sqsubseteq t_1$ iff $t_0 \dagger^{\emptyset} t_1$.

E.g. $\mathsf{aic} \dagger^{\emptyset} \{|\mathsf{aic}|\}_{\mathsf{EK}}$, although $\mathsf{aic} \odot^{S_1} \{|\mathsf{aic}|\}_{\mathsf{EK}}$, where $S_1 = \{\, \{|\mathsf{aic}|\}_{\mathsf{EK}} \,\}$. The TPM transforms aic, transmitting a t such that $\mathsf{aic} \dagger^{S_1} t$, namely $t = \mathsf{aic}\,\hat{}\,\mathsf{keyrec}$.

Protocols. A *protocol Π* is a finite set of strands, representing the roles of the protocol. Three of the roles of the MAIP are the strands shown in Fig. 1. Their instances result by replacing I, K, etc., by any name, asymmetric key, etc., and replacing x by any (possibly compound) message. The fourth role is the *listener* role $\mathsf{Lsn}[y]$ with a single reception node in which y is received. The instances of $\mathsf{Lsn}[y]$ are used to document that values are available without cryptographic protection. For instance, $\mathsf{Lsn}[K]$ would document that K is compromised. Every protocol contains the role $\mathsf{Lsn}[y]$.

Indeterminates represent messages received from protocol peers, or passed down as parameters from higher-level protocols. Thus, we require:

If n_1 is a node on $\rho \in \Pi$, with an indeterminate $x \ll \mathsf{msg}(n_1)$,
then $\exists n_0$, $n_0 \Rightarrow^* n_1$, where n_0 is a reception node and $x \sqsubseteq \mathsf{msg}(n_0)$.

[1] $\sqsubseteq$ was formerly called the "subterm" relation [18], causing some confusion. A key is not an ingredient in its ciphertexts, but an aspect of how they were prepared, so $K \not\sqsubseteq \{|t|\}_K$ unless $K \sqsubseteq t$. Also, $\sqsubseteq \cap (\mathfrak{A}_0 \times \mathfrak{A}_0) = \mathsf{Id}_{\mathfrak{A}_0}$, so e.g. $a \not\sqsubseteq \mathsf{sk}(a)$.

So, an indeterminate is received as an ingredient before appearing in any other way. The initial node on the TPM AIC role in Fig. 1 shows x being received from the higher level protocol that has invoked the TPM activity.

A principal executing a role such as the PCA's role in MAIP may be partway through its run; for instance, it may have executed the first receive event without "yet" having executed its second event, the transmission node.

Definition 2. *Node n is a* role node *of Π if n lies on some $\rho \in \Pi$.*

Let n_j be a role node of Π of the form $n_1 \Rightarrow \ldots \Rightarrow n_j \Rightarrow \ldots$. Node m_j is an instance *of n_j if, for some homomorphism α, the strand of m_j, up to m_j, takes the form: $\alpha(n_1) \Rightarrow \ldots \Rightarrow \alpha(n_j) = m_j$.* □

That is, messages and their directions—transmission or reception—must agree up to node j. However, any remainders of the two strands beyond node j are unconstrained. They need not be compatible. When a protocol allows a principals to decide between different behaviors after step j, based on the message contents of their run, then this definition represents branching [14,16]. At step j, one doesn't yet know which branch will be taken.

Skeletons. A *skeleton* $\mathbb{A}$ consists of (possibly partially executed) role instances, i.e. a finite set of nodes, $\mathsf{nodes}(\mathbb{A})$, with two additional kinds of information:

1. A partial ordering $\preceq_{\mathbb{A}}$ on $\mathsf{nodes}(\mathbb{A})$;
2. Finite sets $\mathsf{unique}_{\mathbb{A}}, \mathsf{non}_{\mathbb{A}}$ of atomic values assumed uniquely originating and respectively non-originating in $\mathbb{A}$.

$\mathsf{nodes}(\mathbb{A})$ and $\preceq_{\mathbb{A}}$ must respect the strand order, i.e. if $n_1 \in \mathsf{nodes}(\mathbb{A})$ and $n_0 \Rightarrow n_1$, then $n_0 \in \mathsf{nodes}(\mathbb{A})$ and $n_0 \preceq_{\mathbb{A}} n_1$. If $a \in \mathsf{unique}_{\mathbb{A}}$, then a must originate at most once in $\mathsf{nodes}(\mathbb{A})$. If $a \in \mathsf{non}_{\mathbb{A}}$, then a must originate nowhere in $\mathsf{nodes}(\mathbb{A})$, though a or a^{-1} may be the key encrypting some $e \ll \mathsf{msg}(n)$ for $n \in \mathsf{nodes}(\mathbb{A})$.

$\mathbb{A}$ is *realized* if it is a possible run without additional activity of *regular* participants; i.e., for every reception node n, the adversary can construct $\mathsf{msg}(n)$ via the Dolev-Yao adversary actions,[2] using as inputs:

1. all messages $\mathsf{msg}(m)$ where $m \prec_{\mathbb{A}} n$ and m is a transmission node;
2. any atomic values a such that $a \notin (\mathsf{non}_{\mathbb{A}} \cup \mathsf{unique}_{\mathbb{A}})$, or such that $a \in \mathsf{unique}_{\mathbb{A}}$ but a originates nowhere in $\mathbb{A}$.

A homomorphism α yields a partial function on skeletons. We apply α to the messages on all nodes of $\mathbb{A}$, as well as to the sets $\mathsf{unique}_{\mathbb{A}}$ and $\mathsf{non}_{\mathbb{A}}$. We regard α as a homomorphism from $\mathbb{A}$ to $\alpha(\mathbb{A})$, when this is defined. However, α must not identify $K \in \mathsf{non}_{\mathbb{A}}$ with any atom that is an ingredient in any message in $\mathbb{A}$, or identify $a \in \mathsf{unique}_{\mathbb{A}}$ with another atom if this would give $\alpha(a)$ two originating nodes in $\alpha(\mathbb{A})$. A homomorphism α always acts as a bijection between $\mathsf{nodes}(\mathbb{A})$ and $\mathsf{nodes}(\alpha(\mathbb{A}))$. In [11] we use a compatible although more inclusive notion of homomorphism, since the action on nodes is not always bijective.

[2] The Dolev-Yao adversary actions are: concatenating messages and separating the pieces of a concatenation; encrypting a given plaintext using a given key; and decrypting a given ciphertext using the matching decryption key.

Authentication Tests. The core proof method in the strand space framework is the authentication test idea [11,10].[3] The idea concerns a realized skeleton $\mathbb{A}$ and a value c. If c was found only within a set of encryptions S, up to some point, and is later found outside S, then this "test" must be explained or "solved." Solutions are of two kinds: Either a key is compromised, so the adversary can create an occurrence of c outside S, or else a regular strand has a transmission node m_1 where, for all earlier nodes $m_0 \Rightarrow^+ m_1$,

$$c \odot^S \mathsf{msg}(m_0), \quad \text{but} \quad c \dagger^S \mathsf{msg}(m_1).$$

Since there are only finitely many roles in Π, unification on their nodes can find all candidates for regular solution nodes m_1. We formalize this using *cuts.*

Definition 3. *Let c be an atom or an encryption, and S be a set of encryptions.* $\mathsf{Cut}(c, S, \mathbb{A})$*, the* test cut *for c, S in $\mathbb{A}$, is defined if $\exists n_1 \in$* $\mathsf{nodes}(\mathbb{A})$ *such that $c \dagger^S$* $\mathsf{msg}(n_1)$*. In this case,*

$$\mathsf{Cut}(c, S, \mathbb{A}) = \{n \in \mathsf{nodes}(\mathbb{A}) \colon \exists m \,.\, m \preceq_{\mathbb{A}} n \wedge c \dagger^S \mathsf{msg}(m)\}. \qquad \square$$

For instance, in any skeleton $\mathbb{A}$ containing a full run of the TPM role, its fourth node n_4 is in the cut $\mathsf{Cut}(\mathsf{aic}, S, \mathbb{A})$ for *every* S, since n_4 transmits aic outside every encryption. Letting $\mathbb{A}_0$ be the skeleton containing all the activity in Fig. 1, with the visually apparent ordering, the third TPM node $n_3 \in \mathsf{Cut}(\mathsf{aic}, \emptyset, \mathbb{A})$ but $n_3 \notin \mathsf{Cut}(\mathsf{aic}, S_1, \mathbb{A})$ where $S_1 = \{\, \{\!|\mathsf{aic}|\!\}_{\mathsf{EK}} \,\}$. In all $m \preceq_{\mathbb{A}_0} n_3$, $\mathsf{aic} \odot^{S_1} \mathsf{msg}(m)$.

Definition 4. $U = \mathsf{Cut}(c, S, \mathbb{A})$ *is* solved *if for every $\preceq_{\mathbb{A}}$-minimal $m_1 \in U$:*

1. *either m_1 is a transmission node;*
2. *or there is a listener node $m = \mathsf{Lsn}[t]$ with $m \prec_{\mathbb{A}} m_1$, and either*
   (a) *$c = \{\!|t_0|\!\}_{t_1}$ and $t_1 = t$, or else*
   (b) *for some $\{\!|t_0|\!\}_{t_1} \in S$, t is the corresponding decryption key $t = t_1^{-1}$.* $\square$

In the skeleton $\mathbb{A}_0$ from Fig. 1, the cut $\mathsf{Cut}(\mathsf{aic}, S_1, \mathbb{A})$ is solved by n_4. The cut $\mathsf{Cut}(\mathsf{aic}, \emptyset, \mathbb{A})$ is solved by the second PCA node, which transmits $\{\!|\mathsf{aic}|\!\}_{\mathsf{EK}}$, which means that it occurs outside the empty set at this point.

In MAIP, these are the two important cuts. In skeleton $\mathbb{A}_0$, they are solved by regular strands emitting the aic (clause 1) in new forms, but in other skeletons they could be solved by a listener node $m = \mathsf{Lsn}[\mathsf{privk}(PCA)]$ (clause 2a), or, for S_1, by $m' = \mathsf{Lsn}[\mathsf{EK}^{-1}]$ (clause 2b). In skeletons in which EK^{-1} and $\mathsf{privk}(PCA)$ are non-originating, then the TPM and PCA strands offer the only solutions.

Theorem 1 ([10]).

1. *If every well-defined cut in $\mathbb{A}$ is solved, $\mathbb{A}$ is realized.*
2. *If $\mathbb{A}$ is realized, then $\mathbb{A}$ has an extension $\mathbb{A}'$, obtained by adding only listener nodes, in which every well-defined cut is solved.*

Clauses 1 and 2 assert *completeness* [10, Prop. 5], and *soundness* [10, Props. 2,3], respectively.

[3] A fine point is that [11] worked in a framework without indeterminates X, while [10] established its completeness result for a stronger framework including them.

2 The Goal Language $\mathcal{L}(\Pi)$

$\mathcal{L}(\Pi)$ is a language for talking about executions of a protocol Π. We use typewriter font $\mathtt{x}, \mathtt{m}$, etc. for syntactic items including metasymbols such as $\mathtt{\Phi}, \mathtt{RhoJ}$.

Definition 5. *$\mathcal{L}(\Pi)$ is the classical quantified language with vocabulary:*

Variables *(unsorted) ranging over messages in $\mathfrak{A}$ and nodes;*
Function symbols $\mathtt{sk}$*,* $\mathtt{inv}$ *for the functions on $\mathfrak{A}_0$;*
Predicate symbols *equality* $\mathtt{u} = \mathtt{v}$*, falsehood* $\mathtt{false}$ *(no arguments), and:*

- $\mathtt{Non(v)}$*,* $\mathtt{Unq(v)}$*, and* $\mathtt{UnqAt(n, v)}$*;*
- $\mathtt{DblArrw(m, n)}$ *and* $\mathtt{Prec(m, n)}$*;*
- *One role predicate* $\mathtt{RhoJ}$ *for each role $\rho \in \Pi$ and each j with $1 \leq j \leq \mathsf{length}(\rho)$. The predicate $\mathtt{RhoJ(m, v_1, \ldots, v_i)}$ for the j^{th} node on ρ has as arguments: a variable $\mathtt{m}$ for the node, and variables for each of the i parameters that have appeared in any of ρ's first j messages.* □

Suppose for the moment that a message value v is associated with each variable $\mathtt{v}$, and the nodes m, n are associated with the variables $\mathtt{m}, \mathtt{n}$. Then the predicates $\mathtt{Non(v)}$, $\mathtt{Unq(v)}$, and $\mathtt{UnqAt(n, v)}$ are (respectively) true in a skeleton $\mathbb{A}$ when v is assumed non-originating in $\mathsf{non}_{\mathbb{A}}$; when v is assumed uniquely originating in $\mathsf{unique}_{\mathbb{A}}$; and when v is assumed uniquely originating in $\mathsf{unique}_{\mathbb{A}}$ and moreover originates at the node n in $\mathbb{A}$. The predicates $\mathtt{DblArrw(m, n)}$ and $\mathtt{Prec(m, n)}$ are (respectively) true in a skeleton $\mathbb{A}$ when the node m lies immediately before the node n, i.e. $m \Rightarrow n$; and when $m \prec_{\mathbb{A}} n$.

Role predicate $\mathtt{RhoJ(m, v_1, \ldots, v_i)}$ is true in a skeleton $\mathbb{A}$ when m is the j^{th} node of an instance of role ρ, with its parameters (in some conventional order) instantiated by the associated values $v_1, \ldots, v_i$. The role predicates are akin to the role state facts of multiset rewriting [13].

In particular, since every protocol Π contains the listener role $\mathsf{Lsn}[y]$, $\mathcal{L}(\Pi)$ always has a role predicate $\mathtt{Lsn1(m, x)}$, meaning that m is the first node on a listener strand receiving message x. It is used to express confidentiality goals.

The MAIP TPM role has four role predicates; the first two are:

- $\mathtt{maip\_tpm1(m, x)}$, meaning that m is a reception node not preceded by any other on its strand, and the message received is on node m is just the parameter x, as dictated by the definition of the MAIP TPM role;
- $\mathtt{maip\_tpm2(m, x, i, k, f, e)}$, meaning that m lies on the second position on its strand after a node m' such that $\mathtt{maip\_tpm1(m', x)}$, and m transmits message: $i \hat{\ } k \hat{\ } x \hat{\ } [\![\, ekc \;\; f \hat{\ } e \,]\!]_{\mathsf{sk}(f)}$. These are not independent; a valid formula is:

$$\mathtt{maip\_tpm2(m_2, x, i, k, f, e)} \supset \exists \mathtt{m_1} \,.\, \mathtt{DblArrw(m_1, m_2)} \wedge \mathtt{maip\_tpm1(m_1, x)}.$$

If Π_1 is a subprotocol of Π in the sense that every role of Π_1 is a role of Π, then $\mathcal{L}(\Pi_1)$ is a sublanguage of $\mathcal{L}(\Pi)$.

Two ingredients are conspicuously missing from $\mathcal{L}(\Pi)$. First, $\mathcal{L}(\Pi)$ has no function symbols for the constructors of $\mathfrak{A}$, namely encryption and concatenation. Second, $\mathcal{L}(\Pi)$ has no function which, given a node, would return the message sent or received on that node. We omitted them for two reasons.

$\forall \mathtt{m}, \mathtt{I}, \mathtt{K}, \mathtt{x}, \mathtt{PCA}, \mathtt{MF}, \mathtt{SRK}\,.$

$$\begin{array}{ll}
\mathtt{if}\ \mathtt{Store1}(\mathtt{m}, \mathtt{I}, \mathtt{K}, \mathtt{x}, \mathtt{PCA}, \mathtt{SRK}) & (\Phi_1)\\
\wedge\ \mathtt{Non}(\mathtt{skMF}) \wedge \mathtt{Non}(\mathtt{skPCA}) \wedge \mathtt{Non}(\mathtt{SRK}) & (\Phi_2)\\
\wedge\ \mathtt{Unq}(\mathtt{K}) \wedge \mathtt{Unq}(\mathtt{inv}(\mathtt{K})) & (\Phi_3)\\
\mathtt{then}\ \exists \mathtt{n}_1, \mathtt{n}_2, \mathtt{EK}\,. & \\
\quad \mathtt{Pca2}(\mathtt{n}_1, \mathtt{I}, \mathtt{K}, \mathtt{x}, \mathtt{PCA}, \mathtt{EK}) \wedge \mathtt{Tpm4}(\mathtt{n}_2, \mathtt{I}, \mathtt{K}, \mathtt{x}, \mathtt{PCA}, \mathtt{EK}, \mathtt{SRK}). &
\end{array}$$

$\forall \mathtt{m}, \mathtt{n}, \mathtt{I}, \mathtt{K}, \mathtt{x}, \mathtt{PCA}, \mathtt{MF}, \mathtt{SRK}\,.\ \mathtt{if}\ \Phi_1 \wedge \Phi_2 \wedge \Phi_3 \wedge \mathtt{Lsn1}(\mathtt{n}, \mathtt{inv}(K))\ \mathtt{then}\ \mathtt{false}.$

Fig. 3. Authentication and confidentiality goals in $\mathcal{L}(\Pi)$

First, the security goals we want to express need not be explicit about the forms of the messages sent and received. They need only refer to the underlying parameters. The definition of the protocol determines uniformly what the participants send and receive, as a function of these parameters. Moreover, assertions about compound messages embedded within parameters would provide artificial ways to construct counterexamples to our protocol independence theorem.

Second, $\mathcal{L}(\Pi)$ should be insensitive to the notational specifics of the protocol Π, describing the goals of the protocol without prejudicing the message syntax.

However, to reason axiomatically about protocols, we would work within an expanded language $\mathcal{L}'(\Pi)$ with message constructors for encryption and concatenation, and with a function to extract the message sent or received on a node. Goals would still be expressed in the sublanguage $\mathcal{L}(\Pi_1)$.

What Is a Security Goal? A security goal is either an authentication or a confidentiality property. An authentication goal requires a peer to have executed some behavior. A confidentiality goal requires some desired secret t not be shared as a parameter of another strand. Usually, this is a listener strand $\mathsf{Lsn}[t]$, so the goal ensures that t can never be transmitted unencrypted, in plaintext.[4]

Definition 6

1. *A* security claim *is a conjunction of atomic formulas of $\mathcal{L}(\Pi)$.*
2. *Suppose that G_0 is $\Phi \supset \exists \mathtt{v}_0 \ldots \mathtt{v}_\mathtt{j}\,.\,(\Psi_1 \vee \ldots \vee \Psi_\mathtt{k})$, where Φ and each $\Psi_\mathtt{i}$ is a security claim. Suppose that, for every variable $\mathtt{n}$ over nodes occurring free in G_0, some conjunct of Φ is a role predicate $\mathtt{RhoJ}(\mathtt{n}, \mathtt{u}, \ldots, \mathtt{w})$. Then the universal closure G of G_0 is a* security goal *of Π.*
3. *G is an authentication goal if $k > 0$ and a confidentiality goal if $k = 0$.* □

We identify the empty disjunction $\bigvee_{i\in\emptyset} \Psi_\mathtt{i}$ with $\mathtt{false}$. We identify the unit conjunction $\bigwedge_{i\in\{1\}} \Phi_\mathtt{i}$ with its sole conjunct $\Phi_\mathtt{i}$, and $\bigvee_{i\in\{1\}} \Phi_\mathtt{i}$ with $\Phi_\mathtt{i}$.

As examples, we formalize the authentication and confidentiality goals of Section 1 as two separate goals in Fig. 3. The authentication goal has a unit disjunction, i.e. Ψ_1 is everything inside the existential quantifier, and the confidentiality goal uses the vacuous disjunction $\mathtt{false}$, where $k = 0$.

[4] We consider only "full disclosure" goals, rather than "partial information" goals, in which a party learns that some values of t are possible, but not others. On the relation between full disclosure goals and partial information goals, see e.g. [3,7].

Semantics. The semantics for $\mathcal{L}(\Pi)$ are classical, with each structure a skeleton for the protocol Π. This requirement builds the permissible behaviors of Π directly into the semantics without requiring an explicit axiomatization.

Definition 7. *Let $\mathbb{A}$ be a skeleton for Π. An* assignment *σ for $\mathbb{A}$ is a function from variables of $\mathcal{L}(\Pi_1)$ to $\mathfrak{A} \cup \mathsf{nodes}(\Pi)$. Extend σ to terms of $\mathcal{L}(\Pi)$ via the rules: $\sigma(\mathtt{sk(t)}) = \mathsf{sk}(\sigma(\mathtt{t}))$, $\sigma(\mathtt{inv(t)}) = (\sigma(\mathtt{t}))^{-1}$.*

Satisfaction *$\mathbb{A}, \sigma \models \Phi$ is defined via the standard Tarski inductive clauses for the classical first order logical constants, and the base clauses:*

$\mathbb{A}, \sigma \models \mathtt{u} = \mathtt{v}$	*iff*	$\sigma(\mathtt{u}) = \sigma(\mathtt{v})$;
$\mathbb{A}, \sigma \models \mathtt{Non(v)}$	*iff*	$\sigma(\mathtt{v}) \in \mathsf{non}_{\mathbb{A}}$;
$\mathbb{A}, \sigma \models \mathtt{Unq(v)}$	*iff*	$\sigma(\mathtt{v}) \in \mathsf{unique}_{\mathbb{A}}$;
$\mathbb{A}, \sigma \models \mathtt{UnqAt(m, v)}$	*iff*	$\sigma(\mathtt{m}) \in \mathsf{nodes}(\mathbb{A})$, *and* $\sigma(\mathtt{v}) \in \mathsf{unique}_{\mathbb{A}}$, *and* $\sigma(\mathtt{v})$ *originates at node* $\sigma(\mathtt{m})$;
$\mathbb{A}, \sigma \models \mathtt{DblArrw(m, n)}$	*iff*	$\sigma(\mathtt{m}), \sigma(\mathtt{n}) \in \mathsf{nodes}(\mathbb{A})$, *and* $\sigma(\mathtt{m}) \Rightarrow \sigma(\mathtt{n})$;
$\mathbb{A}, \sigma \models \mathtt{Prec(m, n)}$	*iff*	$\sigma(\mathtt{m}) \prec_{\mathbb{A}} \sigma(\mathtt{n})$;

and, for each role $\rho \in \Pi$ and index j on ρ, the predicate $\mathtt{RhoJ}(\mathtt{m}, \mathtt{v_1}, \ldots, \mathtt{v_k})$ obeys the clause

$\mathbb{A}, \sigma \models \mathtt{RhoJ}(\mathtt{m}, \mathtt{v_1}, \ldots, \mathtt{v_k})$ *iff* $\sigma(\mathtt{m}) \in \mathsf{nodes}(\mathbb{A})$, *and* $\sigma(\mathtt{m})$ *is an instance of the* j^{th} *node on role* ρ, *with the parameters* $\sigma(\mathtt{v_1}), \ldots, \sigma(\mathtt{v_k})$.

We write $\mathbb{A} \models \Phi$ when $\mathbb{A}, \sigma \models \Phi$ for all σ. □

When $\mathtt{n}$ is a variable over nodes, although $\sigma(\mathtt{n}) \notin \mathsf{nodes}(\mathbb{A})$ is permitted, in that case, whenever $\phi(\mathtt{n})$ is an atomic formula, $\mathbb{A}, \sigma \not\models \phi(\mathtt{n})$.

In protocols where there are two different roles ρ, ρ' that differ only after their first j nodes—typically, because they represent different choices at a branch point after the j^{th} node [16,14]—the two predicates $\mathtt{RhoJ}$ and $\mathtt{Rho'J}$ are equivalent.

Lemma 1. *If ϕ is an atomic formula and $\mathbb{A}, \sigma \models \phi$, then $\alpha(\mathbb{A}), \alpha \circ \sigma \models \phi$.*

If α is injective, and if ϕ is an atomic formula other than a role predicate $\mathtt{RhoJ}$*, and if $\alpha(\mathbb{A}), \alpha \circ \sigma \models \phi$, then $\mathbb{A}, \sigma \models \phi$.* □

3 Multiprotocols, Disjointness, and Authentication Tests

Given a primary protocol Π_1, as well as a protocol Π which includes it, we have written Π in the form $\Pi_1 \cup \Pi_2$, but this is imprecise. We distinguish the nodes of Π_1 from nodes of Π that do not belong to Π_1.

Definition 8

1. *(Π, Π_1) is a* multiprotocol *if Π, Π_1 are protocols, and every role of Π_1 is a role of Π.*
2. *Role node n_j is* primary *if it is an instance of a node of Π_1 (Def. 2). Role node n_2 is* secondary *if it is an instance of a node of Π, but it is not primary.*

3. *Instances of encryptions e_1 R-related to role nodes of Π_1 are in $E^R(\Pi_1)$:*

$$E^R(\Pi_1) = \{\alpha(e_1) \colon \exists n_1 \,.\, R(e_1, \mathsf{msg}(n_1)) \wedge n_1 \text{ is a role node of } \Pi_1\}. \qquad \square$$

Below, we use the cases $\sqsubseteq$ and $\ll$ for R, i.e. $E^{\sqsubseteq}(\Pi_1)$ and $E^{\ll}(\Pi_1)$.

$E^{\ll}(\Pi_1) \neq \{e \colon \exists n_1, \alpha \,.\, e \ll \mathsf{msg}(\alpha(n_1)) \wedge n_1$ is a role node of $\Pi_1\}$, since the latter contains *all* encryptions, whenever any role of Π_1 uses an indeterminate (blank slots). $E^{\ll}(\Pi_1)$ requires that an encryption is *syntactically present* in a roles of Π_1, not instantiated from an indeterminate. The more naïve generalization of [17] would be useless for protocols with indeterminates. Refining the definition was easy, but proving it correct required a new method.

The secondary nodes of (Π, Π_1) do not form a protocol. Π_1 contains the listener role, so listener nodes are primary, not secondary. However, $(\Pi_1 \cup \Pi_2, \Pi_1)$ is a multiprotocol. Its secondary nodes are some of the instances of role nodes of Π_2, namely, those that are not also instances of role nodes of Π_1.

Strong Disjointness. To ensure that a Π does not interfere with the goals of Π_1, we control how the secondary nodes transform encryptions. To create an encryption is one way to transform it, or another way is to extract some ingredient—such as a smaller encryption or a nonce or key—from inside it.

Definition 9

1. *If any $e \in E^{\ll}(\Pi_1)$ originates on a secondary transmission node n_2, then n_2 is an* encryption creation conflict.
2. *A secondary transmission node n is an* extraction conflict *if $t_1 \dagger^S \mathsf{msg}(n)$ for some $S \subseteq E^{\sqsubseteq}(\Pi_1)$ where $t_1 \sqsubseteq e \in S$, and:*

$$(\exists m \,.\; m \Rightarrow^+ n \;\wedge\; e \sqsubseteq \mathsf{msg}(m)) \quad \wedge \quad (\forall m \,.\; m \Rightarrow^+ n \;\supset\; t_1 \odot^S \mathsf{msg}(m)).$$

3. *Π, Π_1 has* strongly disjoint encryption *(s.d.e.) iff it has neither encryption creation conflicts nor extraction conflicts.*
4. *Π_1 and Π_2 are* symmetrically disjoint *if, letting $\Pi = \Pi_1 \cup \Pi_2$, both Π, Π_1 and Π, Π_2 have s.d.e.* $\square$

Creation conflicts and extraction conflicts are the two ways that Π could create new ways to solve authentication tests already present in Π_1. Thus, s.d.e. ensures that only Π_1 solutions are needed for the tests in a Π_1 skeleton.

Strong disjointness is a syntactic property, even though its definition talks about all strands of the protocols Π and Π_1. We can check it using unification, as, in Figs. 1–2, we can quickly observe that the check-cashing protocol never creates an ekc, aic, or keyrec, and never extracts an aic from an encryption. Protocol analysis tools such as CPSA [11] can be programmed to check for it.

4 Protocol Independence

For any goal $G_1 \in \mathcal{L}(\Pi_1)$, we want to squeeze a Π_1-counterexample $\mathbb{A}_1$ out of a Π-counterexample $\mathbb{B}$. We do this in two steps: First, we restrict $\mathbb{B}$ to its primary nodes $\mathbb{B} \restriction \Pi_1$. Then, we remove all non-primary encryptions $e_2 \notin E^{\ll}(\Pi_1)$ from $\mathbb{B} \restriction \Pi_1$, by replacing them with indeterminates. In the reverse direction, this is a homomorphism. I.e., there is a $\mathbb{A}_1$ and a homomorphism α such that no secondary encryptions e_2 appear in $\mathbb{A}_1$, and $\mathbb{B} \restriction \Pi_1 = \alpha(\mathbb{A}_1)$. We call this second step "removal."

Definition 10. *Let $\{e_i\}_{i \in I}$ be the indexed family of all secondary encryptions appearing in a Π-skeleton $\mathbb{B}$, without repetitions, and let $\{x_i\}_{i \in I}$ be an indexed family of indeterminates without repetitions, none of which appear in $\mathbb{B}$.*

The homomorphism α that maps $x_i \mapsto e_i$, and is the identity for all atoms and all indeterminates not in $\{x_i\}_{i \in I}$ is a removal *for $\mathbb{B}$.* $\square$

For a removal α, there are $\mathbb{A}$s with $\mathbb{B} = \alpha(\mathbb{A})$. To compute a canonical one, for each $n \in \mathsf{nodes}(\mathbb{B})$, we walk the tree of $\mathsf{msg}(n)$ from the top. We copy structure until we reach a subtree equal to any e_i, when we insert x_i instead. The resulting $\mathbb{A}$ is the *result* of the removal α for $\mathbb{B}$. The *result* of α for a node $m \in \mathbb{B}$ means the $n \in \mathsf{nodes}(\mathbb{A})$ such that $\alpha(n) = m$.

Lemma 2. *Let Π, Π_1 be a multiprotocol, with $G_1 \in \mathcal{L}(\Pi_1)$ a Π_1 security goal.*

1. *If $\mathbb{B} \models \neg G_1$, then $\mathbb{B} \restriction \Pi_1 \models \neg G_1$.*
2. *Let α be a removal for $\mathbb{B}$ with result $\mathbb{A}$.*
   (a) *If $\alpha(n)$ is a primary node, then n is a primary node for the same role.*
   (b) *If ϕ is a role predicate and $\mathbb{B}, \alpha \circ \sigma \models \phi$, then $\mathbb{A}, \sigma \models \phi$.*
   (c) *If $\mathbb{B} \models \neg G_1$, then $\mathbb{A} \models \neg G_1$.*

The definition of strong disjointness (Def. 9) implies, using Thm. 1:

Lemma 3. *Let Π, Π_1 have s.d.e., and let α be a removal for $\mathbb{B} \restriction \Pi_1$ with result $\mathbb{A}$. If $\mathbb{B}$ is a realized Π-skeleton, then $\mathbb{A}$ is a realized Π_1-skeleton.*

The essential idea here is to show that a solution to a primary cut lies on a primary node, which will be preserved in the restriction, and then again preserved by the removal. From the two preceding results, we obtain our main theorem:

Theorem 2. *Let Π, Π_1 have s.d.e., and let $G_1 \in \mathcal{L}(\Pi_1)$ be a security goal. If $\mathbb{A} \models \neg G_1$ and is realized, then for some realized Π_1-skeleton $\mathbb{A}_1$, $\mathbb{A}_1 \models \neg G_1$.*

Thm. 2 implies, for instance, that the check-cashing protocol of Fig. 2 preserves the goals of MAIP.

Conclusion. Our result Thm. 2 uses a new, model-theoretic approach. It combines reasoning about the logical form of formulas—the security goals G—with operations on the structures that furnish models of these formulas. These operations are restriction, homomorphisms, and removals. The authentication tests suggest the definition of strong disjointness (Def. 9).

Thm. 2 simplifies establishing that two protocols combine to achieve their goals. Goals of the joint protocol Π expressed in $\mathcal{L}(\Pi_1)$ may be verified without reference to $\Pi \setminus \Pi_1$. Second, our composition result can also be read as a prescription—or a heuristic—for protocol design. Protocols can be built from subprotocols that provide some of the intermediate cryptographic values that they require. Thm. 2 gives the constraints that a protocol designer must adhere to, in enriching an existing suite of protocols. His new operations must be strongly disjoint from the existing protocols, regarded as a primary protocol.

Related Work. An outstanding group of articles by Datta, Derek, Mitchell, and Pavlovic, including [9], concern protocol derivation and composition. The authors explore a variety of protocols with common ingredients, showing how they form a sort of family tree, related by a number of operations on protocols.

Our definition of multiprotocol covers both [9]'s *parallel composition* and its *sequential composition. Refinement* enriches the message structure of a protocol. *Transformation* moves information between protocol messages, either to reduce the number of messages or to provide a tighter binding among parameters.

Despite their rich palette of operations, their main results are restricted to parallel and sequential composition [9, Thms. 4.4, 4.8]. Each result applies to particular proofs of particular security goals G_1. Each proof relies on a set Γ of invariant formulas that Π_1 preserves. If a secondary protocol Π_2 respects Γ, then G_1 holds of the parallel composition $\Pi_1 \cup \Pi_2$ (Thm. 4.4). Thm 4.8, on sequential composition, is more elaborate but comparable. By contrast, our Thm. 2 is one uniform assertion about all security goals, independent of their proofs. It ensures that Π_2 will respect all usable invariants of Π_1. This syntactic property, checked once, suffices permanently, without looking for invariants to re-establish.

Universal composability [6] is a related property, although expressed in a very different, very strong, underlying model. It is often implemented by randomly choosing a tag to insert in all messages of a protocol, this tag being chosen at session set-up time. Thus, the symmetric disjointness of any two sessions, whether of the same or of different protocols, holds with overwhelming probability.

Andova, et al. [1] study sequential and parallel composition, using tags or distinct keys as implementation strategies, as in our [17]. They independently propose a definition [1, Def. 25] like the symmetric definition of [8].

Related but narrower problems arise from *type-flaw attacks*, situations in which a participant may parse a message incorrectly, and therefore process it in inappropriate ways [19]. Type flaw attacks concern a single protocol, although a protocol that may be viewed with different degrees of explicit tagging.

Future Work. Is there a theorem like Thm. 2 for the refinement and transformation operations [9]? For specific, limited formulations, the answer should be affirmative, and the model-theoretic approach is promising for establishing that answer. Such a result would provide a strong guide for protocol design.

Acknowledgments. I am grateful to Stéphanie Delaune and to my colleagues Leonard G. Monk, John D. Ramsdell, and F. Javier Thayer. An extremely perceptive anonymous referee report from CSF provoked a radical reworking. MITRE release number: 07-0029.

References

1. Andova, S., Cremers, C.J.F., Gjøsteen, K., Mauw, S., Mjølsnes, S.F., Radomirović, S.: Sufficient conditions for composing security protocols. Information and Computation (2007)
2. Backes, M., Maffei, M., Unruh, D.: Zero-knowledge in the applied pi-calculus and automated verification of the Direct Anonymous Attestation protocol. In: IEEE Symposium on Security and Privacy (2008)
3. Backes, M., Pfitzmann, B.: Relating cryptographic and symbolic key secrecy. In: Proceedings of 26th IEEE Symposium on Security and Privacy (May 2005)
4. Balacheff, B., Chen, L., Pearson, S., Plaquin, D., Proudler, G.: Trusted Computing Platforms: TCPA Technology in Context. Prentice Hall PTR, NJ (2003)
5. Brickell, E., Camenisch, J., Chen, L.: Direct anonymous attestation. In: ACM Conference on Communications and Computer Security (CCS) (2004)
6. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS, IACR 2000/067 (October 2001)
7. Canetti, R., Herzog, J.: Universally composable symbolic analysis of mutual authentication and key-exchange protocols. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 380–403. Springer, Heidelberg (2006)
8. Cortier, V., Delaitre, J., Delaune, S.: Safely composing security protocols. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 352–363. Springer, Heidelberg (2007)
9. Datta, A., Derek, A., Mitchell, J.C., Pavlovic, D.: A derivation system and compositional logic for security protocols. Journal of Computer Security 13(3), 423–482 (2005)
10. Doghmi, S.F., Guttman, J.D., Thayer, F.J.: Completeness of the authentication tests. In: Biskup, J., López, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 106–121. Springer, Heidelberg (2007)
11. Doghmi, S.F., Guttman, J.D., Thayer, F.J.: Searching for shapes in cryptographic protocols. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 523–537. Springer, Heidelberg (2007), Extended version, http://eprint.iacr.org/2006/435
12. Dolev, D., Yao, A.: On the security of public-key protocols. IEEE Transactions on Information Theory 29, 198–208 (1983)
13. Durgin, N., Lincoln, P., Mitchell, J., Scedrov, A.: Multiset rewriting and the complexity of bounded security protocols. Journal of Computer Security 12(2), 247–311 (2004)
14. Fröschle, S.: Adding branching to the strand space model. In: Proceedings of EXPRESS 2008. Electronic Notes in Theoretical Computer Science. Elsevier, Amsterdam (2008)
15. Goguen, J.A., Meseguer, J.: Order-sorted algebra I. Theoretical Computer Science 105(2), 217–273 (1992)
16. Guttman, J.D., Herzog, J.C., Ramsdell, J.D., Sniffen, B.T.: Programming cryptographic protocols. In: De Nicola, R., Sangiorgi, D. (eds.) TGC 2005. LNCS, vol. 3705, pp. 116–145. Springer, Heidelberg (2005)
17. Guttman, J.D., Thayer, F.J.: Protocol independence through disjoint encryption. In: Proceedings of 13th Computer Security Foundations Workshop. IEEE Computer Society Press, Los Alamitos (2000)
18. Guttman, J.D., Thayer, F.J.: Authentication tests and the structure of bundles. Theoretical Computer Science 283(2), 333–380 (2002)
19. Heather, J., Lowe, G., Schneider, S.: How to prevent type flaw attacks on security protocols. In: Proceedings of 13th Computer Security Foundations Workshop. IEEE Computer Society Press, Los Alamitos (2000)

Bisimulation for Demonic Schedulers*

Konstantinos Chatzikokolakis[1], Gethin Norman[2], and David Parker[2]

[1] Eindhoven University of Technology
[2] Oxford Computing Laboratory

Abstract. Bisimulation between processes has been proven a successful method for formalizing security properties. We argue that in certain cases, a scheduler that has full information on the process and collaborates with the attacker can allow him to distinguish two processes even though they are bisimilar. This phenomenon is related to the issue that bisimilarity is not preserved by refinement. As a solution, we introduce a finer variant of bisimulation in which processes are required to simulate each other under the "same" scheduler. We formalize this notion in a variant of CCS with explicit schedulers and show that this new bisimilarity can be characterized by a refinement-preserving traditional bisimilarity. Using a third characterization of this equivalence, we show how to verify it for finite systems. We then apply the new equivalence to anonymity and show that it implies strong probabilistic anonymity, while the traditional bisimulation does not. Finally, to illustrate the usefulness of our approach, we perform a compositional analysis of the Dining Cryptographers with a non-deterministic order of announcements and for an arbitrary number of cryptographers.

1 Introduction

Process algebra provides natural models for security protocols in which non-determinism plays an essential role, allowing implementation details to be abstracted ([1,2,3]). In this setting, security properties are often stated in using equivalence relations, with bisimulation commonly used. Its application takes two distinct forms. In the first, the protocol is shown to be bisimilar to a specification which satisfies the required security property, then since the protocol and specification are equivalent, we conclude that the protocol satisfies the property. An example is the formalization of *authenticity* in [2], in which A sends a message m and B wants to ensure that it receives m and not a different message from some other agent. In the specification, we allow B to test the received message against the real m (as if B knew it beforehand). Showing the protocol is bisimilar to the specification ensures B receives the correct message.

The second form is substantially different: we establish a bisimulation relation between two distinct instances of the protocol. From this, we conclude that the instances are *indistinguishable* to the attacker, that is the difference between the two instances remains hidden from the attacker. An example of this approach is the formalization of

* This work was carried out while Konstantinos Chatzikokolakis was visiting Oxford University. Chatzikokolakis wishes to thank Marta Kwiatkowska for giving him the opportunity to collaborate with her group. Authors Norman and Parker where supported in part by EPSRC grants EP/D077273/1 and EP/D07956X/2.

secrecy in [2]. If $P(m)$ is a protocol parametrized by a message m, and we demonstrate that $P(m)$ and $P(m')$ are bisimilar, then the message remains secret. Another example is *privacy* in voting protocols ([4]). The votes of A and B remain private if an instance of the protocol is bisimilar to the instance in which A and B have exchanged votes.

In this paper we focus on the second use of bisimulation and we argue that, in the presence of a scheduler who has full view of the process, an attacker could actually distinguish bisimilar processes. The reason is that, in the definition of bisimulation, non-determinism is treated in a partially *angelic* way. Letting $\sim$ denote bisimilarity, when $P{\sim}Q$ one requires that if P can make a transition α to P', then Q can make a transition α to some Q' such that $Q{\sim}Q'$ (and vice-versa). In this definition, there are two implicit quantifiers, the second being *existential*:

$$\begin{array}{rl} \textit{for all}\text{ transitions} & P \xrightarrow{\alpha} P' \\ \textit{there exists}\text{ a transition} & Q \xrightarrow{\alpha} Q' \qquad \text{s.t. } P' \sim Q' \end{array}$$

In other words, Q is not forced to simulate P, it only has to have the possibility to do so. For P and Q to remain indistinguishable in the actual execution, we have to count on the fact that the scheduler of Q will guide it in a way that simulates P, that is the scheduler acts in favour of the process. In security, however, we want to consider the worst case scenario, thus we typically assume that the scheduler collaborates with the attacker. A "demonic" scheduler in the implementation of Q can choose to do something different, allowing the attacker to distinguish the two processes.

Consider the simple example of an agent broadcasting a message m on a network, received by agents A and B and then acknowledged (with each agent including its identity in the acknowledgement). We can model this protocol as:

$$P(m) = (\nu c)(\bar{c}\langle m\rangle.\bar{c}\langle m\rangle \mid A \mid B) \quad \text{where} \;\; A = c(x).a \text{ and } B = c(x).b$$

Clearly, $P(m) \sim P(m')$, but does m remain secret? Both instances can perform two visible actions, a and b, the order of which is chosen non-deterministically. The indistinguishability of the two processes relies on the schedulers of $P(m)$ and $P(m')$ choosing the same order for the visible actions. If, however, one scheduler chooses a different order, then we can distinguish m from m' through on the output of the protocol. This could be the case, for example, if an operation is performed upon reception whose execution time is message dependent.

This consequence of angelic non-determinism can be also formulated in terms of *refinement*, where Q refines P if it contains "less" non-determinism. While an implementation of a protocol is a refinement of it, bisimulation need not be preserved by refinement, thus security properties might no longer hold in the implementation, an issue often called the "refinement paradox" ([5,6]). In the example above, $P(m)$ and $P(m')$ could be implemented by $(\nu c)(\bar{c}\langle m\rangle.\bar{c}\langle m\rangle \mid c(x).a.c(x).b)$ and $(\nu c)(\bar{c}\langle m'\rangle.\bar{c}\langle m'\rangle \mid c(x).b.c(x).a)$ respectively which can be distinguished. Note that, in the specification-based use of bisimulation this issue does not appear: as the protocol and specification are bisimilar, the implementation will be a refinement of the specification which is usually enough to guarantee the required security property.

It should be noted that classical bisimulation does offer some control over non-determinism as it is closed under contexts and contexts can restrict available actions.

More precisely, bisimulation is robust against schedulers that can be expressed through contexts. However, contexts cannot control internal choices, like the selection of the first receiver in the above example. The example could be rewritten to make this selection external, however, for more complex protocols this solution becomes challenging. Also, when using contexts, it is not possible to give some information to the scheduler, without making the corresponding action visible, that is, without revealing it to an observer. This could be problematic when, for example, verifying a protocol in which the scheduler, even if he knows some secret information, has no possibility to communicate it to the outside.

In this paper we propose an approach based on a variant of bisimulation, called *demonic bisimulation*, in which non-determinism is treated in a purely demonic way. In principle, we would like to turn the existential quantifier into a universal one, but this is too restrictive and the relation would not even be reflexive. Instead we require Q to simulate a transition α of P, not under *any* scheduler but under the *same* scheduler that produced α:

$$\begin{array}{rll} \textit{for all schedulers } S\textit{, if} & P \xrightarrow{\alpha} P' & \\ \textit{then under the same scheduler } S\textit{:} & Q \xrightarrow{\alpha} Q' & \quad \text{with } P' \sim Q' \end{array}$$

Note that, in general, it is not possible to speak about the "same" scheduler, since different processes have different choices. Still, this is reasonable if the processes have a similar structure (as in the case of $P(m)$ and $P(m')$), in this paper we give a framework that allows us to formalize this concept. The basic idea is that we can choose the scheduler that can break our property, however we must test both processes under the same one. This requirement is both realistic, as we are interested in the indistinguishability of two processes when put in the same environment, and strong, since it leaves no angelic non-determinism.

To formalize demonic bisimulation we use a variant of probabilistic CCS with explicit schedulers, which was introduced in [7] to study the information that a scheduler has about the state of a process. This calculus allows us to speak of schedulers independently from processes, leading to a natural definition of demonic bisimulation. Then, we discuss how we can view a scheduler as a refinement operator that restricts the non-determinism of a process. We define a refinement operator, based on schedulers, and we show that demonic bisimilarity can be characterized as a refinement-preserving classical bisimilarity, for this type of refinement. Afterwards, we give a third characterization of demonic bisimilarity, that allows us to obtain an algorithm to verify finite processes. Finally, we apply the demonic bisimulation to the analysis of anonymity protocols and show that demonic bisimulation, in contrast to the classical one, implies strong probabilistic anonymity. This enables us to perform a compositional analysis of the Dining Cryptographers protocol demonstrating that it satisfies anonymity for an arbitrary number of cryptographers.

2 Preliminaries

We denote by $Disc(X)$ the set of all discrete probability measures over X, and by $\delta(x)$ (called the *Dirac measure* on x) the probability measure that assigns probability 1 to

$\{x\}$. We will also denote by $\sum_i [p_i]\mu_i$ the probability measure obtained as a convex sum of the measures μ_i.

A *simple probabilistic automaton* is a tuple $(S, q, A, \mathcal{D})$ where S is a set of states, $q \in S$ is the *initial state*, A is a set of actions and $\mathcal{D} \subseteq S \times A \times Disc(S)$ is a *transition relation*. Intuitively, if $(s, a, \mu) \in \mathcal{D}$, also written $s \xrightarrow{a} \mu$, then there is a transition from the state s performing the action a and leading to a distribution μ over the states of the automaton. A probabilistic automaton M is *fully probabilistic* if from each state of M there is at most one transition available. An execution α of a probabilistic automaton is a (possibly infinite) sequence $s_0 a_1 s_1 a_2 s_2 \ldots$ of alternating states and actions, such that $q = s_0$, and for each $i : s_i \xrightarrow{a_{i+1}} \mu_i$ and $\mu_i(s_{i+1}) > 0$. A *scheduler* of a probabilistic automaton $M = (S, q, A, \mathcal{D})$ is a function $\zeta : exec^*(M) \mapsto \mathcal{D}$ where $exec^*(M)$ is the set of finite executions of M, such that $\zeta(\alpha) = (s, a, \mu) \in \mathcal{D}$ implies that s is the last state of α. The idea is that a scheduler selects a transition among the ones available in $\mathcal{D}$ and it can base its decision on the history of the execution. A scheduler induces a probability space on the set of executions of M.

If $\mathcal{R}$ is a relation over a set S, then we can lift the relation to probability distributions over S using the standard weighting function technique (see [8] for details). If $\mathcal{R}$ is an equivalence relation then the lifting can be simplified: $\mu_1 \;\mathcal{R}\; \mu_2$ iff for all equivalence classes $\mathcal{E} \in S/\mathcal{R}$, $\mu_1(\mathcal{E}) = \mu_2(\mathcal{E})$. We can now define simulation and bisimulation for simple probabilistic automata.

Definition 1. *Let $(S, q, A, \mathcal{D})$ be a probabilistic automaton. A relation $\mathcal{R} \subseteq S \times S$ is a* simulation *iff for all $(s_1, s_2) \in \mathcal{R}, a \in A$: if $s_1 \xrightarrow{a} \mu_1$ then there exists μ_2 such that $s_2 \xrightarrow{a} \mu_2$ and $\mu_1 \;\mathcal{R}\; \mu_2$. A simulation $\mathcal{R}$ is a* bisimulation *if it is also symmetric (thus, it is an equivalence relation). We define $\sqsubseteq, \sim$ as the largest simulation and bisimulation on S respectively.*

CCS with Internal Probabilistic Choice. Let a range over a countable set of *channel names* and let α stand for $a, \bar{a}$ or τ. The syntax of CCS$_p$ is:

$$P, Q \;\;::=\;\; a.P \mid P \mid Q \mid P + Q \mid \textstyle\sum_i p_i P_i \mid (\nu a)P \mid !a.P \mid 0$$

The term $\sum_i p_i P_i$ represents an *internal probabilistic choice*, all the remaining operators are from standard CCS. We will also use the notation $P_1 +_p P_2$ to represent a binary sum $\sum_i p_i P_i$ with $p_1 = p$ and $p_2 = 1 - p$. Finally, we use replicated input instead of replication or recursion, as this simplifies the presentation. The semantics of CCS$_p$ is standard and has been omitted due to space constraints. The full semantics can be found in the report version of this paper ([9]). We denote this transition system by $\longrightarrow_c$ to distinguish it from other transition systems defined later in the paper.

3 A Variant of CCS$_p$ with Explicit Scheduler

In this section we present a variant of CCS$_p$ in which the scheduler is explicit, in the sense that it has a specific syntax and its behaviour is defined by the operational semantics of the calculus. This calculus was proposed in [7]; we will refer to it as CCS$_\sigma$. Processes in CCS$_\sigma$ contain labels that allow us to refer to a particular sub-process. A

$I ::= 0\,I \mid 1\,I \mid \epsilon$	**label indexes**	$S, T ::=$	**scheduler**
$L ::= l^I$	**labels**	$L.S$	schedule single action
		$\mid (L, L).S$	synchronization
$P, Q ::=$	**processes**	$\mid$ **if** L	label test
$L{:}\alpha.P$	prefix	**then** S	
$\mid P \mid Q$	parallel	**else** S	
$\mid P + Q$	nondeterm. choice	$\mid 0$	nil
$\mid L{:}\sum_i p_i P_i$	internal prob. choice		
$\mid (\nu a)P$	restriction	$CP ::= P \parallel S$	**complete process**
$\mid\ !L{:}a.P$	replicated input		
$\mid L{:}0$	nil		

Fig. 1. The syntax of CCS$_\sigma$

scheduler also behaves like a process, using however a different and much simpler syntax, and its purpose is to guide the execution of the main process using the labels that the latter provides.

3.1 Syntax

Let a range over a countable set of *channel names* and l over a countable set of *atomic labels*. The syntax of CCS$_\sigma$, shown in Figure 1, is the same as the one of CCS$_p$ except for the presence of labels. These are used to select the subprocess which "performs" a transition. Since only the operators with an initial rule can originate a transition, we only need to assign labels to the prefix and to the probabilistic sum. We use labels of the form l^s where l is an atomic label and the index s is a finite string of 0 and 1, possibly empty. Indexes are used to avoid multiple copies of the same label in case of replication. As explained in the semantics, each time a process is replicated we relabel it using appropriate indexes. To simplify the notation, we use base labels of the form $l_1, \ldots, l_n$, and we write ${}^i a.P$ for $l_i{:}a.P$.

A scheduler selects a sub-process for execution on the basis of its label, so we use $l.S$ to represent a scheduler that selects the process with label l and continues as S. In the case of synchronization we need to select two processes simultaneously, hence we need a scheduler of the form $(l_1, l_2).S$. We will use S_l to denote a scheduler of one of these forms (that is, a scheduler that starts with a label or pair of labels). The **if-then-else** construct allows the scheduler to test whether a label is available in the process (in the top-level) and act accordingly. A complete process is a process put in parallel with a scheduler, for example $l_1{:}a.l_2{:}b \parallel l_1.l_2$. We define $\mathcal{P}, \mathcal{CP}$ to be the sets of all processes and all complete CCS$_\sigma$ processes respectively. Note that for processes with an infinite execution path we need schedulers of infinite length.

3.2 Semantics for Complete Processes

The semantics of CCS$_\sigma$ is given in terms of a probabilistic automaton whose state space is $\mathcal{CP}$ and whose transitions are given by the rules in Figure 2. We denote the transitions by $\longrightarrow_s$ to distinguish it from other transition systems.

ACT is the basic communication rule. In order for $l{:}\alpha.P$ to perform α, the scheduler should select this process for execution, so the scheduler needs to be of the form $l.S$.

$$\text{ACT}\ \frac{}{l{:}\alpha.P \parallel l.S \xrightarrow{\alpha}_s \delta(P \parallel S)} \qquad \text{RES}\ \frac{P \parallel S_l \xrightarrow{\alpha}_s \mu \quad \alpha \neq a, \overline{a}}{(\nu a)P \parallel S_l \xrightarrow{\alpha}_s (\nu a)\mu}$$

$$\text{SUM1}\ \frac{P \parallel S_l \xrightarrow{\alpha}_s \mu}{P + Q \parallel S_l \xrightarrow{\alpha}_s \mu} \qquad \text{PAR1}\ \frac{P \parallel S_l \xrightarrow{\alpha}_s \mu}{P \mid Q \parallel S_l \xrightarrow{\alpha}_s \mu \mid Q}$$

$$\text{COM}\ \frac{P \parallel l_1 \xrightarrow{a}_s \delta(P' \parallel 0) \qquad Q \parallel l_2 \xrightarrow{\overline{a}}_s \delta(Q' \parallel 0)}{P \mid Q \parallel (l_1, l_2).S \xrightarrow{\tau}_s \delta(P' \mid Q' \parallel S)}$$

$$\text{PROB}\ \frac{}{l{:}\sum_i p_i P_i \parallel l.S \xrightarrow{\tau}_s \sum_i p_i \delta(P_i \parallel S)} \qquad \text{REP}\ \frac{}{!l{:}a.P \parallel l.S \xrightarrow{\alpha}_s \delta(\rho_0 P \mid !l{:}a.\rho_1 P \parallel S)}$$

$$\text{IF1}\ \frac{l \in tl(P) \qquad P \parallel S_1 \xrightarrow{\alpha}_s \mu}{P \parallel \textbf{if}\ l\ \textbf{then}\ S_1\ \textbf{else}\ S_2 \xrightarrow{\alpha}_s \mu} \qquad \text{IF2}\ \frac{l \notin tl(P) \qquad P \parallel S_2 \xrightarrow{\alpha}_s \mu}{P \parallel \textbf{if}\ l\ \textbf{then}\ S_1\ \textbf{else}\ S_2 \xrightarrow{\alpha}_s \mu}$$

Fig. 2. The semantics of complete CCS$_\sigma$ processes. SUM1 and PAR1 have corresponding right rules SUM2 and PAR2, omitted for simplicity.

After this execution the complete process will continue as $P \parallel S$. The RES rule models restriction on channel a: communication on this channel is not allowed by the restricted process. We denote by $(\nu a)\mu$ the measure μ' such that $\mu'((\nu a)P \parallel S) = \mu(P \parallel S)$ for all processes P and $\mu'(R \parallel S) = 0$ if R is not of the form $(\nu a)P$. SUM1 models nondeterministic choice. If $P \parallel S$ can perform a transition to μ, which means that S selects one of the labels of P, then $P + Q \parallel S$ will perform the same transition, i.e. the branch P of the choice will be selected and Q will be discarded. For example $l_1{:}a.P + l_2{:}b.Q \parallel l_1.S \xrightarrow{a}_s \delta(P \parallel S)$. Note that the operands of the sum do not have labels, the labels belong to the subprocesses of P and Q. In the case of nested choices, the scheduler must select the label of a prefix, thus resolving all the choices at once.

PAR1, modelling parallel composition, is similar: the scheduler selects P to perform a transition on the basis of the label. The difference is that in this case Q is not discarded; it remains in the continuation. $\mu \mid Q$ denotes the measure μ' such that $\mu'(P \mid Q \parallel S) = \mu(P \parallel S)$. COM models synchronization. If $P \parallel l_1$ can perform the action a and $Q \parallel l_2$ can perform $\bar{a}$, then $(l_1, l_2).S$ can synchronize the two by scheduling both l_1 and l_2 at the same time. PROB models internal probabilistic choice. Note that the scheduler cannot affect the outcome of the choice, it can only schedule the choice as a whole (this is why a probabilistic sum has a label) and the process will move to a measure containing all the operands with corresponding probabilities.

REP models replicated input. This rule is the same as in CCS, with the addition of a re-labeling operator ρ_i. The reason for this is that we want to avoid ending up with multiple copies of the same label as the result of replication, since this would create ambiguities in scheduling as explained in Section 3.3. $\rho_i P$ appends $i \in \{0, 1\}$ to the index of all labels of P, for example: $\rho_i l^s{:}\alpha.P = l^{si}{:}\alpha.\rho_i P$ and similarly for the other operators. Note that we relabel only the resulting process, not the continuation of the scheduler: there is no need for relabeling the scheduler since we are free to choose the continuation as we please.

Finally **if-then-else** allows the scheduler to adjust its behaviour based on the labels that are available in P. $tl(P)$ gives the set of top-level labels of P and is defined as: $tl(l : \alpha.P) = tl(l : \sum_i p_i P_i) = tl(!l : a.P) = tl(l : 0) = \{l\}$ and as the union of the top-level labels of all sub-processes for the other operators. Then **if** l **then** S_1 **else** S_2 behaves like S_1 if l is available in P and as S_2 otherwise.

A process is blocked if it cannot perform a transition under any scheduler. A scheduler S is *non-blocking* for a process P if it always schedules some transition, except when P itself is blocked.

3.3 Deterministic Labelings

The idea in CCS$_\sigma$ is that a *syntactic* scheduler will be able to completely resolve the nondeterminism of the process, without needing to rely on a *semantic* scheduler at the level of the automaton. To achieve this we impose a condition on the labels of CCS$_\sigma$ processes. A *labeling* for P is an assignment of labels to the subprocesses of P that require a label. A labeling for P is *deterministic* iff for all schedulers S there is at most one transition of $P \parallel S$ enabled at any time, in other words the corresponding automaton is fully probabilistic. In the rest of the paper, we only consider processes with deterministic labelings.

A simple case of deterministic labelings are the *linear* ones, containing pairwise distinct labels (a more precise definition of linear labelings requires an extra condition and can be found in [10]). It can be shown that linear labelings are preserved by transitions and are deterministic. However, the interesting case is that we can construct labelings that are deterministic without being linear. The usefulness of non-linear labelings is that they limit the power of the scheduler, since the labels provide information about the current state and allow the scheduler to choose different strategies through the use of **if-then-else**. Consider, for example, the following process whose labeling is deterministic but not linear:

$$l{:}({}^1\bar{a}.R_1 +_p {}^1\bar{a}.R_2) \mid {}^2a.P \mid {}^3a.Q \tag{1}$$

Since both branches of the probabilistic sum have the same label l_1, the scheduler cannot resolve the choice between P and Q based on the outcome of the probabilistic choice. Another use of non-linear labeling is the encoding of “private” value passing ([7]):

$$l{:}c(x).P \;\triangleq\; \textstyle\sum_i l{:}cv_i.P[v_i/x] \qquad\qquad l{:}\bar{c}\langle v\rangle.P \;\triangleq\; l{:}\overline{cv}.P$$

This is the usual encoding of value passing in CCS except that we use the same label in all the branches of the nondeterministic sum. Thus, the reception of a message is visible to the scheduler, but not the received value.

4 Demonic Bisimulation

As discussed in the introduction, classical bisimulation treats non-determinism in a partially angelic way. In this section we define a strict variant of bisimulation, called demonic bisimulation, which treats non-determinism in a purely demonic way. We characterize this equivalence in two ways, first in terms of schedulers, then in terms of refinement.

4.1 Definition Using Schedulers

An informal definition of demonic bisimulation was already given in the introduction: P is demonic-bisimilar to Q, written $P \sim_D Q$ if *for all schedulers* S, if $P \xrightarrow{\alpha} P'$ then *under the same scheduler* S: $Q \xrightarrow{\alpha} Q'$ with $P' \sim_D Q$. To define demonic bisimulation concretely, we need a framework that allows a single scheduler to be used with different processes. CCS$_\sigma$ does exactly this: it gives semantics to $P \parallel S$ for any process P and scheduler S (of course, S might be blocking for some processes and non-blocking for others).

If μ is a discrete measure on $\mathcal{P}$, we denote by $\mu \parallel S$ the discrete measure μ' on $\mathcal{CP}$ such that $\mu'(P \parallel S) = \mu(P)$ for all $P \in \mathcal{P}$ and $\mu'(P \parallel S') = 0$ for all $S' \neq S$ (note that all transition rules of Fig. 2 produce measures of this form).

Definition 2 (Demonic bisimulation). *An equivalence relation $\mathcal{R}$ on $\mathcal{P}$ is a* demonic bisimulation *iff for all $(P_1, P_2) \in \mathcal{R}, a \in A$ and all schedulers S: if S is non-blocking for P_1 and $P_1 \parallel S \xrightarrow{\alpha} \mu_1 \parallel S'$ then the same scheduler S is non-blocking for P_2 and $P_2 \parallel S \xrightarrow{\alpha} \mu_2 \parallel S'$ with $\mu_1 \ \mathcal{R} \ \mu_2$. We define demonic bisimlarity $\sim_D$ as the largest demonic bisimulation on $\mathcal{P}$.*

Consider again the example of the introduction. We define:

$$A = {}^1c(x).{}^2a \qquad B = {}^3c(x).{}^4b \qquad P(m) = (\nu c)({}^5\overline{c}\langle m\rangle.{}^6\overline{c}\langle m\rangle \mid A \mid B)$$

Note that $P(m), P(m')$ share the same labels. This choice of labels states that whenever a scheduler chooses an action in $P(m)$, it has to schedule the same action in $P(m')$. Then it is easy to see that $P(m) \sim_D P(m')$. A scheduler that selects A first in $P(m)$ will also select A first in $P(m')$, leading to the same order of actions. Under this definition, we do not rely on angelic non-determinism for $P(m')$ to simulate $P(m)$, we have constructed our model in a way that forces a scheduler to perform the same action in both processes. Note that we could also choose to put different labels in $\overline{c}\langle m\rangle, \overline{c}\langle m'\rangle$, hence allowing them to be scheduled in a different way. In this case $\sim_D$ will no longer hold, exactly because we can now distinguish the two processes using an **if-then-else** scheduler that depends on the message. Finally we perform a usual sanity check:

Proposition 1. *$\sim_D$ is a congruence.*

4.2 Characterization Using Refinement

Another way of looking at schedulers is in terms of refinement: a process Q refines P if it contains "less" non-determinism. A typical definition is in terms of simulation: Q refines P if $Q \sqsubseteq P$. For example, a is a refinement of $a+b$ where the non-deterministic choice has been resolved. Thus, a scheduler can be seen as a way to refine a process by resolving the non-determinism. For example, $l_1 : a$ can be seen as the refinement of $l_1 : a + l_2 : b$ under the scheduler l_1. Moreover, partial schedulers can be considered as resolving only part of the non-determinism. For example, for the process ${}^1a.({}^3c + {}^4d) + {}^2b$, the scheduler l_1 will resolve the first non-deterministic choice but not the second.

It has been observed that many security properties are not preserved by refinement, a phenomenon often called the "refinement paradox". If we define security properties

$$\varphi_0(P) = P \tag{2}$$

$$\varphi_{\lambda.S}(\lambda{:}\alpha.P) = \lambda{:}\alpha.\varphi_S(P) \tag{3}$$

$$\varphi_{\lambda.S}(P+Q) = \varphi_{\lambda.S}(P) + \varphi_{\lambda.S}(Q) \tag{4}$$

$$\varphi_{\lambda.S}((\nu a)P) = (\nu a)\varphi_{\lambda.S}(P) \tag{5}$$

$$\varphi_{l.S}(l{:}\textstyle\sum_i p_i P_i) = l{:}\sum_i p_i \varphi_S(P_i) \tag{6}$$

$$\varphi_{\lambda.S}(P \mid Q) = \begin{cases} \lambda{:}\alpha.\varphi_S(P' \mid Q) & \text{if } \varphi_\lambda(P) \xrightarrow{\alpha}_c \delta(P') \\ \lambda{:}\sum_i p_i \varphi_S(P_i \mid Q) & \text{if } \varphi_\lambda(P) \xrightarrow{\tau}_c \sum_i [p_i]\delta(P_i) \\ \lambda{:}\tau.\varphi_S(P' \mid Q') & \text{if } \lambda = (l_1, l_2) \text{ and} \\ & \varphi_{l_1}(P) \xrightarrow{a}_c \delta(P'), \varphi_{l_2}(Q) \xrightarrow{\overline{a}}_c \delta(Q') \end{cases} \tag{7}$$

$$\varphi_{l.S}(!l{:}a.P) = l{:}a.\varphi_S(\rho_0 P \mid !l{:}\ a.\rho_1 P) \tag{8}$$

$$\varphi_S(P) = \begin{cases} \varphi_{S_1}(P) & \text{if } l \in tl(P) \text{ where } S = \textbf{if } l \textbf{ then } S_1 \textbf{ else } S_2 \\ \varphi_{S_2}(P) & \text{if } l \notin tl(P) \end{cases} \tag{9}$$

$$\varphi_S(P) = 0 \quad \text{if none of the above is applicable (e.g. } \varphi_{l_1}(l_2{:}\alpha.P) = 0) \tag{10}$$

Fig. 3. Refinement of CCS$_\sigma$ processes under a scheduler. The symmetric cases for the parallel operator have been omitted for simplicity.

using bisimulation this issue immediately arises. For example, $a|b$ is bisimilar to $a.b + b.a$ but if we refine them to $a.b$ and $b.a$ respectively, they are no longer bisimilar. Clearly, if we want to preserve bisimilarity we have to refine both processes in a consistent way. In this section, we introduce a refinement operator based on schedulers. We are then interested in processes that are not only bisimilar, but also preserve bisimilarity under this refinement. We show that this stronger equivalence coincides with demonic bisimilarity.

With a slight abuse of notation we extend the transitions $\longrightarrow_c$ (the traditional transition system for CCS$_p$) to CCS$_\sigma$ processes, by simply ignoring the labels, which are then only used for the refinement. Let S be a finite scheduler and P a CCS$_\sigma$ process. The refinement of P under S, denoted by $\varphi_S(P)$, is a new CCS$_\sigma$ process. The function $\varphi_S : \mathcal{P} \to \mathcal{P}$ is defined in Figure 3. Note that φ_S does not perform transitions, it only blocks the transitions that are not enabled by S. Thus, it reduces the non-determinism of the process. The scheduler might be partial: a scheduler 0 leaves the process unaffected (2). Thus, the resulting process might still have non-deterministic choices. A prefix is allowed in the refined process only if its label is selected by the scheduler (3), otherwise the refined process is equal to 0 (10). Case (4) applies the refinement to both operands. Note that, if the labeling is deterministic, at most one of the two will have transitions enabled. The most interesting case is the parallel operator (7). There are three possible executions for $P \mid Q$. An execution of P alone, of Q alone or a synchronization between the two. The refined version enforces the one selected by the scheduler (the symmetric cases have been omitted for simplicity). This is achieved by explicitly prefixing the selected action, for example $l_1{:}a \mid l_2{:}b$ refined by l_1 becomes $l_1{:}a.(0 \mid l_2{:}b)$. If P performs a probabilistic choice, then we have to use a probabilistic sum instead of an action prefix. The case of $!P$ (8) is similar to the prefix (3) and the rest of the cases are self-explanatory.

$$\text{ACT}\ \frac{}{l{:}\alpha.P \xrightarrow{l:\alpha}_a \delta(P)} \qquad \text{RES}\ \frac{P \xrightarrow{l:\alpha}_a \mu \quad \alpha \neq a, \overline{a}}{(\nu a)P \xrightarrow{l:\alpha}_a (\nu a)\mu}$$

$$\text{SUM1}\ \frac{P \xrightarrow{l:\alpha}_a \mu}{P + Q \xrightarrow{l:\alpha}_a \mu} \qquad \text{PAR1}\ \frac{P \xrightarrow{l:\alpha}_a \mu}{P \mid Q \xrightarrow{l:\alpha}_a \mu \mid Q}$$

$$\text{COM}\ \frac{P \xrightarrow{l_1:a}_a \delta(P') \quad Q \xrightarrow{l_2:\overline{a}}_a \delta(Q')}{P \mid Q \xrightarrow{(l_1,l_2):\tau}_a \delta(P' \mid Q')} \qquad \text{PROB}\ \frac{}{l{:}\sum_i p_i P_i \xrightarrow{l:\tau}_a \sum_i p_i \delta(P_i)}$$

$$\text{REP}\ \frac{}{!l{:}a.P \xrightarrow{l:a}_a \delta(\rho_0 P \mid !l{:}a.\rho_1 P)}$$

Fig. 4. Semantics for CCS$_\sigma$ processes without schedulers. SUM1 and PAR1 have corresponding right rules SUM2 and PAR2, omitted for simplicity.

The intention behind the definition of ϕ_S is to refine CCS$_\sigma$ processes: $\phi_S(P)$ contains only the choices of P that are selected by the scheduler S. We now show that the result is indeed a refinement:

Proposition 2. *For all CCS$_\sigma$ processes P and schedulers S: $\phi_S(P) \sqsubseteq P$*

Note that $\sqsubseteq$ is the simulation relation on $\mathcal{P}$ wrt the classical CCS semantics $\longrightarrow_c$. Also, let $\sim$ be the bisimilarity relation on $\mathcal{P}$ wrt $\longrightarrow_c$. A nice property of this type of refinement is that it allows one to refine two processes in a consistent way. This enables us to define a refinement-preserving bisimulation.

Definition 3. *An equivalence relation $\mathcal{R}$ on $\mathcal{P}$ is an* R-bisimulation *iff for all $(P_1, P_2) \in \mathcal{R}$ and all finite schedulers S: $\varphi_S(P_1) \sim \varphi_S(P_2)$. We denote by $\sim_R$ the largest R-bisimulation.*

Note that $P_1 \sim_R P_2$ implies $P_1 \sim P_2$ (for $S = 0$). We now show that processes that preserve bisimilarity under this type of refinement are exactly the ones that are demonic-bisimilar.

Theorem 1. *The equivalence relations $\sim_R$ and $\sim_D$ coincide.*

5 Verifying Demonic Bisimilarity for Finite Processes

The two characterizations of demonic bisimilarity, given in the previous section, have the drawback of quantifying over all schedulers. This makes the verification of the equivalence difficult, even for finite state processes. To overcome this difficulty, we give a third characterization, this one based on traditional bisimilarity on a modified transition system where labels annotate the performed actions. We then use this characterization to adapt an algorithm for verifying probabilistic bisimilarity to our settings.

5.1 Characterization Using a Modified Transition System

In this section we give a modified semantics for CCS$_\sigma$ processes without schedulers. The semantics are given by means of a simple probabilistic automaton with state space

$\mathcal{P}$, displayed in Figure 4 and denoted by $\longrightarrow_a$. The difference is that now the labels annotate the actions instead of being used by the scheduler. Thus, we have actions of the form $\lambda : \alpha$ where λ is l or (l_1, l_2), and α is a channel, an output on a channel or τ. Note that, in the case of synchronization (COM), we combine the labels l_1, l_2 of the actions $a, \overline{a}$ and we annotate the resulting τ action by (l_1, l_2). All rules match the corresponding transitions for complete processes. Since no schedulers are involved here, the rules IF1 and IF2 are completely removed.

We can now characterize demonic bisimilarity using this transition system.

Definition 4. *An equivalence relation $\mathcal{R}$ on $\mathcal{P}$ is an* A-bisimulation *iff*

i) it is a bisimulation wrt $\longrightarrow_a$, and
ii) $tl(P_1) = tl(P_2)$ for all non-blocked $P_1, P_2 \in \mathcal{R}$

We define $\sim_A$ as the largest A-bisimulation on $\mathcal{P}$.

Theorem 2. *The equivalence relations $\sim_D$ and $\sim_A$ coincide.*

Essentially, we have encoded the schedulers in the actions of the transition system $\longrightarrow_a$. Thus, if two processes perform the same action in $\longrightarrow_a$ it means that they perform the same action with the same scheduler in $\longrightarrow_s$. Note that the relation $\sim_A$ is stricter that the classical bisimilarity. This is needed because schedulers have the power to check the top-level labels of a process, even if this label is not "active", that is it does not correspond to a transition. We could modify the semantics of the **if-then-else** operator, in order to use the traditional bisimilarity in the above theorem. However, this would make the schedulers less expressive. Indeed, it can be shown ([7]) that for any semantic scheduler (that is, one defined on the automaton) of a CCS$_p$ process P, we can create a syntactic scheduler that has the same behaviour on P labeled with a linear labeling. This property, though, is lost under the modified **if-then-else**.

5.2 An Algorithm for Finite State Processes

We can now use $\sim_A$ to verify demonic bisimilarity for finite state processes. For this, we adapt the algorithm of Baier ([11]) for probabilistic bisimilarity. The adaptation is straightforward and has been omitted due to space constraints. The only interesting part is to take into account the additional requirement of $\sim_A$ that related non-blocked processes should have the same set of top-level labels. This can be done in a pre-processing step where we partition the states based on their top-level labels. More details can be found in the report version of this paper ([9]). The algorithm has been implemented and used to verify some of the results of the following section.

6 An Application to Security

In this section, we apply the demonic bisimulation to the verification of anonymity protocols. First, we formalize anonymity in terms of equivalence between different instances of the protocol. We then show that this definition implies strong probabilistic anonymity, which was defined in [12] in terms of traces. This allows us to perform an easier analysis of protocols by exploiting the algebraic properties of an equivalence. We perform such a compositional analysis on the Dining Cryptographers protocol with non-deterministic order of announcements.

6.1 Probabilistic Anonymity

Consider a protocol in which a set $\mathcal{A}$ of *anonymous events* can occur. An event $a_i \in \mathcal{A}$ could mean, for example, that user i performed an action of interest. In each execution, the protocol produces an observable event $o \in \mathcal{O}$. The goal of the attacker is to deduce a_i from o. Strong anonymity was defined in [12], here we use this definition in a somewhat informal way, a more formal treatment is available in the report version of the paper ([9]).

Definition 5 (strong anonymity). *A protocol is strongly anonymous iff for all schedulers, for all $a_i, a_j \in \mathcal{A}$ and all $o \in \mathcal{O}$, the probability of producing o when a_i is selected is equal to the probability of producing o when a_j is selected.*

Let $Prot_i$ be the CCS$_\sigma$ process modelling the instance of the protocol when a_i occurs. Typically, the selection of anonymous event is performed in the beginning of the protocol (for example a user i decides to send a message) and then the protocol proceeds as $Prot_i$. Thus, the complete protocol is modelled by $Prot \triangleq l : \sum_i p_i \ Prot_i$. The observable events correspond to the traces of $Prot$. We can now give a definition of strong anonymity based on demonic bisimulation:

Definition 6 (equivalence based anonymity). *A protocol satisfies anonymity iff for all anonymous events $a_i, a_j \in \mathcal{A} : Prot_i \sim_D Prot_j$.*

The idea behind this definition is that, if $Prot_i, Prot_j$ are demonic-bisimilar, they should behave in the same way under all schedulers, thus producing the same observation. Indeed, we can show that the above definition implies Def. 5.

Proposition 3. *If $Prot_i \sim_D Prot_j$ for all i, j then the protocol satisfies strong probabilistic anonymity (Def. 5)*

It is worth noting that, on the other hand, $Prot_i \sim Prot_j$ does not imply Def. 5, as we see in the next section.

6.2 Analysis of the Dining Cryptographers Protocol

The problem of the Dining Cryptographers is the following: Three cryptographers dine together. After the dinner, the bill has to be paid by either one of them or by another agent called the master. The master decides who will pay and then informs each of them separately whether he has to pay or not. The cryptographers would like to find out whether the payer is the master or one of them. However, in the latter case, they wish to keep the payer anonymous.

The Dining Cryptographers Protocol (DCP) solves the above problem as follows: each cryptographer tosses a fair coin which is visible to himself and his neighbour to the right. Each cryptographer checks the two adjacent coins and, if he is not paying, announces *agree* if they are the same and *disagree* otherwise. However, the paying cryptographer says the opposite. It can be proved that the master is paying if and only if the number of *disagrees* is even ([13]).

$$CryptP_i \triangleq {}^{1,i}c_i(coin_1).{}^{2,i}c_i(coin_2).{}^{3,i}\overline{out}_i\langle coin_1 \otimes coin_2\rangle$$

$$Crypt_i \triangleq {}^{1,i}c_i(coin_1).{}^{2,i}c_i,(coin_2).{}^{3,i}\overline{out}_i\langle coin_1 \otimes coin_2 \otimes 1\rangle$$

$$Coin_i \triangleq l_{4,i}{:}(({}^{5,i}\bar{c}_i\langle 0\rangle \mid {}^{6,i}\bar{c}_{i\oplus 1}\langle 0\rangle) +_{0.5} ({}^{5,i}\bar{c}_i\langle 1\rangle \mid {}^{6,i}\bar{c}_{i\oplus 1}\langle 1\rangle))$$

$$Prot_i \triangleq (\nu \boldsymbol{c})(CryptP_i \mid \textstyle\prod_{j\neq i} Crypt_j \mid \prod_{j=0}^{n-1} Coin_j)$$

Fig. 5. Encoding of the dining cryptographers protocol

We model the protocol, for the general case of a ring of n cryptographers, as shown in Figure 5. The symbols $\oplus, \otimes$ represent the addition modulo n and modulo 2 (xor) respectively. $Crypt_i, CryptP_i$ model the cryptographer i acting as non-payer or payer respectively. $Coin_i$ models the i-th coin, shared between cryptographers i and $i \oplus 1$. Finally, $Prot_i$ is the instance of the protocol when cryptographer i is the payer, and consists of $CryptP_i$, all other cryptographers as non-payers, and all coins. An external observer can only see the announcements $\overline{out}_i\langle\cdot\rangle$. As discussed in [12], DCP satisfies anonymity if we abstract from their order. If their order is observable, on the contrary, a scheduler can reveal the identity of the payer to the observer by forcing the payer to make his announcement first, or by selecting the order based on the value of the coins.

In CCS$_\sigma$ we can be precise about the information that is revealed to the scheduler. In the encoding of Fig. 5, we have used the same labels on both sides of the probabilistic choice in $Coin_i$. As a consequence, after performing the choice, the scheduler cannot use an **if-then-else** to find out which was the outcome, so his decision will be independent of the coin's value. Similarly, the use of private value passing (see Section 3.3) guarantees that the scheduler will not see which value is transmitted by the coin to the cryptographers. Then we can show that for any number of cryptographers:

$$Prot_i \sim_D Prot_j \quad \forall 1 \leq i, j \leq n \tag{11}$$

For a fixed number of cryptographers, (11) can be verified automatically using the algorithm of Section (5.2). We have used a prototype implementation to verify demonic bisimilarity for a very small number of cryptographers (after that, the state space becomes too big). However, using the algebraic properties of $\sim_D$ we can perform a compositional analysis and prove (11) for any number of cryptographers. This approach is described in the report version of the paper.

This protocol offers a good example of the difference between classical and demonic bisimulation. Wrt the $\longrightarrow_s$ transition system, $Prot_i, Prot_j$ are both bisimilar and demonic-bisimilar, and strong anonymity holds. Now let $Coin_i'$ be the same as $Coin_i$ but with different labels on the left-hand and right-hand side, meaning that now a scheduler can depend its behaviour on the value of the coin. The resulting $Prot_i', Prot_j'$ processes are no longer demonic-bisimilar and strong-anonymity is violated. However, classic bisimulation still holds, showing that it fails to capture the desired security property.

7 Related Work

Various works in the area of probabilistic automata introduce restrictions to the scheduler to avoid violating security properties ([14,15,16]). Their approach is based on dividing the actions of each component of the system in equivalence classes (*tasks*). The order of execution of different tasks is decided in advance by a so-called *task scheduler*. The remaining nondeterminism within a task is resolved by a second demonic schedule. In our approach, the order of execution is still decided non-deterministically by a demonic scheduler, but we impose that the scheduler will make the same decision in both processes.

Refinement operators that preserve various security properties are given in [17,18]. In our approach, we impose that the refinement operator should preserve bisimilarity, obtaining a stronger equivalence.

In the probabilistic setting, a bisimulation that quantifies over all schedulers is used in [19]. In this work, however, the scheduler only selects the action and the remaining nondeterminism is resolved probabilistically (using a uniform distribution). This avoids the problem of angelic non-determinism but weakens the power of the scheduler.

On the other hand, [20] gives an equivalence-based definition of anonymity for the Dining Cryptographers, but in a possibilistic setting. In this case the scheduler is clearly angelic, since anonymity relies on a non-deterministic selection of the coins. Our definition is the probabilistic counterpart of this work, which was problematic due to the angelic use of non-determinism.

8 Conclusion and Future Work

We introduced a notion of bisimulation where processes are required to simulate each other under the same scheduler. We characterized this equivalence in three different ways: using syntactic schedulers, using a refinement operator based on schedulers and using a modified transition system where labels annotate the actions. We applied this notion to anonymity showing that strong anonymity can be defined in terms of equivalence, leading to a compositional analysis of the dining cryptographers with non-deterministic order of announcements.

As future work, we want to investigate the effect of angelic non-determinism to other process equivalences. Many of them are defined based on the general schema: when P does an action of interest (passes a test, produces a barb, etc) then Q should be able to match it, employing an existential quantifier. Moreover, we would like to investigate models in which both angelic and demonic non-determinism are present. One approach would be to use two separate schedulers, one acting in favour and one against the process, along the lines of [21].

References

1. Roscoe, A.W.: Modelling and verifying key-exchange protocols using CSP and FDR. In: Proc. CSFW, pp. 98–107. IEEE Computer Soc. Press, Los Alamitos (1995)
2. Abadi, M., Gordon, A.D.: A calculus for cryptographic protocols: The spi calculus. Information and Computation 148, 1–70 (1999)

3. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: Proceedings of POPL 2001, pp. 104–115. ACM, New York (2001)
4. Kremer, S., Ryan, M.D.: Analysis of an electronic voting protocol in the applied pi calculus. In: Sagiv, M. (ed.) ESOP 2005. LNCS, vol. 3444, pp. 186–200. Springer, Heidelberg (2005)
5. McLean: A general theory of composition for a class of "possibilistic" properties. IEEETSE: IEEE Transactions on Software Engineering 22 (1996)
6. Roscoe, B.: CSP and determinism in security modelling. In: Proc. of 1995 IEEE Symposium on Security and Privacy. IEEE Computer Society Press, Los Alamitos (1995)
7. Chatzikokolakis, K., Palamidessi, C.: Making random choices invisible to the scheduler. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 42–58. Springer, Heidelberg (2007)
8. Segala, R.: Modeling and Verification of Randomized Distributed Real-Time Systems. PhD thesis, MIT (1995)
9. Chatzikokolakis, K., Norman, G., Parker, D.: Bisimulation for demonic schedulers. Technical report (2009), `http://www.win.tue.nl/~kostas/`
10. Chatzikokolakis, K.: Probabilistic and Information-Theoretic Approaches to Anonymity. PhD thesis, Ecole Polytechnique, Paris (2007)
11. Baier, C.: Polynomial-time algorithms for testing probabilistic bisimulation and simulation. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 50–61. Springer, Heidelberg (1996)
12. Bhargava, M., Palamidessi, C.: Probabilistic anonymity. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 171–185. Springer, Heidelberg (2005)
13. Chaum, D.: The dining cryptographers problem: Unconditional sender and recipient untraceability. Journal of Cryptology 1, 65–75 (1988)
14. Canetti, R., Cheung, L., Kaynar, D., Liskov, M., Lynch, N., Pereira, O., Segala, R.: Task-structured probabilistic i/o automata. In: Proceedings the 8th International Workshop on Discrete Event Systems (WODES 2006), Ann Arbor, Michigan (2006)
15. Canetti, R., Cheung, L., Kaynar, D.K., Liskov, M., Lynch, N.A., Pereira, O., Segala, R.: Time-bounded task-PIOAs: A framework for analyzing security protocols. In: Dolev, S. (ed.) DISC 2006. LNCS, vol. 4167, pp. 238–253. Springer, Heidelberg (2006)
16. Garcia, F.D., van Rossum, P., Sokolova, A.: Probabilistic anonymity and admissible schedulers, arXiv:0706.1019v1 (2007)
17. Jürjens, J.: Secrecy-preserving refinement. In: Oliveira, J.N., Zave, P. (eds.) FME 2001. LNCS, vol. 2021, p. 135. Springer, Heidelberg (2001)
18. Mantel, H.: Possibilistic definitions of security - an assembly kit. In: CSFW, pp. 185–199 (2000)
19. Lincoln, P., Mitchell, J., Mitchell, M., Scedrov, A.: A probabilistic poly-time framework for protocol analysis. In: Proceedings of the 5th ACM Conference on Computer and Communications Security, pp. 112–121. ACM Press, New York (1998)
20. Schneider, S., Sidiropoulos, A.: CSP and anonymity. In: Martella, G., Kurth, H., Montolivo, E., Bertino, E. (eds.) ESORICS 1996. LNCS, vol. 1146, pp. 198–218. Springer, Heidelberg (1996)
21. Chatzikokolakis, K., Knight, S., Panangaden, P.: Epistemic strategies and games on concurrent processes. In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910. Springer, Heidelberg (2008)

On Omega-Languages Defined by Mean-Payoff Conditions

Rajeev Alur[1], Aldric Degorre[2], Oded Maler[2], and Gera Weiss[1]

[1] Dept. of Computer and Information Science, University of Pennsylvania, USA
{alur,gera}@cis.upenn.edu
[2] CNRS - Verimag, University of Grenoble, France
{aldric.degorre,oded.maler}@imag.fr

Abstract. In *quantitative* verification, system states/transitions have associated payoffs, and these are used to associate *mean-payoffs* with infinite behaviors. In this paper, we propose to define ω-languages via Boolean queries over mean-payoffs. Requirements concerning averages such as "the number of messages lost is negligible" are not ω-regular, but specifiable in our framework. We show that, for closure under intersection, one needs to consider multi-dimensional payoffs. We argue that the acceptance condition needs to examine the set of *accumulation points* of sequences of mean-payoffs of prefixes, and give a precise characterization of such sets. We propose the class of *multi-threshold mean-payoff languages* using acceptance conditions that are Boolean combinations of inequalities comparing the minimal or maximal accumulation point along some coordinate with a constant threshold. For this class of languages, we study expressiveness, closure properties, analyzability, and Borel complexity.

1 Introduction

In algorithmic verification of reactive systems, the system is modeled as a finite-state transition system (possibly with fairness constraints), and requirements are captured as languages of infinite words over system observations [8,9]. The most commonly used framework for requirements is the class of ω-regular languages. This class is expressive enough to capture many natural requirements, and has well-understood and appealing theoretical properties: it is closed under Boolean operations, it is definable by finite automata (such as deterministic parity automata or nondeterministic Büchi automata), it contains Linear Temporal Logic LTL, and decision problems such as emptiness, language inclusion are decidable [10,11].

The classical verification framework only captures *qualitative* aspects of system behavior, and in order to describe *quantitative* aspects, for example, consumption of resources such as CPU and energy, a variety of extensions of system models, logics, and automata have been proposed and studied in recent years [1,3,5,7]. The best known, and the most relevant to our work, approach is as follows: a *payoff* (or a cost) is associated with each state (or transition) of the model, the *mean-payoff* of a finite run is simply the average of the payoffs of all the states in the run, and the mean-payoff of an infinite run is the limit, as n goes to infinity, of the mean-payoff of the prefix of length n. The notion of mean-payoff objectives was first studied in classical game theory, and more recently in verification literature [5, 6, 12]. Most of this work is focused on computing the optimal mean-payoff value, typically in the setting of two-player games, and the fascinating connections between the mean-payoff and parity games.

In this paper, we propose and study ways of defining languages of infinite words based on the mean-payoffs. As a motivating example, suppose 1 denotes the condition that "message is delivered" and 0 denotes the condition that "message is lost." A behavior of the network is an infinite sequence over $\{0, 1\}$. Requirements such as "no message is ever lost" (always 1), "only finitely many messages are lost" (eventually-always 1), and "infinitely many messages are delivered" (infinitely-often 1), are all ω-regular languages. However, the natural requirement that "the number of lost messages is negligible" is not ω-regular. Such a requirement can be formally captured if we can refer to averages. For this purpose, we can associate a payoff with each symbol, payoff 0 with message lost and payoff 1 with message delivered, and require that mean-payoff of every infinite behavior is 1. As this example indicates, using mean-payoffs to define acceptance conditions can express meaningful, non-regular, and yet analyzable, requirements.

The central technical question for this paper is to define a precise query language for mapping mean-payoffs of infinite runs into Boolean answers so that the resulting class of ω-languages has desirable properties concerning closure, expressiveness, and analyzability. The obvious candidate for turning mean-payoffs into acceptance criteria is threshold queries of the form "is mean-payoff above (or below) a given threshold t". Indeed, this is implicitly the choice in the existing literature on decision problems related to mean-payoff models [5, 6, 12]. A closer investigation indicates that this is not a satisfactory choice for queries.

First, closure under intersection requires that we should be able to model multiple payoff functions. For this purpose, we define *d-payoff automata*, where d is the dimension of the payoffs, and each edge is annotated with a d-dimensional vector of payoffs. We prove that expressiveness strictly increases with the dimension. From the applications point of view, multi-payoffs allow to model requirements involving multiple quantities. Because we allow unbounded dimensions, one can also add coordinates that model weighted sums of the quantities, and put bounds on these coordinates too.

Second, the limit of the mean-payoffs of prefixes of an infinite run may not exist. This leads us to consider the set of *accumulation points* corresponding to a run. For single-dimensional payoffs, the set of these points is an interval. For multi-dimensional payoffs, we are not aware of existing work on understanding the structure of accumulation points. We establish a precise characterization of the structure of accumulation points: a set can be a set of accumulation points of a run of a payoff automaton if and only if it is closed, bounded, and connected.

Third, if we use mp to refer to the mean-payoff of a run, and consider four types of queries of the form $mp < t$, $mp \leq t$, $mp > t$, and $mp \geq t$, where t is a constant, we prove that the resulting four classes of ω-languages have incomparable expressiveness. Consequently acceptance condition needs to support combination of all such queries.

After establishing a number of properties of the accumulation points of multi-dimensional payoff automata, we propose the class of *multi-threshold mean-payoff languages*. For this class, the acceptance condition is a Boolean combination of constraints of the form "is there an accumulation point whose ith projection is less than a given threshold t". We show that the expressive power of this class is incomparable to that of the class of ω-regular languages, that this class is closed under Boolean operations and has decidable emptiness problem. We also study its Borel complexity.

2 Definitions

2.1 Multi-payoff Automata

Multi-payoff automata are defined as automata with labels, called payoffs, attached to transitions. In this paper, payoffs are vectors in a finite dimensional Euclidean space.

Definition 1 (d-Payoff automaton). *A d-payoff automaton, with $d \in \mathbb{N}$, is a tuple $\langle A, Q, q_0, \delta, w\rangle$ where A and Q are finite sets, representing the alphabet and states of the automaton, respectively; $q_0 \in Q$ is an initial state; $\delta \in Q \times A \to Q$ is a total transition function (also considered as a set of transitions $(q, a, \delta(q, a))$) and $w\colon \delta \to \mathbb{R}^d$ is a function that maps each transition to a d-dimensional vector, called payoff.*

Note that we consider only deterministic complete automata.

Definition 2. *The following notions are defined for payoff automata:*

- *A finite* run *of an automaton is a sequence of transitions of the following type: $(q_1, a_1, q_2)(q_2, a_2, q_3)\dots(q_i, a_i, q_{i+1})$. An infinite run is an infinite sequence of transitions such that any prefix is a finite run.*
- *We denote by $\lambda(r)$ the word of the symbols labelling the successive transitions of the run r, i.e. $\lambda((q_1, a_1, q_2)\cdots(q_n, a_n, q_{n+1})) = a_1\cdots a_n$.*
- *A run is* initial *if $q_1 = q_0$.*
- *By* $\mathrm{run}_{\mathcal{A}}(u)$ *we denote the initial run r in $\mathcal{A}$ such that $u = \lambda(r)$*
- *A* cycle *is a run $(q_1, a_1, q_2)(q_2, a_2, q_3)\dots(q_i, a_i, q_{i+1})$ such that $q_1 = q_{i+1}$. A cycle is* simple *if no proper subsequence is a cycle.*
- *For a word or run u, $u\restriction n$ denotes the prefix of length n of u, and $u[n]$ the nth element of u.*
- *The payoff of a finite run r is* $\mathrm{payoff}(r) = \sum_{i=1}^{|r|} w(r[i])$.
- *The mean-payoff of a run r is* $\mathrm{mp}(r) = \mathrm{payoff}(r)/|r|$.
- *A subset of the states of an automaton is* strongly connected *if, for any two elements of that subset, there is a path from one to the other.*
- *A* strongly connected component *(SCC) is a strongly connected subset that is not contained in any other strongly connected subset.*
- *A SCC is* terminal *if it is reachable and there is no path from the SCC to any other SCC.*

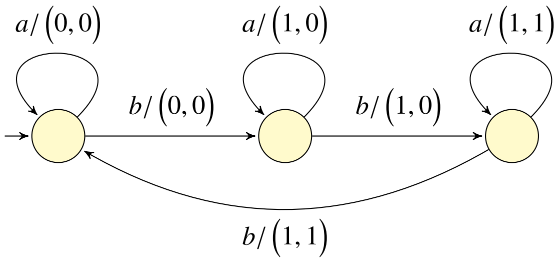
2.2 Acceptance

In the literature, the mean-payoff value of a run is generally associated to the "limit" of the averages of the prefixes of the run. As that limit does not always exist, standard definitions only consider the lim inf of that sequence (or sometimes lim sup) and, more specifically, threshold conditions comparing those quantities with fixed constants [2,4,5,12]. As that choice is arbitrary, and more can be said about the properties of that sequence than the properties of just its lim inf or even lim sup, in particular when $d > 1$, we choose to consider the entire set of accumulation points of that sequence.

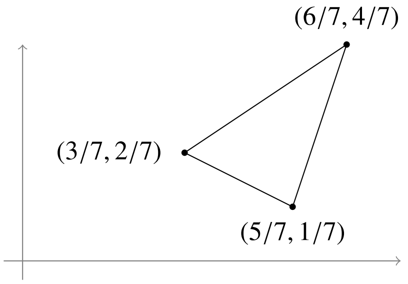
A point x is an accumulation point of the sequence $x_0, x_1, \dots$ if, for every open set containing x, there are infinitely many indices such that the corresponding elements of the sequence belong to the open set.

Definition 3. *We denote by* $\mathrm{Acc}(x_n)_{n=1}^{\infty}$ *the set of accumulation points of the sequence* $(x_n)_{n=1}^{\infty}$. *If r is a run of a d-payoff automaton* $\mathcal{A}$, $\mathrm{Acc}_{\mathcal{A}}(r) = \mathrm{Acc}(\mathrm{mp}(r{\upharpoonright}n))_{n=1}^{\infty}$, *and for a word w,* $\mathrm{Acc}_{\mathcal{A}}(w) = \mathrm{Acc}_{\mathcal{A}}(\mathrm{run}(w))$.

Example 1. Consider the 2-payoff automaton



where edges are annotated with expression of the form σ/v meaning that the symbol σ triggers a transition whose payoff is v. Let $w = \prod_{i=0}^{\infty} a^{2^i-1}b$ be an infinite word where b's are isolated by sequences of a's with exponentially increasing lengths. The set $\mathrm{Acc}_{\mathcal{A}}(w)$ is the triangle

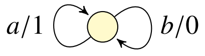


as we show next. By direct calculation we get that $\lim_{n\to\infty} \mathrm{mp}(w{\upharpoonright} \sum_{i=0}^{3n} 2^i) = (6/7, 4/7)$, $\lim_{n\to\infty} \mathrm{mp}(w{\upharpoonright} \sum_{i=0}^{3n+1} 2^i) = (3/7, 2/7)$, and $\lim_{n\to\infty} \mathrm{mp}(w{\upharpoonright} \sum_{i=0}^{3n+2} 2^i) = (5/7, 1/7)$. Furthermore, for every $n \in \mathbb{N}$, $j \in \{0,1,2\}$ and $k \in \{0,\dots,2^{3n+j+1}\}$, the vector $\mathrm{mp}(w{\upharpoonright} k + \sum_{i=0}^{3n+j} 2^i)$ is in the convex hull of $\mathrm{mp}(w{\upharpoonright} \sum_{i=0}^{3n+j} 2^i)$ and $\mathrm{mp}(w{\upharpoonright} \sum_{i=0}^{3n+j+1} 2^i)$ and the maximal distance between points visited on this line goes to zero as n goes to infinity. Together, we get that the points to which the mean-payoff gets arbitrarily close are exactly the points on the boundary of the above triangle. Similarly, if we choose the word $w' = \prod_{i=0}^{\infty} a^{3^i-1}b$, we get that $\mathrm{Acc}_{\mathcal{A}}(w')$ is the boundary of the triangle $(4/13, 3/13), (10/13, 1/13), (12/13, 9/13)$. □

We say that a word or run is *convergent*, whenever its set of accumulation points is a singleton, i.e. when its sequence of mean payoffs converges. For instance, periodic runs are convergent because the mean-payoffs of the prefixes $r\restriction n$ of an infinite run $r = r_1 r_2^\omega$ converge to the mean-payoff of the finite run r_2, when n goes to infinity.

Definition 4. *An infinite run r is accepted by a d-payoff automaton $\mathcal{A}$ with condition F, where F is a predicate on $2^{\mathbb{R}^d}$, if and only if $F(\mathrm{Acc}_{\mathcal{A}}(r))$. An infinite word u is accepted if and only if* $\mathrm{run}(u)$ *is accepted. We denote by $L(\mathcal{A}, F)$ the language of words accepted by $\mathcal{A}$ with condition F. In the following, we call* mean-payoff language*, any language accepted by a d-payoff automaton with such a condition. If d is one and $F(S)$ is of the form* $\mathrm{extr}\, S \bowtie C$ *where* $\mathrm{extr} \in \{\inf, \sup\}$, $\bowtie \in \{<, \leq, >, \geq\}$*, and C is a real constant; we say that F is a threshold condition.*

Example 2. For the 1-payoff automaton



let the acceptance condition $F(S)$ be true iff $S = \{0\}$. This defines the language of words having negligibly many a's. □

3 Expressiveness

3.1 Comparison with ω-Regular Languages

Before proving specific results on the class of mean-payoff languages, we show that it is incomparable with the class of ω-regular languages. In this context, we call specification types incomparable if each type of specification can express properties that are not expressible in the other type. Incomparability of mean-payoff and ω-regular specifications is, of course, a motivation for studying mean-payoff languages.

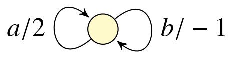
We will need the following ad-hoc pumping lemma for ω-regular languages.

Lemma 1 (Pumping lemma). *Let L be an ω-regular language. There exists $p \in \mathbb{N}$ such that, for each $w = u_1 w_1 u_2 w_2 \ldots u_i w_i \cdots \in L$ such that $|w_i| \geq p$ for all i, there are sequences of finite words $(x_i)_{i\in\mathbb{N}}, (y_i)_{i\in\mathbb{N}}, (z_i)_{i\in\mathbb{N}}$ such that, for all i, $w_i = x_i y_i z_i$, $|x_i y_i| \leq p$ and $|y_i| > 0$ and for every sequence of pumping factors $(j_i)_{i\in\mathbb{N}} \in \mathbb{N}^{\mathbb{N}}$, the pumped word $u_1 x_1 y_1^{j_1} z_1 u_2 x_2 y_2^{j_2} z_2 \ldots u_i x_i y_i^{j_i} z_i \ldots$ is in L.*

Proof. Similar to the proof of the pumping lemma for finite words. □

Proposition 1. *There exists a mean-payoff language, defined by a 1-payoff automaton and a threshold acceptance condition, that is not ω-regular.*

Proof. Consider the 1-payoff automaton



We show that $L = \{w | \inf \mathrm{mp}_{\mathcal{A}}(w) \leq 0\}$ is not regular. For any p, the word $w = (a^p b^{2p})^\omega$ is in that language. Assuming, towards contradiction, that the language is regular and using the pumping Lemma 1 on w, we can select as factors w_i the sequences of a an choose $j_i = 2$ to obtain a word w' that should be in L. But since $\mathrm{mp}_A(w')$ is a singleton bigger than zero, w' does not satisfy the acceptance condition and therefore is not in L, a contradiction. □

Proposition 2. *There exists an ω-regular language that is not a mean-payoff language.*

Proof. Let $L = (a^*b)^\omega$. We will show that, in any payoff automaton, we can find two words u_1 and u_2, u_1 having infinitely often b and u_2 having eventually only a, and such that $Acc(u_1) = Acc(u_2)$. Then obviously no general mean-payoff acceptance condition can distinguish those two words although $u_1 \in L$ and $u_2 \notin L$.

Let us construct the counter-example. Suppose $\mathcal{A}$ is a payoff automaton recognizing L with some predicate F. Let c_a be a cycle such that $\lambda(c_a)$ contains only a's and c_b a cycle such that $\lambda(c_b)$ contains at least one b, both starting in some state q in a terminal strongly connected component of $\mathcal{A}$, and let p be an initial run leading to q.

The mean-payoffs of the run $r = p \prod_{i=1}^{\infty} c_a^i c_b$, which should be accepted, converge to $\mathrm{mp}_{\mathcal{A}}(c_a)$, which is also the mean-payoff of pc_a^ω, which should be rejected but has to be accepted by $\mathcal{A}$, since it has the same mean-payoff as r. □

3.2 Topology of Mean-Payoff Accumulation Points

In this section we discuss the structure of the set of accumulation points. In particular we characterize the sets that are the accumulation points of some run of a payoff automaton.

If S is a strongly connected component of an automaton, and C is the set of simple cycles in S, then we denote by $\mathrm{ConvHull}(S)$ the convex hull of $\{\mathrm{mp}(c) | c \in C\}$.

Theorem 1. *Let r be an infinite run of a d-payoff automaton, then* $\mathrm{Acc}(r)$ *is a closed, connected and bounded subset of* $\mathbb{R}^d$.

Proof.
Closed: True for any set of accumulation points: let (a_n) be a sequence in a topological space, and $(x_n) \in \mathrm{Acc}(a_n)_{n=1}^{\infty}$ be a sequence of accumulation points converging to a point x. For any x_i, we can choose a sub-sequence (a_{i_n}) converging to x_i. Now we can construct a sub-sequence of elements that converges to x: for every i, take the first element $a_{i_{f(i)}}$ of a_{i_n} which is at a distance smaller than 2^{-i} from x_i such that $f(i) > f(i-1)$. Then the sequence $(a_{i_{f(i)}})_{i \in \mathbb{N}}$ converges to x.

Bounded: As we are speaking of a sequence of averages of the (finite) set of payoffs, it is clear that the sequence of mean-payoffs remains in the convex hull of that set, which is bounded.

Connected: Proof by contradiction. Suppose there exists two disjoint open sets O_1 and O_2 such that $Acc(r) \subseteq O_1 \cup O_2$. Let d be the distance between O_1 and O_2. As those sets are open and disjoint, $d > 0$. But the vector between two successive averages is $\mathrm{payoff}(r \upharpoonright n)/n - \mathrm{payoff}(r \upharpoonright n-1)/n-1 = (1/n)(\mathrm{payoff}(r \upharpoonright n-1)) + w(r[n]) - n/(n-1)\,\mathrm{payoff}(r \upharpoonright n-1)) = (1/n)(w(r[n]) - \mathrm{mp}(r \upharpoonright n))$, whose norm is smaller than $\varDelta/n$, where $\varDelta = \max\{\|w(t) - w(t')\| | t, t' \in \delta\}$. If a run has accumulations points in both O_1 and O_2, then there exist $n > \varDelta/d$ such that the nth step is in O_1 and the $(n+1)$th in O_2. The distance between those two points has to be both greater than d and smaller than Δ/n, which is not possible. □

As a remark, we can say more than boundedness: indeed a run eventually comes into a SCC it never leaves. The contribution of the payoffs of the SCC becomes then dominant as n goes to the infinity. Even better, actually, the contribution of the simple cycles of that SCC is dominant. Thus the set of accumulation points is included in the convex hull of the simple cycles of the SCC.

The following theorem is a converse to Theorem 1.

Theorem 2. *For every non-empty, closed, bounded and connected set $D \subset \mathbb{R}^d$, there is a d-payoff automaton and a run r of that automaton such that* $\mathrm{Acc}(r) = D$.

Proof. Take any automaton with a reachable SCC such that D is contained in the convex hull of the cycles of the SCC.

For every integer $n > 0$, let $\{O_{i,n} \colon i = 1, \ldots, l_n\}$ be a finite coverage of D by open sets of diameter smaller than $1/n$. Such a coverage exists, for example, by covering D by spheres of diameter $1/n$.

Suppose p is a finite initial run going into the SCC. For every n and every i, we can prolong p with a suffix c such that $\mathrm{mp}(pc) \in O_{n,i}$ and pc is in the SCC (form the end of p onwards). For that, we need c to be long enough and have the right proportions of simple cycles. Furthermore, as $\mathrm{mp}(pc \upharpoonright l + 1) - \mathrm{mp}(pc \upharpoonright l)$ becomes smaller when l goes to infinity, we can make the distance of $\mathrm{mp}(pc \upharpoonright l)$ from D converge to zero when l goes to infinity.

As the set $(O_{n,i})_{n,i \in \mathbb{N}\times\mathbb{N}}$ is countable, we can construct recursively the successive suffixes $c_{1,1}, c_{1,2}, \ldots, c_{2,1}, c_{2,2}, \ldots$ such that $\mathrm{mp}(pc_{1,1}c_{1,2} \ldots_{2,1} c_{2,2} \ldots c_{n,i})$ is in $O_{n,i}$, and such that for every l, $\mathrm{mp}(p \prod_{ji \in \mathbb{N}\times\mathbb{N}} c_{ji} \upharpoonright l)$ is at a distance smaller than K/l from D.

Let $x \in D$. Then for every n, $x \in O_{n,i}$ for some i, thus for every n, the sequence of mean-payoffs comes within a radius $1/n$ from x, which means x is an accumulation point. Conversely, if $y \notin D$, as D is closed, it is at a distance $\delta > 0$ from D, moreover there exist a l such that $\mathrm{mp}(pc \upharpoonright l)$ never exits a radius $\epsilon < \delta$ around D and therefore the sequence of mean-payoff will never come in a radius $\delta - \epsilon$ from y. So y is not an accumulation point. We conclude that $\mathrm{Acc}_{\mathcal{A}}(r)$ is exactly D. □

Actually, like for Theorem 1, a careful examination of the proof reveals that a stronger statement is true. Specifically, it is easy to verify that any closed, bounded and connected set contained in any of the SCC of an automaton is the set of accumulation points of some run of that automaton.

3.3 Comparison of Threshold Mean-Payoff Languages

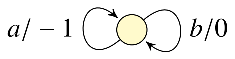
We study mean-payoff languages where the minimum and maximum of the set of accumulation points are compared with a given threshold. We assume, without loss of generality, that the threshold is zero because changing the threshold is equivalent to an affine transformation of the payoffs. We show that the different threshold languages are incomparable in expressive power.

Definition 5. *We denote by $\mathcal{L}_{\bowtie}$ the class of mean-payoff languages accepted by a* 1-*payoff automaton with the condition* $\min \mathrm{Acc}(w) \bowtie 0$*, where* $\bowtie$ *is* $<, >, \leq$ *or* $\geq$.

Note that these languages are the winning conditions used to define mean-payoff games, e.g. in [12], because $\min \mathrm{Acc}(w) = \liminf_{n\to\infty} \mathrm{mp}_{\mathcal{A}}(w \upharpoonright n)$. We do not need to discuss the class of languages defined as complements of these conditions because $\mathcal{L}_{>}$ is $\mathrm{co}\,\mathcal{L}_{\leq}$ and $\mathcal{L}_{\geq}$ is $\mathrm{co}\,\mathcal{L}_{<}$, where $\mathrm{co}\,\mathcal{L}_{\bowtie}$ is the set of languages defined as sets of words that do not satisfy $\min \mathrm{Acc}(w) \bowtie 0$ for some automaton.

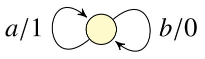
Theorem 3. *The classes $\mathcal{L}_{<}$, $\mathcal{L}_{\leq}$,$\mathcal{L}_{\geq}$ and $\mathcal{L}_{>}$ are incomparable.*

Proof. We begin by showing that $\mathcal{L}_{<}$ and $\mathcal{L}_{\leq}$ are incomparable. Consider the automaton



and the language $L = \{w | \min \mathrm{Acc}(w) < 0\}$. Suppose, towards contradiction, that there exists an automaton $\mathcal{A}'$ accepting L with a $\mathcal{L}_{\leq}$ condition. Consider c_a and c_b two cycles in $\mathcal{A}'$, labelled respectively with a word in $a(a+b)^i$ and with b^j for some integers i and j, and start from a same reachable state q (two such cycles exist at least in any terminal strongly connected component). Let p be a finite initial run ending at q. As pc_a^{ω} is a run of $\mathcal{A}'$ which should be accepted, it is necessary that $\mathrm{payoff}_{\mathcal{A}'}(c_a) \leq 0$, and as pc_b^{ω} should not be accepted, it is necessary that $\mathrm{payoff}_{\mathcal{A}'}(c_b) > 0$. For all k, the run $p(c_a c_b^k)^{\omega}$ should be accepted. So it is necessary that for all k, $\mathrm{payoff}_{\mathcal{A}'}(c_a c_b^k) \leq 0$. Thus $\mathrm{payoff}_{\mathcal{A}'}(c_a) + k\,\mathrm{payoff}_{\mathcal{A}'}(c_b) \leq 0$, which is possible if and only if $\mathrm{payoff}_{\mathcal{A}'}(c_b) \leq 0$ and contradicts $\mathrm{payoff}_{\mathcal{A}'}(c_b) > 0$. Thus L cannot be accepted by a $\mathcal{L}_{\leq}$ condition.

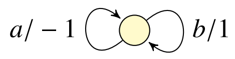
Conversely, consider the automaton



with the language L defined by the $\mathcal{L}_{\leq}$ acceptance condition. Towards contradiction, assume that $\mathcal{A}'$ is an automaton accepting L with the $\mathcal{L}_{<}$ acceptance contradiction. If c_a, c_b and p are defined in $\mathcal{A}'$, the same way as before, we should have $\mathrm{payoff}_{\mathcal{A}'}(c_a) \geq 0$ and $\mathrm{payoff}_{\mathcal{A}'}(c_b) < 0$. For all k, the run $p(c_a c_b^k)$ should be rejected, so it is necessary that $\mathrm{payoff}_{\mathcal{A}'}(c_a c_b^k) \geq 0$. Thus for all k, $\mathrm{payoff}_{\mathcal{A}'}(c_a) + k\,\mathrm{payoff}_{\mathcal{A}'}(c_b) \geq 0$, which is possible if and only if $\mathrm{payoff}_{\mathcal{A}'}(c_b) \geq 0$ and contradicts $\mathrm{payoff}_{\mathcal{A}'}(c_b) < 0$. Therefore L cannot be expressed by a $\mathcal{L}_{<}$ condition.

These counter examples can be adapted for proving that any class with strict inequality symbol is incomparable to any class with a weak inequality symbol. It remains to prove the incompatibility of $\mathcal{L}_{<}$ and $\mathcal{L}_{>}$ and that of $\mathcal{L}_{\leq}$ and $\mathcal{L}_{\geq}$.

Consider the language L defined by the automaton



(denoted by $\mathcal{A}$) with the $\mathcal{L}_{\leq}$ acceptance condition. Suppose, towards contradiction, that $\mathcal{A}'$ is an automaton defining the same language with a $\mathcal{L}_{\geq}$ condition. Choose two cycles c_a and c_b starting from two states q_a and q_b in a same terminal strongly connected component of $\mathcal{A}'$, such that c_a is labelled by a^l and c_b is labelled by b^m for some l and m, an initial run p going to q_b and two runs u_{ab} and u_{ba} going from q_a to q_b and from q_b to q_a, respectively. Because $pu_{ba}c_a^{\omega}$ should be accepted and pc_b^{ω}

should be rejected, we must have $\mathrm{payoff}_{\mathcal{A}'}(c_b) < 0$ and $\mathrm{payoff}_{\mathcal{A}'}(c_a) \geq 0$. Consider $r = p \prod_{i \in \mathbb{N}} c_b^{2^i l} u_{ba} c_a^{2^i m} u_{ab}$. Then $\mathrm{mp}_{\mathcal{A}}(\lambda(p \prod_{i=0}^{n} c_b^{2^i l} u_{ba} c_a^{2^i m} u_{ab}))$ converges to 0 as n goes to infinity, thus $\lambda(r) \in L$, and therefore r should be accepted by $\mathcal{A}'$ with the $\mathcal{L}_{\geq}$ condition. But $\mathrm{mp}_{\mathcal{A}'_3}(word(p \prod_{i=0}^{n} c_b^{2^i l} u_{ba} c_a^{2^i m} u_{ab} c_b^{2^{n+1} l}))$ converges to a negative limit, thus r cannot be accepted by $\mathcal{A}'_3$ with the $\mathcal{L}_{\geq}$ condition, leading to contradiction.

Conversely, consider the language L, defined with the same automaton, but with the $\mathcal{L}_{\geq}$ condition. Suppose $\mathcal{A}'$ is an automaton defining the same L with the $\mathcal{L}_{\leq}$ condition. We can choose two cycles c_a and c_b starting from two states q_a and q_b in a same terminal strongly connected component of $\mathcal{A}'$ such that c_a is labelled by a^l and c_b is labelled by b^m for some l and m, an initial run p going to q_b and two runs u_{ab} and u_{ba} going from q_a to q_b and from q_b to q_a. Then we should have $\mathrm{payoff}_{\mathcal{A}'}(c_b) \leq 0$ and $\mathrm{payoff}_{\mathcal{A}'}(c_a) > 0$ (because $p u_{ba} c_a^{\omega}$ should be rejected and $p c_b^{\omega}$ should be accepted). Consider $r = p \prod_{i \in \mathbb{N}} u_{ba} c_a^{2^i m} u_{ab} c_b^{2^i l}$. Then $\mathrm{mp}_{\mathcal{A}_3}(\lambda(p(\prod_{i=0}^{n} u_{ba} c_a^{2^i m} u_{ab} c_b^{2^i l}) u_{ba} c_a^{2^{n+1} m}))$ converges to a negative limit as n goes to infinity, thus $\lambda(r) \notin L$, and therefore r should be rejected by $\mathcal{A}'$ with the $\mathcal{L}_{\leq}$ condition. But $\mathrm{mp}_{\mathcal{A}'}(\lambda(p \prod_{i=0}^{n} u_{ba} c_a^{2^i m} u_{ab} c_b^{2^i l}))$ converges to 0, thus r is accepted by $\mathcal{A}'$ with the $\mathcal{L}_{\leq}$ condition, which contradicts the fact that it recognizes L.

We have thus established the incomparability of $\mathcal{L}_{\geq}$ and $\mathcal{L}_{\leq}$. As $\mathcal{L}_{<}$ and $\mathcal{L}_{>}$ are the classes of the complements of languages in respectively $\mathcal{L}_{\geq}$ and $\mathcal{L}_{\leq}$, it also implies the incomparability of the latter pair. □

The incompatibility of threshold classes shows how arbitrary the choice of only one of them as a standard definition would be. This suggests definition of a larger class including all of them.

3.4 Mean-Payoff Languages in the Borel Hierarchy

For a topological space X, we denote by Σ_1^0 the set of open subsets by and Π_1^0 the set of closed subsets. The Borel hierarchy is defined inductively as the two sequences (Σ_α^0) and (Π_α^0), where $\Sigma_\alpha^0 = (\bigcup_{\beta<\alpha} \Pi_\beta^0)_\sigma$, and $\Pi_\alpha^0 = (\bigcup_{\beta<\alpha} \Sigma_\beta^0)_\delta$, where α and β are ordinals and $(\bullet)_\sigma$, $(\bullet)_\delta$ denote closures under countable intersections and unions, respectively.

We consider the standard topology over A^ω with the base $\{wA^\omega \colon w \in A^*\}$, i.e. a subset of A^ω is open if and only if it is a union of sets, each set consists of all possible continuations of a finite word.

Theorem 4. *The following facts hold:* $\mathcal{L}_{\leq} \subset \Pi_2^0$, $\mathcal{L}_{\leq} \not\subseteq \Sigma_2^0$, $\mathcal{L}_{<} \subset \Sigma_3^0$ *and* $\mathcal{L}_{<} \not\subseteq \Pi_3^0$.

Proof. – Let $L \in \mathcal{L}_{\leq}$, then there exists $d \in \mathbb{N}$, a 1-payoff automaton $\mathcal{A}$ such that $L = \{w \in A^\omega | \min(Acc_{\mathcal{A}}(w)) \leq 0\}$. Therefore we can write L as

$$\begin{aligned} L &= \bigcap_{N \in \mathbb{N}} \{w \in A^\omega | \forall m \in \mathbb{N} \exists n > m \, \mathrm{mp}_{\mathcal{A}}(w \restriction n) < 1/N)\} \\ &= \bigcap_{N \in \mathbb{N}, m \in \mathbb{N}} \bigcup_{n>m} \{w \in A^\omega | \, \mathrm{mp}_{\mathcal{A}}(w \restriction n) < 1/N)\}. \end{aligned}$$

For any N and m the condition $\mathrm{mp}_{\mathcal{A}}(w \restriction n) < 1/N)$ is independent of the suffix past the nth symbol of w and therefore the set $\{w \in A^\omega | \, \mathrm{mp}_{\mathcal{A}}(w \restriction n) < 1/N)\}$ is clopen. We get that $\mathcal{L}_{\leq} \in \Pi_2^0$.

- We prove $\mathcal{L}_> \not\subseteq \Pi^0_2$, which is the same as $\mathcal{L}_\le \not\subseteq \Sigma^0_2$ because $\mathcal{L}_> = \text{co}\,\mathcal{L}_\le$. Let L be the set of words on alphabet $A = \{a, b\}$ having more than negligibly many b. We already demonstrated that $L \in \mathcal{L}_>$. Suppose $L \in \Pi^0_2$. Then $L = \bigcap_{i\in\mathbb{N}} L_i A^\omega$ for some family of languages of finite words L_i. We can assume without loss of generality that the words of L_i have all length i. For all m, the word $w_m = (\prod_{j=1}^{m-1} a^{2^j} b)(a^{2^m} b)^\omega \in L$ (as it is ultimately periodic with a period where the proportion of b is not 0). For the word $w = \prod_{j=1}^{\infty} a^{2^j} b$, it means that any prefix $w\restriction i$ of length i is in L_i. This is a contradiction, because $w \notin L$.
- For the two last items of the theorem: Chatterjee exhibited in [2] a Π^0_3-hard language in $\mathcal{L}_\ge$. He also established that this class is included in Π^0_3. As $\mathcal{L}_\ge = \text{co}\,\mathcal{L}_<$, that proves what we need. □

3.5 Dimensionality

In this section we analyze closure properties of mean-payoff languages defined by automata with a fixed dimension.

The following lemma shows that, for any d, the class of mean-payoff languages definable by d-payoff automata is not closed under intersection.

Lemma 2. *If d_1 and d_2 are two integers, then there exists L_1 and L_2, two mean-payoff languages of dimensions d_1 and d_2 such that L_1 and L_2 contain only convergent words and $L_1 \cap L_2$ is not definable as a dimension d mean-payoff language with $d < d_1 + d_2$.*

Proof. Let $A = \{a_1, \dots, a_{d_1}\}$ and $B = \{b_1, \dots, b_{d_2}\}$ be two disjoint alphabets. Let $\mathcal{A}_1$ be the one-state d_1-payoff automaton on alphabet $A \cup B$, such that the payoff of the transition (q_0, a_i, q_0) is 1 on the ith coordinate and 0 in the other coordinates and the payoff of the transition (q_0, b_i, q_0) is 0. And let $\mathcal{A}_2$ be the d_2-payoff one-state automaton defined similarly by swapping a and b.

Let L_i be the language defined on $\mathcal{A}_i$ by predicate F_i, testing equality with the singleton $\{l_i\}$, where l_i is in the simplex defined by the $d_i + 1$ different payoffs of the transitions of $\mathcal{A}_i$. In the proof of Theorem 2 we establish that the L_i are not empty.

Let $w \in (A + B)^\omega$, then w is in L_1 if and only if the proportion of a_i in every prefix tends to the ith coordinate of l_1, and it is in L_2 if and only if the proportion of b_i in every prefix tends to the ith coordinate of l_2.

Then for w to be in $L_1 \cap L_2$, it is necessary that the proportion of every symbols tends to either a coordinate of l_1, if that symbol is a a_i, or a coordinate of l_2, if it is a b_i.

Now suppose $L_1 \cap L_2$ is recognized by a d-payoff automaton with $d < d_1 + d_2$. Choose one terminal strongly connected component of $\mathcal{A}$ and consider for every symbol a of the alphabet a cycle labeled by a word in a^*, starting at some state q_a. Let also be p an initial run going to q_a and for every pair of symbols a, b a path u_{ab} going from q_a to q_b.

Only looking at the runs in the language $p\{u_{a_1 a} c_a^* u_{a a_1} | a \in A \cup B\}^\omega$, it is possible to converge to any proportion of the symbols of $A \cup B$, and thus have runs whose labeling word is in L. But as the payoffs are in dimension d, and the number of symbols is $d_1 + d_2 > d$, that language also contains runs converging to different symbol proportions but still having the same mean-payoff limit. Those runs are accepted by $\mathcal{A}$ but are not labeled by a word in L. □

Next, we prove that the intersection of two languages of dimensions d_1 and d_2 is a language of dimension $d_1 + d_2$. This will be proved constructively, by showing that the intersection language is the language defined on the product automaton with the "product" condition. Before going to the statement of the lemma, we need to define what those products are.

Definition 6. *If F_1 and F_2 are predicates on $2^{\mathbb{R}^{d_1}}$ and $2^{\mathbb{R}^{d_2}}$, we denote by $F_1 \Cap F_2$ the predicate on $2^{\mathbb{R}^{d_1+d_2}}$ which is true for $X \subseteq \mathbb{R}^{d_1+d_2}$ if and only if $F_1(p_1(X))$ and $F_2(p_2(X))$, where p_1 is the projection on the d_1 first coordinates and p_2 on the d_2 last.*

Definition 7 (Weighted automata product). *If $\mathcal{A}_1 = \langle A, Q_1, q_0^1, \delta_1, w_1 \rangle$ is a d_1-payoff automaton and $\mathcal{A}_2 = \langle A, Q_2, q_0^2, \delta_2, w_2 \rangle$, a d_2-payoff automaton, then we define $\mathcal{A}_1 \otimes \mathcal{A}_2 = \langle A, Q_1 \times Q_2, (q_0^1, q_0^2), \delta_{1\otimes 2}, w_{1\otimes 2} \rangle$, the product of $\mathcal{A}_1$ and $\mathcal{A}_2$, a $(d_1 + d_2)$-payoff automaton such that*

– $\delta_{1\otimes 2} = \{((q_1, q_2), a, (q_1', q_2')) | (q_1, a, q_1') \in \delta_1 \wedge (q_2, a, q_2') \in \delta_2 \wedge a \in A\}$,
– $w_{1\otimes 2} \colon \delta_{1\otimes 2} \to \mathbb{R}^{d_1+d_2}$ *is such that if* $w_1(q_1, a, q_1') = (x_1, \dots x_{d_1})$ *and* $w_2(q_2, a, q_2') = (y_1, \dots y_{d_2})$, *then* $w((q_1, q_2), a, (q_1', q_2')) = (x_1, \dots x_{d_1}, y_1, \dots y_{d_2})$.

But before we state the theorem, we need the following lemma:

Lemma 3. *If r is a run of a d-payoff automaton $\mathcal{A}$ and p is a projection from $\mathbb{R}^d$ to $\mathbb{R}^{d'}$, with $d' < d$, then* $\mathrm{Acc}(p(\mathrm{mp}_{\mathcal{A}}(r{\upharpoonright}n))) = p(\mathrm{Acc}(\mathrm{mp}_{\mathcal{A}}(r{\upharpoonright}n)))$

Proof. Let $x' \in \mathrm{Acc}(p(\mathrm{mp}_{\mathcal{A}}(r{\upharpoonright}n)))$. For any $i \in \mathbb{N}$, $p(\mathrm{mp}_{\mathcal{A}}(r{\upharpoonright}n))$ eventually comes into a distance $1/i$ from x', for some index n_i. For $j > i$ $\mathrm{mp}_{\mathcal{A}}(r{\upharpoonright}n_j)$ remains in $B(x', 1/i) \times K$ (where K is a compact of $\mathbb{R}^{d-d'}$), as this product is compact, it has at least one accumulation point. Thus the distance from x' to $p(\mathrm{Acc}(\mathrm{mp}_{\mathcal{A}}(r{\upharpoonright}n)))$ is 0. But $\mathrm{Acc}(\mathrm{mp}_{\mathcal{A}}(r{\upharpoonright}n))$ is a closed set and p, being a projection, is continuous, so $p(\mathrm{Acc}(\mathrm{mp}(r{\upharpoonright}n)))$ is closed too, which means $x' \in p(\mathrm{Acc}(\mathrm{mp}_{\mathcal{A}}(r{\upharpoonright}n)))$, and so $\mathrm{Acc}(p(\mathrm{mp}(r{\upharpoonright}n))) \subseteq p(\mathrm{Acc}(\mathrm{mp}_{\mathcal{A}}(r{\upharpoonright}n)))$.

Conversely, if $x' \in p(\mathrm{Acc}(\mathrm{mp}_{\mathcal{A}}(r \upharpoonright n)))$ a sub-sequence $\mathrm{mp}_{\mathcal{A}}(r \upharpoonright n_i)$ converges to a x such that $x' = p(x)$, and thus $p(\mathrm{mp}_{\mathcal{A}}(r \upharpoonright n_i))$ converges to x', which means $x' \in \mathrm{Acc}(p(\mathrm{mp}(r{\upharpoonright}n)))$. We conclude that $\mathrm{Acc}(p(\mathrm{mp}_{\mathcal{A}}(r{\upharpoonright}n))) = p(\mathrm{Acc}(\mathrm{mp}_{\mathcal{A}}(r{\upharpoonright}n)))$. □

Now we have all the needed tools, we can characterize the intersection of two mean-payoff languages as another mean-payoff language defined on an automaton whose dimension is known.

Proposition 3. *For any two d_1-payoff and d_2-payoff automata $\mathcal{A}_1$ and $\mathcal{A}_2$ and any two predicates F_1 and F_2 on respectively $2^{\mathbb{R}^{d_1}}$ and $2^{\mathbb{R}^{d_2}}$, the following equality holds: $L(\mathcal{A}_1, F_1) \cap L(\mathcal{A}_2, F_2) = L(\mathcal{A}_1 \otimes \mathcal{A}_2, F_1 \Cap F_2)$.*

Proof. Suppose $u \in A^\omega$, then the sequence of mean-payoffs of run r of u in $\mathcal{A}_1 \otimes \mathcal{A}_2$ are the projections by p_1 and p_2 of the sequence of mean-payoffs of some runs r_1 and r_2 in $\mathcal{A}_1$ and r_2 in $\mathcal{A}_2$ whose labeling is u. And conversely, if u has runs r_1 and r_2 in $\mathcal{A}_1$ and r_2 in $\mathcal{A}_2$, then it has a run r in $\mathcal{A}_1 \otimes \mathcal{A}_2$ whose sequence of mean-payoffs projects by p_1 and p_2 onto those of r_1 and r_2.

If r, r_1, and r_2 are such runs (the payoffs of r projecting on those of r_1 and r_2), then using lemma 3, we find that $\mathrm{Acc}_{\mathcal{A}}(r_1) = \mathrm{Acc}(p_1(\mathrm{mp}(r{\upharpoonright}n))) = p_1(\mathrm{Acc}(\mathrm{mp}(r{\upharpoonright}n)))$ and that $\mathrm{Acc}_{\mathcal{A}}(r_2) = \mathrm{Acc}(p_2(\mathrm{mp}(r{\upharpoonright}n))) = p_2(\mathrm{Acc}(\mathrm{mp}(r{\upharpoonright}n)))$.

But by definition $(F_1 \Cap F_2)(\text{Acc}(\text{mp}(r \upharpoonright n)))$ holds iff $F_1(p_1(\text{Acc}(\text{mp}(r \upharpoonright n))))$ and $F_2(p_2(\text{Acc}(\text{mp}(r{\upharpoonright}n))))$ hold, thus it holds iff $F_1(\text{Acc}_{\mathcal{A}_1}(r_1))$ and $F_2(\text{Acc}_{\mathcal{A}_2}(r_2))$ hold.

From that we deduce that a word is in $L(\mathcal{A} \otimes \mathcal{A}, F_1 \Cap F_2)$ if and only if it is both in $L(\mathcal{A}_1, F_1)$ and $L(\mathcal{A}_2, F_2)$. □

4 An Analyzable Class of Mean-Payoff Languages

4.1 The Class of Multi-threshold Mean-Payoff Languages

As a candidate for a class of mean-payoff languages that is closed under complementation and includes all the expected standard mean-payoff language classes, we propose the following definition.

Definition 8. *A language L is a multi-threshold mean-payoff language (denoted by $L \in \mathcal{L}_{mt}$) if it is the mean-payoff language defined on some d-payoff automaton $\mathcal{A}$, with a predicate F such that $F(S)$ is a Boolean combination of threshold conditions on $p_i(S)$ where p_i is the projection along the ith coordinate.*

Example 3. Consider the automaton given in Example 1 and the multi-threshold mean-payoff language $L = \{w |\min p_1(\text{Acc}(w)) > .1 \wedge \max p_1(\text{Acc}(w)) < .9 \wedge \min p_2(\text{Acc}(w)) > .1 \wedge \max p_2(\text{Acc}(w)) < .9\}$. For the word w, defined in Example 1, the set of accumulation points is shown to be a triangle that is contained in the box $\{x|.1 < p_1(x) < .9 \wedge .1 < p_2(x) < .9\}$ and therefore $w \in L$.

Geometrically, multi-threshold acceptance conditions can be visualized as specifying constraints on the maximal and minimal projection of $\text{Acc}(w)$ on the axes. Since we can extend the payoff vectors by adding a coordinate whose values are a linear combination of the other coordinates, also threshold constraints involving minimal and maximal elements of the projection of $\text{Acc}(w)$ on other lines are expressible, as shown in the following example.

Example 4. Assume that, with the automaton given in Example 1, we want to accept the words w such that $\text{Acc}(w)$ is contained in the triangle $(.2, .2)-(.8, .2)-(.2, .8)$. We can do so by extending the dimension of the payoffs and renaming $(0, 0) \mapsto (0, 0, 0)$, $(1, 0) \mapsto (1, 0, 1)$, and $(1, 1) \mapsto (1, 1, 2)$. Namely, by adding a coordinate whose value is the sum of the other two coordinates. Then, $L = \{w |\min p_1(\text{Acc}(w)) > .2 \wedge \min p_2(\text{Acc}(w)) > .2 \wedge \max p_3(\text{Acc}(w)) < 1\}$ is the wanted language.

4.2 Closure under Boolean Operations

We prove here that $\mathcal{L}_{mt}$ is in fact the Boolean closure of $\mathcal{L}_{\lesseqgtr} \triangleq \mathcal{L}_{<} \cup \mathcal{L}_{\leq} \cup \mathcal{L}_{>} \cup \mathcal{L}_{\geq}$.

Theorem 5. *$\mathcal{L}_{mt}$ is closed under Boolean operations and any language in $\mathcal{L}_{mt}$ is a Boolean combination of languages in $\mathcal{L}_{\lesseqgtr}$.*

Proof. Closure by complementation: let L be a $\mathcal{L}_{mt}$ language, defined on some automaton $\mathcal{A}$ by a predicate P. $w \in L$ iff $P(\text{Acc}_{\mathcal{A}}(w))$. So $w \in L^c$ iff $w \notin L$, that is iff

$\neg P(\mathrm{Acc}_{\mathcal{A}}(w))$. But $\neg P$ is also a Boolean combination of threshold conditions, thus L^c is a $\mathcal{L}_{mt}$ language.

Closure by intersection: let L_1 and L_2 be two $\mathcal{L}_{mt}$ languages defined respectively on the automata $\mathcal{A}$ and $\mathcal{A}$ by the predicates P_1 and P_2. Then $L_1 \cap L_2 = L(\mathcal{A} \otimes \mathcal{A}, P_1 \Cap P_2)$ (Theorem 3). It is easy to see that $P_1 \Cap P_2$ is still a Boolean combination of thresholds, and thus $L(\mathcal{A} \otimes \mathcal{A}, P_1 \Cap P_2)$ is in $\mathcal{L}_{mt}$.

The other Boolean operations can be written as a combination of complementation and intersection.

Now we show, by induction on height of the formula of a predicate, that any $\mathcal{L}_{mt}$ language is a Boolean combination of $\mathcal{L}_{\lesseqgtr}$ languages.

We can, without loss of generality, suppose that every threshold concerns a different coordinate (if a coordinate has several thresholds, we can duplicate that coordinate, keeping the same values, and the language will remain the same). We can also assume that the predicate is written only with combinations of conjunctions and negations of thresholds.

- If the height is 0, that means that the condition is only one threshold on a multi-payoff automaton. The recognized language is the same as that of the automaton projected on the tested coordinate, so it is in $\mathcal{L}_{\lesseqgtr}$.
- If the predicate is $\neg P$, then the recognized language is the complement of $L(\mathcal{A}, P)$, which is a Boolean combination of $\mathcal{L}_{mt}$ languages of lesser height.
- If the predicate is $P = P_1 \wedge P_2$, let us call $\mathcal{A}_i$, a copy of $\mathcal{A}$ whose payoffs are projected on the subspace tested in P_i. Then $\mathcal{A}$ is isomorphic to $\mathcal{A}_1 \otimes \mathcal{A}_2$. Furthermore, as the set of coordinates that are tested in P_1 and P_2 are disjoint, their exists some P'_1 and P'_2 with the same heights as P_1 and P_2, such that $P = P'_1 \Cap P'_2$. Thus $L = L(\mathcal{A}_1, P'_1) \cap L(\mathcal{A}_2, P'_2)$ (Theorem 3), which is a Boolean combination of $\mathcal{L}_{mt}$ languages of lesser height. □

4.3 Decidability

Theorem 6. *The emptiness of a language of $\mathcal{L}_{mt}$ is decidable.*

Proof. We can assume the predicate of acceptance is written in disjunctive normal form (if not, we can find an equivalent DNF formula). Then we can see that a run is accepted whenever its set of accumulation points satisfies at least one of the disjuncts, and in a disjunct, every literal has to be satisfied. If we know how to check if a literal is satisfied, then this provides an obvious decision algorithm for one run.

Then it is easy to see that there are two types of literal. Some say that there must exist an accumulation point whose tested coordinate is greater or smaller than the threshold, we call those existential literals. The other literals say that every accumulation point should have the tested coordinate above or below the threshold, those we call universal literals.

For checking the emptiness of $L(\mathcal{A}, F)$, we propose the following algorithm: Try, for every a disjunct of F and every reachable SCC C of $\mathcal{A}$, to compute $P(C)$,the convex hull of the payoffs of its transitions, then compute C' the intersection of $P(C)$ with every universal literal, and finally check whether it intersects with every existential literal of the disjunct. If it does, then return true, else loop. If you exhausted the combinations, return false.

If this algorithm returns true, because C' is a convex polyhedron included in C and intersecting with every existential literal, we can construct D which is connected, closed, included in C', and intersects with every existential literal (take for instance the convex hull of a family consisting in one point in every intersection of C' with an existential literal). We can see that $F(D)$ holds. Then, Theorem 2 says there exist a run r such that $\mathrm{Acc}_{\mathcal{A}}(r) = D$, and thus there exist a word which that run and therefore is in $L(\mathcal{A}, F)$.

If that algorithm returns false, for every reachable SCC C, if you choose a closed connected subset D of $P(C)$ (as Theorem 1 says sets of accumulation points have to be), then for every disjunct, D either is not completely included in some universal literal, either does not intersect with some existential literal. In both case, D does not make the disjunct true. So F holds for no set of accumulation points of a run of $\mathcal{A}$, which implies that $L(\mathcal{A}, F)$ is empty. □

5 Summary and Future Directions

We proposed a definition of ω-languages using Boolean combination of threshold predicates over mean-payoffs. This type of specifications allows to express requirements concerning averages such as “no more than 10% of the messages are lost” or “the number of messages lost is negligible”. The later is not expressible by ω-regular requirements. We showed that if closure under intersection is needed, multi-dimensional payoffs have to be considered. For runs of d-payoff automata, we studied acceptance conditions that examine the set of *accumulation points* and characterized those sets as all closed, bounded and connected subsets of $\mathbb{R}^d$.

The class of *multi-threshold mean-payoff languages* was proposed, using acceptance conditions that are Boolean combinations of inequalities comparing the minimal or maximal accumulation point along some coordinate with a constant threshold. We studied expressiveness, closure properties, analyzability, and Borel complexity.

Possible direction for future include extension to non-deterministic automata, and the study of multi-mean-payoff games.

Acknowledgments

This research was partially supported by NSF grants CNS 0524059 and CPA 0541149, and the French Minalogic project Athole.

References

1. Alur, R., Kanade, A., Weiss, G.: Ranking automata and games for prioritized requirements. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 240–253. Springer, Heidelberg (2008)
2. Chatterjee, K.: Concurrent games with tail objectives. Theor. Comput. Sci. 388(1-3), 181–198 (2007)

3. Chatterjee, K., de Alfaro, L., Henzinger, T.: The complexity of quantitative concurrent parity games. In: Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms, pp. 678–687 (2006)
4. Chatterjee, K., Doyen, L., Henzinger, T.A.: Quantitative languages. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 385–400. Springer, Heidelberg (2008)
5. Chatterjee, K., Henzinger, T., Jurdziński, M.: Mean-payoff parity games. In: Proceedings of the 20th Annual Symposium on Logic in Computer Science, pp. 178–187. IEEE Computer Society Press, Los Alamitos (2005)
6. Gimbert, H., Zielonka, W.: Deterministic priority mean-payoff games as limits of discounted games. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 312–323. Springer, Heidelberg (2006)
7. Kupferman, O., Lustig, Y.: Lattice automata. In: Cook, B., Podelski, A. (eds.) VMCAI 2007. LNCS, vol. 4349, pp. 199–213. Springer, Heidelberg (2007)
8. Manna, Z., Pnueli, A.: The temporal logic of reactive and concurrent systems: Specification. Springer, Heidelberg (1991)
9. Pnueli, A.: The temporal logic of programs. In: 18th IEEE Symposium on the Foundations of Computer Science (FOCS 1977), pp. 46–57 (1977)
10. Thomas, W.: Automata on infinite objects. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B, pp. 133–191. Elsevier, Amsterdam (1990)
11. Vardi, M., Wolper, P.: Reasoning about infinite computations. Information and Computation 115(1), 1–37 (1994)
12. Zwick, U., Paterson, M.: The complexity of mean payoff games on graphs. Theoretical Computer Science 158, 343–359 (1996)

Minimal Cost Reachability/Coverability in Priced Timed Petri Nets

Parosh Aziz Abdulla[1] and Richard Mayr[2]

[1] Uppsala University, Sweden
[2] University of Edinburgh, UK

Abstract. We extend discrete-timed Petri nets with a cost model that assigns token storage costs to places and firing costs to transitions, and study the minimal cost reachability/coverability problem. We show that the minimal costs are computable if all storage/transition costs are non-negative, while even the question of zero-cost coverability is undecidable in the case of general integer costs.

1 Introduction

Petri nets are one of the most widely used models for the study and analysis of concurrent systems. Furthermore, several different models have been introduced in [1,7,14,13,4,9,6] which extend the classical model by introducing timed behaviors.

We consider *Timed Petri Nets (TPNs)* in which each token has an 'age' represented by a natural number [1,7]. A marking of the net is a mapping which assigns a multiset of natural numbers to each place. The multiset represents the numbers and ages of the tokens in the corresponding place. Each arc of the net is equipped with an interval defined by two natural numbers (or ω). A transition may fire iff its input places have tokens with ages satisfying the intervals of the corresponding arcs. Tokens generated by transitions will have ages in the intervals of the output arcs. In fact, this model is a generalization of the one in [7], since the latter only allows generating tokens of age 0.

In parallel, there have been several works on extending the model of timed automata [2] with *prices* (*weights*) (see e.g., [3,11,5]). Weighted timed automata are suitable models for embedded systems, where we have to take into consideration the fact that the behavior of the system may be constrained by the consumption of different types of resources. Concretely, weighted timed automata extend classical timed automata with a cost function C that maps every location and every transition to a nonnegative integer (or rational) number. For a transition, C gives the cost of performing the transition. For a location, C gives the cost per time unit for staying in the location. In this manner, we can define, for each run of the system, the accumulated cost of staying in locations and performing transitions along the run.

We study a natural extension of TPNs, namely *Priced TPNs (PTPNs)*. We allow the cost function to map transitions and places of the Petri net into vectors of integers (of some given length k). Again, for a transition, C gives the cost of performing the transition; while for a place, C gives the cost per time unit per token in the place. We consider the *cost-optimal problem* for PTPNs where, given an *initial marking* M_0 and a set F of *final markings*, the task is to compute the minimal accumulated cost of a run that reaches F from M_0. We consider two variants of the problem: the *reachability* problem in which F is a single marking; and the *coverability* problem in which F is an upward closed set of markings. Since the set of costs within which we can reach a set F from a set M_0 is upward closed (regardless of the form of F), the cost-optimality problem can be reduced, using the construction of Valk and Jantzen [17], to the *cost threshold* problem. In the latter, we are given a cost vector v, and we want to check whether it is possible to reach F from M_0 with a cost that does not exceed v.

We also consider two models related to PTPNs. The first, called *Priced Petri Nets (PPNs)*, is a priced extension of classical (untimed) Petri nets, and is a special case of PTPNs. The other model is an (interesting and nontrivial) extension of classical Petri nets, in which a fixed place p and a fixed transition t are connected by a so called *inhibitor arc*. In this case, the transition t can fire only if p is empty. It has been shown that the reachability problem for Petri nets with one inhibitor arc is decidable [15].

For the above mentioned models, we show a number of (un)decidability results. First, we recall that the reachability problem is undecidable for TPNs [16], which immediately implies that the cost threshold reachability problem is undecidable for PTPNs. With this undecidability result in mind, the two natural (and simpler) problems to consider are the cost threshold coverability problem for PTPNs, and the cost threshold reachability problem for PPNs. We prove that cost threshold coverability problem is decidable for PTPNs with non-negative costs (where all components of the cost vectors are non-negative). We show that this gives in a straightforward manner also a proof of the decidability of the coverability problem for Petri nets with one inhibitor arc. Furthermore, we show that the cost threshold reachability problem for PPNs and the reachability problem for Petri nets with one inhibitor arc are reducible to each other. These results show a close (and surprising) relationship between our models and that of Petri nets with one inhibitor arc. Finally, we show that if we allow negative costs then even the cost threshold coverability problem for PPNs becomes undecidable.

2 Preliminaries

2.1 Priced Timed Petri Nets

The timed Petri net model (TPN) defined by Escrig et al. [7,16] is an extension of Petri nets where tokens have integer ages measuring the time since their creation, and transition arcs are labeled with time-intervals (whose bounds are natural numbers or ω) which restrict the ages of tokens which can be consumed and produced. We extend this model to priced timed Petri nets (PTPN) by assigning multidimensional costs to both transitions (action costs) and places (storage costs). Each firing of a discrete transition costs the assigned cost vector. The cost of a timed transition depends on the marking. Storing k_1 tokens for k_2 time units on a place with cost vector v costs $k_1 * k_2 * v$.

Let $\mathbb{N}$ denote the non-negative integers and $\mathbb{N}^k$ and $\mathbb{N}^k_\omega$ the set of vectors of dimension k over $\mathbb{N}$ and $\mathbb{N} \cup \{\omega\}$, respectively ($\omega$ represents the first limit ordinal). We use a set *Intrv* of intervals $\mathbb{N} \times \mathbb{N}_\omega$. We view a multiset M over A as a mapping $M : A \mapsto \mathbb{N}$.

Given a set A with an ordering $\preceq$ and a subset $B \subseteq A$, B is said to be *upward closed* in A if $a_1 \in B, a_2 \in A$ and $a_1 \preceq a_2$ implies $a_2 \in B$. Given a set $B \subseteq A$, we define the *upward closure* $B \uparrow$ to be the set $\{a \in A | \; \exists a' \in B : \; a' \preceq a\}$.

Definition 1. *A* Priced Timed Petri Net (PTPN) *is a tuple* $N = (P, T, In, Out, C)$ *where* P *is a finite set of places,* T *is a finite set of transitions,* In, Out *are partial functions from* $T \times P$ *to Intrv and* $C : P \cup T \to \mathbb{Z}^k$ *is the cost function assigning (multidimensional, and possibly also negative) firing costs to transitions and storage costs to places.*

If $In(t, p)$ (respectively $Out(t, p)$) is defined, we say that p is an *input (respectively output) place* of t. Let max denote the maximal finite number $\in \mathbb{N}$ appearing on the time intervals of the given PTPN.

A *marking* M of N is a finite multiset over $P \times \mathbb{N}$. It defines the numbers and ages of tokens in each place in the net. We identify a token in a marking M by the pair (p, x) representing its place and age in M. Then, $M((p, x))$ defines the number of tokens with age x in place p. Abusing notation, we define, for each place p, a multiset $M(p)$ over $\mathbb{N}$ where $M(p)(x) = M((p, x))$. We sometimes denote multisets as lists. For a marking M of the form $[(p_1, x_1), \ldots, (p_n, x_n)]$ we use M^{+1} to denote the marking $[(p_1, x_1 + 1), \ldots, (p_n, x_n + 1)]$. For PTPN, let $\leq$ denote the partial order on markings given by multiset-inclusion.

Transitions. We define two transition relations on the set of markings: timed and discrete. The *timed transition relation* increases the age of each token by one. Formally, $M_1 \to_{time} M_2$ iff $M_2 = M_1^{+1}$.

We define the *discrete transition relation* $\to_D$ as $\bigcup_{t \in T} \longrightarrow_t$, where $\longrightarrow_t$ represents the effect of *firing* the discrete transition t. More precisely, $M_1 \longrightarrow_t M_2$ if the set of input arcs $\{(p, \mathcal{I}) | \; In(t, p) = \mathcal{I}\}$ is of the form $\{(p_1, \mathcal{I}_1), \ldots, (p_k, \mathcal{I}_k)\}$, the set of output arcs $\{(p, \mathcal{I}) | \; Out(t, p) = \mathcal{I}\}$ is of the form $\{(q_1, \mathcal{J}_1), \ldots, (q_\ell, \mathcal{J}_\ell)\}$, and there are multisets $b_1 = [(p_1, x_1), \ldots, (p_k, x_k)]$ and $b_2 = [(q_1, y_1), \ldots, (q_\ell, y_\ell)]$ over $P \times \mathbb{N}$ such that the following holds:

- $b_1 \leq M_1$ and $M_2 = (M_1 - b_1) + b_2$
- $x_i \in \mathcal{I}_i$, for $i : 1 \leq i \leq k$ and $y_i \in \mathcal{J}_i$, for $i : 1 \leq i \leq \ell$

We say that t is *enabled* in M if such a b_1 exists. A transition t may be fired only if for each incoming arc, there is a token with the right age in the corresponding input place. These tokens will be removed when the transition is fired. The newly produced tokens have ages which are chosen nondeterministically from the relevant intervals on the output arcs of the transition.

We write $\longrightarrow = \to_{time} \cup \to_D$ to denote all transitions, $\xrightarrow{*}$ to denote the reflexive-transitive closure of $\longrightarrow$ and $\to_D^+$ to denote the transitive closure of $\to_D$. It is easy to extend $\xrightarrow{*}$ to sets of markings. We define $Reach(M) := \{M' \,|\, M \xrightarrow{*} M'\}$ as the set of markings reachable from M.

The cost of computations. A computation $\sigma := M_1 \longrightarrow M_2 \longrightarrow \ldots \longrightarrow M_n$ is a sequence of transitions, and also denoted by $M_1 \xrightarrow{\sigma} M_n$. The cost of a discrete transition t is defined as $Cost(M \longrightarrow_t M') := C(t)$ and the cost of a timed transition is defined as $Cost(M \to_{time} M^{+1}) := \sum_{p \in P} |M(p)| * C(p)$. The cost of a computation σ is the sum of all transition costs, i.e., $Cost(\sigma) := \sum_{i=1}^{n-1} Cost(M_i \longrightarrow M_{i+1})$.

If the prices are ignored in PTPN then the model becomes equivalent to the timed Petri nets of [7,16], except that we also allow the creation of tokens with nonzero ages.

2.2 Priced Petri Nets

Priced Petri Nets (PPN) are a simple extension of standard Petri nets (i.e., without token clocks and time constraint arcs) by adding prices and transition delays. Later we will show that PPN are a weaker model than PTPN (Lemma 3), but most undecidability results hold even for the weaker PPN model (Theorem 14).

Definition 2. *A* Priced Petri Net (PPN) *is a tuple* $N = (P, T, T_0, T_1, In, Out, C)$ *where P is a finite set of places, $T = T_0 \uplus T_1$ is a disjoint union of the sets of instantaneous transitions and timed transitions, $In, Out : T \to \{0, 1\}^P$, and $C : P \cup T \to \mathbb{Z}^k$ is the cost function assigning (multidimensional) firing costs to transitions and storage costs to places.*

The markings $M : P \to \mathbb{N}$ and the firing rules are exactly as in standard Petri nets. Transition t is enabled at marking M iff $M \geq In(t)$ (componentwise), and firing t yields the marking $M - In(t) + Out(t)$.

The cost of an instantaneous transition $t \in T_0$ is defined as $Cost(M_1 \longrightarrow_t M_2) := C(t)$ and the cost of a timed transition $t \in T_1$ is defined by $Cost(M_1 \longrightarrow_t M_2) := C(t) + \sum_{p \in P} M(p) * C(p)$. As before, the cost of a computation is the sum of all transition costs in it.

2.3 The Priced Reachability/Coverability Problem

We study the problem of computing the minimal cost for reaching a marking in a given target set from the initial marking.

COST-THRESHOLD

Instance: A PTPN (or PPN) N with an initial marking M_0, a set of final markings F and a vector $v \in \mathbb{N}_\omega^k$.

Question: Does there exist a marking $M_f \in F$ and a computation $M_0 \xrightarrow{\sigma} M_f$ s.t. $Cost(\sigma) \leq v$?

We call this problem cost-threshold-coverability if F is upward-closed, and cost-threshold-reachability if F is a single marking, i.e., $F = \{M_f\}$ for a fixed marking M_f.

Lemma 3. *The cost threshold reachability/coverability problem for PPN is polynomial time reducible to the cost threshold reachability/coverability problem for PTPN.*

If all costs are non-negative (i.e., in $\mathbb{N}^k$ rather than $\mathbb{Z}^k$) then the standard componentwise ordering on costs is a well-order and thus every upward-closed set of costs has finitely many minimal elements. Furthermore, if we have a positive instance of cost-threshold with some allowed cost v then any modified instance with some allowed cost $v' \geq v$ will also be positive. Thus the set of possible costs in the cost-threshold problem is upward-closed. In this case the Valk-Jantzen Theorem [17] implies that the set of minimal possible costs can be computed iff the Cost-Threshold problem is decidable.

Theorem 4. *(Valk & Jantzen [17]) Given an upward-closed set $V \subseteq \mathbb{N}^k$, the finite set V_{min} of minimal elements of V is computable iff for any vector $\boldsymbol{v} \in \mathbb{N}_\omega^k$ the predicate $\boldsymbol{v}\!\downarrow \cap\, V \neq \emptyset$ is decidable.*

COMPUTING MINIMAL POSSIBLE COSTS

Instance: A PTPN (or PPN) N with $C : P \cup T \to \mathbb{N}^k$, initial marking M_0, and a set of final markings F.

Question: Compute the minimal possible costs of reaching F, i.e., the finitely many minimal elements of $\{v \in \mathbb{N}^k \mid \exists M_f \in F, \sigma.\, M_0 \xrightarrow{\sigma} M_f \wedge \mathit{Cost}(\sigma) \leq v\}$.

2.4 Petri Nets with Control-States and Petri Nets with One Inhibitor Arc

Several other versions of Petri nets will be used in our constructions. We define Petri nets with an extra finite control. This does not increase their expressive power, since they can easily be encoded into standard Petri nets with some extra places. However, for some constructions, we need to make the distinction between infinite memory and finite memory in Petri nets explicit.

A *Petri net with control-states (cPN)* is a tuple $N = (P, Q, T, \mathit{In}, \mathit{Out})$ where P is a finite set of places, Q a finite set of control-states, T a finite set of transitions and $\mathit{In}, \mathit{Out} : T \to Q \times \mathbb{N}^P$.

A marking is a tuple (q, M) where $q \in Q$ and $M : P \to \mathbb{N}$. A transition t with $\mathit{In}(t) = (q_1, M_1)$ and $\mathit{Out}(t) = (q_2, M_2)$ is enabled at marking (q, M) iff $q_1 = q$ and $M_1 \leq M$. If t fires then the resulting marking is $(q_2, M - M_1 + M_2)$.

The reachability problem for Petri nets is decidable [12] and a useful generalization to sets of markings was shown by Jančar [10].

Theorem 5. *([10]) Given a cPN, we can use a simple constraint logic to describe properties of markings (q, M). Any formula Φ in this logic is a boolean combination of predicates of the following form: $q = q_i$ (the control-state is q_i), $M(p_i) \geq k$, or $M(p_i) \leq k$, where $k \in \mathbb{N}$. In particular, the logic can describe all upward-closed sets of markings. Given an initial marking and a constraint logic formula Φ, it is decidable if there exists a reachable marking that satisfies Φ.*

A *Petri Net with One Inhibitor Arc* [15] is defined as an extension of cPN. We fix a place p and a transition t, which are connected by a so-called inhibitor arc (p, t), and modify the firing rule for t such that t can only fire if place p is empty. Decidability of the reachability problem for Petri nets with one inhibitor arc has been shown in [15]. This result can be extended to the case where one place inhibits several transitions.

Lemma 6. *The reachability problem for Petri nets with many inhibitor arcs $(p, t_1), \ldots, (p, t_k)$ which all connect to the same place p can be reduced to the reachability problem for Petri nets with just one inhibitor arc.*

3 Decidability for Non-negative Costs

Theorem 7. *The cost-threshold coverability problem is decidable for PTPN with non-negative costs.*

Consider an instance of the problem. Let $N = (P, T, \mathit{In}, \mathit{Out}, C)$ be a PTPN with $C(P \cup T) \subseteq \mathbb{N}^k$, M_0 the initial marking, F an upward-closed target set (represented by the finite set F_{min} of its minimal elements) and $v = (v_1, \ldots, v_k) \in \mathbb{N}_\omega^k$.

First, for every i, if $v_i = \omega$ then we replace v_i by 0 and set the i-th component of the cost function C of N to 0, too. This does not change the result of the cost-threshold problem. So we can assume without restriction that $v = (v_1, \ldots, v_k) \in \mathbb{N}^k$ and let $b := \max_{1 \leq i \leq k} v_i \in \mathbb{N}$. Let $max \in \mathbb{N}$ be the maximal finite constant appearing on time intervals of the PTPN N.

Our proof has two parts. In part 1 we construct a cPN N' that simulates the behavior of the PTPN N w.r.t. discrete transitions. In part 2 we define an operation g on markings of N' which encodes the effect of timed transitions in N.

Construction (part 1). We now construct the cPN $N' = (P', Q, T', In', Out')$ that encodes discrete transitions of N.

The set of places $P = \{p_1, \ldots, p_n\}$ of N can be divided into two disjoint subsets $P = P_1 \uplus P_0$ where $\forall p \in P_0 .\, C(p) = 0$ and $\forall p \in P_1 .\, C(p) > 0$. We call the places in P_0 *free-places* and the places in P_1 *cost-places*. Let $m := |P_1|$. Without restriction let $p_1, \ldots, p_m$ be cost-places and $p_{m+1}, \ldots, p_n$ be free-places.

The places of N' are defined as $P' := \{p(j,x) \mid p_j \in P \wedge 0 \leq x \leq max + 1\}$. The index x is used to encode the age of the token. So if in N there is a token of age x on place p_j then in N' there is a token on place $p(j,x)$. The number of tokens on place $p(j, max + 1)$ in N' encodes the number of tokens with ages $> max$ on p_j in N. This is because token ages $> max$ cannot be distinguished from each other in N. In the following we always consider equivalence classes of markings of N by identifying all token ages $> max$.

The set of control-states Q of N' is defined as a tuple $Q = Q' \times R$, where $Q' = \{0, \ldots, b\}^{\{1,\ldots,m\} \times \{0,\ldots,max+1\}}$. The intuition is that for every cost-place $p(j,x)$ of N' the number of tokens on $p(j,x)$ is partly stored in the control-state $q(j,x)$ of Q' up-to a maximum of b, while only the rest is stored on the place $p(j,x)$ directly. So if place $p(j,x)$ contains $y \geq b$ tokens then this is encoded as just $y - b$ tokens on place $p(j,x)$ and a control-state q with $q(j,x) = b$. If place $p(j,x)$ contains $y \leq b$ tokens then this is encoded as 0 tokens on place $p(j,x)$ and a control-state q with $q(j,x) = y$. Furthermore, $R = \{0, \ldots, b\}^k$ and is used to store the remaining maximal allowed cost.

A marking M' of N' is given as $M' = ((q,r), M'')$ where $(q,r) \in Q$ and $M'' : \{1, \ldots, n\} \times \{0, \ldots, max + 1\} \rightarrow \mathbb{N}$. For every $r \in R$ we define a mapping f_r of markings of N to markings of N'. Given a marking M of N, let $M' = ((q,r), M'') := f_r(M)$ be defined as follows.

- $q(j,x) = \min\{M((p_j,x)), b\}$ for $1 \leq j \leq m, 0 \leq x \leq max$
- $M''(j,x) = \max\{0, M((p_j,x)) - b\}$ for $1 \leq j \leq m, 0 \leq x \leq max$
- $q(j, max + 1) = \min\{\sum_{x > max} M((p_j,x)), b\}$ for $1 \leq j \leq m$
- $M''(j, max + 1) = \max\{0, \sum_{x > max} M((p_j,x)) - b\}$ for $1 \leq j \leq m$
- $M''(j,x) = M((p_j,x))$ for $j > m, 0 \leq x \leq max$
- $M''(j, max + 1) = \sum_{x > max} M((p_j,x))$ for $j > m$

This ensures that token numbers up-to b on cost-places are encoded in the control-state and only the rest is kept on the cost-places themselves. Free-places are unaffected. The parameter $r \in R$ assigns the allowed remaining cost, which is independent of M, but also stored in the finite control of M'. The initial marking M'_0 of N' is defined as $M'_0 = f_v(M_0)$. The upward-closed set of final markings F of N is represented by the finite set F_{min} of its minimal elements. We also represent the upward-closed set of final markings F' of N' by the finite set F'_{min} of its minimal elements. Let

$F'_{min} := \bigcup_{\mathbf{0} \le r \le v} f_r(F_{min})$, i.e., we take the union over all possibilities of remaining allowed (unused) cost r.

The Petri net N' only encodes the effect of discrete transitions of the PTPN N' (the effect of timed transitions will be handled separately). The set T' of transitions of N' is defined as follows. Let $t \in T$. We say that a pair of functions $I, O : \{1, \ldots, n\} \times \{0, \ldots, max + 1\} \to \{0, 1\}$ are compatible with t iff $\forall p_j \in P = \{p_1, \ldots, p_n\}$

- If $In(t, p_j)$ is defined then $\exists_{=1} x \in In(t, p_j) \cap \{\mathit{0}, \ldots, max + \mathit{1}\}$ s.t. $I(j, x) = 1$ and $I(j, x') = 0$ for every $x' \neq x$. Otherwise $I(j, x) = 0$ for all x.
- If $Out(t, p_j)$ is defined then $\exists_{=1} x \in Out(t, p_j) \cap \{\mathit{0}, \ldots, max + \mathit{1}\}$ s.t. $O(j, x) = 1$ and $I(j, x') = 0$ for every $x' \neq x$. Otherwise $O(j, x) = 0$ for all x.

The set of all these compatible pairs of functions represents all possible ways of choosing the age of tokens consumed/produced by t out of the specified time intervals. All ages $> max$ are lumped together under $max + 1$, since they are indistinguishable in N.

Then for every combination of matrices $v_1, v_3 \in \{0, 1\}^{\{1, \ldots, m\} \times \{0, \ldots, max+1\}}$ and $v_2, v_4 \in \{0, 1\}^{\{1, \ldots, n\} \times \{0, \ldots, max+1\}}$ and every pair of functions $I, O : \{1, \ldots, n\} \times \{0, \ldots, max+1\} \to \{0, 1\}$ which are compatible with t and every control-state $(q, r) \in Q$ we have a transition $t' \in T'$ with $In(t') = ((q, r), I')$ and $Out(t') = ((q', r'), O')$ iff the following conditions are satisfied.

- $(v_1, \mathbf{0}) + v_2 = I$ and $(v_3, \mathbf{0}) + v_4 = O$
- $q \geq v_1$
- $q' = q - v_1 + v_3 \in Q'$ (in particular, every component of q' is $\leq b$).
- $r' = r - C(t) \geq \mathbf{0}$ (the cost of t is deducted from the remaining allowed cost).
- $I'(p(j, x)) = v_2(j, x)$ for $1 \leq j \leq n, 0 \leq x \leq max + 1$
- $O'(p(j, x)) = v_4(j, x)$ for $1 \leq j \leq n, 0 \leq x \leq max + 1$

The choice of the v_1, v_2, v_3, v_4 encodes all possible ways of splitting the effect of the transition on places between the part which is encoded in the finite control and the part remaining on the places. Consume v_1 from the finite control and v_2 from the real places. Produce v_3 in the finite control and v_4 on the real places. Note that v_1, v_3 have a smaller dimension than v_2, v_4, because they operate only on cost-places. So v_1, v_3 are extended from dimension $\{1, \ldots, m\} \times \{0, \ldots, max + 1\}$ to $\{1, \ldots, n\} \times \{0, \ldots, max + 1\}$ by filling the extra entries (the free places) with zero to yield $(v_1, \mathbf{0})$ and $(v_3, \mathbf{0})$ which can be combined with v_2, v_4. The transitions cannot guarantee that cost-places are always properly stored up-to b in the finite control. E.g., if $b = 7$ there could be reachable markings where a cost-place holds 5 tokens, but only 3 are encoded in the finite control while 2 remain on the place. However, proper encodings as described above are always possible. Our constructions will ensure that such non-standard encodings as in this example do not change the final result. Intuitively, the reason is the following. Non-standard encodings differ from proper encodings by having more tokens on the real cost-places and fewer encoded in the finite control. However, at key points (i.e., where timed transitions happen) our constructions enforce that the real cost-places are empty, thus filtering out the computations with non-standard encodings. Furthermore, by forcing the cost-places to be empty, we ensure that all contents of cost-places are encoded in the finite control and that they are below the bound b. This makes it possible to deduct the correct place-costs during timed transitions (see Construction (part 2) below).

Lemma 8. *Let M_1, M_2 be markings of N. Then there is a computation σ using only discrete transitions s.t. $M_1 \xrightarrow{\sigma} M_2$ with $Cost(\sigma) \leq v$ if and only if in N' there are computations σ' where $f_r(M_1) \xrightarrow{\sigma'} f_{r'}(M_2)$ for every r, r' with $v \geq r$ and $r' = r - Cost(\sigma) \geq \mathbf{0}$.*

Proof. Directly from the construction of N' and induction over the lengths of σ, σ'. □

Construction (part 2). The cPN N' only encodes the behavior of N during discrete transitions. It does not encode the effect of timed transitions, nor the place-costs of delays. A crucial observation is that, in computations of N, since the maximal allowed cost v is bounded by b (componentwise), the maximal number of tokens on any cost-place must be $\leq b$ before (and thus also after) every timed transition, or else the cost would exceed the limit v. Since in N' token numbers on cost-places are encoded in the finite control up-to b, we can demand without restriction that in N' all cost-places are empty before and after every simulated timed transition. These simulated timed transitions are not encoded into N' directly, but handled in the following construction.

We define a (non-injective) function g on markings of N' which encodes the effect of a timed transition. For technical reasons we restrict the domain of g to markings of N' which are empty on all cost-places. Let $((q, r), M)$ be a marking of N' where M is empty on all cost-places. The marking $((q', r'), M') := g(((q, r), M))$ is defined by

- $q'(j, 0) = 0$ for $1 \leq j \leq m$. (No token has age 0 after time has passed.)
- $q'(j, x + 1) = q(j, x)$ for $1 \leq j \leq m$, $0 \leq x < max$. (All tokens age by 1.)
- $q'(j, max + 1) = q(j, max) + q(j, max + 1)$ for $1 \leq j \leq m$.
- $M'(j, 0) = 0$ for $1 \leq j \leq n$. (No token has age 0 after time has passed.)
- $M'(j, x + 1) = M(j, x)$ for $1 \leq j \leq n$, $0 \leq x < max$. (All tokens age by 1.)
- $M'(j, max + 1) = M(j, max) + M(j, max + 1)$ for $1 \leq j \leq n$.
- $r' = r - \sum_{j=1}^{m} \sum_{x=0}^{max+1} q(j, x) * C(p_j)$ (Deduct the correct place costs).
- $r \leq v$. (Costs above the limit v are not allowed.)
- $M(j, x) = 0$ for $1 \leq j \leq m$. (All cost-places are empty in M and thus also in M'.)

The last two conditions ensure that g is only defined for $r \leq v$ and markings where all cost-places are empty. Also g is not injective, since ages $> max$ are encoded as max+1.

Lemma 9. *Let M_1, M_2 be markings of N. Then $M_1 \rightarrow_{time} M_2$ with $Cost(M_1 \rightarrow_{time} M_2) \leq v$ if and only if in N' we have $f_{r'}(M_2) = g(f_r(M_1))$ for every r, r' with $v \geq r$ and $r' = r - Cost(M_1 \rightarrow_{time} M_2) \geq \mathbf{0}$.*

Proof. Since $Cost(M_1 \rightarrow_{time} M_2) \leq v$, the content of the cost-places in M_1 and M_2 is below b. Thus the cost places are completely encoded in the finite control in $f_r(M_1)$ and $f_{r'}(M_2)$, while the cost-places themselves are empty. Therefore the remaining cost r' is computed correctly, and depends only on the finite control. The rest follows directly from the definition of N' and g. □

Lemma 10. *The following three conditions are equivalent.*

1. *The PTPN N with M_0, F and v is a positive instance of cost-threshold.*
2. *There exist markings $M_1, \ldots, M_{j-1}$ and $A_1, \ldots, A_j$ of N with $M_i \xrightarrow{\sigma_i} A_{i+1}$ and $A_i \rightarrow_{time} M_i$ and $A_j \in F$ where σ_i consists only of discrete transitions and $\sum_{i=0}^{j-1} Cost(\sigma_i) + \sum_{i=1}^{j-1} Cost(A_i \rightarrow_{time} M_i) \leq v$.*
3. *There exist markings $M'_0 = f_v(M_0)$ and $M'_1, \ldots, M'_{j-1}$ and $A'_1, \ldots, A'_j$ of N' with $M'_i \xrightarrow{\sigma'_i} A'_{i+1}$ and $M'_i = g(A'_i)$ and $A'_j \in F'$.*

Proof. Conditions 1. and 2. are equivalent by definition. For the equivalence of 2. and 3. let $r_0 = v$, $a_{i+1} = r_i - Cost(\sigma_i)$ and $r_i = a_i - Cost(A_i \rightarrow_{time} M_i)$. Then $M'_i = f_{r_i}(M_i)$ and $A'_i = f_{a_i}(A_i)$. The proof follows directly from Lemmas 8 and 9.

In the following we consider an extended notion of computations of N' which contain both normal transitions of N' and applications of function g.

Let F^i be the set of markings M' of N' where: (1) M' can reach F' via an extended computation that starts with an application of function g and contains i applications of function g (i.e., i encoded timed transitions), and (2) M' is 0 on all cost-places, i.e., $M'((j,x)) = 0$ for $1 \leq j \leq m$ and all x. The set G^i is defined analogously, except that the extended computation must start with a normal transition of N'. We have $G^0 = \{M' = (\mathbf{0}, \boldsymbol{x}) \mid M' \xrightarrow{*} M'' \in F'\}$ and $G^i = \{M' = (\mathbf{0}, \boldsymbol{x}) \mid M' \xrightarrow{*} M'' = (\mathbf{0}, \boldsymbol{x}') \in F^i\}$ for $i > 0$, and $F^{i+1} = g^{-1}(G^i)$ for $i \geq 0$.

Since F is upward-closed w.r.t. the (multiset-inclusion) order on markings of N, the set F' is upward-closed w.r.t. the order on markings of N'. Therefore, all sets F^i and G^i are upward-closed w.r.t. the free-places (i.e., their projection on the free-places), by monotonicity of Petri nets and the monotonicity of function g. Furthermore, the markings in F^i and G^i are all zero on the cost-places. So F^i and G^i can be characterized by their finitely many minimal elements. The finitely many minimal elements of G^0 (resp. G^i) can be computed from F' (resp. F^i) by generalized Petri net reachability (Theorem 5) and the Valk-Jantzen Theorem (Theorem 4) applied to the Petri net N'.

The step from G^i to F^{i+1} encodes the effect of a timed transition. Let G^i_{min} be the finite set of minimal elements of G^i. We compute the finite set F^{i+1}_{min} of minimal elements of F^{i+1} as $F^{i+1}_{min} := g^{-1}(G^i_{min})$. Even though g is not injective, $g^{-1}(G^i_{min})$ is still finite, because G^i_{min} is finite and every marking in G^i_{min} contains only finitely many tokens. Finally, let $H^l := \bigcup_{i \leq l} F^i$. Now we can prove the main theorem.

Proof. (of Theorem 7) Given the instance of cost-threshold, we construct the Petri N' and the sets of markings H^l for $l = 0, 1, 2, \ldots$. The markings in H^l are all empty on the cost-places, but the sets H^l are upward-closed w.r.t. the free-places. Thus, by Dickson's Lemma [8], the construction converges at H^y for some finite index y. By the construction of F^i we obtain that the set H^y contains all markings which are empty on the cost-places and which can reach F' via some extended computation that begins with an application of function g and costs at most v.

Thus, by Lemma 10, the instance of cost-threshold is positive iff $M'_0 = f_v(M_0)$ can reach $H^y \cup F'$ by normal transitions in Petri net N'. This is decidable by Theorem 5. □

It was shown in [15] that reachability, and thus also coverability, is decidable for Petri nets with one inhibitor arc. However, our result on PTPN also yields a more direct proof of decidability of the coverability problem.

Corollary 11. *Given a Petri net with one inhibitor arc, an initial marking M_0 and an upward-closed set of final markings F, it is decidable if $M_0 \rightarrow^* F$.*

Proof. We reduce the coverability problem for a Petri net with one inhibitor arc N to the cost threshold problem for a PPN N' which is constructed as follows. Let (p, t) be the inhibitor arc. We remove the inhibitor arc, make t a timed transition and all other transitions instantaneous transitions. Place p has cost 1 while all other places and transitions have cost 0. In N', any computation with cost 0 has the property that transition t is only used if place p is empty. Therefore, in N' the set F is reachable with cost 0 iff the set F is reachable in N.

Furthermore, by Lemma 3, the cost threshold problem for PPN is reducible to the cost threshold problem for PTPN. Since F is upward-closed and all costs are non-negative the problem is decidable by Theorem 7. □

Now we consider the cost-threshold reachability problem. This is the case where F is not upward-closed, but a fixed marking, i.e., $F = \{M_f\}$.

Theorem 12. *The cost-threshold reachability problem is undecidable for PTPN, even if all costs are zero.*

Proof. Directly from the undecidability of the reachability problem for TPN [16]. □

However, for the simpler PPN model, the cost-threshold reachability problem is equivalent to the reachability problem for Petri nets with one inhibitor arc. The reduction from Petri nets with one inhibitor arc to the cost-threshold reachability problem for PPN is similar to the construction in Corollary 11. Now we show the other direction.

Theorem 13. *The cost-threshold reachability problem for PPN is decidable.*

Proof. Consider a PPN $N = (P, T, T_0, T_1, In, Out, C)$, cost v, initial marking M_0 and a target marking M_f. We construct a Petri net $N' = (P', Q, T', In', Out')$, with inhibitor arcs (p_0, t) (for many different transitions t, but always the same place p_0), an initial marking M'_0 and markings $M'_f, \widehat{M_f}$ s.t. $M'_0 \rightarrow^* M'_f \rightarrow \widehat{M_f}$ in N' iff N is a positive instance of cost-threshold.

Construction. Let $v = (v_1, \ldots, v_k) \in \mathbb{N}_\omega^k$. First, for every i, if $v_i = \omega$ then we replace v_i by 0 and set the i-th component of the cost function C of N to 0, too. This does not change the result of the cost-threshold problem. Thus we can assume without restriction that $v = (v_1, \ldots, v_k) \in \mathbb{N}^k$. Let $b := \max_{1 \le i \le k} v_i \in \mathbb{N}$.

The set of places $P = \{p_1, \ldots, p_n\}$ of N can be divided into two disjoint subsets $P = P_1 \uplus P_0$ where $\forall p \in P_0. C(p) = 0$ and $\forall p \in P_1. C(p) > 0$. We call the places in P_0 *free-places* and the places in P_1 *cost-places*. Let $m := |P_1|$. Without restriction let $p_1, \ldots, p_m$ be cost-places and $p_{m+1}, \ldots, p_n$ be free-places.

The set of control-states Q of N' is defined as a tuple $Q = Q' \times R$, where $Q' = \{0, \ldots, b\}^m$. The intuition is that for every cost-place p_j the number of tokens on p_j is partly stored in the j-th component of Q' up-to a maximum of b, while only the rest is stored on the place directly. So if place p_j contains $x \ge b$ tokens then this is encoded as just $x - b$ tokens on place p_j and a control-state where the j-th component is b. If place p_j contains $x \leq b$ tokens then this is encoded as just 0 tokens on place p_j and a control-state where the j-th component is x. Furthermore, $R = \{0, \ldots, b\}^k$ and is used to store the remaining maximal allowed cost of the computation.

Let $P' := P \cup \{p_0\}$. The extra place p_0 will be used to store the sum of all cost-places. So every marking M' of N' will satisfy the invariant $M'(p_0) = \sum_{p \in P_1} M'(p)$. In particular $M'(p_0) = 0 \Leftrightarrow \forall p \in P_1. M'(p) = 0$.

The set T' of transitions of N' is defined as follows. Let $t \in T_0$. Then for every combination of vectors $v_1, v_3 \in \mathbb{N}^m$ and $v_2, v_4 \in \mathbb{N}^n$ and every control-state $(q, r) \in Q$ we have a transition $t' \in T'$ with $In(t') = ((q, r), I)$ and $Out(t') = ((q', r'), O)$ iff the following conditions are satisfied. (The intuition for the vectors v_1, v_2, v_3, v_4 is to model all possible ways of splitting the consumption/production of tokens by the transition between tokens encoded in the finite control and tokens present on the real places; similarly as in Construction (part 1) of the proof of Theorem 7.)

- $(v_1, \mathbf{0}) + v_2 = In(t)$, and $(v_3, \mathbf{0}) + v_4 = Out(t)$
- $q \geq v_1$
- $q' = q - v_1 + v_3 \leq (b, \ldots, b)$
- $r' = r - C(t) \geq \mathbf{0}$
- $I(p_j) = v_2(p_j)$ for $j \geq 1$ and $I(p_0) = \sum_{i=1}^{m} v_2(p_i)$
- $O(p_j) = v_4(p_j)$ for $j \geq 1$ and $O(p_0) = \sum_{i=1}^{m} v_4(p_i)$

Let $t \in T_1$. Then for every combination of vectors $v_1, v_3 \in \mathbb{N}^m$ and $v_2, v_4 \in \mathbb{N}^n$ and every control-state $(q, r) \in Q$ we have a transition $t' \in T'$ with $In(t') = ((q, r), I)$ and $Out(t') = ((q', r'), O)$ and inhibitor arc (p_0, t') iff the following conditions hold.

- $(v_1, \mathbf{0}) + v_2 = In(t)$, and $(v_3, \mathbf{0}) + v_4 = Out(t)$
- $q \geq v_1$
- $q' = q - v_1 + v_3 \leq (b, \ldots, b)$
- $r' = r - C(t) - \sum_{i=1}^{m} q_i * C(p_i) \geq \mathbf{0}$
- $I(p_j) = v_2(p_j)$ for $j \geq 1$ and $I(p_0) = \sum_{i=1}^{m} v_2(p_i) = 0$
- $O(p_j) = v_4(p_j)$ for $j \geq 1$ and $O(p_0) = \sum_{i=1}^{m} v_4(p_i)$

Finally, for every $(q, r) \in Q'$, we add another transition t' to T with $In(t') = ((q, r), \mathbf{0})$ and $Out(t') = ((q, \mathbf{0}), \mathbf{0})$. This makes it possible to set the remaining allowed cost to zero at any time.

For every $r \in R$ we define a mapping f_r of markings of N to markings of N'. Given a marking M of N, let $M' := f_r(M)$ be defined as follows. $M' = ((q, r), M'')$ where $q_i = \min\{M(p_i), b\}$ for $1 \leq i \leq m$ and $M''(p_i) = \max\{0, M(p_i) - b\}$ for $1 \leq i \leq m$ and $M''(p_i) = M(p_i)$ for $i > m$ and $M''(p_0) = \sum_{i=1}^{m} M''(p_i)$. This ensures that token numbers up-to b on cost-places are encoded in the control-state and only the rest is kept on the cost-places themselves. Free-places are unaffected. The parameter $r \in R$ assigns the allowed remaining cost, which is independent of M, but also stored in the finite control of M'. The initial marking M'_0 of N' is defined as $M'_0 = f_v(M_0)$ and the final marking is defined as $\widehat{M_f} = f_{\mathbf{0}}(M_f)$.

Proof of correctness. Assume that there is a computation σ of N of the form $M_0 \rightarrow M_1 \rightarrow M_2 \rightarrow \cdots \rightarrow M_f$ such that $Cost(\sigma) \leq v$. Then there is a corresponding computation σ' in N' of the form $M'_0 \rightarrow M'_1 \rightarrow M'_2 \rightarrow \cdots \rightarrow M'_f \rightarrow \widehat{M_f}$ such that

$M_i' = f_{r_i}(M_i)$, where $r_i = v - Cost(M_0 \rightarrow \cdots \rightarrow M_i)$ and $\widehat{M_f} = f_{\mathbf{0}}(M_f)$. The step $M_f' \rightarrow \widehat{M_f}$ uses the special transition that sets the remaining allowed cost to zero. The crucial observation is that whenever a timed transition $M_i \xrightarrow{t} M_{i+1}$ is used in σ then the number of tokens on every cost-place p_j in M_i is $\leq b$, because $Cost(\sigma) \leq v$. Therefore, in M_i' every cost-place p_j is empty, since all the $\leq b$ tokens are encoded into the finite control. Thus $M_i'(p_0) = 0$ and the inhibitor arc (p_0, t) does not prevent the transition from $M_i' \xrightarrow{t} M_{i+1}'$. Furthermore, the remaining allowed cost r_i is always non-negative, because, by our assumption, $v \geq Cost(\sigma)$. Thus the cost restrictions do not inhibit transitions in σ' either. Finally, we apply the special transition which sets the remaining allowed cost to zero and thus we reach $\widehat{M_f}$ as required.

In order to show the other direction, we need a reverse mapping g from markings M' of N' to markings M of N. For $M' = ((q, r, M'')$ we define $M = g(M')$ as follows. For cost-places p_j (with $1 \leq j \leq m$) we have $M(p_j) = M''(p_j) + q_j$. For free-places p_j (with $j > m$) we have $M(p_j) = M''(p_j)$. Assume now that we have a computation σ' of N' of the form $M_0' \rightarrow M_1' \rightarrow M_2' \rightarrow \cdots \rightarrow M_f' \rightarrow \widehat{M_f}$. Without restriction we can assume that the special transition which sets the remaining allowed cost to zero is used exactly only in the last step $M_f' \rightarrow \widehat{M_f}$, because the set of possible computations is monotone increasing in the allowed remaining cost. In the special case where the remaining allowed cost is already 0 in M_f' we have $M_f' = \widehat{M_f}$. There then exists a computation σ of N of the form $M_0 \rightarrow M_1 \rightarrow M_2 \rightarrow \cdots \rightarrow M_f$ such that $M_i = g(M_i')$ and $Cost(M_0 \rightarrow \cdots \rightarrow M_i) = v - r_i$, where $M_i' = ((q, r_i), M_i'')$ (for some $q \in Q'$). The crucial argument is about the timed-transition steps $M_i' \xrightarrow{t} M_{i+1}'$. The inhibitor arc (p_0, t) in N' ensures that $M_i'(p_0) = 0$. Thus all cost-places p_j are empty in M_i' and only the part up-to b which is encoded in the finite control part q remains. Therefore, we deduct the correct cost $C(t) + \sum_{i=1}^{m} q_i * C(p_i)$ from r_i to obtain r_{i+1} and so we maintain the above invariant by $Cost(M_0 \rightarrow \cdots \rightarrow M_{i+1}) = v - r_{i+1}$. So we obtain $Cost(M_0 \rightarrow \cdots \rightarrow M_f) = v - r_f \geq \mathbf{0}$ and thus $Cost(\sigma) \leq v$.

Finally, since all the inhibitor arcs in N' connect to the same place p_0, the reachability problem for N' can be reduced to the reachability problem for some Petri net with just one inhibitor arc by Lemma 6, and this is decidable by [15]. □

4 Undecidability for Negative Costs

The cost threshold coverability problem for PTPN is undecidable if negative transition costs are allowed. This holds even for the simpler PPN and one-dimensional costs.

Theorem 14. *The cost threshold problem for PPN $N = (P, T, T_0, T_1, In, Out, C)$ is undecidable even if $C(P) \subseteq \mathbb{N}$ and $C(T) \subseteq \mathbb{Z}_{\leq 0}$ and F is upward-closed.*

Proof. We prove undecidability of the problem through a reduction from the control-state reachability problem for 2-counter machines. We recall that a 2-counter machine M, operating on two counters c_0 and c_1, is a triple (Q, δ, q_{init}), where Q is a finite set of *control states*, δ is a finite set of *transitions*, and $q_{init} \in Q$ is the *initial control-state*. A transition $r \in \delta$ is a triple (q_1, op, q_2), where op is of one of the three forms (for $i = 0, 1$): (i) c_i++ which increments the value of c_i by one; (ii) c_i−− which decrements the value of c_i by one; or (iii) $c_i = 0?$ which checks whether the value of c_i is equal to zero. A *configuration* γ of M is a triple (q, x, y), where $q \in Q$ and $x, y \in \mathbb{N}$. The configuration gives the control-state together with the values of the counters c_0 and c_1. The initial configuration c_{init} is $(q_{init}, 0, 0)$. The operational semantics of M is defined in the standard manner. In the control-state reachability problem, we are given a counter machine M and a (final) control-state q_F, and check whether we can reach a configuration of the form (q_F, x, y) (for arbitrary x and y) from γ_{init}.

Given an instance of the control-state reachability problem for 2-counter machines, we derive an instance of the cost threshold coverability problem for a PPN with only non-negative costs on places and only non-positive costs on transitions; and where the threshold vector is given by (0) (i.e., the vector is one-dimensional, and its only component has value 0). We define the PPN $N = (P, T, T_0, T_1, \mathit{In}, \mathit{Out}, C)$ as follows. For each control-state $q \in Q$ we have a place $q \in P$. A token in the place q indicates that M is in the corresponding control-state. For each counter c_i we have a place $c_i \in P$ with $C(c_i) = 1$. The number of tokens in the place c_i gives the value of the corresponding counter. We define the set F as $(q_F, 0, 0)\uparrow$. Increment and decrement transitions are simulated in a straightforward manner. For a transition $r = (q_1, c_i{+}{+}, q_2) \in \delta$ there is a transition $r \in T_0$ such that $\mathit{In}(r) = \{q_1\}$, $\mathit{Out}(r) = \{q_2, c_i\}$, and $C(t) = 0$. For a transition $r = (q_1, c_i{-}{-}, q_2) \in \delta$ there is a transition $r \in T_0$ such that $\mathit{In}(r) = \{q_1, c_i\}$, $\mathit{Out}(r) = \{q_2\}$, and $C(t) = 0$. The details of simulating a zero testing transition $r = (q_1, c_i = 0?, q_2)$ are shown in Figure 1. The main idea is to put a positive cost on the counter places c_i and c_{1-i}. If the system 'cheats' and takes the transition from a configuration where the counter value c_i is positive (the corresponding place is not empty), then the transition will impose a cost which cannot be compensated in the remainder of the computation. Since the other counter c_{1-i} also has a positive cost, we will also pay an extra (unjustified) price corresponding to the number of tokens in c_{1-i}. Therefore, we use a number of auxiliary places and transitions which make it possible to reimburse unjustified cost for tokens on counter c_{1-i}. The reimbursement is carried out (at most completely, but possibly just partially) by cycling around the tokens in c_{1-i}. Concretely, we have three transitions $t_1^r, t_2^r, t_3^r \in T_0$ and two transitions $t_4^r, t_5^r \in T_1$. Furthermore, we have three extra places $p_1^r, p_2^r, p_3^r \in P$. The costs are given by $C(t_2^r) = -2, C(p_3^r) = 2, C(c_{1-i}) = C(c_{1-i}) = 1$; while the cost of the other places and transitions are all equal to 0. Intuitively, N simulates the transition r by first firing the transition t_4^r which will add a cost which is equal to the number of tokens in c_i and c_{1-i}. Notice that the (only) token in the place q_i has now been removed. This means that there is no token in any place corresponding to a control-state in M and hence the rest of the net has now become passive. We observe also that t_4^r puts a token in p_1^r. This token will enable the next phase which will make it possible to reclaim the (unjustified) cost we have for the tokens in the place c_{1-i}. Let n be the number of tokens in place c_{1-i}. Then, firing t_4^r costs n. We can now fire the transition t_2^r m times, where $m \leq n$, thus moving m tokens from c_{1-i} to p_3^r and gaining $2m$ (i.e., paying $-2m$). Eventually, t_1^r will fire enabling t_3^r. The latter can fire k times (where $k \leq m$) moving k tokens back to c_{1-i}. The places p_3^r and c_{1-i} will now contain $m - k$ resp. $n + k - m$ tokens. Finally, the transition t_5^r will fire, costing $2(m - k) + (n + k - m) = n + m - k$, moving a token to q_2, and resuming the "normal" simulation of M. The total cost ℓ

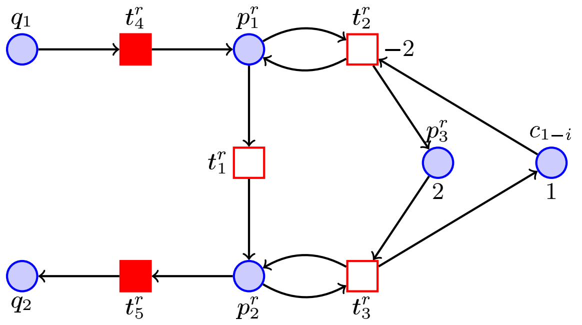


Fig. 1. Simulating zero testing in a PPN. Timed transitions are filled. The cost of a place or transition is shown only if it is different from 0.

for the whole sequence of transitions is $n - 2m + n + m - k = 2n - m - k$. This means $0 \leq \ell$ and that $\ell = 0$ only if $k = m = n$, i.e., all the tokens of c_{1-i} are moved to p_3^r and back to c_{1-i}. In other words, we can reimburse all the unjustified cost (but not more than that). This implies correctness of the construction as follows. Suppose that the given instance of the control-state reachability problem has a positive answer. Then there is a faithful simulation in N (which will eventually put a token in the place q_F). In particular, each time we perform a transition which tests the value of counter c_i, the corresponding place is indeed empty and hence we pay no cost for it. We can also choose to reimburse all the unjustified cost paid for counter c_{1-i}. Recall that all transitions are instantaneous except the ones which are part of simulations of zero testing (the ones of the form t_4^r and t_5^r). It follows that the computation has an accumulated cost equal to 0. On the other hand, suppose that the given instance of the control-state reachability problem has a negative answer. Then the only way for N to put a token in q_F is to 'cheat' during the simulation of a zero testing transition (as described above). However, in such a case either $m < n$ or $k < n$. In either case, the accumulated cost for simulating the transition is positive. Since simulations of increment and decrement transition have zero costs, and simulations of zero testing transitions have non-negative costs, the extra cost paid for cheating cannot be recovered later in the computation. This means that the accumulated cost for the whole computation will be strictly positive, and thus we have a negative instance of the cost threshold coverability problem. □

Corollary 15. *The cost threshold problem for PTPN $N = (P, T, T_0, T_1, In, Out, C)$ is undecidable even if $C(P) \subseteq \mathbb{N}$ and $C(T) \subseteq \mathbb{Z}_{\leq 0}$ and F is upward-closed.*

Proof. Directly from Theorem 14 and Lemma 3. □

5 Conclusion

We have considered Priced Timed Petri nets (PTPN), which is an extension of discrete-timed Petri nets with a cost model. We have shown decidability of the priced coverability problem when prices of places and transitions are given as vectors of natural numbers. On the other hand, allowing negative costs, even the priced coverability problem becomes undecidable even for the simpler model of Priced Petri Nets (PPNs) which is an extension of the the classical model of Petri nets with prices.

The (un)decidability results of can be extended in several ways, using constructions similar to the ones provided in the paper. For instance, if we consider a model where we allow control-states and where the place-costs depend on the control-state, then the coverability problem is undecidable for PPNs even if there are no costs on the transitions. In fact, the result can be shown using a simplified version of the proof of Theorem 14. The idea is to use the control-state to avoid paying wrongly-paid costs for the counter which is not currently tested for zero. Furthermore, if all place-costs are 0, then the cost threshold reachability/coverability problem can be encoded into the standard Petri net problems, even if transition-costs can be negative. Finally, if all places have non-positive costs then everything is still decidable, even for general integer transition costs. The reason is that, instead of negative place costs, we could in every time-passing phase 'cycle' the tokens on cost-places at most once through negative transitions. Since the costs are negative, there is an 'incentive' to do this cycling fully.

A challenging problem which we are currently considering is to extend our results to the case of dense-timed Petri nets.

References

1. Abdulla, P.A., Nylén, A.: Timed Petri Nets and BQOs. In: Colom, J.-M., Koutny, M. (eds.) ICATPN 2001. LNCS, vol. 2075, pp. 53–70. Springer, Heidelberg (2001)
2. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science 126, 183–235 (1994)
3. Alur, R., Torre, S.L., Pappas, G.J.: Optimal paths in weighted timed automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 49–62. Springer, Heidelberg (2001)
4. Berthomieu, B., Diaz, M.: Modeling and verification of time dependent systems using time Petri nets. IEEE Trans. on Software Engineering 17(3), 259–273 (1991)
5. Bouyer, P., Brihaye, T., Bruyère, V., Raskin, J.-F.: On the optimal reachability problem of weighted timed automata. Formal Methods in System Design 31(2), 135–175 (2007)
6. Bowden, F.D.J.: Modelling time in Petri nets. In: Proc. Second Australian-Japan Workshop on Stochastic Models (1996)
7. de Frutos Escrig, D., Ruiz, V.V., Alonso, O.M.: Decidability of Properties of Timed-Arc Petri Nets. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 187–206. Springer, Heidelberg (2000)
8. Dickson, L.E.: Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. Amer. J. Math. 35, 413–422 (1913)
9. Ghezzi, C., Mandrioli, D., Morasca, S., Pezzè, M.: A unified high-level Petri net formalism for time-critical systems. IEEE Trans. on Software Engineering 17(2), 160–172 (1991)
10. Jančar, P.: Decidability of a temporal logic problem for Petri nets. Theoretical Computer Science 74, 71–93 (1990)
11. Larsen, K.G., Behrmann, G., Brinksma, E., Fehnker, A., Hune, T., Pettersson, P., Romijn, J.: As cheap as possible: Efficient cost-optimal reachability for priced timed automata. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, p. 493. Springer, Heidelberg (2001)
12. Mayr, E.: An algorithm for the general Petri net reachability problem. SIAM Journal of Computing 13, 441–460 (1984)

13. Merlin, P., Farber, D.: Recoverability of communication protocols - implications of a theoretical study. IEEE Trans. on Computers, COM 24, 1036–1043 (1976)
14. Razouk, R., Phelps, C.: Performance analysis using timed Petri nets. In: Protocol Testing, Specification, and Verification, pp. 561–576 (1985)
15. Reinhardt, K.: Reachability in Petri nets with inhibitor arcs. In: Proc. RP 2008, 2 nd Workshop on Reachability Problems (2008)
16. Ruiz, V.V., Gomez, F.C., de Frutos Escrig, D.: On non-decidability of reachability for timed-arc Petri nets. In: Proc. 8th Int. Workshop on Petri Net and Performance Models (PNPM 1999), Zaragoza, Spain, 8-10 October 1999, pp. 188–196 (1999)
17. Valk, R., Jantzen, M.: The residue of vector sets with applications to decidability problems in Petri nets. Acta Inf., 21 (1985)

Delayed Nondeterminism in Continuous-Time Markov Decision Processes[⋆]

Martin R. Neuhäußer[1,2], Mariëlle Stoelinga[2], and Joost-Pieter Katoen[1,2]

[1] MOVES Group, RWTH Aachen University, Germany
[2] FMT Group, University of Twente, The Netherlands

Abstract. Schedulers in randomly timed games can be classified as to whether they use timing information or not. We consider continuous-time Markov decision processes (CTMDPs) and define a hierarchy of positional (P) and history-dependent (H) schedulers which induce strictly tighter bounds on quantitative properties on CTMDPs. This classification into time abstract (TA), total time (TT) and fully time-dependent (T) schedulers is mainly based on the kind of timing details that the schedulers may exploit. We investigate when the resolution of nondeterminism may be deferred. In particular, we show that TTP and TAP schedulers allow for delaying nondeterminism for all measures, whereas this does neither hold for TP nor for any TAH scheduler. The core of our study is a transformation on CTMDPs which unifies the speed of outgoing transitions per state.

1 Introduction

Continuous-time Markov decision processes (CTMDPs) which are also known as controlled Markov chains, have originated as continuous-time variants of finite-state probabilistic automata [1], and have been used for, among others, the control of queueing systems, epidemic, and manufacturing processes. The analysis of CTMDPs is mainly focused on determining optimal schedulers for criteria such as expected total reward and expected (long-run) average reward, cf. the survey [2].

As in discrete-time MDPs, nondeterminism in CTMDPs is resolved by schedulers. An important criterion for CTMDP schedulers is whether they use timing information or not. For time-bounded reachability objectives, e.g., timed schedulers are optimal [3]. For simpler criteria such as unbounded reachability or average reward, time-abstract (TA) schedulers will do. For such objectives, it suffices to either abstract the timing information in the CTMDP (yielding an "embedded" MDP) or to transform the CTMDP into an equivalent discrete-time MDP, see e.g., [4, p. 562] [2]. The latter process is commonly referred to as uniformization. Its equivalent on continuous-time Markov chains, a proper subclass of CTMDPs, is pivotal to probabilistic model checking [5].

The main focus of this paper is on defining a hierarchy of positional (P) and history-dependent (H) schedulers which induce strictly tighter bounds on quantitative properties on CTMDPs. This hierarchy refines the notion of generic measurable schedulers [6]. An important distinguishing criterion is the level of detail of timing information the

⋆ Supported by the NWO projects QUPES (612.000.420) and FOCUS/BRICKS (642.000.505) (MOQS) and by the EU grants IST-004527 (ARTIST2) and FP7-ICT-2007-1 (QUASIMODO).

schedulers may exploit, e.g., the delay in the last state, total time (TT), or all individual state residence times (T).

In general, the delay to jump to a next state in a CTMDP is determined by the action selected by the scheduler on entering the current state. We investigate under which conditions this resolution of nondeterminism may be deferred. Rather than focusing on a specific objective, we consider this delayed nondeterminism for generic (measurable) properties. The core of our study is a transformation —called local uniformization— on CTMDPs which unifies the speed of outgoing transitions per state. Whereas classical uniformization [7,8,9] adds self-loops to achieve this, local uniformization uses auxiliary copy-states. In this way, we enforce that schedulers in the original and uniformized CTMDP have (for important scheduler classes) the same power, whereas classical loop-based uniformization allows a scheduler to change its decision when re-entering a state through the added self-loop. Therefore, locally uniform CTMDPs allow to defer the resolution of nondeterminism, i.e., they dissolve the intrinsic dependency between state residence times and schedulers, and can be viewed as MDPs with exponentially distributed state residence times.

In particular, we show that TTP and TAP schedulers allow to delay nondeterminism for all measures. As TTP schedulers are optimal for time-bounded reachability objectives, this shows that local uniformization preserves the probability of such objectives. Finally, we prove that TP and TAH schedulers do not allow for delaying nondeterminism. This results in a hierarchy of time-dependent schedulers and their inclusions. Moreover, we solve an open problem in [3] concerning TAP schedulers.

The paper is organized as follows: Sec. 2 introduces CTMDPs and a general notion of schedulers which is refined in Sec. 3. In Sec. 4, we define local uniformization and prove its correctness. Sec. 5 summarizes the main results and Sec. 6 proves that deferring nondeterministic choices induces strictly tighter bounds on quantitative properties.

2 Continuous-Time Markov Decision Processes

We consider CTMDPs with finite sets $\mathscr{S} = \{s_0, s_1, \dots\}$ and $Act = \{\alpha, \beta, \dots\}$ of states and actions; $Distr(\mathscr{S})$ and $Distr(Act)$ are the respective sets of probability distributions.

Definition 1 (Continuous-time Markov decision process). *A* continuous-time Markov decision process *(CTMDP) is a tuple* $\mathscr{C} = (\mathscr{S}, Act, \mathbf{R}, \nu)$ *where* $\mathscr{S}$ *and* Act *are finite, nonempty sets of* states *and* actions*,* $\mathbf{R} : \mathscr{S} \times Act \times \mathscr{S} \to \mathbb{R}_{\geq 0}$ *is a three-dimensional* rate matrix *and* $\nu \in Distr(\mathscr{S})$ *is an* initial distribution.

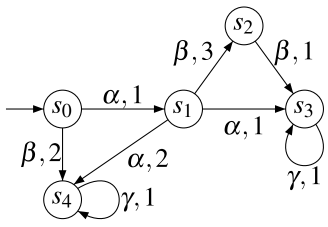


Fig. 1. A well-formed CTMDP

If $\mathbf{R}(s, \alpha, s') = \lambda$ and $\lambda > 0$, an α-transition leads from state s to state s'. λ is the *rate* of an exponential distribution which defines the transition's delay. Hence, it executes in time interval $[a, b]$ with probability $\eta_\lambda([a,b]) = \int_a^b \lambda e^{-\lambda t}\, dt$; note that η_λ directly extends to the Borel σ-field $\mathfrak{B}(\mathbb{R}_{\geq 0})$. Further, $Act(s) = \{\alpha \in Act \mid \exists s' \in \mathscr{S}.\ \mathbf{R}(s, \alpha, s') > 0\}$ is the set of *enabled* actions in state s and

$E(s,\alpha) = \sum_{s' \in \mathscr{S}} \mathbf{R}(s,\alpha,s')$ is its *exit rate* under action α. A CTMDP is *well-formed* if $Act(s) \neq \emptyset$ for all $s \in \mathscr{S}$. As this is easily achieved by adding self-loops, we restrict to well-formed CTMDPs. The time-abstract *branching probabilities* are captured by a matrix $\mathbf{P}$ where $\mathbf{P}(s,\alpha,s') = \frac{\mathbf{R}(s,\alpha,s')}{E(s,\alpha)}$ if $E(s,\alpha) > 0$ and $\mathbf{P}(s,\alpha,s') = 0$ otherwise.

Example 1. If α is chosen in state s_0 of the CTMDP in Fig. 1, we enter state s_1 after a delay which is exponentially distributed with rate $\mathbf{R}(s_0,\alpha,s_1) = E(s_0,\alpha) = 1$. For state s_1 and action α, a *race* decides which of the two α-transitions executes; in this case, the *sojourn time* of state s_1 is exponentially distributed with rate $E(s_1,\alpha) = 3$. The time-abstract probability to move to state s_3 is $\frac{\mathbf{R}(s_1,\alpha,s_3)}{E(s_1,\alpha)} = \frac{1}{3}$.

2.1 The Probability Space

In a CTMDP $\mathscr{C} = (\mathscr{S}, Act, \mathbf{R}, \nu)$, a *finite path* π of length n (denoted $|\pi| = n$) is a sequence $\pi = s_0 \xrightarrow{\alpha_0,t_0} s_1 \xrightarrow{\alpha_1,t_1} \cdots \xrightarrow{\alpha_{n-1},t_{n-1}} s_n$ where $s_i \in \mathscr{S}$, $\alpha_i \in Act$ and $t_i \in \mathbb{R}_{\geq 0}$. With $\pi[k] = s_k$ and $\delta(\pi,k) = t_k$ we refer to its k-th state and the associated sojourn time. Accordingly, $\Delta(\pi) = \sum_{k=0}^{n-1} t_k$ is the total time spent on π. Finally, $\pi{\downarrow} = s_n$ denotes the last state of π and $\pi[i..k]$ is the path infix $s_i \xrightarrow{\alpha_i,t_i} \cdots \xrightarrow{\alpha_{k-1},t_{k-1}} s_k$. The path π is built by a state and a sequence of *combined transitions* from the set $\Omega = Act \times \mathbb{R}_{\geq 0} \times \mathscr{S}$: It is the concatenation $s_0 \circ m_0 \circ m_1 \cdots \circ m_{n-1}$ where $m_i = (\alpha_i, t_i, s_{i+1}) \in \Omega$. Thus $Paths^n(\mathscr{C}) = \mathscr{S} \times \Omega^n$ yields the set of paths of length n in $\mathscr{C}$ and analogously, $Paths^\star(\mathscr{C})$, $Paths^\omega(\mathscr{C})$ and $Paths(\mathscr{C})$ denote the sets of finite, infinite and all paths of $\mathscr{C}$. We use $abs(\pi) = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_{n-1}} s_n$ to refer to the *time-abstract* path induced by π and define $Paths^n_{abs}(\mathscr{C})$ accordingly. For simplicity, we omit the reference to $\mathscr{C}$ wherever possible.

Events in $\mathscr{C}$ are measurable sets of paths; as paths are sequences of combined transitions, we first define the σ-field $\mathfrak{F} = \sigma\big(\mathfrak{F}_{Act} \times \mathfrak{B}(\mathbb{R}_{\geq 0}) \times \mathfrak{F}_{\mathscr{S}}\big)$ on subsets of Ω where $\mathfrak{F}_{\mathscr{S}} = 2^{\mathscr{S}}$ and $\mathfrak{F}_{Act} = 2^{Act}$. Based on $(\Omega, \mathfrak{F})$, we derive the product σ-field $\mathfrak{F}_{Paths^n} = \sigma\big(\{S_0 \times M_0 \times \cdots \times M_{n-1} \mid S_0 \in \mathfrak{F}_{\mathscr{S}}, M_i \in \mathfrak{F}\}\big)$ for paths of length n. Finally, the cylinder-set construction [10] allows to extend this to a σ-field over infinite paths: A set $B \in \mathfrak{F}_{Paths^n}$ is a *base* of the infinite *cylinder* C if $C = Cyl(B) = \{\pi \in Paths^\omega \mid \pi[0..n] \in B\}$. Now the desired σ-field $\mathfrak{F}_{Paths^\omega}$ is generated by the set of all cylinders, i.e. $\mathfrak{F}_{Paths^\omega} = \sigma\big(\bigcup_{n=0}^\infty \{Cyl(B) \mid B \in \mathfrak{F}_{Paths^n}\}\big)$. For an in-depth discussion, we refer to [10,11,6].

2.2 Probability Measure

The probability measures on $\mathfrak{F}_{Paths^n}$ and $\mathfrak{F}_{Paths^\omega}$ are defined using schedulers that resolve the nondeterminism in the underlying CTMDP.

Definition 2 (Generic measurable scheduler). *Let $\mathscr{C}$ be a CTMDP with actions in Act. A* generic scheduler *on $\mathscr{C}$ is a mapping $D : Paths^\star \times \mathfrak{F}_{Act} \to [0,1]$ where $D(\pi,\cdot) \in Distr(Act(\pi{\downarrow}))$. It is* measurable *(gm-scheduler) iff the functions $D(\cdot,A) : Paths^\star \to [0,1]$ are measurable for all $A \in \mathfrak{F}_{Act}$.*

On reaching state s_n via path π, $D(\pi,\cdot)$ defines a distribution over $Act(s_n)$ and thereby resolves the nondeterminism in state s_n. The measurability condition in Def. 2 states that $\{\pi \in Paths^\star | D(\pi, A) \in B\} \in \mathfrak{F}_{Paths^\star}$ for all $A \in \mathfrak{F}_{Act}$ and $B \in \mathfrak{B}([0,1])$; it is required for the Lebesgue-integral in Def. 4 to be well-defined.

To define a probability measure on sets of paths, we proceed stepwise and first derive a probability measure on sets of combined transitions:

Definition 3 (Probability on combined transitions). *Let $\mathscr{C}=(\mathscr{S}, Act, \mathbf{R}, \nu)$ be a CTMDP and D a gm-scheduler on $\mathscr{C}$. For all $\pi \in Paths^\star(\mathscr{C})$, define the probability measure $\mu_D(\pi, \cdot) : \mathfrak{F} \to [0,1]$ where*

$$\mu_D(\pi, M) = \int_{Act} D(\pi, d\alpha) \int_{\mathbb{R}_{\geq 0}} \eta_{E(\pi\downarrow, \alpha)}(dt) \int_{\mathscr{S}} \mathbf{I}_M(\alpha, t, s') \ \mathbf{P}(s, \alpha, ds'). \tag{1}$$

Here, $\mathbf{I}_M$ denotes the characteristic function of $M \in \mathfrak{F}$. A proof that $\mu_D(\pi, \cdot)$ is indeed a probability measure can be found in [6, Lemma 1]. Intuitively, $\mu_D(\pi, M)$ is the probability to continue on path π under scheduler D with a combined transition in M. With $\mu_D(\pi, \cdot)$ and ν, we can define the probability of sets of paths:

Definition 4 (Probability measure). *Let $\mathscr{C} = (\mathscr{S}, Act, \mathbf{R}, \nu)$ be a CTMDP and D a gm-scheduler on $\mathscr{C}$. For $n \geq 0$, we define the probability measures $Pr^n_{\nu,D}$ on the measurable space $(Paths^n, \mathfrak{F}_{Paths^n})$ inductively as follows:*

$$Pr^0_{\nu,D} : \mathfrak{F}_{Paths^0} \to [0,1] \ : \Pi \mapsto \sum_{s \in \Pi} \nu(\{s\}) \quad \text{and for } n > 0$$

$$Pr^n_{\nu,D} : \mathfrak{F}_{Paths^n} \to [0,1] \ : \Pi \mapsto \int_{Paths^{n-1}} Pr^{n-1}_{\nu,D}(d\pi) \int_{\Omega} \mathbf{I}_\Pi(\pi \circ m) \ \mu_D(\pi, dm).$$

Intuitively, we measure sets of paths Π of length n by multiplying the probability $Pr^{n-1}_{\nu,D}(d\pi)$ of path prefixes π with the probability $\mu_D(\pi, dm)$ of a combined transition m that extends π to a path in Π. Together, the measures $Pr^n_{\nu,D}$ extend to a unique measure on $\mathfrak{F}_{Paths^\omega}$: if $B \in \mathfrak{F}_{Paths^n}$ is a measurable base and $C = Cyl(B)$, we define $Pr^\omega_{\nu,D}(C) = Pr^n_{\nu,D}(B)$. Due to the inductive definition of $Pr^n_{\nu,D}$, the Ionescu–Tulcea extension theorem [10] is applicable and yields a unique extension of $Pr^\omega_{\nu,D}$ from cylinders to arbitrary sets in $\mathfrak{F}_{Paths^\omega}$.

As we later need to split a set of paths into a set of prefixes I and a set of suffixes Π, we define the set of path prefixes of length $k > 0$ by $PPref^k = (\mathfrak{F}_{\mathscr{S}} \times \mathfrak{F}_{Act} \times \mathfrak{B}(\mathbb{R}_{\geq 0}))^k$ and provide a probability measure on its σ-field $\mathfrak{F}_{PPref^k}$:

Definition 5 (Prefix measure). *Let $\mathscr{C} = (\mathscr{S}, Act, \mathbf{R}, \nu)$ be a CTMDP and D a gm-scheduler on $\mathscr{C}$. For $I \in \mathfrak{F}_{PPref^k}$ and $k > 0$, define*

$$\mu^k_{\nu,D}(I) = \int_{Paths^{k-1}} Pr^{k-1}_{\nu,D}(d\pi) \int_{Act} D(\pi, d\alpha) \int_{\mathbb{R}_{\geq 0}} \mathbf{I}_I(\pi \xrightarrow{\alpha,t}) \ \eta_{E(\pi\downarrow,\alpha)}(dt).$$

As $Pr^{k-1}_{\nu,D}$ is a probability measure, so is $\mu^k_{\nu,D}$. If $I \in \mathfrak{F}_{PPref^k}$ and $\Pi \in \mathfrak{F}_{Paths^n}$, their concatenation is the set $I \times \Pi \in \mathfrak{F}_{Paths^{k+n}}$; its probability $Pr^{k+n}_{\nu,D}(I \times \Pi)$ is obtained by multiplying the measure of prefixes $i \in I$ with the suffixes in Π:

Lemma 1. *Let $\Pi \in \mathfrak{F}_{Paths^n}$ and $I \in \mathfrak{F}_{PPref^k}$. If $i = s_0 \xrightarrow{\alpha_0, t_0} \cdots s_{k-1} \xrightarrow{\alpha_{k-1}, t_{k-1}}$, define $\nu_i = \mathbf{P}(s_{k-1}, \alpha_{k-1}, \cdot)$ and $D_i(\pi, \cdot) = D(i \circ \pi, \cdot)$. Then*

$$Pr^{k+n}_{\nu,D}(I \times \Pi) = \int_{PPref^k} \mu^k_{\nu,D}(di) \int_{Paths^n} \mathbf{I}_{I \times \Pi}(i \circ \pi) \ Pr^n_{\nu_i, D_i}(d\pi).$$

Lemma 1 justifies to split sets of paths and to measure the components of the resulting Cartesian product; therefore, it abstracts from the inductive definition of $Pr^n_{\nu,D}$.

3 Scheduler Classes

Section 2.2 defines the probability of sets of paths w.r.t. a gm-scheduler. However, this does not fully describe a CTMDP, as a single scheduler is only one way to resolve nondeterminism. Therefore we define scheduler classes according to the information that is available when making a decision. Given an event $\Pi \in \mathfrak{F}_{Paths^\omega}$, a scheduler class induces a set of probabilities which reflects the CTMDP's possible behaviours. In this paper, we investigate which classes in Fig. 2 preserve minimum and maximum probabilities if nondeterministic choices are delayed.

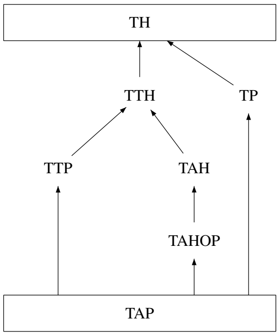
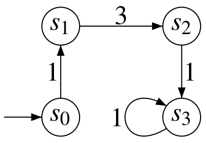


Fig. 2. Scheduler classes

As proved in [6], the most general class is the set of all gm-schedulers: If paths $\pi_1, \pi_2 \in Paths^\star$ of a CTMDP end in state s, a gm-scheduler $D : Paths^\star \times \mathfrak{F}_{Act} \to [0,1]$ may yield different distributions $D(\pi_1, \cdot)$ and $D(\pi_2, \cdot)$ over the next action, depending on the entire histories π_1 and π_2. We call this the class of *timed, history dependent (TH)* schedulers.

On the contrary, D is a *time-abstract positional* (TAP) scheduler, if $D(\pi_1, \cdot) = D(\pi_2, \cdot)$ for all $\pi_1, \pi_2 \in Paths^\star$ that end in the same state. As $D(\pi, \cdot)$ only depends on the current state, it is specified by a mapping $D : \mathscr{S} \to Distr(Act)$.

Example 2. For TAP scheduler D with $D(s_0) = \{\alpha \mapsto 1\}$ and $D(s_1) = \{\beta \mapsto 1\}$, the induced stochastic process of the CTMDP in Fig. 1 is the CTMC depicted in Fig. 3. Note that in general, randomized schedulers do not yield CTMCs as the induced sojourn times are hyper-exponentially distributed.


$$\begin{tikzcd} s_1 \arrow[r, "3"] & s_2 \arrow[d, "1"] \\ \arrow[r] & s_0 \arrow[ul, "1"] & s_3 \arrow[loop left, "1"] \end{tikzcd}$$
Fig. 3. Induced CTMC

For TAHOP schedulers, the decision may depend on the current state s and the length of π_1 and π_2 (*hop-counting schedulers*); accordingly, they are isomorphic to mappings $D : \mathscr{S} \times \mathbb{N} \to Distr(Act)$. Moreover, D is a *time-abstract history-dependent* scheduler (TAH), if $D(\pi_1, \cdot) = D(\pi_2, \cdot)$ for all histories $\pi_1, \pi_2 \in Paths^\star$ with $abs(\pi_1) = abs(\pi_2)$; given history π, TAH schedulers may decide based on the sequence of states and actions in $abs(\pi)$. In [3], the authors show that TAHOP and TAH induce the same probability bounds for timed reachability which are tighter than the bounds induced by TAP.

Time-dependent scheduler classes generally induce probability bounds that exceed those of the corresponding time-abstract classes [3]: If we move from state s to s', a *timed positional* scheduler (TP) yields a distribution over $Act(s')$ which depends on s' and the time to go from s to s'; thus TP extends TAP with information on the delay of the last transition.

Similarly, *total time history-dependent* schedulers (TTH) extend TAH with information on the time that passed up to the current state: If $D \in TTH$ and $\pi_1, \pi_2 \in Paths^\star$ are histories with $abs(\pi_1) = abs(\pi_2)$ and $\Delta(\pi_1) = \Delta(\pi_2)$, then $D(\pi_1, \cdot) = D(\pi_2, \cdot)$. Note that $TTH \subseteq TH$, as TTH schedulers may depend on the accumulated time but not on sojourn times in individual states of the history. Generally the probability bounds of TTH are less strict than those of TH.

Table 1. Proposed scheduler classes for CTMDPs

	scheduler class	scheduler signature
time abstract	positional (TAP)	$D : \mathscr{S} \to Distr(Act)$
	hop-counting (TAHOP)	$D : \mathscr{S} \times \mathbb{N} \to Distr(Act)$
	time abstract history dependent (TAH)	$D : Paths^{\star}_{abs} \to Distr(Act)$
time dependent	timed history dependent (TH)	full timed history $D : Paths^{\star} \to Distr(Act)$
	total time history dependent (TTH)	sequence of states & total time $D : Paths^{\star}_{abs} \times \mathbb{R}_{\geq 0} \to Distr(Act)$
	total time positional (TTP)	last state & total time $D : \mathscr{S} \times \mathbb{R}_{\geq 0} \to Distr(Act)$
	timed positional (TP)	last state & delay of last transition $D : \mathscr{S} \times \mathbb{R}_{\geq 0} \to Distr(Act)$

In this paper, we focus on *total time positional* schedulers (TTP) which are given by mappings $D : \mathscr{S} \times \mathbb{R}_{\geq 0} \to Distr(Act)$. They are similar to TTH schedulers but abstract from the state-history. For $\pi_1, \pi_2 \in Paths^{\star}$, $D(\pi_1, \cdot) = D(\pi_2, \cdot)$ if π_1 and π_2 end in the same state and have the same simulated time $\Delta(\pi_1) = \Delta(\pi_2)$. TTP schedulers are of particular interest, as they induce optimal bounds w.r.t. timed reachability: To see this, consider the probability to reach a set of goal states $G \subseteq \mathscr{S}$ within t time units. If state s is reached via $\pi \in Paths^{\star}$ (without visiting G), the maximal probability to enter G is given by a scheduler which maximizes the probability to reach G from state s within the remaining $t - \Delta(\pi)$ time units. Obviously, a TTP scheduler is sufficient in this case.

Example 3. For $t \in \mathbb{R}_{\geq 0}$, let the TTP-scheduler D for the CTMDP of Fig. 1 be given by $D(s_0, 0) = \{\alpha \mapsto 1\}$ and $D(s_1, t) = \{\alpha \mapsto 1\}$ if $t \leq 0.64$ and $D(s_1, t) = \{\beta \mapsto 1\}$, otherwise. It turns out that D maximizes the probability to reach s_3 within time t. For now, we only note that the probability induced by D is obtained by the gm-scheduler $D'(\pi) = D(\pi{\downarrow}, \Delta(\pi))$.

Note that we can equivalently specify any gm-scheduler $D : Paths^{\star} \times \mathfrak{F}_{Act} \to [0,1]$ as a mapping $D' : Paths^{\star} \to Distr(Act)$ by setting $D'(\pi)(A) = D(\pi, A)$ for all $\pi \in Paths^{\star}$ and $A \in \mathfrak{F}_{Act}$; to further simplify notation, we also use $D(\pi, \cdot)$ to refer to this distribution.

Definition 6 (Scheduler classes). *Let $\mathscr{C}$ be a CTMDP and D a gm-scheduler on $\mathscr{C}$. For $\pi, \pi' \in Paths^{\star}(\mathscr{C})$, the scheduler classes are defined as follows:*

$$\begin{aligned}
D{\in}TAP &\iff \forall \pi, \pi'. \pi{\downarrow} = \pi'{\downarrow} \Rightarrow D(\pi, \cdot) = D(\pi', \cdot)\\
D{\in}TAHOP &\iff \forall \pi, \pi'. \big(\pi{\downarrow} = \pi'{\downarrow} \wedge |\pi| = |\pi'|\big) \Rightarrow D(\pi, \cdot) = D(\pi', \cdot)\\
D{\in}TAH &\iff \forall \pi, \pi'. abs(\pi) = abs(\pi') \Rightarrow D(\pi, \cdot) = D(\pi', \cdot)\\
D{\in}TTH &\iff \forall \pi, \pi'. \big(abs(\pi){=}abs(\pi') \wedge \Delta(\pi){=}\Delta(\pi')\big) \Rightarrow D(\pi, \cdot){=}D(\pi', \cdot)\\
D{\in}TTP &\iff \forall \pi, \pi'. \big(\pi{\downarrow}{=}\pi'{\downarrow} \wedge \Delta(\pi){=}\Delta(\pi')\big) \Rightarrow D(\pi, \cdot){=}D(\pi', \cdot)\\
D{\in}TP &\iff \forall \pi, \pi'. \big(\pi{\downarrow}{=}\pi'{\downarrow} \wedge \delta(\pi, |\pi{-}1|){=}\delta(\pi', |\pi'{-}1|)\big) \Rightarrow D(\pi, \cdot){=}D(\pi', \cdot).
\end{aligned}$$

Def. 6 justifies to restrict the domain of the schedulers to the information the respective class exploits. In this way, we obtain the characterization in Table 1. We now come to the transformation on CTMDPs that unifies the speed of outgoing transitions and thereby allows to defer the resolution of nondeterministic choices.

4 Local Uniformization

Generally, the exit rate of a state depends on the action that is chosen by the scheduler when entering the state. This dependency requires that the scheduler decides directly when entering a state, as otherwise the state's sojourn time distribution is not well-defined. An exception to this are *locally uniform* CTMDPs which allow to delay the scheduler's choice up to the point when the state is left:

Definition 7 (Local uniformity). *A CTMDP* $(\mathscr{S}, Act, \mathbf{R}, \nu)$ *is* locally uniform *iff there exists* $u : \mathscr{S} \to \mathbb{R}_{>0}$ *such that* $E(s,\alpha) = u(s)$ *for all* $s \in \mathscr{S}, \alpha \in Act(s)$.

In locally uniform CTMDPs the exit rates are state-wise constant with rate $u(s)$; hence, they do not depend on the action that is chosen. Therefore locally uniform CTMDPs allow to delay the scheduler's decision until the current state is left. To generalize this idea, we propose a transformation on CTMDPs which attains local uniformity; further, in Sec. 4.2 we investigate as to which scheduler classes local uniformization preserves quantitative properties.

Definition 8 (Local uniformization). *Let* $\mathscr{C} = (\mathscr{S}, Act, \mathbf{R}, \nu)$ *be a CTMDP and define* $u(s) = \max\{E(s,\alpha) \mid \alpha \in Act\}$ *for all* $s \in \mathscr{S}$. *Then* $\overline{\mathscr{C}} = (\overline{\mathscr{S}}, Act, \overline{\mathbf{R}}, \overline{\nu})$ *is the locally uniform CTMDP induced by* $\mathscr{C}$ *where* $\overline{\mathscr{S}} = \mathscr{S} \mathbin{\dot\cup} \mathscr{S}_{cp}$, $\mathscr{S}_{cp} = \{s^\alpha \mid E(s,\alpha) < u(s)\}$ *and*

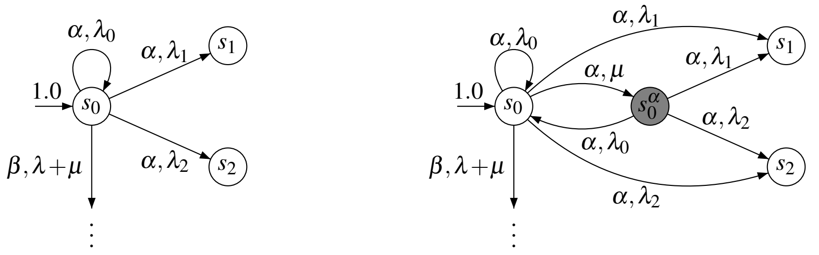
$$\overline{\mathbf{R}}(s,\alpha,s') = \begin{cases} \mathbf{R}(s,\alpha,s') & \text{if } s,s' \in \mathscr{S} \\ \mathbf{R}(t,\alpha,s') & \text{if } s = t^\alpha \wedge s' \in \mathscr{S} \\ u(s) - E(s,\alpha) & \text{if } s \in \mathscr{S} \wedge s' = s^\alpha \\ 0 & \text{otherwise.} \end{cases}$$

Further, $\overline{\nu}(s) = \nu(s)$ *if* $s \in \mathscr{S}$ *and* 0*, otherwise.*

Local uniformization is done for each state s separately with uniformization rate $u(s)$. If the exit rate of s under action α is less than $u(s)$, we introduce a copy-state s^α and an α-transition which carries the missing rate $\mathbf{R}(s,\alpha,s^\alpha) = u(s) - E(s,\alpha)$. Regarding s^α, only the outgoing α-transitions of s carry over to s^α. Hence s^α is deterministic in the sense that $Act(s^\alpha) = \{\alpha\}$.

Example 4. Consider the fragment CTMDP in Fig. 4(a) where $\lambda = \sum \lambda_i$ and $\mu > 0$. It is not locally uniform as $E(s_0,\alpha) = \lambda$ and $E(s_0,\beta) = \lambda + \mu$. Applying our transformation, we obtain the locally uniform CTMDP in Fig. 4(b).

Local uniformization of $\mathscr{C}$ introduces new states and transitions in $\overline{\mathscr{C}}$. The paths in $\overline{\mathscr{C}}$ reflect this and differ from those of $\mathscr{C}$; more precisely, they may contain sequences of transitions $s \xrightarrow{\alpha,t} s^\alpha \xrightarrow{\alpha,t'} s'$ where s^α is a copy-state. Intuitively, if we identify s and s^α, this corresponds to a single transition $s \xrightarrow{\alpha,t+t'} s'$ in $\mathscr{C}$. To formalize this correspondence,



(a) Fragment of a non-uniform CTMDP. (b) Local uniformization of state s_0.

Fig. 4. How to obtain locally uniform CTMDPs by introducing copy states

we derive a mapping *merge* on all paths $\overline{\pi} \in Paths^\star(\overline{\mathscr{C}})$ with $\overline{\pi}[0], \overline{\pi}\downarrow \in \mathscr{S}$: If $|\overline{\pi}| = 0$, $merge(\overline{\pi}) = \overline{\pi}[0]$. Otherwise, let

$$merge\big(s \xrightarrow{\alpha,t} \overline{\pi}\big) = \begin{cases} s \xrightarrow{\alpha,t} merge(\overline{\pi}) & \text{if } \overline{\pi}[0] \in \mathscr{S} \\ merge(s \xrightarrow{\alpha,t+t'} \overline{\pi}') & \text{if } \overline{\pi} = s^\alpha \xrightarrow{\alpha,t'} \overline{\pi}'. \end{cases}$$

Naturally, *merge* extends to infinite paths if we do not require $\overline{\pi}\downarrow \in \mathscr{S}$; further, merging a set of paths $\overline{\Pi}$ is defined element-wise and denoted $merge(\overline{\Pi})$.

Example 5. Let $\overline{\pi} = s_0 \xrightarrow{\alpha_0,t_0} s_0^{\alpha_0} \xrightarrow{\alpha_0,t_0'} s_1 \xrightarrow{\alpha_1,t_1} s_2 \xrightarrow{\alpha_2,t_2} s_2^{\alpha_2} \xrightarrow{\alpha_2,t_2'} s_3$ be a path in $\overline{\mathscr{C}}$. Then $merge(\overline{\pi}) = s_0 \xrightarrow{\alpha_0,t_0+t_0'} s_1 \xrightarrow{\alpha_1,t_1} s_2 \xrightarrow{\alpha_2,t_2+t_2'} s_3$.

For the reverse direction, we map sets of paths in $\mathscr{C}$ to sets of paths in $\overline{\mathscr{C}}$; formally, if $\Pi \subseteq Paths(\mathscr{C})$ we define $extend(\Pi) = \{\overline{\pi} \in Paths(\overline{\mathscr{C}}) \mid merge(\overline{\pi}) \in \Pi\}$.

Lemma 2. *Let $\mathscr{C}$ be a CTMDP and $\Pi_1, \Pi_2, \dots \subseteq Paths(\mathscr{C})$. Then*

1. *$\Pi_1 \subseteq \Pi_2 \Longrightarrow extend(\Pi_1) \subseteq extend(\Pi_2)$,*
2. *$\Pi_1 \cap \Pi_2 = \emptyset \Longrightarrow extend(\Pi_1) \cap extend(\Pi_2) = \emptyset$ and*
3. *$\bigcup extend(\Pi_k) = extend\big(\bigcup \Pi_k\big)$.*

Our goal is to construct gm-schedulers such that the path probabilities in $\mathscr{C}$ and $\overline{\mathscr{C}}$ are equal. Therefore, we first adopt a local view and prove that the probability of a single step in $\mathscr{C}$ equals the probability of the corresponding steps in $\overline{\mathscr{C}}$.

4.1 One-Step Correctness of Local Uniformization

Consider the CTMDP in Fig. 4(a) where $\lambda = \sum \lambda_i$. Assume that action α is chosen in state s_0; then $\frac{\mathbf{R}(s_0,\alpha,s_i)}{E(s_0,\alpha)} = \frac{\lambda_i}{\lambda}$ is the probability to move to state s_i (where $i \in \{0,1,2\}$). Hence the probability to reach s_i in time interval $[0,t]$ is

$$\frac{\lambda_i}{\lambda} \int_0^t \eta_\lambda(dt_1). \tag{2}$$

Let us compute the same probability for $\overline{\mathscr{C}}$ depicted in Fig. 4(b): The probability to go from s_0 to s_i directly (with action α) is $\frac{\overline{\mathbf{R}}(s_0,\alpha,s_i)}{\overline{E}(s_0,\alpha)} = \frac{\lambda_i}{\lambda+\mu}$; however, with probability

$\frac{\overline{\mathbf{R}}(s_0,\alpha,s_0^\alpha)}{\overline{E}(s_0,\alpha)} \cdot \frac{\overline{\mathbf{R}}(s_0^\alpha,\alpha,s_i)}{\overline{E}(s_0^\alpha,\alpha)} = \frac{\mu}{\lambda+\mu} \cdot \frac{\lambda_i}{\lambda}$ we instead move to state s_0^α and only then to s_i. In this case, the probability that in time interval $[0,t]$ an α-transition of s_0 executes, followed by one of s_0^α is $\int_0^t (\lambda+\mu)e^{-(\lambda+\mu)t_1} \int_0^{t-t_1} \lambda e^{-\lambda t_2}\, dt_2\, dt_1$. Hence, we reach state s_i with action α in at most t time units with probability

$$\frac{\lambda_i}{\lambda+\mu}\int_0^t \eta_{\lambda+\mu}(dt_1) + \frac{\mu}{\lambda+\mu}\cdot\frac{\lambda_i}{\lambda}\int_0^t \eta_{\lambda+\mu}(dt_1)\int_0^{t-t_1}\eta_\lambda(dt_2). \tag{3}$$

It is easy to verify that (2) and (3) are equal. Thus the probability to reach a (non-copy) successor state in $\{s_0,s_1,s_2\}$ is the same for $\mathscr{C}$ and $\overline{\mathscr{C}}$. It can be computed by replacing λ_i with $\sum\lambda_i$ in (2) and (3). This straightforwardly extends to the Borel σ-field $\mathfrak{B}(\mathbb{R}_{\geq 0})$; further, the equality of (2) and (3) is preserved even if we integrate over a Borel-measurable function $f:\mathbb{R}_{\geq 0}\to[0,1]$. In the following, we consider the probability to reach an arbitrary non-copy state within time $T\in\mathfrak{B}(\mathbb{R}_{\geq 0})$; thus in the following lemma, we replace λ_i with $\sum\lambda_i = \lambda$:

Lemma 3 (One-step timing). *Let $f:\mathbb{R}_{\geq 0}\to[0,1]$ be a Borel measurable function and $T\in\mathfrak{B}(\mathbb{R}_{\geq 0})$. Then*

$$\int_T f(t)\,\eta_\lambda(dt) = \frac{\lambda}{\lambda+\mu}\int_T f(t)\,\eta_{\lambda+\mu}(dt) + \frac{\mu}{\lambda+\mu}\int_{\mathbb{R}_{\geq 0}}\eta_{\lambda+\mu}(dt_1)\int_{T\ominus t_1} f(t_1+t_2)\,\eta_\lambda(dt_2)$$

where $T\ominus t = \{t'\in\mathbb{R}_{\geq 0}\mid t+t'\in T\}$.

The equality of (2) and (3) proves that the probability of a single step in $\mathscr{C}$ equals the probability of one or two transitions (depending on the copy-state) in $\overline{\mathscr{C}}$. In the next section, we lift this argument to sets of paths in $\mathscr{C}$ and $\overline{\mathscr{C}}$.

4.2 Local Uniformization Is Measure Preserving

We prove that for any gm-scheduler D (on $\mathscr{C}$) there exists a gm-scheduler $\overline{D}$ (on $\overline{\mathscr{C}}$) such that the induced probabilities for the sets of paths Π and *extend*(Π) are equal. However, as $\overline{\mathscr{C}}$ differs from $\mathscr{C}$, we cannot use D to directly infer probabilities on $\overline{\mathscr{C}}$. Instead, given a history $\overline{\pi}$ in $\overline{\mathscr{C}}$, we define $\overline{D}(\overline{\pi},\cdot)$ such that it mimics the decision that D takes in $\mathscr{C}$ for history *merge*$(\overline{\pi})$: For all $\overline{\pi}\in Paths^\star(\overline{\mathscr{C}})$,

$$\overline{D}(\overline{\pi},\cdot) = \begin{cases} D(\pi,\cdot) & \text{if } \overline{\pi}[0],\overline{\pi}{\downarrow}\in\mathscr{S}\wedge merge(\overline{\pi})=\pi \\ \{\alpha\mapsto 1\} & \text{if } \overline{\pi}{\downarrow}=s^\alpha\in\mathscr{S}_{\text{cp}} \\ \gamma_{\overline{\pi}} & \text{otherwise,}\end{cases}$$

where $\gamma_{\overline{\pi}}$ is an arbitrary distribution over $Act(\overline{\pi}{\downarrow})$: If *merge* is applicable to $\overline{\pi}$ (i.e. if $\overline{\pi}[0],\pi{\downarrow}\in\mathscr{S}$), then $\overline{D}(\overline{\pi},\cdot)$ is the distribution that D yields for path *merge*$(\overline{\pi})$ in $\mathscr{C}$; further, if $\overline{\pi}{\downarrow}=s^\alpha$ then $Act(s^\alpha)=\{\alpha\}$ and thus $\overline{D}$ chooses action α. Finally, $\overline{\mathscr{C}}$ contains paths that start in a copy-state s^α. But as $\overline{\nu}(s^\alpha)=0$ for all $s^\alpha\in\mathscr{S}_{\text{cp}}$, they do not contribute any probability, independent of $\overline{D}(\overline{\pi},\cdot)$.

Based on this, we consider a measurable base B of the form $B=S_0\times A_0\times T_0\times\ldots\times S_n$ in $\mathscr{C}$. This corresponds to the set *extend*(B) of paths in $\overline{\mathscr{C}}$. As *extend*(B) contains paths of different lengths, we resort to its induced (infinite) cylinder $Cyl(extend(B))$ and prove that its probability equals that of B :

Lemma 4 (Measure preservation under local uniformization). *Let $\mathscr{C} = (\mathscr{S}, Act, \mathbf{R}, \nu)$ be a CTMDP, D a gm-scheduler on $\mathscr{C}$ and $B = S_0 \times A_0 \times T_0 \times \cdots \times S_n \in \mathfrak{F}_{Paths^n(\mathscr{C})}$. Then there exists a gm-scheduler $\overline{D}$ such that*

$$Pr^n_{\nu,D}(B) = \overline{Pr}^{\omega}_{\overline{\nu},\overline{D}}(Cyl(extend(B)))$$

where $\overline{Pr}^{\omega}_{\overline{\nu},\overline{D}}$ is the probability measure induced by $\overline{D}$ and $\overline{\nu}$ on $\mathfrak{F}_{Paths^{\omega}(\overline{\mathscr{C}})}$.

Proof. To shorten notation, let $\overline{B} = extend(B)$ and $\overline{C} = Cyl(\overline{B})$. In the induction base $B = S_0$ and $Pr^0_{\nu,D}(B) = \sum_{s\in B} \nu(s) = \sum_{s\in \overline{B}} \overline{\nu}(s) = \overline{Pr}^0_{\overline{\nu},\overline{D}}(\overline{B}) = \overline{Pr}^{\omega}_{\overline{\nu},\overline{D}}(\overline{C})$. In the induction step, we extend B with a set of initial path prefixes $I = S_0 \times A_0 \times T_0$ and consider the base $I \times B$ which contains paths of length $n+1$:

$$\begin{aligned}
Pr^{n+1}_{\nu,D}(I\times B) &= \int_I Pr^n_{\nu_i,D_i}(B)\,\mu^1_{\nu,D}(di) && \text{by Lemma 1}\\
&= \int_I \overline{Pr}^{\omega}_{\overline{\nu_i},\overline{D_i}}(\overline{C})\,\mu^1_{\nu,D}(di) && \text{by ind. hyp.}\\
&= \sum_{s\in S_0} \nu(s) \sum_{\alpha\in A_0} D(s,\alpha) \int_{T_0} \overline{Pr}^{\omega}_{\overline{\nu_i},\overline{D_i}}(\overline{C})\,\eta_{E(s,\alpha)}(dt) && \text{where } i=(s,\alpha,t)\\
&= \sum_{s\in S_0} \overline{\nu}(s) \sum_{\alpha\in A_0} \overline{D}(s,\alpha) \int_{T_0} \underbrace{\overline{Pr}^{\omega}_{\overline{\nu_i},\overline{D_i}}(\overline{C})}_{f(s,\alpha,t)}\,\eta_{E(s,\alpha)}(dt) && \text{by Def. of } \overline{\nu},\overline{D}.
\end{aligned}$$

The probabilities $\overline{Pr}^{\omega}_{\overline{\nu_i},\overline{D_i}}(\overline{C})$ define a measurable function $f(s,\alpha,\cdot) : \mathbb{R}_{\geq 0} \to [0,1]$ where $f(s,\alpha,t) = \overline{Pr}^{\omega}_{\overline{\nu_i},\overline{D_i}}(\overline{C})$ if $i = (s,\alpha,t)$. Therefore we can apply Lemma 3 and obtain

$$\begin{aligned}
Pr^{n+1}_{\nu,D}(I\times B) = \sum_{s\in S_0} \overline{\nu}(s) \sum_{\alpha\in A_0} \overline{D}(s,\alpha)\cdot \Big[&\overline{\mathbf{P}}(s,\alpha,\mathscr{S}) \int_{T_0} f(s,\alpha,t)\,\eta_{\overline{E}(s,\alpha)}(dt)\\
&+\overline{\mathbf{P}}(s,\alpha,s^{\alpha}) \int_{\mathbb{R}_{\geq 0}} \eta_{\overline{E}(s,\alpha)}(dt_1) \int_{T_0\ominus t_1} f(s,\alpha,t_1+t_2)\,\eta_{\overline{E}(s^{\alpha},\alpha)}(dt_2)\Big].
\end{aligned} \tag{4}$$

To rewrite this further, note that any path prefix $i = (s,\alpha,t)$ in $\mathscr{C}$ induces the sets of path prefixes $\overline{I}_1(i) = \{s \xrightarrow{\alpha,t}\}$ and $\overline{I}_2(i) = \{s \xrightarrow{\alpha,t_1} s^{\alpha} \xrightarrow{\alpha,t_2} \mid t_1+t_2 = t\}$ in $\overline{\mathscr{C}}$, where $\overline{I}_1(i)$ corresponds to directly reaching a state in $\mathscr{S}$, whereas in $\overline{I}_2(i)$ the detour via copy-state s^{α} is taken. As defined in Lemma 1, $\nu_i(s') = \mathbf{P}(s,\alpha,s')$ is the probability to go to state s' when moving along prefix i in $\mathscr{C}$. Similarly, for $\overline{\mathscr{C}}$ we define $\overline{\nu}_{\overline{i}}(s')$ as the probability to be in state $s' \in \mathscr{S}$ after a path prefix $\overline{i} \in \overline{I}_1(i) \cup \overline{I}_2(i)$: If $\overline{i} \in \overline{I}_1(i)$ then we move to a state $s' \in \mathscr{S}$ directly and do not visit copy-state s^{α}. Thus $\overline{\nu}_{\overline{i}}(s') = \overline{\mathbf{P}}(s,\alpha,s')$ for $\overline{i} \in \overline{I}_1(i)$. Further, $\mathbf{P}(s,\alpha,s')$ in $\mathscr{C}$ equals the conditional probability $\frac{\overline{\mathbf{P}}(s,\alpha,s')}{\overline{\mathbf{P}}(s,\alpha,\mathscr{S})}$ to enter s' in $\overline{\mathscr{C}}$ given that we move there directly. Therefore $\overline{\nu}_{\overline{i}}(s') = \overline{\mathbf{P}}(s,\alpha,\mathscr{S}) \cdot \nu_i(s')$ if $\overline{i} \in \overline{I}_1(i)$.

If instead $\overline{i} \in \overline{I}_2(i)$ then $\overline{i}$ has the form $s \xrightarrow{\alpha,t_1} s^{\alpha} \xrightarrow{\alpha,t_2}$ and $\overline{\nu}_{\overline{i}}(s') = \overline{\mathbf{P}}(s^{\alpha},\alpha,s')$ is the probability to end up in state s' after $\overline{i}$. By the definition of s^{α}, this is equal to the probability to move from state s to s' in $\mathscr{C}$. Hence $\overline{\nu}_{\overline{i}}(s') = \nu_i(s')$ if $\overline{i} \in \overline{I}_2(i)$.

As defined in Lemma 1, $D_i(\pi,\cdot) = D(i\circ\pi,\cdot)$ and $\overline{D}_{\overline{i}}(\overline{\pi},\cdot) = \overline{D}(\overline{i}\circ\overline{\pi},\cdot)$. From the definition of $\overline{D}$ we obtain $D_i(\pi,\cdot) = \overline{D}_{\overline{i}}(\overline{\pi},\cdot)$ for all $\overline{i} \in \overline{I}_1(i) \cup \overline{I}_2(i)$ and $\overline{\pi} \in extend(\pi)$. Hence it follows that if $i = (s,\alpha,t)$ and $\overline{i} \in \overline{I}_1(i) \cup \overline{I}_2(i)$ it holds

$$\overline{Pr}^{\omega}_{\overline{\nu}_{\bar{i}},\overline{D}_{\bar{i}}}(\overline{C}) = \begin{cases} \overline{\mathbf{P}}(s,\alpha,\mathscr{S}) \cdot \overline{Pr}^{\omega}_{\overline{\nu}_i,\overline{D}_i}(\overline{C}) & \text{if } \bar{i} \in \overline{I}_1(i) \\ \overline{Pr}^{\omega}_{\overline{\nu}_i,\overline{D}_i}(\overline{C}) & \text{if } \bar{i} \in \overline{I}_2(i). \end{cases} \tag{5}$$

Note that the first summand in (4) corresponds to the set $\overline{I_1}(s,\alpha,t)$ and the second to $\overline{I}_2(s,\alpha,t_1+t_2)$. Applying equality (5) to the right-hand side of (4) we obtain

$$\begin{aligned} Pr^{n+1}_{\nu,D}(I\times B) &= \sum_{s\in S_0} \overline{\nu}(s) \sum_{\alpha\in A_0} \overline{D}(s,\alpha) \int_{T_0} \overline{Pr}^{\omega}_{\overline{\nu}_{\bar{i}},\overline{D}_{\bar{i}}}(\overline{C})\ \eta_{\overline{E}(s,\alpha)}(dt) \\ &+ \sum_{s\in S_0} \overline{\nu}(s) \sum_{\alpha\in A_0} \overline{D}(s,\alpha) \cdot \overline{\mathbf{P}}(s,\alpha,s^{\alpha}) \int_{\mathbb{R}_{\geq 0}} \eta_{\overline{E}(s,\alpha)}(dt_1) \int_{T_0\ominus t_1} \overline{Pr}^{\omega}_{\overline{\nu}_{\bar{i}},\overline{D}_{\bar{i}}}(\overline{C})\ \eta_{\overline{E}(s^{\alpha},\alpha)}(dt_2). \end{aligned}$$

Applying Def. 5 allows to integrate over the sets of path prefixes $\overline{I}_1 = \bigcup_{i\in I} \overline{I}_1(i)$ and $\overline{I}_2 = \bigcup_{i\in I} \overline{I}_2(i)$ which are induced by $I = S_0 \times A_0 \times T_0$ and to obtain

$$Pr^{n+1}_{\nu,D}(I\times B) = \int_{\overline{I}_1} \overline{Pr}^{\omega}_{\overline{\nu}_{\bar{i}},\overline{D}_{\bar{i}}}(\overline{C})\ \overline{\mu}^1_{\overline{\nu},\overline{D}}(d\bar{i}) + \int_{\overline{I}_2} \overline{Pr}^{\omega}_{\overline{\nu}_{\bar{i}},\overline{D}_{\bar{i}}}(\overline{C})\ \overline{\mu}^2_{\overline{\nu},\overline{D}}(d\bar{i}).$$

Rewriting the right-hand side yields $Pr^{n+1}_{\nu,D}(I\times B) = \overline{Pr}^{\omega}_{\overline{\nu},\overline{D}}\big(Cyl(extend(I\times B))\big)$. □

Lemma 4 holds for all measurable rectangles $B = S_0 \times A_0 \times T_0 \times \ldots \times S_n$; however, we aim at an extension to arbitrary bases $B \in \mathfrak{F}_{Paths^n(\mathscr{C})}$. Thus let $\mathfrak{G}_{Paths^n(\mathscr{C})}$ be the class of all finite disjoint unions of measurable rectangles. Then $\mathfrak{G}_{Paths^n(\mathscr{C})}$ is a *field* [10, p. 102]:

Lemma 5. *Let $\mathscr{C} = (\mathscr{S}, Act, \mathbf{R}, \nu)$ be a CTMDP, D a gm-scheduler on $\mathscr{C}$ and $n \in \mathbb{N}$. Then $Pr^n_{\nu,D}(B) = \overline{Pr}^{\omega}_{\overline{\nu},\overline{D}}\big(Cyl(extend(B))\big)$ for all $B \in \mathfrak{G}_{Paths^n(\mathscr{C})}$.*

With the monotone class theorem [10], the preservation property extends from $\mathfrak{G}_{Paths^n}$ to the σ-field $\mathfrak{F}_{Paths^n}$: A class $\mathfrak{C}$ of subsets of $Paths^n$ is a monotone class if it is closed under in- and decreasing sequences: if $\Pi_k \in \mathfrak{C}$ and $\Pi \subseteq Paths^n$ such that $\Pi_0 \subseteq \Pi_1 \subseteq \cdots$ and $\bigcup_{k=0}^{\infty} \Pi_k = \Pi$, we write $\Pi_k \uparrow \Pi$ (similary for $\Pi_k \downarrow \Pi$). Then $\mathfrak{C}$ is a monotone class iff for all $\Pi_k \in \mathfrak{C}$ and $\Pi \subseteq Paths^n$ with $\Pi_k \uparrow \Pi$ or $\Pi_k \downarrow \Pi$ it holds that $\Pi \in \mathfrak{C}$.

Lemma 6 (Monotone class). *Let $\mathscr{C} = (\mathscr{S}, Act, \mathbf{R}, \nu)$ be a CTMDP with gm-scheduler D. The set $\mathfrak{C} = \left\{ B \in \mathfrak{F}_{Paths^n(\mathscr{C})} \mid Pr^n_{\nu,D}(B) = \overline{Pr}^{\omega}_{\overline{\nu},\overline{D}}\big(Cyl(extend(B))\big) \right\}$ is a monotone class.*

Lemma 7 (Extension). *Let $\mathscr{C} = (\mathscr{S}, Act, \mathbf{R}, \nu)$ be a CTMDP, D a gm-scheduler on $\mathscr{C}$ and $n \in \mathbb{N}$. Then $Pr^n_{\nu,D}(B) = \overline{Pr}^{\omega}_{\overline{\nu},\overline{D}}\big(Cyl(extend(B))\big)$ for all $B \in \mathfrak{F}_{Paths^n(\mathscr{C})}$.*

Proof. By Lemma 6, $\mathfrak{C}$ is a monotone class and by Lemma 5 it follows that $\mathfrak{G}_{Paths^n(\mathscr{C})} \subseteq \mathfrak{C}$. Thus, the Monotone Class Theorem [10, Th. 1.3.9] applies and $\mathfrak{F}_{Paths^n} \subseteq \mathfrak{C}$. Hence $Pr^n_{\nu,D}(B) = \overline{Pr}^{\omega}_{\overline{\nu},\overline{D}}\big(Cyl(extend(B))\big)$ for all $B \in \mathfrak{F}_{Paths^n}$. □

Lemma 4 and its measure-theoretic extension to the σ-field are the basis for the major results of this work as presented in the next section.

5 Main Results

The first result states the correctness of the construction of scheduler $\overline{D}$, i.e. it asserts that D and $\overline{D}$ assign the same probability to corresponding sets of paths.

Theorem 1. *Let $\mathscr{C} = (\mathscr{S}, Act, \mathbf{R}, \nu)$ be a CTMDP and D a gm-scheduler on $\mathscr{C}$. Then $Pr^{\omega}_{\nu,D}(\Pi) = \overline{Pr}^{\omega}_{\overline{\nu},\overline{D}}\big(extend(\Pi)\big)$ for all $\Pi \in \mathfrak{F}_{Paths^{\omega}}$.*

Proof. Each cylinder $\Pi \in \mathfrak{F}_{Paths^{\omega}(\mathscr{C})}$ is induced by a measurable base [10, Thm. 2.7.2]; hence $\Pi = Cyl(B)$ for some $B \in \mathfrak{F}_{Paths^{n}(\mathscr{C})}$ and $n \in \mathbb{N}$. But then, $Pr^{\omega}_{\nu,D}(\Pi) = Pr^{n}_{\nu,D}(B)$ and $Pr^{n}_{\nu,D}(B) = \overline{Pr}^{\omega}_{\overline{\nu},\overline{D}}\big(extend(\Pi)\big)$ by Lemma 7. $\square$

With Lemma 4 and its extension, we are now ready to prove that local uniformization does not alter the CTMDP in a way that we leak probability mass with respect to the most important scheduler classes:

Theorem 2. *Let $\mathscr{C} = (\mathscr{S}, Act, \mathbf{R}, \nu)$ be a CTMDP and $\Pi \in \mathfrak{F}_{Paths^{\omega}(\mathscr{C})}$. For scheduler classes $\mathfrak{D} \in \{TH, TTH, TTP, TAH, TAP\}$ it holds that*

$$\sup_{D \in \mathfrak{D}(\mathscr{C})} Pr^{\omega}_{\nu,D}(\Pi) \leq \sup_{D' \in \mathfrak{D}(\overline{\mathscr{C}})} \overline{Pr}^{\omega}_{\overline{\nu},D'}(extend(\Pi)). \tag{6}$$

Proof. By Thm. 1, the claim follows for the class of all gm-schedulers, that is, for $\mathfrak{D} = TH$. For the other classes, it remains to check that the gm-scheduler $\overline{D}$ used in Lemma 4 also falls into the respective class. Here, we state the proof for *TTP*: If $D : \mathscr{S} \times \mathbb{R}_{\geq 0} \to Distr(Act) \in TTP$, define $\overline{D}(s,\Delta) = D(s,\Delta)$ if $s \in \mathscr{S}$ and $\overline{D}(s^{\alpha},\Delta) = \{\alpha \mapsto 1\}$ for $s^{\alpha} \in \mathscr{S}_{\text{cp}}$. Then Lemma 4 applies verbatim. $\square$

Thm. 4 proves that (6) does not hold for *TP* and *TAHOP*. Although we obtain a gm-scheduler $\overline{D}$ on $\overline{\mathscr{C}}$ for any $D \in TP(\mathscr{C}) \cup TAHOP(\mathscr{C})$ by Thm. 1, $\overline{D}$ is generally not in $TP(\overline{\mathscr{C}})$ (or $TAHOP(\overline{\mathscr{C}})$, resp.). For the main result, we identify the scheduler classes, that do not gain probability mass by local uniformization:

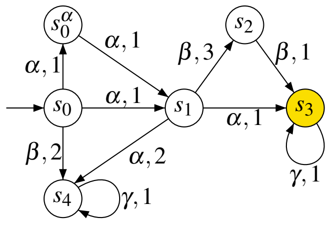
Theorem 3. *Let $\mathscr{C} = (\mathscr{S}, Act, \mathbf{R}, \nu)$ be a CTMDP and $\Pi \in \mathfrak{F}_{Paths^{\omega}(\mathscr{C})}$. Then*

$$\sup_{D \in \mathfrak{D}(\mathscr{C})} Pr^{\omega}_{\nu,D}(\Pi) = \sup_{D' \in \mathfrak{D}(\overline{\mathscr{C}})} \overline{Pr}^{\omega}_{\overline{\nu},D'}(extend(\Pi)) \quad \text{for } \mathfrak{D} \in \{TTP, TAP\}.$$

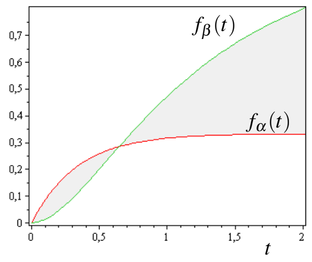
Proof. Thm. 2 proves the direction from left to right. For the reverse, let $D' \in TTP(\overline{\mathscr{C}})$ and define $D \in TTP(\mathscr{C})$ such that $D(s,\Delta) = D'(s,\Delta)$ for all $s \in \mathscr{S}, \Delta \in \mathbb{R}_{\geq 0}$. Then $\overline{D} = D'$ and $\overline{Pr}^{\omega}_{\overline{\nu},D'}(extend(\Pi)) = Pr^{\omega}_{\nu,D}(\Pi)$ by Thm. 1. Hence the claim for *TTP* follows; analogue for $D' \in TAP(\overline{\mathscr{C}})$. $\square$

Conjecture 1. We conjecture that Thm. 3 also holds for *TH* and *TTH*. For $D' \in TH(\overline{\mathscr{C}})$, we aim at defining a scheduler $D \in TH(\mathscr{C})$ that induces the same probabilities on $\mathscr{C}$. However, a history $\pi \in Paths^{\star}(\mathscr{C})$ corresponds to the uncountable set $extend(\pi)$ in $\overline{\mathscr{C}}$ s.t. $D'(\overline{\pi},\cdot)$ may be different for each $\overline{\pi} \in extend(\pi)$. As D can only decide once on history π, in order to mimic D' on $\overline{\mathscr{C}}$, we propose to weigh each distribution $D'(\overline{\pi},\cdot)$ with the conditional probability of $d\overline{\pi}$ given $extend(\pi)$.

In the following, we disprove (6) for *TP* and *TAHOP* schedulers. Intuitively, *TP* schedulers rely on the sojourn time in the last state; however, local uniformization changes the exit rates of states by adding transitions to copy-states.



(a) Local uniformization of Fig. 1



(b) From state s_1 to state s_3.

Fig. 5. Timed reachability of state s_3 (starting in s_1) in $\mathscr{C}$ and $\overline{\mathscr{C}}$

Theorem 4. *For $\mathfrak{G} \in \{TP, TAHOP\}$, there exists $\mathscr{C}$ and $\Pi \in \mathfrak{F}_{Paths^\omega(\mathscr{C})}$ such that*

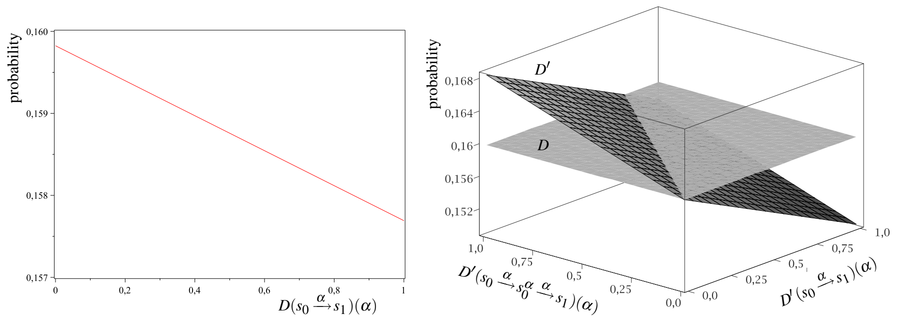
$$\sup_{D \in \mathfrak{G}(\mathscr{C})} Pr^\omega_{\nu,D}(\Pi) > \sup_{D' \in \mathfrak{G}(\overline{\mathscr{C}})} \overline{Pr}^\omega_{\overline{\nu},D'}\big(extend(\Pi)\big).$$

Proof. We give the proof for TP: Consider the CTMDPs $\mathscr{C}$ and $\overline{\mathscr{C}}$ in Fig. 1 and Fig. 5(a), resp. Let $\Pi \in \mathfrak{F}_{Paths^\omega(\mathscr{C})}$ be the set of paths in $\mathscr{C}$ that reach state s_3 in 1 time unit and let $\overline{\Pi} = extend(\Pi)$. To optimize $Pr^\omega_{\nu,D}(\Pi)$ and $\overline{Pr}^\omega_{\overline{\nu},D'}(\overline{\Pi})$, any scheduler D (resp. D') must choose $\{\alpha \mapsto 1\}$ in state s_0. Nondeterminism only remains in state s_1; here, the optimal distribution over $\{\alpha, \beta\}$ depends on the time t_0 that was spent to reach state s_1: In $\mathscr{C}$ and $\overline{\mathscr{C}}$, the probability to go from s_1 to s_3 in the remaining $t = 1 - t_0$ time units is $f_\alpha(t) = \frac{1}{3} - \frac{1}{3}e^{-3t}$ for α and $f_\beta(t) = 1 + \frac{1}{2}e^{-3t} - \frac{3}{2}e^{-t}$ for β. Fig. 5(b) shows the cdfs of f_α and f_β; as any convex combination of α and β results in a cdf in the shaded area of Fig. 5(b), we only need to consider the extreme distributions $\{\alpha \mapsto 1\}$ and $\{\beta \mapsto 1\}$ for maximal reachability. Let d be the unique solution (in $\mathbb{R}_{>0}$) of $f_\alpha(t) = f_\beta(t)$, i.e. the point where the two cdfs cross. Then $D_{opt}(s_0 \xrightarrow{\alpha,t_0} s_1, \cdot) = \{\alpha \mapsto 1\}$ if $1 - t_0 \le d$ and $\{\beta \mapsto 1\}$ otherwise, is an optimal gm-scheduler for Π on $\mathscr{C}$ and $D_{opt} \in TP(\mathscr{C}) \cap TTP(\mathscr{C})$ as it depends only on the delay of the last transition.

For $\overline{\Pi}$, D' is an optimal gm-scheduler on $\overline{\mathscr{C}}$ if $D'(s_0 \xrightarrow{\alpha,t_0} s_1, \cdot) = D_{opt}(s_0 \xrightarrow{\alpha,t_0} s_1, \cdot)$ as before and $D'(s_0 \xrightarrow{\alpha,t_0} s_0^\alpha \xrightarrow{\alpha,t_1} s_1, \cdot) = \{\alpha \mapsto 1\}$ if $1 - t_0 - t_1 \le d$ and $\{\beta \mapsto 1\}$ otherwise. Note that by definition, $D' = \overline{D_{opt}}$ and $\overline{D_{opt}} \in TTP(\overline{\mathscr{C}})$, whereas $D' \notin TP(\overline{\mathscr{C}})$ as any $TP(\overline{\mathscr{C}})$ scheduler is independent of t_0. For history $\pi = s_0 \xrightarrow{\alpha,t_0} s_0^\alpha \xrightarrow{\alpha,t_1} s_1$, the best approximation of t_0 is the expected sojourn time in state s_0, i.e. $\frac{1}{E(s_0,\alpha)}$. For the induced scheduler $D'' \in TP(\overline{\mathscr{C}})$, it holds $D''(s_1, t_1) \ne D'(s_0 \xrightarrow{\alpha,t_0} s_0^\alpha \xrightarrow{\alpha,t_1} s_1)$ almost surely. But as $\overline{D_{opt}}$ is optimal, there exists $\varepsilon > 0$ such that $\overline{Pr}^\omega_{\overline{\nu},D''}(\overline{\Pi}) = \overline{Pr}^\omega_{\overline{\nu},\overline{D_{opt}}}(\overline{\Pi}) - \varepsilon$. Therefore

$$\sup_{D'' \in TP(\overline{\mathscr{C}})} \overline{Pr}^\omega_{\overline{\nu},D''}(\overline{\Pi}) < \overline{Pr}^\omega_{\overline{\nu},\overline{D_{opt}}}(\overline{\Pi}) = Pr^\omega_{\nu,D_{opt}}(\Pi) = \sup_{D \in TP(\mathscr{C})} Pr^\omega_{\nu,D}(\Pi).$$

For *TAHOP*, a similar proof applies that relies on the fact that local uniformization changes the number of transitions needed to reach a goal state. $\square$



(a) *TAH*-schedulers on $\mathscr{C}$ (b) *TAH*-schedulers on $\overline{\mathscr{C}}$

Fig. 6. Optimal *TAH*-schedulers for time-bounded reachability

This proves that by local uniformization, essential information for *TP* and *TAHOP* schedulers is lost. In other cases, schedulers from *TAH* and *TAHOP* gain information by local uniformization:

Theorem 5. *There exists CTMDP* $\mathscr{C} = (\mathscr{S}, Act, \mathbf{R}, \nu)$ *and* $\Pi \in \mathfrak{F}_{Paths^\omega(\mathscr{C})}$ *such that*

$$\sup_{D \in \mathfrak{G}(\mathscr{C})} Pr^\omega_{\nu,D}(\Pi) < \sup_{D' \in \mathfrak{G}(\overline{\mathscr{C}})} \overline{Pr}^\omega_{\overline{\nu},D'}\big(extend(\Pi)\big) \quad \textit{for } \mathfrak{G} = \{TAH, TAHOP\}.$$

Proof. Consider the CTMDPs $\mathscr{C}$ and $\overline{\mathscr{C}}$ in Fig. 1 and Fig. 5(a), resp. Let Π be the time-bounded reachability property of state s_3 within 1 time unit and let $\overline{\Pi} = extend(\Pi)$. We prove the claim for *TAH*: Therefore we derive $D \in TAH(\mathscr{C})$ such that $Pr^\omega_{\nu,D}(\Pi) = \sup_{D' \in TAH(\mathscr{C})} Pr^\omega_{\nu,D'}(\Pi)$. For this, $D(s_0) = \{\alpha \mapsto 1\}$ must obviously hold. Thus the only choice is in state s_1 for time-abstract history $s_0 \xrightarrow{\alpha} s_1$ where $D(s_0 \xrightarrow{\alpha} s_1) = \mu$, $\mu \in Distr(\{\alpha, \beta\})$. For initial state s_0, Fig. 6(a) depicts $Pr^\omega_{\nu,D}(\Pi)$ for all $\mu \in Distr(\{\alpha, \beta\})$; obviously, $D(s_0 \xrightarrow{\alpha} s_1) = \{\beta \mapsto 1\}$ maximizes $Pr^\omega_{\nu,D}(\Pi)$. On $\overline{\mathscr{C}}$, we prove that there exists $D' \in TAH(\overline{\mathscr{C}})$ such that $Pr^\omega_{\nu,D}(\Pi) < \overline{Pr}_{\overline{\nu},D'}(\overline{\Pi})$: To maximize $\overline{Pr}^\omega_{\overline{\nu},D'}(\overline{\Pi})$, define $D'(s_0) = \{\alpha \mapsto 1\}$. Note that D' may yield different distributions for the time-abstract paths $s_0 \xrightarrow{\alpha} s_1$ and $s_0 \xrightarrow{\alpha} s_0^\alpha \xrightarrow{\alpha} s_1$; for $\mu, \mu_c \in Distr(\{\alpha, \beta\})$ such that $\mu = D'(s_0 \xrightarrow{\alpha} s_1)$ and $\mu_c = D'(s_0 \xrightarrow{\alpha} s_0^\alpha \xrightarrow{\alpha} s_1)$ the probability of $\overline{\Pi}$ under D' is depicted in Fig. 6(b) for all $\mu, \mu_c \in Distr(\{\alpha, \beta\})$. Clearly, $\overline{Pr}^\omega_{\overline{\nu},D'}(\overline{\Pi})$ is maximal if $D'(s_0 \xrightarrow{\alpha} s_1) = \{\beta \mapsto 1\}$ and $D'(s_0 \xrightarrow{\alpha} s_0^\alpha \xrightarrow{\alpha} s_1) = \{\alpha \mapsto 1\}$. Further, Fig. 6(b) shows that with this choice of D', $\overline{Pr}^\omega_{\overline{\nu},D'}(\overline{\Pi}) > Pr^\omega_{\nu,D}(\Pi)$ and the claim follows. For *TAHOP*, the proof applies analogously. □

6 Delaying Nondeterministic Choices

To conclude the paper, we show how local uniformization allows to derive the class of *late schedulers* which resolve nondeterminism only when leaving a state. Hence, they may exploit information about the current state's sojourn time and, as a consequence, induce more accurate probability bounds than gm-schedulers.

More precisely, let $\mathscr{C} = (\mathscr{S}, Act, \mathbf{R}, \nu)$ be a locally uniform CTMDP and D a gm-scheduler on $\mathscr{C}$. Then $E(s,\alpha) = u(s)$ for all $s \in \mathscr{S}$ and $\alpha \in Act$ (cf. Def. 7). Thus the measures $\eta_{E(s,\alpha)}$ in Def. 3 do not depend on α and we may exchange their order of integration in (1) by applying [10, Thm. 2.6.6]. Hence for locally uniform CTMDPs let

$$\mu_D(\pi,M) = \int_{\mathbb{R}_{\geq 0}} \eta_{u(\pi\downarrow)}(dt) \int_{Act} D(\pi,d\alpha) \int_{\mathscr{S}} \mathbf{I}_M(\alpha,t,s')\ \mathbf{P}(s,\alpha,ds'). \tag{7}$$

Formally, (7) allows to define *late schedulers* as mappings $D : Paths^\star(\mathscr{C}) \times \mathbb{R}_{\geq 0} \times \mathfrak{F}_{Act} \to [0,1]$ that extend gm-schedulers with the sojourn-time in $\pi\downarrow$. Note that local uniformity is essential here: In the general case, the measures $\eta_{E(s,\alpha)}(dt)$ and a late scheduler $D(\pi,t,d\alpha)$ are inter-dependent in t and α; hence, in Def. 3, $\mu_D(\pi,\cdot)$ is not well-defined for late-schedulers. Intuitively, the sojourn time t of the current state s depends on D while D depends on t.

Let *LATE* and *GM* denote the classes of late and gm-schedulers, respectively. For all $\Pi \in Paths^\omega(\mathscr{C})$:

$$\sup_{D\in GM} Pr^\omega_{\nu,D}(\Pi) \leq \sup_{D\in LATE} Pr^\omega_{\nu,D}(\Pi) \tag{8}$$

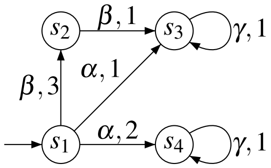


Fig. 7. Example

holds as $GM \subseteq LATE$. By Thm. 3, *TTP* and *TAP* preserve probability bounds; hence, late-schedulers are well-defined for those classes and yield better probability bounds than gm-schedulers, i.e., in general inequality (8) is strict: Let $\mathscr{C}$ be as in Fig. 7 and Π be timed-reachability for s_3 in 1 time unit. Then $\sup_{D\in GM} Pr^\omega_{\nu,D}(\Pi) = 1+\frac{1}{2}e^{-3}-\frac{3}{2}e^{-1}$. On the other hand, the optimal late scheduler is given by $D(s_1,t,\cdot) = \{\beta \mapsto 1\}$ if $t < 1+\ln 2-\ln 3$ and $\{\alpha \mapsto 1\}$ otherwise. Then $Pr^\omega_{\nu,D}(\Pi) = 1+\frac{19}{24}e^{-3}-\frac{3}{2}e^{-1}$ and the claim follows.

7 Conclusion

We studied a hierarchy of scheduler classes for CTMDPs, and investigated their sensitivity for general measures w.r.t. local uniformization. This transformation is shown to be measure-preserving for TAP and TTP schedulers. In addition, in contrast to TP and TAHOP schedulers, TH, TTH, and TAH schedulers cannot lose information to optimize their decisions. TAH and TAHOP schedulers can also gain information. We conjecture that our transformation is also measure-preserving for TTH and TH schedulers. Finally, late schedulers are shown to be able to improve upon generic schedulers [6].

References

1. Knast, R.: Continuous-time probabilistic automata. Inform. and Control 15, 335–352 (1969)
2. Guo, X., Hernández-Lerma, O., Prieto-Rumeau, T.: A survey of recent results on continuous-time Markov decision processes. TOP 14, 177–261 (2006)
3. Baier, C., Hermanns, H., Katoen, J.P., Haverkort, B.R.: Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes. Theor. Comp. Sci. 345(1), 2–26 (2005)
4. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley and Sons, Chichester (1994)

5. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.P.: Model-checking algorithms for continuous-time Markov chains. IEEE TSE 29(6), 524–541 (2003)
6. Wolovick, N., Johr, S.: A characterization of meaningful schedulers for continuous-time Markov decision processes. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 352–367. Springer, Heidelberg (2006)
7. Grassmann, W.K.: Finding transient solutions in Markovian event systems through randomization. In: Stewart, W.J. (ed.) Numerical Solutions of Markov Chains, pp. 357–371 (1991)
8. Gross, D., Miller, D.R.: The randomization technique as a modeling tool and solution procedure for transient Markov processes. Oper. Res. 32(2), 343–361 (1984)
9. Jensen, A.: Markov chains as an aid in the study of Markov processes. Skand. Aktuarietidskrift 3, 87–91 (1953)
10. Ash, R., Doléans-Dade, C.: Probability & Measure Theory, 2nd edn. Academic Press, London (2000)
11. Neuhäußer, M.R., Katoen, J.P.: Bisimulation and logical preservation for continuous-time Markov decision processes. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 412–427. Springer, Heidelberg (2007)

Concurrency, σ-Algebras, and Probabilistic Fairness

Samy Abbes and Albert Benveniste

[1] PPS/Université Paris 7 Denis Diderot. 175, rue du Chevaleret, 75013 Paris, France
samy.abbes@pps.jussieu.fr
http://www.pps.jussieu.fr/~abbes
[2] INRIA/IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France
Albert.Benveniste@inria.fr
http://www.irisa.fr/distribcom/benveniste

Abstract. We extend previous constructions of probabilities for a prime event structure E by allowing arbitrary confusion. Our study builds on results related to fairness in event structures that are of interest per se.

Executions of E are captured by the set Ω of maximal configurations. We show that the information collected by observing only fair executions of E is confined in some σ-algebra $\mathfrak{F}_0$, contained in the Borel σ-algebra $\mathfrak{F}$ of Ω. Equality $\mathfrak{F}_0 = \mathfrak{F}$ holds when confusion is finite (formally, for the class of *locally finite* event structures), but inclusion $\mathfrak{F}_0 \subseteq \mathfrak{F}$ is strict in general. We show the existence of an increasing chain $\mathfrak{F}_0 \subseteq \mathfrak{F}_1 \subseteq \mathfrak{F}_2 \subseteq \ldots$ of sub-σ-algebras of $\mathfrak{F}$ that capture the information collected when observing executions of increasing unfairness. We show that, if the event structure unfolds a 1-safe net, then unfairness remains quantitatively bounded, that is, the above chain reaches $\mathfrak{F}$ in finitely many steps.

The construction of probabilities typically relies on a Kolmogorov extension argument. Such arguments can achieve the construction of probabilities on the σ-algebra $\mathfrak{F}_0$ only, while one is interested in probabilities defined on the entire Borel σ-algebra $\mathfrak{F}$. We prove that, when the event structure unfolds a 1-safe net, then unfair executions all belong to some set of $\mathfrak{F}_0$ of zero probability. Whence $\mathfrak{F}_0 = \mathfrak{F}$ modulo 0 *always* holds, whereas $\mathfrak{F}_0 \neq \mathfrak{F}$ in general. This yields a new construction of Markovian probabilistic nets, carrying a natural interpretation that "unfair executions possess zero probability".

Keywords: Probabilistic Petri nets, probabilistic event structures, true-concurrency, probabilistic fairness.

Introduction

The distinction between interleaving and partial orders semantics (also called *true-concurrency* semantics), has a deep impact when considering probabilistic aspects. In true-concurrency models, executions are modeled by *traces* or *configurations*, i.e., partial orders of events. Corresponding probabilistic models thus consist in randomizing maximal configurations, not infinite firing sequences. It turns out that a central issue in developing true-concurrency probabilistic models is to localize choices made while the executions progress. In a previous work [4,6], the authors have introduced *branching cells*, which dynamically localize choices along the progress of configurations. In this context, it is natural to introduce the class of *locally finite* event structures, in which each choice is causally connected to only finitely many other choices—as a particular case, every confusion free event structure is locally finite [20,3]. In locally finite event structures, maximal configurations are tiled by branching cells. A recursive and non deterministic procedure allows to scan this set of tiling branching cells—of course, non determinism in the procedure is due to concurrency within the configuration. This tiling shows that every execution may be seen as a *partial order of choices.* Therefore, it is natural to proceed to the randomization of executions by randomizing local choices and imposing probabilistic independence to concurrent choices.

Although quite natural, the class of locally finite event structures is not general enough. Finite 1-safe nets may unfold to non locally finite event structures. Worse, the class of locally finite event structures is not stable under natural operations such as synchronization product. In this paper, to free our theory from external constraints on confusion, we consider general event structures with arbitrary confusion. We still try to build a coherent theory of choice for these, with applications to probabilistic event structures.

As a first contribution, we show that the branching cells that tile a configuration may require infinite ordinals greater than $\boldsymbol{\omega}$ for their enumeration. We classify configurations according to their *height*, that is the number of limit ordinals greater than $\boldsymbol{\omega}$ needed to enumerate the branching cells that tile the configuration—thus, for a locally finite event structure, all configurations have height zero. We show that, for event structures unfolding a finite 1-safe net, configurations have their height bounded by the number of transitions of the net. Configurations of strictly positive height turn out to exhibit lack of fairness. Thus our results provide an analysis of the structure of choice in relation with fairness in that the height of a configuration can be seen as a measure of its "unfairness degree".

A second contribution of our paper concerns the construction of probabilities for event structures with arbitrary confusion. When equipping concurrent systems with probabilities, the partial orders semantics attaches probabilities to partial orders of events, not to sequences. Randomizing an event structure is performed by equipping each "local zone" where a choice occurs with a local "routing" probability. Accordingly, local probabilities are attached to branching cells. An event structure is said to be *probabilistic* when a probability measure is attached to the space $(\Omega, \mathfrak{F})$ of maximal configurations equipped with its Borel σ-algebra. For locally finite event structures, we have shown in [4] that a Kolmogorov extension argument allows to infer the existence and uniqueness of a probability $\mathbb{P}$ on $(\Omega, \mathfrak{F})$ coherent with a given family of local "routing" probabilities attached to branching cells—see also [20] for a similar result valid for confusion free event structures. For event structures with possibly infinite confusion, however, this construction is not sufficient, mainly because branching cells do not entirely tile maximal configurations.

The novel idea of this paper is to introduce an increasing family $\mathfrak{F}_n$ of σ-algebras, where index n ranges over the set of all possible heights for configurations. $\mathfrak{F}_0$ captures the information obtained by observing only configurations of height 0 (the fair ones) and $\mathfrak{F}_n$ captures the information obtained by observing only configurations of height up to n. In particular, if the maximal height for configurations is finite and equal to N, then $\mathfrak{F}_N = \mathfrak{F}$, the Borel σ-algebra—we show in this paper that this property holds for unfoldings of 1-safe nets.

The Kolmogorov extension argument always allows to construct a probability $\mathbb{P}_0$ over $\mathfrak{F}_0$. However, $\mathfrak{F}_0 \subseteq \mathfrak{F}$ holds with strict inclusion unless the event structure is locally finite. The second important result of this paper consists in showing that, for Markovian probabilistic nets, "unfair executions have zero probability". Formally, we show that, for every Borel set $A \in \mathfrak{F}$, there exist two measurable sets $B, B' \in \mathfrak{F}_0$ such that $B \subseteq A \subseteq B'$ and $\mathbb{P}_0(B' - B) = 0$. Consequently, $\mathbb{P}_0$ extends trivially to the Borel σ-algebra $\mathfrak{F}$ by adding to $\mathfrak{F}_0$ all zero probability sets. With these results we fill the gap that remained in our previous studies and therefore complete the landscape of true-concurrency probabilistic systems.

Related Work. Our study of related work is structured according to the two contributions of this paper.

The first contribution is concerned with the structure of choice in prime event structures and nets. Confusion freeness and its variants have been extensively considered for Petri nets, particularly in the context of stochastic Petri nets [7]. Regarding prime event structures, the notion of *cell* has been introduced by Varacca *et al.* in [20] as equivalence classes of the minimal conflict relation. For this construction to work, confusion-freeness of the event structure is required. Cells are minimal zones of the event structure where local choices occur. Independently, the authors of this paper have developed in [2,4,6] the theory of *locally finite* event structures, in which confusion freeness is relaxed to kind of a "bounded confusion". *Branching cells* generalize cells in this context. They still represent zones of local choice. However, unlike cells in confusion free event structures, branching cells are dynamically defined in that they depend on the configuration enabling them. Local finiteness guarantees that branching cells are finite. Restricting ourselves to confusion free or locally finite event structures ensures that the structure of choice is "simple" enough. With the present paper, however, we show that the concept of local choice is valid and useful for general prime event structures and is still adequately captured by the notion of branching cell. Thus branching cells appear as *the* central concept when dealing with choice in general event structures. In addition, we have characterized fairness by means of the infinite ordinal (but still countable) needed when incrementally tiling configurations with branching cells. While most authors characterize fairness with topological tools [19,11], our use of σ-algebras for fairness related issues is quite new.

The second contribution of this paper relates to probabilistic models for systems involving concurrency. The definition and specification of probabilistic systems can be done through process algebra techniques. Probabilistic process algebra allow to retain performance information on a system while giving its specifications. According to the different modeling constraints, the definition of synchronization for probabilistic processes will differ. Several variants have thus been proposed, such as PCCS [17], TIPP [14], MPA [9], the probabilistic π-calculus [15], PEPA [16], or the κ-calculus [21] developed for biological applications. The above theories have been developed in the framework of interleaving semantics, where a probability is assigned to a sequence of events once proper scheduling of non deterministic choices has been performed. In contrast our work addresses the construction of true concurrency probabilistic models in which probabilities are assigned to partially ordered executions, not sequences.

In the context of interleaving probabilistic semantics, the main focus has been and remains on finding appropriate bisimulation relations for correctly testing and monitoring systems. The original probabilistic bisimulation relation from the seminal paper [18] has thus been extensively developed and generalized until recently [13,10]. As an instance of this series of developments, in [10] simulation relations as well as approximations are studied, relying on techniques of σ-algebras and conditional expectations. The objective is to approximate the state space by a possibly non-injective labeling of its states, thus giving rise to a sub-σ-algebra. Our present work also makes use of σ-algebras but in a totally different way. Our σ-algebras are not attached to the state space but rather to the space of trajectories (i.e., the maximal configurations) and they capture the increasing flow of information gathered while observing the system. Our objectives are not to obtain simulation relations but rather *1/* to develop the bases needed to equip prime structures with probabilities with no restriction, *2/* to further study their properties when the event structure originates from a 1-safe net, thus yielding Markov nets, and *3/* to carry over to Markov nets the fundamental and highly powerful statistical apparatus attached to infinite behaviours (Law of Large numbers, Central Limit Theorem, etc.). In this paper we address the two first issues; the reader is referred to [6] for the third one. Note that the present work shows that the ergodic results of the latter reference also hold without the local finiteness assumption.

Organization of the Paper. The paper is organized as follows. Section 1 quickly reviews the decomposition of event structures through branching cells, and recalls the probabilistic construction for locally finite event structures. A generalized induction is introduced in §2, to deal with choices in case of infinite confusion. Probabilistic applications are given in §3. Finally, §4 discusses further research perspectives. Omitted proofs may be found in the research report [5].

1 Background on Probability and Concurrency

We first describe how choices may be distributed on simple examples of nets. We explain in the same way the randomization that comes with the decomposition of choices.

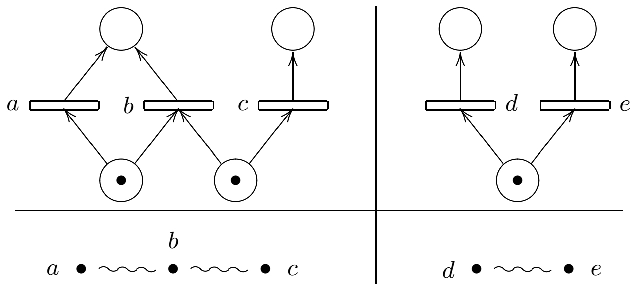


Fig. 1. Two nets (top) and their associated event structures (bottom)

1.1 Branching Cells by Example

We recall the construction of branching cells through examples. Formal definitions and results will also be given. Branching cells are best understood in the case of a *finite* event structure. In a sense, local finiteness is just the most natural extension of the finite case.

Consider thus the net $\mathcal{N}_1$ depicted in Figure 1, top left, and its (quite trivial) unfolding event structure E_1 depicted on bottom left. Remember that we randomize *maximal configurations* of unfoldings, hence the space to be randomized here is simply the set with two elements $\Omega_1 = \{(ac), (b)\}$, where we note (ac) for the configuration with events a and c, the order between a and c being of no importance. Note that, although a and c are *concurrent* events, they are not independent. On the contrary, their occurrences are strongly correlated, since any maximal configuration ω_1 has the following property: $a \in \omega_1$ if and only if $c \in \omega_1$. Obviously, the set Ω_1 with 2 elements cannot be further decomposed; this shows that concurrency and independence are distinct notions. This also shows that choices, here between (ac) or (b), are not supported by transitions, places or events of nets or event structures. Here, the event structure must be considered as a whole. We shall therefore randomize $\mathcal{N}_1$ by means of a finite probability μ_1, i.e., two non-negative numbers $\mu_1(ac)$ and $\mu_1(b)$ such that $\mu_1(ac) + \mu_1(b) = 1$.

In the same way, consider also the net $\mathcal{N}_2$ depicted on the right column of Figure 1, top, and its event structure equivalent E_2 depicted at bottom-right. Here, the set to be randomized is $\Omega_2 = \{(d), (e)\}$, so we are given a probability μ_2 on Ω_2: $\mu_2(d) + \mu_2(e) = 1$.

Consider now the net $\mathcal{N}'$ consisting of the two above nets $\mathcal{N}_1$ and $\mathcal{N}_2$ put side by side—mentally erase the vertical rule of Fig. 1 to get the picture of net $\mathcal{N}'$. The corresponding event structure, say E', has the property that any event in E_1 is concurrent *and* independent of any event in E_2. To verify this, just observe that the occurrence of any event in E_1 is compatible with the occurrence of any event in E_2; and *vice versa*. Hence $\mathcal{N}_1$ and $\mathcal{N}_2$, being atomic units of choice, are the branching cells that form net $\mathcal{N}'$. As a consequence, the set Ω' of maximal configurations of $\mathcal{N}'$ has the natural product decomposition $\Omega' = \Omega_1 \times \Omega_2$. It is thus natural to consider the product probability $\mu' = \mu_1 \otimes \mu_2$ on Ω'. Hence, for instance, the probability of firing a, c and d is given by $\mu'(acd) = \mu_1(ac) \times \mu_2(d)$. Observe the

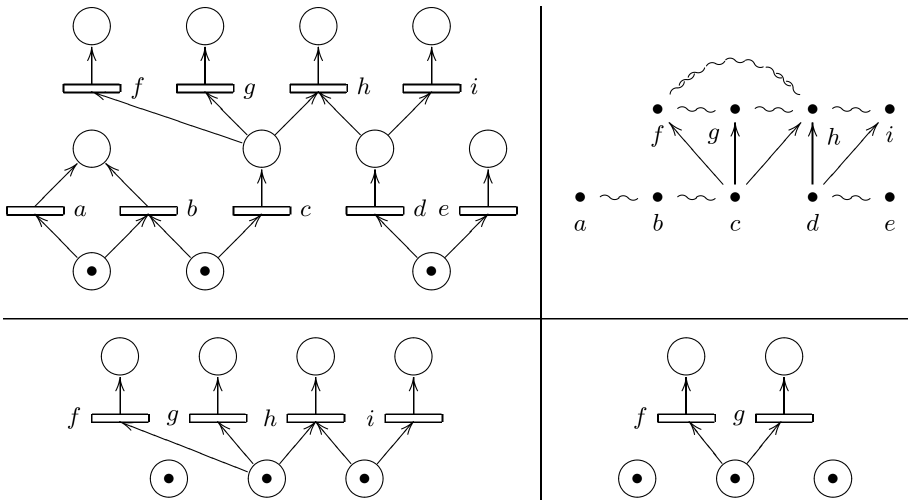


Fig. 2. Illustrating the decomposition of nets

application here of the principle of correspondence between concurrency and probabilistic independence–see [4,6] for a discussion of this idea.

It remains to continue the construction in case of synchronisation. For this, consider the net $\mathcal{N}$ depicted on the top line of Figure 2, with the event structure equivalent E on the right. Observe that net $\mathcal{N}'$, itself composed of $\mathcal{N}_1$ and $\mathcal{N}_2$, stands as the "beginning" of net $\mathcal{N}$. We already know how to randomize events that occur in the $\mathcal{N}'$ area of $\mathcal{N}$, thanks to the product decomposition of $\mathcal{N}'$. What happens "next" will be randomized by a classical conditioning process. Let for instance the probability of executing maximal configuration $\omega = (ac\,d\,gi)$ to be computed. The prefix of ω in $\mathcal{N}'$ is $v = (ac\,d)$. Since we know already the probability of execution of $v = (ac\,d)$ in $\mathcal{N}'$, we consider the system *after* configuration v. Hence we delete from $\mathcal{N}$ all transitions that either have already been fired during the execution of v, or either that are now unable to fire. The resulting net is depicted on bottom left of Figure 2—in the event structure model, we would call it the *future* E^v of v, to be detailed below in §1.2. We now start again the analysis we made in the beginning, and realize that f, g, h and i being correlated, they belong to a same third branching cell, say $\mathcal{N}_3$, or E_3 in the event structure model, and we shall consider a third probability distribution μ_3 on the set Ω_3 of maximal configurations of E_3. Hence, if μ denotes the global probability on the set Ω of maximal configurations of E, we get that $\mu(ac\,d\,gi) = \mu_1(ac) \times \mu_2(d) \times \mu_3(gi)$.

Now assume that $w = (ace)$ had fired instead of (ade). Erasing events incompatible with w only leave events f and g (see the result on bottom right of Figure 2). Hence f and g are now still two competing events, but they do not compete in the same context than previously. We have to consider they form a fourth branching cell, to which we attach a fourth probability distribution μ_4 on associated set $\Omega_4 = \{(f), (g)\}$ of maximal configurations. We would have for instance $\mu(ac\,d\,f) = \mu_1(ac) \times \mu_2(d) \times \mu_4(f)$. Since a same event, here f or g, may appear in *different* branching cells according to the context brought by the configuration, we say that the decomposition of configurations through branching cells is *dynamic*. It is part of the theory that the function μ for which we have explained the construction does indeed sum up to 1 over the set Ω of maximal configurations of E—a fact that can be easily checked by hand on this example. Let us now formalise the construction.

1.2 Formalisation: Stopping Prefixes and Branching Cells

We refer to the research report [5] and to our original publications [3,4] for the detailed construction and properties of branching cells. Here we will recall some essential definitions.

Recall that the relation $\#_\mu$ of *minimal conflict* has been defined by several authors for an event structure $(E, \leq, \#)$ as follows:

$$\forall x, y \in E, \quad x \#_\mu y \iff (\downarrow x \times \downarrow y) \cap \# = \{(x, y)\},$$

where $\downarrow x = \{e \in E \;:\; e \leq x\}$ represents the set of predecessors of event x. Define a *stopping prefix* of event structure E as a subset $B \subseteq E$ such that:

1. B is downward closed: $\forall x \in B,\ \forall y \in E, \quad y \leq x \Rightarrow y \in B$;
2. B is $\#_\mu$-closed: $\forall x \in B,\ \forall y \in E, \quad y \#_\mu x \Rightarrow y \in B$.

Stopping prefixes of E form a complete lattice with $\emptyset$ and E as minimal and maximal elements. Say that a stopping prefix is *initial* if it is minimal among non empty stopping prefixes. In the above example depicted in Fig. 2, E_1 and E_2 were the two initial prefixes of E. Any event structure may not have an initial stopping prefix—see the research report [5] for an example of event structure without initial stopping prefix. However if E is the non empty unfolding of a finite Petri net, then any stopping prefix B of E contains an initial stopping prefix—in particular, E itself contains initial stopping prefixes. This is a particular case of the following result:

Theorem 1. *Let E be a non empty event structure with the following property: there is a constant $K \geq 0$ such that, for any finite configuration v of E, at most K events $e \in E \setminus v$ are such that $v \cup \{e\}$ is a configuration. Then for every nonempty stopping prefix B of E, there is an initial stopping prefix $A \subseteq B$.*

We will always consider event structures satisfying the assumption of Theorem 1, even if it is not explicitly formulated.

Finally, if v is a configuration of E (that is, a subset of E downward closed and conflict free), we define the *future* E^v of v in E as the following sub-event structure of E:

$$E^v = \{e \in E \;:\; e \text{ is compatible with } v\} \setminus v.$$

If z is a configuration of E^v, then the set-theoretic union $v \cup z$ is a configuration of E, that we denote $v \oplus z$ to emphasize that we form the concatenation of v and z.

Consider the following recursive construction:

1. Pick an initial stopping prefix of E, pick a maximal configuration x_0 in it, and consider the future E^{x_0};
2. Pick an initial stopping prefix of E^{x_0}, pick a maximal configuration x_1 in it, and consider the future $E^{x_0 \oplus x_1}$;
3. And so on.

Any configuration that can be obtained as some $x_0 \oplus \ldots \oplus x_n$ as in the above construction, or as an increasing union of such, we call a *stopped configuration*[1] of E. A configuration obtained as some $x_0 \oplus \ldots \oplus x_n$ as in the above construction is called *finitely stopped*. The reader that would not know about branching cells is encouraged to apply this construction to the previous examples.

The several initial stopping prefixes of nested event structures that appear in the decomposition of some stopped configuration v are called the *branching cells* in the decomposition of v. Although there is range for non determinism in the decomposition of stopped configurations, it is a result that branching cells encountered in the decomposition of some stopped configuration v *only depend* on v. Branching cells are thus *intrinsic* to stopped configurations. We denote by $\Delta(v)$ the set of branching cells that occur in the decomposition of any stopped configuration v. If v is a finitely stopped configuration, any initial stopping prefix of E^v is called a branching cell *enabled* at v.

Specializing to the case where E is the unfolding of some finite 1-safe net, it is easy to realize in this case that branching cells of E are finitely many, up to isomorphism of labeled event structures—the labeling originates of course from the unfolding structure. Furthermore, the isomorphism of labelled event structures between isomorphic branching cells is *unique*. If $\mathcal{N}$ is the net being unfolded, we say that the isomorphism classes of branching cells of E are the *local states* of $\mathcal{N}$. We use the generic notation $\mathbf{x}$ to denote local states of nets.

1.3 The Case of Locally Finite Event Structures and Markov Nets

Additional properties of branching cells hold if the event structure satisfies the following property: *any event $e \in E$ belongs to some finite stopping prefix of E.* In that case, event structure E is said to be *locally finite* [3,4]. In the remaining of this paragraph, we consider a locally finite event structure E, maybe originating from the unfolding of a 1-safe Petri net.

The first property is that any branching cell is *finite*. Furthermore, *any maximal configuration of E is stopped.* We will give an interpretation of the latter fact through σ-algebras in a next section (§2).

The next steps forward to get to the randomization of locally finite event structures are the following—the following definition of a probabilistic event structure is general, and does not require E to be locally finite. We denote by Ω_E the set of maximal configurations of event structure E—this set is *always*

[1] Such configurations are called *recursively stopped* in [4,6].

non empty. The Borel σ-algebra on E is the σ-algebra generated by subsets of the form

$$\uparrow v = \{\omega \in \Omega \,:\, v \subseteq \omega\},$$

for v ranging over the *finite* configurations of E. We denote by $\mathfrak{F}$ the Borel σ-algebra on Ω_E. We say the event structure E is *probabilistic* if we are given a probability measure $\mathbb{P}$ on the measurable space $(\Omega_E, \mathfrak{F})$. Next, consider for each branching cell x of E the set Ω_x of maximal configurations of x, and a finite probability distribution p_x on Ω_x. Then define the following function p, for v ranging over the set of finitely stopped configurations of E:

$$p(v) = \prod_{x \in \Delta(v)} p_x(v \cap x), \tag{1}$$

where we recall that $\Delta(v)$ denotes the set of branching cells involved in the decomposition of v. Then $v \cap x$ belongs to Ω_x, and thus the finite product above is well defined. It is a result that *there is a unique probability measure* $\mathbb{P}$ *on* $(\Omega, \mathfrak{F})$ *such that* $\mathbb{P}(\uparrow v) = p(v)$ *for any finitely stopped configuration* v [3,4]. This result makes use of the local finiteness assumption, the crucial point being that maximal configurations of E are stopped.

Assume, furthermore, that the locally finite event structure E is the unfolding of some 1-safe net $\mathcal{N}$. Then we require the family $(p_x)_x$ to satisfy the following additional property: if x and x' are isomorphic branching cells, then so are p_x and $p_{x'}$. Formally, $p_{x'}(\omega') = p_x(\omega)$, where ω is an arbitrary maximal configuration of x, $\omega' = \phi_{x,x'}(\omega)$, and $\phi_{x,x'}$ is the *unique* isomorphism of labelled event structures from x to x'. Let $\mathbf{x}$ denote the local state associated with x and x'. Since $\phi_{x,x'}$ is unique, it makes sense to consider the set $\Omega_{\mathbf{x}}$ of maximal configurations of $\mathbf{x}$, and the probability distribution $p_{\mathbf{x}}$ attached to it, derived from the various p_x's. Such a $p_{\mathbf{x}}$ is called a *local transition probability.*

According to the previous result, the (finite) family of local transition probabilities defines a unique probability measure $\mathbb{P}$ on the space $(\Omega, \mathfrak{F})$. Call *Markov net* a net equipped with such a probability measure. Markovian and ergodic properties of Markov nets were studied in [1,6].

The aim of this paper is to generalise the above construction to an arbitrary 1-safe net, without the local finiteness assumption.

2 Non Locally Finite Unfoldings and the Height of Nets

In this section we introduce a new notion of *height* for nets, which formalizes our informal discussion in the introduction regarding fairness.

Let us first analyze non locally finite unfoldings on an example. Let $\mathcal{N}$ be the 1-safe net depicted in Fig. 3, top. The unfolding E of $\mathcal{N}$ is depicted in bottom-left. Events a_i, b_i and c_i, for $i = 1, 2, \ldots$, are respectively labeled by transitions a, b and c. Events named d, e and f are labeled by transitions d, e and f respectively. E has a unique initial stopping prefix, namely $x_1 = \{a_1, b_1\}$. Observe that the smallest stopping prefix that contains d is $E \setminus \{e, f\}$, since

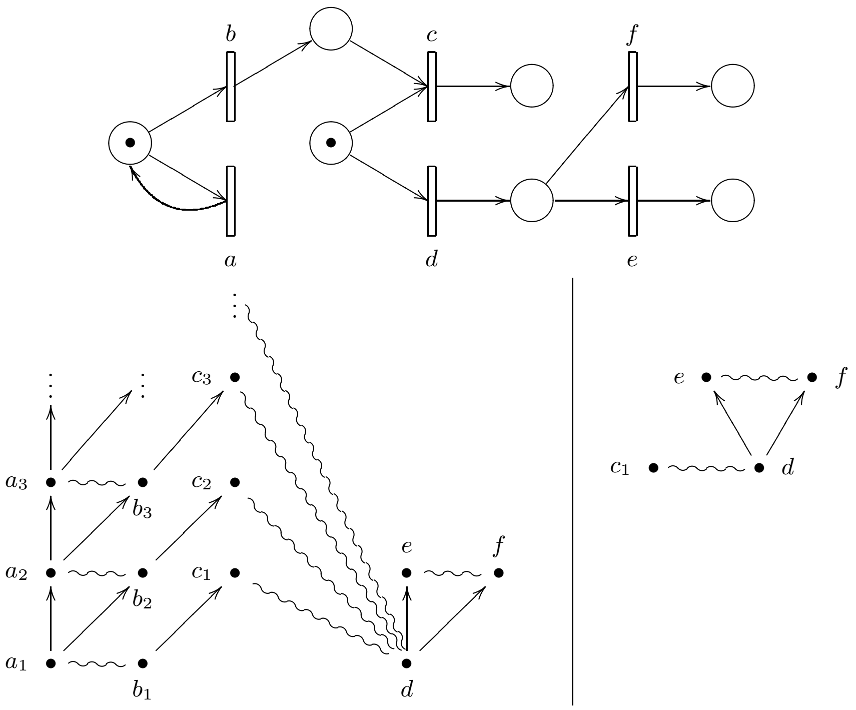


Fig. 3. A 1-safe net that unfolds to a non locally finite event structure

$d\#_\mu c_i$ for all $i = 1, 2, \ldots$, and thus E is not locally finite. The finitely stopped configurations associated with x_1 are (a_1) and (b_1). Now the future $E^{(b_1)}$ is depicted in Fig. 3, bottom-right. It contains the two branching cells $\{c_1, d\}$ and $\{e, f\}$. On the other hand, the future $E^{(a_1)}$ is isomorphic to E. Repeating this process, we find all stopped configurations of E. We describe them as follows: let $r_0 = \emptyset$, and $r_n = a_1 \oplus \cdots \oplus a_n$, for $n = 1, 2, \ldots$. Putting $s_n = r_{n-1} \oplus b_n$ for $n \geq 1$, stopped configurations containing b_n must belong to the following list:

$$s_n, \quad s_n \oplus c_n, \quad s_n \oplus d, \quad s_n \oplus d \oplus e, \quad s_n \oplus d \oplus f, \quad n \geq 1\,. \tag{2}$$

All stopped configurations are those listed in (2), plus all r_n for $n \geq 0$, and finally the infinite configuration $a_\infty = (a_1, a_2, \ldots)$. Branching cells are computed accordingly. They belong to the following list: $x_n = \{a_n, b_n\}$, $x'_n = \{c_n, d\}$, $n \geq 1$, or $x'' = \{e, f\}$. This shows in passing that branching cells can be all finite *without* E being locally finite. On the other hand, the set Ω_E of maximal configurations is described by:

$$\Omega_E = \big\{a_\infty \oplus d \oplus e, a_\infty \oplus d \oplus f\big\} \cup \big\{s_n \oplus c_n, s_n \oplus d \oplus e, s_n \oplus d \oplus f,\ n \geq 1\big\}.$$

As a consequence, *$a_\infty \oplus d \oplus e$ and $a_\infty \oplus d \oplus f$ are two maximal configurations that are not stopped.* This contrasts with the case of locally finite unfoldings, as we mentioned above.

We may however reach the missing maximal configurations $\omega_e = a_\infty \oplus d \oplus e$ and $\omega_f = a_\infty \oplus d \oplus f$ by a recursion of higher order than $\boldsymbol{\omega}$. Indeed, a_∞ is a stopped configuration of E. Its future is the simple event structure with 3 elements $d \preceq e$, $d \preceq f$, and $e \# f$. E^{a_∞} has two branching cells, namely $\{d\}$ and $\{e, f\}$. Hence if we authorize to perform concatenation, not only with finitely stopped configurations as left-concatenated element, but also on stopped configurations such as a_∞, we reach more configurations. In this example, in one additional step, we reach the missing elements ω_e and ω_f of Ω_E. We formalize and extend the above discussion in a general context next.

Let E be the unfolding of a 1-safe net $\mathcal{N}$. We set $\mathcal{X}_{-1} = \{\emptyset\}$, and we define inductively:

$$\text{for } n \geq 0, \quad \mathcal{X}_n = \left\{u \oplus v \,:\, u \in \mathcal{X}_{n-1}, \text{ and } v \text{ is stopped in } E^u\right\}.$$

It follows from this definition that $\mathcal{X}_{n-1} \subseteq \mathcal{X}_n$ for all $n \geq 0$, and that $\mathcal{X}_0$ is the set of stopped configurations of E. Then we define a non-decreasing sequence of associated σ-algebras of Ω_E as follows: For $n \geq 0$, $\mathfrak{F}_n$ is the σ-algebra generated by *arbitrary* unions of subsets of the form $\uparrow (u \oplus v)$, with $u \in \mathcal{X}_{n-1}$ and v finitely stopped in E^u. Then $\mathfrak{F}_n \subseteq \mathfrak{F}_{n+1}$ for all $n \geq 0$ since $\mathcal{X}_n \subseteq \mathcal{X}_{n+1}$. In case of locally finite unfoldings, we have the following:

Proposition 1. *If E is locally finite, then $\mathfrak{F} = \mathfrak{F}_0$.*

Example 1. That $\mathfrak{F} = \mathfrak{F}_0$ is not true in general. For instance, in the above example of Figure 3, consider $A = \uparrow (a_\infty \oplus d \oplus f)$. Then $A \notin \mathfrak{F}_0$. Indeed, considering

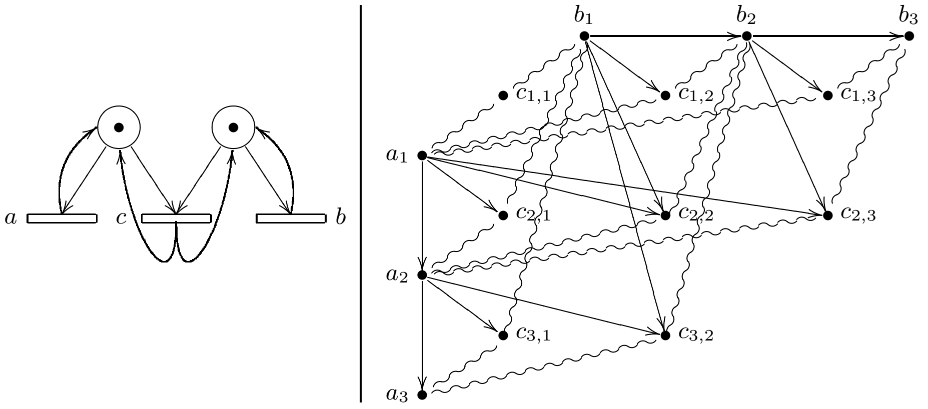


Fig. 4. *A net with height 0 and infinite branching cells.* A prefix of the (unique and infinite) initial stopping prefix x_0 of the unfolding is depicted at right. To get the entire unfolding, add a fresh copy of x_0 after each event $c_{i,j}$, $i, j \geq 1$, and continue recursively. Maximal configurations of x_0 have the form $\omega_{n,m} = a_1 \oplus \cdots \oplus a_n \oplus b_1 \oplus \ldots b_m \oplus c_{n+1,m+1}$, with $n, m \geq 0$, or $\omega_\infty = a_1 \oplus b_1 \oplus a_2 \oplus b_2 \oplus \cdots$. Any maximal configuration ω of the unfolding is a finite concatenation of $\omega_{n,m}$'s, ended with a ω_∞, or an infinite concatenation of $\omega_{n,m}$'s. The net has therefore height zero.

the σ-algebra $\mathfrak{G} = \{\uparrow a_\infty \cap K,\ K \in \mathfrak{F}_0\}$, the description that we gave of finitely stopped configurations shows that $\mathfrak{G} = \{\emptyset, \uparrow a_\infty\}$. This implies that $A \notin \mathfrak{F}_0$.

The following result generalizes the observation made on the above example: maximal configurations are reached after a finite number of (infinite) steps.

Theorem 2. *Let $\mathcal{N}$ be a 1-safe net with p transitions. Let E be the unfolding of $\mathcal{N}$, and construct as above the sequences $(\mathcal{X}_n)_n$ and $(\mathfrak{F}_n)_n$. Then $\Omega_E \subseteq \mathcal{X}_p$ and $\mathfrak{F} \subseteq \mathfrak{F}_{p+1}$.*

Definition 1 (height). *The* height *of a maximal configuration $\omega \in \Omega_E$ is the smallest integer q such that $\omega \in \mathcal{X}_q$. The* height *of a 1-safe net is the smallest integer q such that $\Omega_E \subseteq \mathcal{X}_q$.*

Theorem 2 says that 1-safe nets have finite height, less than the number of transitions. Nets with locally finite unfoldings have height 0, although all nets of height 0 need not to have a locally finite unfolding, as shown by the example of the *double loop* depicted on Fig. 4.

3 Application to the Construction of Probabilistic Nets

From the result on σ-algebras stated in Th. 2, one may wish to construct a probability measure on $(\Omega_E, \mathfrak{F})$ by using recursively and finitely many times formula (1). For locally finite unfoldings, such a construction amounts to taking a projective limit of measures (see [2]). We thus want to take nested projective limits of measures. Although this procedure would apply to any event structure (satisfying the hypotheses of Th. 1), considering unfoldings of nets brings a surprising simplification.

3.1 Analyzing an Example

Let us informally apply this construction to the example depicted in Fig. 3; justifications of the computations that we perform will be given below. We have already listed configurations from $\mathcal{X}_0$ and associated branching cells $x_n = \{a_n, b_n\}$, $x'_n = \{c_n, d\}$, $n \geq 1$, and $x'' = \{e, f\}$. With $a_\infty = (a_1, a_2, \ldots)$, configurations from $\mathcal{X}_1$ are $a_\infty \oplus d$, $a_\infty \oplus d \oplus e$ and $a_\infty \oplus d \oplus f$ (concatenation of a_∞ with stopped configurations of E^{a_∞}). Hence, extending the definition of branching cells to initial stopping prefixes in the future of configurations from $\mathcal{X}_1$, we add $x''' = \{d\}$ and the already known x''. Hence the net has four generalized local states (=classes of generalized branching cells) $\mathbf{x} = \{a, b\}$, $\mathbf{x}' = \{c, d\}$, $\mathbf{x}'' = \{e, f\}$ and $\mathbf{x}''' = \{d\}$. Consider μ, μ', μ'' and μ''', probabilities on the associated sets $\Omega_{\mathbf{x}}$, $\Omega_{\mathbf{x}'}$, $\Omega_{\mathbf{x}''}$ and $\Omega_{\mathbf{x}'''}$. For a finite configuration $v \in \mathcal{X}_0$ as listed in (2) and thereafter, the probability $\mathbb{P}\big(\uparrow v\big)$ is computed by the product formula (1). Every maximal configuration ω belongs to $\mathcal{X}_1$, and that some of them belong to $\mathcal{X}_0$. We may thus ask: what is the probability that $\omega \in \mathcal{X}_0$? Using formula (1), and recalling the notation $r_n = a_1 \oplus \cdots \oplus a_n$, we have:

$$\mathbb{P}\{\omega \notin \mathcal{X}_0\} = \mathbb{P}\{\omega \supseteq a_\infty,\ \omega \neq a_\infty\} \leq \mathbb{P}\{\omega \supseteq a_\infty\} = \lim_{n\to\infty} \mathbb{P}\{\omega \supseteq r_n\} = \lim_{n\to\infty} \alpha^n ,$$

where parameter $\alpha = \mu(a)$ is the probability of choosing transition a for a token sitting on the left most place of the net.

We thus obtain that $\mathbb{P}(\mathcal{X}_1 \setminus \mathcal{X}_0) = 0$ whenever $\alpha < 1$ (note that $\alpha < 1$ is a natural situation). In other words, configurations in $\mathcal{X}_1$ are unfair, since they have infinitely many chances to enable local state $\mathbf{x}'$ but never do, and thus they have probability zero. This is of course an expected result—see, *e.g.*, [12] for an account on probabilistic fairness. We shall now see that this situation is indeed general, for Markov nets.

3.2 Markov Nets of First Order

The first result we have is the following:

Theorem 3. *Let $\mathcal{N}$ be a 1-safe net, and let $\mu_{\mathbf{x}}$ be a local transition probability for every local state $\mathbf{x}$ of $\mathcal{N}$. For each finitely stopped configuration v, let $p(v)$ be defined by:*

$$p(v) = \prod_{x \in \Delta(v)} \mu_{\mathbf{x}}(v \cap x)\,, \tag{3}$$

where $\mathbf{x}$ denotes the isomorphism class of branching cell x. Then there is a unique probability measure $\mathbb{P}_0$ on $(\Omega_E, \mathfrak{F}_0)$ such that $\mathbb{P}_0\big(\uparrow v\big) = p(v)$ for all finitely stopped configurations v. The pair $(\mathcal{N}, (\mu_{\mathbf{x}})_{\mathbf{x}})$, where $\mathbf{x}$ ranges over the set of all local states of $\mathcal{N}$, is called a Markov net of first order*.*

Comment. For simplicity, the above theorem is formulated only for the case where each local state $\mathbf{x}$ has the property that $\Omega_{\mathbf{x}}$ is at most of countable cardinality. In general we would need to consider subsets of the form $\uparrow_{\mathbf{x}} z := \{w \in \Omega_{\mathbf{x}} : z \subseteq w\}$, for z ranging over the finite configurations of $\mathbf{x}$, instead of the mere singletons $\{v \cap \mathbf{x}\}$.

Observe the difference with the result stated in §1.3 for nets with locally finite unfoldings. The probability constructed in Th. 3 is defined only on $\mathfrak{F}_0$, and cannot measure in general all Borel subsets. We will see that this is actually not a restriction (see Th. 4 below). In case E is locally finite, we see that both constructions of probability are the same, since $\mathfrak{F} = \mathfrak{F}_0$ by Prop. 1, and since formula (1) and (3) are the same.

3.3 Completion of Markov Nets of First Order to Markov Nets

We now formalize the result observed on the example above (§3.1), that there is "no room left" for maximal configurations ω not in $\mathcal{X}_0$. For this we use the notions of complete and of completed σ-algebras. Define first the *symmetric difference* $A \triangle A'$ between two sets A and A' by $A \triangle A' = (A \setminus A') \cup (A' \setminus A)$. Let $(\Omega, \mathfrak{F}, \mathbb{P})$ be a probability space. Say that a subset $A \subseteq \Omega$ is $\mathbb{P}$-*negligible* (or simply *negligible* if no confusion can occur) if there is a subset $A' \in \mathfrak{F}$ such that $A \subseteq A'$ and $\mathbb{P}(A') = 0$. Remark that, in this definition, A is not required to be in $\mathfrak{F}$. The σ-algebra $\mathfrak{F}$ is said to be *complete* (with respect to probability $\mathbb{P}$) if $\mathfrak{F}$ contains all $\mathbb{P}$-negligible subsets. For any σ-algebra $\mathfrak{F}$, a σ-algebra $\mathfrak{H}$ is said to be a *completion* of $\mathfrak{F}$ (w.r.t. $\mathbb{P}$) if $\mathfrak{H}$ is complete, and if for every $A' \in \mathfrak{H}$, there is a $A \in \mathfrak{F}$ such that $A \triangle A'$ is negligible. It is well known that every σ-algebra $\mathfrak{F}$ has a unique completion, which is called the *completed σ-algebra* of $\mathfrak{F}$ [8].

Theorem 4. *Let $\mathcal{N}$ and $(\mu_{\mathbf{x}})_{\mathbf{x}}$ define a Markov net of first order. We assume that $\mu_{\mathbf{x}}(\uparrow y) > 0$ for any local state $\mathbf{x}$ and for any finite configuration y of $\mathbf{x}$.*

Let $\mathbb{P}_0$ be the probability on $(\Omega_E, \mathfrak{F}_0)$ constructed as in Th. 3, and let $\mathfrak{H}$ be the completed σ-algebra of $\mathfrak{F}_0$. Then $\mathfrak{F} \subseteq \mathfrak{H}$, and thus $\mathbb{P}_0$ extends to a unique probability $\mathbb{P}$ on $(\Omega_E, \mathfrak{F})$, where $\mathfrak{F}$ is the Borel σ-algebra of Ω_E.

Comment. The case when $\mathfrak{F}_0 \neq \mathfrak{F}$ brings an obstruction to a purely topological or combinatorial construction of the probability $\mathbb{P}$ on $\mathfrak{F}$. A detailed reading of the proof reveals that our construction indeed combines combinatorial arguments that use the notion of height for nets with measure theoretic tools.

4 Conclusion

We have shown how to define and construct probabilistic Petri nets for 1-safe net with arbitrary confusion. The basic idea is that choice is supported by the notion of branching cells, so independent dice can be attached to each branching cell in order to draw maximal configurations at random.

Whereas a countable sequence of drawings is enough for nets with locally finite unfolding, an induction of higher order than $\boldsymbol{\omega}$, although still countable, is needed in the more general case. Surprisingly enough, for Markov nets, this higher order induction is actually not required.

Limitations of this approach are encountered when we try to construct effective local transition probabilities. Although nets with non locally finite unfoldings can have finite branching cells, we face in general the case of infinite branching cells x, with associated spaces Ω_x being infinite also. Worst is when Ω_x is not countable. We hope that such more difficult cases can be reached by regarding them as products of simpler probabilistic nets. Composition of true-concurrent probabilistic processes is a field that we currently explore.

References

1. Abbes, S.: A (true) concurrent Markov property and some applications to Markov nets. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 70–89. Springer, Heidelberg (2005)
2. Abbes, S.: A projective formalism applied to topological and probabilistic event structures. Mathematical Structures in Computer Science 17, 819–837 (2007)
3. Abbes, S., Benveniste, A.: Branching cells as local states for event structures and nets: Probabilistic applications. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 95–109. Springer, Heidelberg (2005); Extended version available as Research Report INRIA RR-5347
4. Abbes, S., Benveniste, A.: Probabilistic models for true-concurrency: branching cells and distributed probabilities for event structures. Information & Computation 204(2), 231–274 (2006)

5. Abbes, S., Benveniste, A.: Concurrency, σ-algebras and probabilistic fairness. Technical report, PPS/Université Paris 7 Denis Diderot (2008), http://hal.archives-ouvertes.fr/hal-00267518/en/
6. Abbes, S., Benveniste, A.: Probabilistic true-concurrency models: Markov nets and a Law of large numbers. Theoretical Computer Science 390, 129–170 (2008)
7. Baccelli, F., Gaujal, B.: Stationary regime and stability of free-choice Petri nets. Springer, Heidelberg (1994)
8. Breiman, L.: Probability. SIAM, Philadelphia (1968)
9. Buchholz, P.: Compositional analysis of a Markovian process algebra. In: Rettelbach, M., Herzog, U. (eds.) Proceedings of 2nd process algebra and performance modelling workshop. Arbeitsberichte des IMMD, vol. 27 (1994)
10. Danos, V., Desharnais, J., Panangaden, P.: Labelled Markov processes: stronger and faster approximations. ENTCS 87, 157–203 (2004)
11. Darondeau, P., Nolte, D., Priese, L., Yoccoz, S.: Fairness, distance and degrees. Theoretical Computer Science 97, 131–142 (1992)
12. de Alfaro, L.: From fairness to chance. Electronic Notes in Theoretical Computer Science 22, 55–87 (1999)
13. Desharnais, J., Gupta, V., Jagadeesan, R., Panangaden, P.: Approximating labeled Markov processes. Information and Computation 184(1), 160–200 (2003)
14. Götz, N., Herzog, U., Rettelbach, M.: Multiprocessor and distributed system design: the integration of functional specification and performance analysis using stochastic process algebras. In: Proceedings of Performance 1993 (1993)
15. Herescu, O.M., Palamidessi, C.: Probabilistic asynchronous π-calculus. In: Tiuryn, J. (ed.) FOSSACS 2000. LNCS, vol. 1784, pp. 146–160. Springer, Heidelberg (2000)
16. Hillston, J.: A compositional approach to performance modelling. Cambridge University Press, Cambridge (1996)
17. Jou, C., Smolka, S.: Equivalences, congruences and complete axiomatizations of probabilistic processes. In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR 1990. LNCS, vol. 458, pp. 367–383. Springer, Heidelberg (1990)
18. Larsen, K., Skou, A.: Bisimulation through probabilistic testing. Information and Computation 94(1), 1–28 (1991)
19. Park, D.: Concurrency and automata on infinite sequences. In: Theoretical Computer Science, pp. 167–183. Springer, Heidelberg (1981)
20. Varacca, D., Völzer, H., Winskel, G.: Probabilistic event structures and domains. Theoretical Computer Science 358(2), 173–199 (2006)
21. Wilkinson, D.: Stochastic modelling for systems biology. Chapman & Hamm/CRC, Boca Raton (2006)

Synthesis from Component Libraries[⋆]

Yoad Lustig[⋆⋆] and Moshe Y. Vardi

Rice University, Department of Computer Science, Houston, TX 77251-1892, USA
{yoad,vardi}@cs.rice.edu
http://www.cs.rice.edu/~yoad,
http://www.cs.rice.edu/~vardi

Abstract. Synthesis is the automated construction of a system from its specification. In the classical temporal synthesis algorithms, it is always assumed the system is "constructed from scratch" rather than "composed" from reusable components. This, of course, rarely happens in real life. In real life, almost every non-trivial commercial system, either in hardware or in software system, relies heavily on using libraries of reusable components. Furthermore, other contexts, such as web-service orchestration, can be modeled as synthesis of a system from a library of components.

In this work we define and study the problem of LTL synthesis from libraries of reusable components. We define two notions of composition: data-flow composition, for which we prove the problem is undecidable, and control-flow composition, for which we prove the problem is 2EXPTIME-complete. As a side benefit we derive an explicit characterization of the information needed by the synthesizer on the underlying components. This characterization can be used as a specification formalism between component providers and integrators.

1 Introduction

The design of almost every non-trivial commercial system, either hardware or software system, involves many sub-systems each dealing with different engineering aspects and each requiring different expertise. For example, a software application for an email client contains sub-systems for managing graphic user interface and sub-systems for managing network connections (as well as many other sub-systems). In practice, the developer of a commercial product rarely develops all the required sub-systems himself. Instead, many sub-systems can be acquired as collections of reusable components that can be integrated into the system. We refer to a collection of reusable components as a *library*.[1]

[⋆] Work supported in part by NSF grants CCR-0124077, CCR-0311326, CCF-0613889, ANI-0216467, and CCF-0728882, by BSF grant 9800096, and by gift from Intel.

[⋆⋆] Part of this research was done while this author was at the Hebrew University in Jerusalem.

[1] In the software industry, every collection of reusable components is referred to as a "library". In the hardware industry, the term "library" is sometimes reserved for collections of components of basic functionality (e.g., logical *and*-gates with fan-in 4), while reusable components with higher functionality (e.g., an ARM CPU) are sometimes referred to by other names (such as IP cores). In this paper we refer to any collection of reusable components as a library, regardless of the level of functionality.

The exact nature of the reusable components in a library may differ. The literature suggest many different types of components. For example: IP cores (in hardware), function libraries (for procedural programming languages), object libraries (for object oriented programming languages), and aspect libraries (for aspect oriented programming languages). Web-services can also be viewed as reusable components used by an orchestrator.

Synthesis is the automated construction of a system from its specification. The basic idea is simple and appealing: instead of developing a system and verifying that it adheres to its specification, we would like to have an automated procedure that, given a specification, constructs a system that is correct by construction. The first formulation of synthesis goes back to Church [1]; the modern approach to that problem was initiated by Pnueli and Rosner who introduced LTL (linear temporal logic) synthesis [2]. In LTL synthesis the specification is given in LTL and the system constructed is a finite-state transducer modeling a reactive system.

In the work of Pnueli and Rosner, and in the many works that followed, it is always assumed that the system is "constructed from scratch" rather than "composed" from reusable components. This, of course, rarely happens in real life. In real life, almost every non-trivial system is constructed using libraries of reusable components. In fact, in many cases the use of reusable components is essential. This is the case when a system is granted access to a reusable component, while the component itself is not part of the system. For example, a software system can be given access to a hard-disk device driver (provided by the operating system), and a web-based system might orchestrate web services to which it has access, but has no control of. Even when it is theoretically possible to design a sub-system from scratch, many times it is desirable to use reusable components. The use of reusable components allows to abstract away most of the detailed behavior of the sub-system, and write a specification that mentions only the aspects of the sub-system relevant for the synthesis of the system at large.

We believe therefore, that one of the prerequisites of wide use of synthesis algorithms is support of synthesis from libraries. In this work, we define and study the problem of LTL synthesis from libraries of reusable components.

As a perquisite to the study of synthesis from libraries of reusable components, we have to define suitable models for the notions of reusable components and their composition. Indeed, there is no one correct model encompassing all possible facets of the problem. The problem of synthesis from reusable components is a general problem to which there are as many facets as there are models for components and types of composition. Components can be composed in many ways: synchronously or asynchronously, using different types of communications, etc. . As an example for the multitude of composition notions see [3], where Sifakis suggests an algebra of various composition forms.

In this work we approach the general problem by choosing two specific concrete notions of models and compositions, each corresponding to a natural facet of the problem. For components, we abstract away the precise details of the components and model a component as a *transducer*, i.e., a finite-state machine with outputs. Transducers constitute a canonical model for a reactive component, abstracting away internal architecture and focusing on modeling input/output behavior.

As for compositions, we define two notions of component composition. One relates to data-flow and is motivated by hardware, while the other relates to control-flow and is motivated by software. We study synthesis from reusable components for these notions, and show that whether or not synthesis is computable depends crucially on the notion of composition.

The first composition notion, in Section 3, is *data-flow* composition, in which the outputs of a component are fed into the inputs of other components. In data-flow composition the synthesizer controls the flow of data from one component to the other. We prove that the problem of LTL synthesis from libraries is undecidable in the case of data-flow composition. In fact, we prove a stronger result. We prove that in the case of data-flow composition, the LTL synthesis from libraries is undecidable even if we restrict ourselves to pipeline architectures, where the output of one component is fed into the input of the next component. Furthermore, it is possible to fix either the formula to be synthesized, or the library of components, and the problem remains undecidable.

The second notion of composition we consider is *control-flow* composition, which is motivated by software and web services. In the software context, when a function is called, the function is given control over the machine. The computation proceeds under the control of the function until the function calls another function or returns. Therefore, it seems natural to consider components that gain and relinquish control over the computation. A control-flow component is a transducer in which some of the states are designated as final states. Intuitively, a control-flow component receives control when entering an initial state and relinquish control when entering a final state. Composing control-flow components amounts to deciding which component will resume control when the control is relinquished by the component that currently is in control.

Web-services orchestration is another context naturally modeled by control-flow composition. A system designer composes web services offered by other parties to form a new system (or service). When referring a user to another web service, the other service may need to interact with the user. Thus, the orchestrator effectively relinquishes control of the interaction with that user until the control is received back from the referred service. Web-services orchestration has been studied extensively in recent years [4,5,6]. In Subsection 1.1, we compare our framework to previously studied models.

We show that the problem of LTL synthesis from libraries in the case of control-flow composition is 2EXPTIME-complete. One of the side benefits of this result is an explicit characterization of the information needed by the synthesis algorithm about the underlying control-flow components. The synthesis algorithm does not have to know the entire structure of the component but rather needs some information regarding the reachable states of an automaton for the specification when it monitors a component's run (the technical details can be found in Section 4). This characterization can be used to define the interface between providers and integrators of components. On the one hand, a component provider such as a web service, can publish the relevant information to facilitate the component use. On the other hand, a system developer, can publish a specification for a needed component as part of a commercial tender or even as an interface with another development group within the same organization.

1.1 Related Work

The synthesis problem was first formulated by Church [1] and solved by Büchi and Landweber [7] and by Rabin [8]. We follow the LTL synthesis problem framework presented by Pnueli and Rosner in [2,9]. We also incorporate ideas from Kupferman and Vardi [10], who suggested a way to work directly with a universal automata for the specification. In [11], Krishnamurthi and Fisler suggest an approach to aspect verification that inspired our approach to control-flow synthesis.

While the synthesis literature does not address the problem of incorporating reusable components, extensive work studies the construction of systems from components. Examples for important work on the subject can be found in Sifakis' work on component based-construction [3], and de Alfaro and Henzinger's work on "interface-based design" [12].

In addition to the work done on the subject by the formal verification community, much work has been done in field of web-services orchestration [4,5,6]. The web-services literature suggests several models for web services; the most relevant to this work is known as the "Roman model", presented in [5]. In the Roman model web services are modeled, as here, by finite-state machines. The abstraction level of the modeling, however, is significantly different. In the Roman model, every interaction with a web-service is abstracted away to a single action and no distinction is made between the inputs of the web service and the outputs of the web service.

In our framework, as in the synthesis literature, there is a distinction between output signals, which the component controls, and input signals, which the component does not control. A system should be able to cope with any value of an input signal, while the output signals can be set to desired values [2]. Therefore, the distinction is critical as the quantification structure on input and output signals is different (see [2] for details). In the Roman model, since no distinction between inputs and outputs is made, the abstraction level of the modeling *must* leave each interaction abstracted as a single atomic action. The Roman model is suitable in cases in which all that is needed to ensure is the availability of web-services actions when these are needed. Many times, however, such high level of abstraction cannot suffice for complete specification of a system.

2 Preliminaries

For a natural number n, we denote the set $\{1, \ldots, n\}$ by $[n]$. For an alphabet Σ, we denote by Σ^* the set of finite words over Σ, by Σ^ω the set of infinite words over Σ, and by Σ^∞ the union $\Sigma^* \cup \Sigma^\omega$.

Given a set D of directions, a *D-tree* is a set $T \subseteq D^*$ such that if $x \cdot c \in T$, where $x \in D^*$ and $c \in D$, then also $x \in T$. For every $x \in T$, the words $x \cdot c$, for $c \in D$, are the *successors* of x. A *path* π of a tree T is a set $\pi \subseteq T$ such that $\varepsilon \in \pi$ and for every $x \in \pi$, either x is a leaf or there exists a unique $c \in D$ such that $x \cdot c \in \pi$. The *full* D-tree is D^*. Given an alphabet Σ, a *Σ-labeled D-tree* is a pair $\langle T, \tau \rangle$ where T is a tree and $\tau : T \to \Sigma$ maps each node of T to a letter in Σ.

A *transducer*, (also known as a Moore machine [13]) is an deterministic finite automaton with outputs. Formally, a transducer is tuple $\mathcal{T} = \langle \Sigma_I, \Sigma_O, Q, q_0, \delta, F, L \rangle$ where: Σ_I is a finite input alphabet, Σ_O is a finite output alphabet, Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma_I \to Q$ is a transition function, F is a set of final states, and $L : Q \to \Sigma_O$ is an output function labelling states with output letters. For a transducer $\mathcal{T}$ and an input word $w = w_1 w_2 \ldots w_n \in \Sigma_I^n$, a *run*, or a *computation* of $\mathcal{T}$ on w is a sequence of states $r = r_0, r_1, \dots r_n \in Q^n$ such that $r_0 = q_0$ and for every $i \in [n]$ we have $r_i = \delta(r_{i-1}, w_i)$. The *trace* $tr(r)$ of the run r is the word $u = u_1 u_2 \dots u_n \in \Sigma_O^n$ where for each $i \in [n]$ we have $u_i = L(r_{i-1})$. The notions of run and trace are extended to infinite words in the natural way.

For a transducer $\mathcal{T}$, we define $\delta^* : \Sigma_I^* \rightarrow Q$ in the following way: $\delta^*(\varepsilon) = q_0$, and for $w \in \Sigma_I^*$ and $\sigma \in \Sigma_I$, we have $\delta^*(w \cdot \sigma) = \delta^*(\delta^*(w), \sigma)$. A Σ_O-labeled Σ_I-tree $\langle \Sigma_I^*, \tau \rangle$ is *regular* if there exists a transducer $\mathcal{T} = \langle \Sigma_I, \Sigma, Q, q_0, \delta, L \rangle$ such that for every $w \in \Sigma_I^*$, we have $\tau(w) = L(\delta^*(w))$.

A transducer $\mathcal{T}$ outputs a letter for every input letter it reads. Therefore, for every input word $w_I \in \Sigma_I^\omega$, the transducer $\mathcal{T}$ induces a word $w \in (\Sigma_I \times \Sigma_O)^\omega$ that combines the input and output of $\mathcal{T}$. A transducer $\mathcal{T}$ *satisfies* an LTL formula φ if for every input word $w_i \in \Sigma_I^\omega$ the induced word $w \in (\Sigma_I \times \Sigma_O)^\omega$ satisfies φ.

For a set X, let $\mathcal{B}^+(X)$ be the set of positive Boolean formulas over X (i.e., Boolean formulas built from elements in X using $\wedge$ and $\vee$), where we also allow the formulas **True** (an empty conjunction) and **False** (an empty disjunction). For a set $Y \subseteq X$ and a formula $\theta \in \mathcal{B}^+(X)$, we say that Y *satisfies* θ iff assigning **True** to elements in Y and assigning **False** to elements in $X \setminus Y$ makes θ true. An *alternating tree automaton* is $\mathcal{A} = \langle \Sigma, D, Q, q_{in}, \delta, \alpha \rangle$, where Σ is the input alphabet, D is a set of directions, Q is a finite set of states, $\delta : Q \times \Sigma \rightarrow \mathcal{B}^+(D \times Q)$ is a transition function, $q_{in} \in Q$ is an initial state, and α specifies the acceptance condition (a condition that defines a subset of Q^ω; we define several types of acceptance conditions below). For a state $q \in Q$, we denote by $\mathcal{A}^q$ the automaton $\langle \Sigma, D, Q, q, \delta, \alpha \rangle$ in which q is the initial state.

The alternating automaton $\mathcal{A}$ runs on Σ-labeled full D-trees. A *run* of $\mathcal{A}$ over a Σ-labeled D-tree $\langle T, \tau \rangle$ is a $(T \times Q)$-labeled $\mathbb{N}$-tree $\langle T_r, r \rangle$. Each node of T_r corresponds to a node of T. A node in T_r, labeled by (x, q), describes a copy of the automaton that reads the node x of T and visits the state q. Note that many nodes of T_r can correspond to the same node of T. The labels of a node and its successors have to satisfy the transition function. Formally, $\langle T_r, r \rangle$ satisfies the following:

1. $\varepsilon \in T_r$ and $r(\varepsilon) = \langle \varepsilon, q_{in} \rangle$.
2. Let $y \in T_r$ with $r(y) = \langle x, q \rangle$ and $\delta(q, \tau(x)) = \theta$. Then there is a (possibly empty) set $S = \{(c_0, q_0), (c_1, q_1), \dots, (c_{n-1}, q_{n-1})\} \subseteq D \times Q$, such that S satisfies θ, and for all $0 \leq i \leq n-1$, we have $y \cdot i \in T_r$ and $r(y \cdot i) = \langle x \cdot c_i, q_i \rangle$.

A run $\langle T_r, r \rangle$ is accepting if all its infinite paths satisfy the acceptance condition. Given a run $\langle T_r, r \rangle$ and an infinite path $\pi \subseteq T_r$, let $inf(\pi) \subseteq Q$ be such that $q \in inf(\pi)$ if and only if there are infinitely many $y \in \pi$ for which $r(y) \in T \times \{q\}$. We consider *Büchi* acceptance in which a path π is accepting iff $inf(\pi) \cap \alpha \neq \emptyset$, and *co-Büchi* acceptance in which a path π is accepting iff $inf(\pi) \cap \alpha = \emptyset$. An automaton accepts a tree iff there exists a run that accepts it. We denote by $L(\mathcal{A})$ the set of all Σ-labeled trees that $\mathcal{A}$ accepts.

The alternating automaton $\mathcal{A}$ is *nondeterministic* if for all the formulas that appear in δ, if (c_1, q_1) and (c_2, q_2) are conjunctively related, then $c_1 \neq c_2$. (i.e., if the transition is rewritten in disjunctive normal form, there is at most one element of $\{c\} \times Q$, for each $c \in D$, in each disjunct). The automaton $\mathcal{A}$ is *universal* if all the formulas that appear in δ are conjunctions of atoms in $D \times Q$, and $\mathcal{A}$ is *deterministic* if it is both nondeterministic and universal. The automaton $\mathcal{A}$ is a *word* automaton if $|D| = 1$.

We denote each of the different types of automata by three-letter acronyms in $\{D, N, U\} \times \{B, C\} \times \{W, T\}$, where the first letter describes the branching mode of the automaton (deterministic, nondeterministic, or universal), the second letter describes the acceptance condition (Büchi or co-Büchi), and the third letter describes the object over which the automaton runs (words or trees). For example, NBT are nondeterministic tree automata and UCW are universal co-Büchi word automata.

Let I be a set of input signals and O be a set of output signals. For a 2^O-labeled full-2^I tree $T = \langle (2^I)^*, \tau \rangle$ we denote by T' the $2^{I \cup O}$-labeled full 2^I-tree in which ε is labeled by $\tau(\varepsilon)$ and for every $x \in (2^I)^*$ and $i \in 2^I$ the node $x \cdot i$ is labeled by $i \cup \tau(x \cdot i)$.

The LTL *realizability* problem is: given an LTL specification φ (with atomic propositions from $I \cup O$), decide whether there is a tree T such that the labelling of every path in T' satisfies φ. It was shown in [7] that if such a tree exists, then a regular such tree exists. The *synthesis* problem is to find the transducer inducing the tree if such a transducer exists [2].

3 Data-Flow Composition

Data-flow composition is the form of composition in which the outputs of a component are fed into other components as inputs. In the general case, each component might have several input and output channels, and these may be connected to various other components. For an exposition of general data-flow composition of transducers we refer the reader to [14]. In this paper, however, the main result is a negative result of undecidability. Therefore, we restrict ourselves to a very simple form of data-flow decomposition: the pipeline architecture. To that end, we model each component as a transducer with a single input channel and single output channel. The composition of such components form the structure of a pipeline. We prove that even for such limited form of data-flow composition the problem remains undecidable.

A *data-flow component*, is a transducer in which the set of final states plays no role. We denote such a component by $C = \langle \Sigma_I, \Sigma_O, Q, q_0, \delta, L \rangle$. For two data-flow components: $C^i = \langle \Sigma_I^i, \Sigma_O^i, Q^i, q_0^i, \delta^i, L^i \rangle$, $i = 1, 2$, where $\Sigma_O^1 \subseteq \Sigma_I^2$, the composition of C^1 and C^2 is the data-flow component $C^1 \circ C^2 = \langle \Sigma_I^1, \Sigma_O^2, Q^1 \times Q^2, \langle q_0^1, q_0^2 \rangle, \delta, L \rangle$ where $\delta(\langle q_1, q_2 \rangle, \sigma) = \langle \delta^1(q_1, \sigma), \delta^2(q_2, L^1(q_1)) \rangle$, and $L(\langle q_1, q_2 \rangle) = L^2(q_2)$. It is not hard to see that the trace of the composition on a word w is the same as the trace of the run of C^2 on the trace of the run of C^1 on w.

A library $\mathcal{L}$ of data-flow component is simply a set of data-flow components. Let $\mathcal{L} = \{C_i\}$ be a collection of data-flow components. A data-flow component C is a *pipeline composition* of $\mathcal{L}$-components if there exists $k \geq 1$ and $C_1, \ldots, C_k \in \mathcal{L}$, not necessarily distinct, such that $C = C_1 \circ C_2 \circ \ldots \circ C_k$. When the library $\mathcal{L}$ is clear from the context, we abuse notation and say that C is a pipeline.

The *data-flow library LTL realizability problem* is: Given a data-flow components library $\mathcal{L}$ and an LTL formula φ, is there a pipeline composition of $\mathcal{L}$-components that satisfies φ.

Theorem 1. *Data-flow library LTL realizability is undecidable.*[2] *Furthermore, the following hold:*

1. *There exists a library $\mathcal{L}$ such that the data-flow library LTL realizability problem with respect to $\mathcal{L}$ is undecidable.*
2. *There exists an LTL formula φ such that the data-flow library φ-realizability is undecidable.*

The standard way to prove undecidability of some machine model is to show that the model can simulate Turing machines. Had we allowed a more general way of composing transducers, for example, as in [14], such an approach, indeed, could have been used. Indeed, the undecidability proof technique in [16] can be cast as an undecidability result for data-flow library realizability, where the component transducers are allowed to communicate in a two-sided manner, each simulating a tape cell of a Turing machine. Here, however, we are considering a pipeline architecture, in which information can be passed only in one direction. Such an architecture seems unable to simulate a Turing machine. In fact, in the context of *distributed* LTL realizability, which is undecidable in general [9], the case of a pipeline architecrure is the decidable case [9].

Nevertheless, data-flow library LTL realizability is undecidable even for pipeline archirecture. We prove undecidability by leveraging an idea used in the undecidability proof in [9] for non-pipeline architectures. The idea is to show that our machine model, though not powerful enough to *simulate* Turing machines, is powerful enough to *check* computations of Turing machines. In this approach, the environment produces an input stream that is a candidate computation of a Turing machine, and the synthesized system checks this computation.

We now proceed with details. Let M be a Turing machine with an RE-hard language. We reduce the language of M to a data-flow library realizability problem. Given a word w, we construct a library of components $\mathcal{L}_w$ and a formula φ, such that φ is realizable by a pipeline of $\mathcal{L}_w$-components iff $w \in L(M)$.

The gist of the proof, is that given w, we construct a pipeline that checks whether its input is an encoding of an accepting computation of M on w. Each component in the pipeline is checking a single cell in M's tape. The detailed proof can be found in the full version below we present an overview of the proof.

Intuitively, the pipeline C checks whether its input is an encoding of an accepting computation of M on w. (To encode terminating computations by infinite words, simply iterate the last configuration indefinitely). The pipeline C produces the signal *ok* either if it succeeds to verify that the input is an accepting computation of M on w, or if the input is not a computation of M on w. That way, if $w \in L(M)$ then on every word *ok* is produced, while if $w \notin L(M)$ then on the computation of M on w, the signal *ok* is never produced.

The input to the transducer is an infinite word u over some alphabet Σ_{tape} defined below. Intuitively, u is broken into chunks where each chunk is assumed to encode a single configuration of M. The general strategy is that every component in the pipeline tracks a single tape cell, and has to verify that letters that are supposed to correspond to the content of the cell "behave properly" throughout the computation.

[2] It is not hard to see that the problem is computationally enumerable, since it is computationally possible to check whether a specific pipeline composition satisfies φ [15].

The input alphabet Σ_{tape} is used to encode configurations in a way that allows the verification of the computation. The content of one cell is either a letter from M's tape alphabet Γ, or both a letter and the state of M encoding the fact that the head of M is on the cell. Each letter in Σ_{tape} encodes the content of a cell and the content of its two adjacent cells. The reason for this encoding is that during a computation the content of a cell can be predicted by the content of the cell and its neighbors in the previous cycle. In addition to letters from Γ, we allow a special separator symbol $\#$ to separate between configurations. For simplicity, assume that all the configurations of the computation are encoded by words of the same length. (In the full version we deal with the general case.)

The library $\mathcal{L}_w$ contains only two types of components C_f and C_s. In the interesting pipelines a single C_f component is followed by one or more C_s components. Intuitively, each C_s component tracks one cell tape (e.g., the third cell) and checks whether the input encodes correctly the content of the tracked cell throughout the computation. The C_f component drives the C_s components.

The alphabet Σ_{tape} is the input alphabet of the C_f component. The output alphabet of C_f as well as the input and output alphabet of C_s is more complicated. We describe this alphabet as Cartesian product of several smaller alphabets. The first of these is Σ_{tape} itself, and both C_f and C_s produce each cycle the Σ_{tape} letter they read (thus the content is propagated through the pipeline).

In order to make sure each component tracks one specific cell (e.g., the third cell), we introduce another alphabet $\Sigma_{clock} = \{pulse, \neg pulse\}$. The components produces a pulse signal as follows: A C_f component produces $pulse$ one cycle after it sees a letter encoding a separator symbol #, and a C_s component produces a $pulse$ signal two cycles it reads a $pulse$. On other cycles $\neg pulse$ is produced. Note that one cycle delay is implied by the definition of transducers. Thus, a C_s component delays the pulse signal for one additional cycle. In the full version we show that this timing mechanism allows each C_s transducer to identify the letters that encode the content of "his" cell.

As for the tracking itself, the content of a tape cell (and the two adjacent cells) in one configuration contains the information needed to predict the content of the cell in the following configuration. Thus, whenever a clock $pulse$ signal is read, each C_s component compares the content of the cell being read to the expected content from the previous cycle in which $pulse$ was read. If the content is different from the expected content a special signal is sent. The special signal $junk$, sent is part of another alphabet $\Sigma_{junk} = \{junk, \neg junk\}$. When $junk$ is produced it is propagated throughout the pipeline. The C_f component is used to check the consistency of adjacent Σ_{tape} letters, as well as that the first configuration is an encoding of the initial configuration. If an inconsistency is found $junk$ is produced.

To discover accepting computations we introduce another signal, *acc*, that is produced by a C_s if M enters the accepting state and is propagated throughout the pipeline. Finally, we introduce the signal ok. A C_s component produces ok if it never saw M's head and either it reads $junk$ (i.e., the word does not encode a computation), or it reads *acc* (i.e., the encoded computation is accepting). Note that the signal ok is never produced if the pipeline is too short to verify the computation.

In the full version we prove the following: if $w \in L(M)$ then there exists a (long enough) pipeline in which ok is produced on every word, while if $w \notin L(M)$ then ok

is never produced (by any pipeline) on the word that encodes M's computation on w. The above shows Theorem 1 for the fixed formula Fok. The proof for a fixed library is similar.

4 Control-Flow Composition

In the case of software systems, another model of composition seems natural. In the software context, when a function is called, the function is given control over the machine. The computation proceeds under the control of the function until the function calls another function or returns. Therefore, in the software context, it seems natural to consider components that gain and relinquish control over the computation.

In our model, during a phase of the computation in which a component C is in control, the input-output behavior of the entire system is governed by the component. An intuitive example is calling a function from a GUI library. Once called, the function governs the interaction with user until it returns. Just as a function might call another function, a component might relinquish control at some point. In fact, there might be several ways in which a component might relinquish control (such as taking one out of several exit points).

The composition of such components amounts to deciding the flow of control. This means that the components have to be composed in a way that specifies which component receives control in what circumstances. Thus, the system synthesizer provides an interface for each component C, where the next component to receive control is specified for every exit point in C (e.g., if C exits in one way then control goes to C_2, if C exists in another way control goes to C_3, etc.). An intuitive example of such interface in real life would be a case statement on the various return values of a function f. In case f returns 1: call function g, in case f returns 2: call function h, and so on.[3]

Below we discuss a model in which the control is passed explicitly from one component to another, as in goto. A richer model would consider also control flow by calls and returns; we leave this to future work. In our model each component is modeled by a transducer and relinquishing control is modeled by entering a final state. The interface is modeled by a function mapping the various final states to other components in the system.

Let Σ_I be an input alphabet, Σ_O be an output alphabet and, $\Sigma = \Sigma_I \cup \Sigma_O$. A *control-flow component* is a transducer $M = \langle \Sigma_I, \Sigma_O, Q, q_0, \delta, F, L \rangle$. Unlike the data-flow component case, in control-flow components the set F of final states is important. Intuitively, control-flow components receives control when entering the initial state and relinquishes control when entering a final state. When a control-flow component is in control, the input-output interaction with the environment is done by the component. For that reason, control-flow components in a system (that interact with the same environment) must share input and output alphabets. A *control-flow components library*

[3] At first sight, it seems that the synthesizer is not given realistic leeway. In real life, systems are composed not only from reusable components but also from some code written by the system creator. This problem is only superficial, however, since one can add to the component library a set of components with the same functionality as the basic commands at the disposal of the system creator.

is a set of control-flow components that share the same input and output alphabets. We assume w.l.o.g. all the final sets in the library are of the same size n_F. We denote the final set of the i-th component in the library by $F_i = \{s^i_1, \dots s^i_{n_F}\}$.

Next, we discuss a notion of composition suitable for control-flow components. When a component is in control the entire system behaves as the component and the system composition plays no role. The composition comes into play, however, when a component relinquishes control. Choosing the "next" component to be given control is the essence of the control-flow composition. A control-flow component relinquishes control by entering one of several final states. A suitable notion of composition should specify, for each of the final states, the next component the control will be given to. Thus, a control-flow composition is a sequence of components, each paired with an interface function that maps the various final states to other components in system. We refer to these pairs of a component coupled with an interface function as *interfaced component*. Note that a system synthesizer might choose to reuse a single component from the library several times, each with a different interface. Therefore, the number of interfaced components might differ from the number of components is the library. Formally, a *composition* of components from a control-flow components library $\mathcal{L}$ is a finite sequence of pairs $\langle C_1, f_1\rangle, \langle C_2, f_2\rangle, \dots, \langle C_n, f_n\rangle$ where the first element in each pair is a control-flow component $C_i = \langle \Sigma_I, \Sigma_O, Q_i, q^i_0, \delta_i, F_i, L_i\rangle \in \mathcal{L}$ and the second element in each pair is an *interface function* $f_i : F_i \to \{1, \dots, n\}$. Each of the pairs $\langle C_i, f_i\rangle$ is an *interfaced component*.

Intuitively, for an interfaced component $\langle C_i, f_i\rangle$, when C_i is in control and enters a final state $q \in F_i$, the control is passed to the interfaced component $\langle C_{f_i(q)}, f_{f_i(q)}\rangle$. Technically, this amounts to moving out of the state as if the state is not the final state q of C_i but rather the initial state $q_0^{f_i(q)}$ of $C_{f_i(q)}$. For technical reasons, we restrict every interface function $f_i : F_i \to \{1, \dots, n\}$ in the composition to map every final state to a component whose initial state agrees with the final state on the labelling.[4] Thus, f_i is an an interface function if for every $j \leq |F_i|$ we have $L_i(s^i_j) = L_{f_i(s^i_j)}(q_0^{f_i(s^i_j)})$.

The fact that a control-flow component C might appear in more than one interfaced component means that each component in the composition can be referred to in more than one way: first, as the i-th component in the library, and second, as the component within the j-th interfaced component in the composition. To avoid confusion, whenever we refer to a component from the library (as in the i-th component from the library) we denote it by $M^i \in \mathcal{L}$, while whenever we refer to a component within an interfaced component in the composition (as in the component within the j-th interfaced component) we denote it by C_j. We denote by $type(j)$ the index, in the library, of the component C_j which is the component within the j-th interfaced component. Thus, C_i is the same reusable component as $M^{type(i)}$.

The *result* of the composition is the transducer $M = \langle \Sigma_I, \Sigma_O, Q, q_0, \delta, L\rangle$ where:

1. The state space Q is $\bigcup_{i=1}^n (Q_i \times \{i\})$.
2. The initial state q_0 is $\langle q^1_0, 1\rangle$.
3. The transition function δ is defined as follows:

[4] This restriction is only a technicality chosen to simplify notation in proofs.

   (a) For a state $\langle q, i\rangle$ in which $q \in Q_i \setminus F_i$, we set $\delta(\langle q, i\rangle, \sigma) = \langle \delta_i(q, \sigma), i\rangle$.
   (b) For $\langle q, i\rangle$, where $q \in F_i$ we set $\delta(\langle q, i\rangle, \sigma) = \delta_{f_i(q)}(\langle q_0^{f_i(q)}, f_i(q)\rangle, \sigma)$.
4. The labelling function L is defined $L(\langle q, i\rangle) = L_i(q)$.

The control-flow library LTL realizability problem is: Given a control-flow components library $\mathcal{L}$ and an LTL formula φ, decide whether there exists a composition of components from $\mathcal{L}$ that satisfies φ. The control-flow library LTL synthesis problem is similar, given a $\mathcal{L}$ and φ, find the composition realizing φ if one exists.

Theorem 2. *The control-flow library LTL synthesis problem can is 2EXPTIME-complete.*

Proof. For the lower bound, we reduce classical synthesis to control-flow library synthesis. Thus, a 2EXPTIME complexity lower bound follows from the classical synthesis lower bound [17]. We proceed to describe this reduction in detail.

As described earlier, the problem of classical synthesis is to construct a transducer such that for every sequence of input signals, the sequence of input and output signals induced by the transducer computation satisfies φ. The reduction from classical synthesis is simply to provide a library of control-flow components of basic functionality, such that every transducer can be composed out of this library.

An *atomic* transducer, is a transducer that has only an initial state and final states. Furthermore, every transition from the initial state enters a final state. Thus, in an atomic transducer we have state set $Q = \{q_0, q_1, \ldots, q_m\}$, where $m = |\Sigma_I|$, final state set $F = \{q_1, \ldots, q_m\}$, and transition function $\delta(q_0, a_i) = q_i$. The different atomic transducers differ only in their output function L.

Consider now the library of all possible atomic transducers. It is not hard to see that every transducer can be composed out of this library (where every state in the transducer has its own interfaced component in the composition). Therefore synthesis is possible from this library of atomic control-flow components iff synthesis is possible at all. This concludes the reduction.

We proceed to prove the upper bound. Before going into further details, we would like to give an overview of the proof and the ideas underlying it. The classical synthesis algorithm [2] considers execution trees. An execution tree is an infinite labelled tree where the structure of the tree represents the possible finite input sequences (i.e., for input signal set I the structure is $(2^I)^*$) and the labelling represents mapping of inputs to outputs. Every transducer induces an execution tree, and every regular execution tree can be implemented by a transducer. Thus, questions regarding transducers can be reduced to questions regarding execution trees. Questions regarding execution trees can be solved using tree automata. Specifically, it is possible to construct a tree automaton whose language is the set of execution trees in which the LTL formula is satisfied, and the realizability problem reduces to checking emptiness of this automaton.

Inspired by the approach described above, we should ask what is the equivalent of an execution tree in the case of control-flow components synthesis? Fixing a library $\mathcal{L}$ of components, we would like to have a type of labelled trees, representing compositions, such that every composition would induce a tree, and every regular tree would induce a composition. To that end, we define control-flow trees. Control-flow trees represent the possible flows of control during computations of a composition. Thus, the structure of control flow trees is simply $[n_F]^*$, each node representing a finite sequence of control-flow choices. (Where each choice picks one final state from which the control is relinquished.) A control-flow tree is labelled by components from $\mathcal{L}$. Thus, a path in a control-flow tree represents the flow of control between components in the system. Note that a control-flow tree also implicitly encodes interface functions. For every node $v \in [n_F]^*$ in the tree, both v and v's sons are labelled by components from $\mathcal{L}$. We denote the labelling function by $\tau : [n_f]^* \to \mathcal{L}$. For a direction $d \in [n_F]$, the labelling $\tau(v \cdot d) \in \mathcal{L}$ of the son of v in direction d, implicitly the flow of control. (Formally, $\tau(v \cdot d)$ defines the component from $\mathcal{L}$, within the interfaced component, to which the control is passed.) Thus, a regular control-flow tree can be used to define a composition of control-flow components from $\mathcal{L}$.

Each path in a control-flow tree stands for many possible executions, all of which share the same control-flow. It is possible , however, to analyse the components in $\mathcal{L}$ and reason about the possible executions represented by a path in the control-flow tree. This allows us to construct a tree automaton that runs on control-flow trees and accept the control-flow trees in which all executions satisfy the specification. Once we have such tree automaton we can take the classical approach to synthesis.

An *infinite tree composition* $\langle [n_F]^*, \tau \rangle$ is an $[|\mathcal{L}|]$-labeled $[n_F]^*$-tree in which $\tau(\varepsilon) = 1$. Intuitively, an infinite tree composition represents possible flow of control in a composition. The root is labeled 1 since the run begins when C_1 is in control. The j-th successor of a node is labeled by $i \in |\mathcal{L}|$ if on arrival to the j-th final state, the control passed to M^i. Every finite composition $\langle C_1, f_1 \rangle, \langle C_2, f_2 \rangle, \ldots, \langle C_n, f_n \rangle$ can be unfolded to an infinite composition tree in the following way: $\tau(\varepsilon) = 1$, and for $x \in [n_F]^*$, and $i \in [n_F]$ we set $\tau(x \cdot i) = f_{\tau(x)}(s_i^{\tau(x)})$. In the proof we construct a tree automaton $\mathcal{A}$ that accepts the infinite tree compositions that satisfy φ. As we show below, if the language of $\mathcal{A}$ is empty then φ cannot be satisfied by any control-flow composition. If, on the other hand, the language of $\mathcal{A}$ is not empty, then there exists a regular tree in the language of $\mathcal{A}$, from which we can extract a finite composition.

The key to understanding the construction, is the observation that the effect of passing the control to a component is best analyzed in terms of the effect on an automaton for the specification. The specification φ has a UCW automaton $\mathcal{A}_\varphi = \langle \Sigma, Q_\varphi, q_0, \delta, \alpha \rangle$ that accepts exactly the words satisfying φ. To construct $\mathcal{A}_\varphi$ we construct an NBW $\mathcal{A}_{\neg\varphi}$ as in [18] and dualize it [10]. The effect of giving the control to a component M^i, with regard to satisfying φ, can be analyzed in terms of questions of the following type: assuming $\mathcal{A}_\varphi$ is in state q when the control is given to M^i, what possible states $\mathcal{A}_\varphi$ might be in when M^i relinquishes control by entering final state s, and whether $\mathcal{A}_\varphi$ visits an accepting state on the path from q to s.

Our first goal is to develop notation for the formalization of questions of the type presented above. For a finite word $w \in \Sigma$, we denote $\delta_\varphi^*(q, w) = \{q' \in Q_\varphi \mid$ there exists a run of $\mathcal{A}_\varphi^q$ on w that ends in $q'\}$. For $q \in Q_\varphi$ and $q' \in \delta_\varphi^*(q, w)$ we denote by $\alpha(q, w, q')$ the value of 1 if there exists a path in the run of $\mathcal{A}_\varphi^q$ on w that ends in q' and traverses through a state in α. Otherwise, $\alpha(q, w, q')$ is 0.

For a word $w \in \Sigma_I^*$ and a component $C = \langle \Sigma_I, \Sigma_O, Q_C, q_0^C, \delta_C, F_C, L_C \rangle$, we denote by $\delta_C^*(w)$ the state C reaches after running on w. We denote by $\Sigma(w, C)$ the word from Σ induced by w and the run of C on w. For $w \in \Sigma_I^*$, we denote by $\delta_\varphi^*(q, C, w)$ the set $\delta_\varphi^*(q, \Sigma(w, C))$ and by $\alpha(q, w, q')$ the bit $\alpha(q, \Sigma(w, C), q')$. Finally, we define $e_C : Q_\varphi \times F_C \rightarrow 2^{Q_\varphi \times \{0,1\}}$ where $e_C(q, s) = \{\langle q', b\rangle \mid \exists w \in \Sigma_I$ s.t. $s = \delta_C^*(w)$ and $q' \in \delta_\varphi^*(q, C, w)$ and $b = \alpha(q, w, q')\}$. Thus, $\langle q', b\rangle$ is in $e_C(q, s)$ if there exists a word $w \in \Sigma_I^*$ such that when C is run on w it relinquishes control by entering s, and if at the start of C's run on w the state of $\mathcal{A}_\varphi$ is q then at the end of C's run the state of $\mathcal{A}_\varphi$ is q'. Furthermore, b is 1 iff there is a run of $\mathcal{A}_\varphi^q$ on $\Sigma(C, w)$ that ends in q' and traverses through an accepting state.

Note, that it also possible that for some component C and infinite input word $w \in \Sigma_I^\omega$ the component C never relinquish control when running on w. For an $\mathcal{A}_\varphi$-state $q \in Q_\varphi$, the component C is a *dead end* if there exists a word $w \in \Sigma_I^\omega$ on which C never enters a final state, and on which $\mathcal{A}_\varphi^q$ rejects $\Sigma(C, w)$.

Next, we define a UCT $\mathcal{A}$ whose language is the set of infinite tree compositions realizing φ.

Let $\mathcal{A} = \langle \mathcal{L}, Q, \Delta, \langle q_0, 1\rangle, \alpha\rangle$ where:

1. The state space Q is $(Q_\varphi \times \{0, 1\}) \cup \{q_{rej}\}$. Where q_{rej} is a new state (a rejecting sink).
2. The transition relation $\Delta : Q \times \mathcal{L} \rightarrow \mathcal{B}^+([n_F] \times Q)$ is defined as follows:
   (a) For every $C \in \mathcal{L}$ and $i \in [n_F]$ we have $\Delta(q_{rej}, C) = \bigwedge_{i \in [n_F]}(i, q_{rej})$.
   (b) For $\langle q, b\rangle \in Q_\varphi \times \{0, 1\}$ and $C \in \mathcal{L}$:
      – If C is a dead end for q, then $\Delta(\langle q, j\rangle, C) = \bigwedge_{i \in [n_F]}(i, q_{rej})$.
      – Otherwise, for every $j \in [n_F]$ the automaton $\mathcal{A}$ sends in direction j the states in $e_C(q, s_j^i)$. Formally, $\Delta(\langle q, b\rangle, C) = \bigwedge_{i \in [n_F]} \bigwedge_{\langle q_i, b_i\rangle \in e_C(q, s_j^i)} (i, \langle q_i, b_i\rangle)$.
3. The initial state is $\langle q_0, 1\rangle$ for q_0 the initial state of $\mathcal{A}_\varphi$.
4. The acceptance condition is $\alpha = \{q_{rej}\} \cup \{Q_\varphi \times \{1\}\}$.

Claim 3. $L(\mathcal{A})$ is empty iff φ is not realizable from $\mathcal{L}$ □

The proof can be found in the full version.

Note that while Claim 3 is phrased in terms realizability, the proof actually yields a stronger result. If the language of $\mathcal{A}$ is not empty, then one can extract a composition realizing φ from a regular tree in the language of $\mathcal{A}$. To solve the emptiness of $\mathcal{A}$ we transform it into an "emptiness equivalent" NBT $\mathcal{A}'$ by the method of [10]. By [10], the language of $\mathcal{A}'$ is empty iff the language of $\mathcal{A}$ is empty. Furthermore, if the language of $\mathcal{A}'$ is not empty, then the emptiness test yields a witness that is in the language of $\mathcal{A}$ (as well as the language of $\mathcal{A}'$). From the witness, which is a transducer labelling $[n_f]^*$ trees with components from $\mathcal{L}$, it is possible to extract a composition.

This concludes the proof of correctness for Theorem 2 and all that is left is the complexity analysis. The input to the problem is a library $\mathcal{L} = \{M^1, \ldots, M^{|\mathcal{L}|}\}$ and a specification φ. The number of states of the UCW $\mathcal{A}_\varphi$ is $2^{O(|\varphi|)}$. The automaton $\mathcal{A}_\varphi$ can be computed in space logarithmic in $2^{O(|\varphi|)}$ (i.e., space polynomial in $|\varphi|$). The main hurdle in computing the UCT $\mathcal{A}$ is computing the transitions by computing the e_C functions for the various components. For a component $M^i \in \mathcal{L}$, an $\mathcal{A}_\varphi$-state $q \in Q_\varphi$, and a final state $s^i_j \in F_i$ the value of $e_C(q, s^i_j)$ can be computed by deciding emptiness of small variants of the product of $\mathcal{A}_\varphi$ and M^i. Thus, computing $e_C(q, s^i_j)$ is nondeterministic logspace in $2^{O(|\varphi|)} \cdot |M^i|$. The complexity of computing $\mathcal{A}$ is nondeterministic logspace in $2^{O(|\varphi|)} \cdot n_F \cdot (\sum_{i=1}^{|\mathcal{L}|} |M^i|)$. The number of states of $\mathcal{A}$ is twice the number of states of $\mathcal{A}_\varphi$, i.e. $2^{O(|\varphi|)}$, and does not depend on the library.

To solve the emptiness of $\mathcal{A}$ we use [10] to transform it into an "emptiness equivalent" NBT $\mathcal{A}'$. The size of $\mathcal{A}'$ is doubly exponential in $|\varphi|$ (specifically, $2^{2^{|\varphi|^2 \cdot \log(|\varphi|)}}$) and the complexity of its computation is polynomial time in the number of its states. Finally, the emptiness problem of an NBT can be solved in quadratic time (see [19]). Thus, the overall complexity of the problem is doubly exponential in $|\varphi|$ and polynomially dependent on the size of the library. □

An interesting side benefit the work presented so far, is the characterization of the information needed by the synthesis algorithm on the underlying components. The only dependence on a component C is by its corresponding e_C functions. Thus, given the e_C functions it is possible to perform synthesis without further knowledge of the component implementation. This suggest that the e_C functions can serve as a specification formalism between component providers and possible users.

5 Discussion

We defined two notions of component composition. Data-flow composition, for which we proved undecidability, and control-flow composition for which we provided a synthesis algorithm.

Control-flow composition required the synthesized system to be constructed only from the components in the library. In real life, system integrators usually add some code, or hardware circuitry, of their own in addition to the components used. The added code is not intended to replace the main functionality of the components, but rather allows greater flexibility in the integration of the components into a system. At first sight it might seem that our framework does not support adding such "integration code". This is not the case, as we now explain.

Recall, from the proof of Theorem 2, that LTL synthesis can be reduced to our framework by providing a library of atomic components. Every system can be constructed from atomic components. Thus, by including atomic components in our library, we enable the construction of integration code.

Note, however, that if *all* the atomic components are added to the input library, then the control-flow library LTL synthesis becomes classical LTL synthesis, as explained in the proof of Theorem 2. Fortunately, integration code typically supports functionality that can be directly manipulated by the system, as opposed to functionality that can only accessed through the components in the library. Therefore, it is possible to add to the input library only atomic components that manipulate signals in direct control of the system. This allows the control-flow library LTL synthesis of systems that contain integration code.

References

1. Church, A.: Logic, arithmetics, and automata. In: Proc. Int. Congress of Mathematicians, 1962, Institut Mittag-Leffler, pp. 23–35 (1963)
2. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Proc. 16th ACM Symp. on Principles of Programming Languages, pp. 179–190 (1989)
3. Sifakis, J.: A framework for component-based construction extended abstract. In: Proc. 3rd Int. Conf. on Software Engineering and Formal Methods (SEFM 2005), pp. 293–300. IEEE Computer Society, Los Alamitos (2005)
4. Alonso, G., Casati, F., Kuno, H.A., Machiraju, V.: Web Services - Concepts, Architectures and Applications. Springer, Heidelberg (2004)
5. Berardi, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Mecella, M.: Automatic composition of e-services that export their behavior. In: Orlowska, M.E., Weerawarana, S., Papazoglou, M.P., Yang, J. (eds.) ICSOC 2003. LNCS, vol. 2910, pp. 43–58. Springer, Heidelberg (2003)
6. Sardiña, S., Patrizi, F., Giacomo, G.D.: Automatic synthesis of a global behavior from multiple distributed behaviors. In: AAAI, pp. 1063–1069 (2007)
7. Büchi, J., Landweber, L.: Solving sequential conditions by finite-state strategies. Trans. AMS 138, 295–311 (1969)
8. Rabin, M.: Automata on infinite objects and Church's problem. Amer. Mathematical Society (1972)
9. Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: Proc. 31st IEEE Symp. on Foundations of Computer Science, pp. 746–757 (1990)
10. Kupferman, O., Vardi, M.: Safraless decision procedures. In: Proc. 46th IEEE Symp. on Foundations of Computer Science, pp. 531–540 (2005)
11. Krishnamurthi, S., Fisler, K.: Foundations of incremental aspect model-checking. ACM Transactions on Software Engineering Methods 16(2) (2007)
12. de Alfaro, L., Henzinger, T.: Interface-based design. In: Broy, M., Grünbauer, J., Harel, D., Hoare, C. (eds.) Engineering Theories of Software-intensive Systems. NATO Science Series: Mathematics, Physics, and Chemistry, vol. 195, pp. 83–104. Springer, Heidelberg (2005)
13. Hopcroft, J., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (1979)
14. Nain, S., Vardi, M.Y.: Branching vs. Linear time: Semantical perspective. In: Namjoshi, K.S., Yoneda, T., Higashino, T., Okamura, Y. (eds.) ATVA 2007. LNCS, vol. 4762, pp. 19–34. Springer, Heidelberg (2007)
15. Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Press, Cambridge (1999)
16. Apt, K., Kozen, D.: Limits for automatic verification of finite-state concurrent systems. Infornation Processing Letters 22(6), 307–309 (1986)
17. Rosner, R.: Modular Synthesis of Reactive Systems. PhD thesis, Weizmann Institute of Science (1992)
18. Vardi, M., Wolper, P.: Reasoning about infinite computations. Information and Computation 115(1), 1–37 (1994)
19. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games. LNCS, vol. 2500. Springer, Heidelberg (2002)

Realizability of Concurrent Recursive Programs*

Benedikt Bollig[1], Manuela-Lidia Grindei[1], and Peter Habermehl[1,2]

[1] LSV, ENS Cachan, CNRS, INRIA, France
{bollig,grindei}@lsv.ens-cachan.fr
[2] LIAFA, CNRS and University Paris Diderot (Paris 7), France
haberm@liafa.jussieu.fr

Abstract. We define and study an automata model of concurrent recursive programs. An automaton consists of a finite number of pushdown systems running in parallel and communicating via shared actions. Actually, we combine multi-stack visibly pushdown automata and Zielonka's asynchronous automata towards a model with an undecidable emptiness problem. However, a reasonable restriction allows us to lift Zielonka's Theorem to this recursive setting and permits a logical characterization in terms of a suitable monadic second-order logic. Building on results from Mazurkiewicz trace theory and work by La Torre, Madhusudan, and Parlato, we thus develop a framework for the specification, synthesis, and verification of concurrent recursive processes.

1 Introduction

The analysis of a concurrent recursive program where several recursive threads access a shared memory is a difficult task due to the typically high complexity of interaction between its components. One general approach is to run a verification algorithm on a finite-state abstract model of the program. As the model usually preserves recursion, this amounts to verifying multi-stack pushdown automata. Unfortunately, even if we deal with a boolean abstraction of data, the control-state reachability problem in this case is undecidable [23]. However, as proved in [22], it becomes decidable if only those states are taken into consideration that can be reached within a bounded number of context switches. A context switch consists of a transfer of control from one process to another. This result allows for the discovery of many errors, since they typically manifest themselves after a few context switches [22]. Other approaches to analyzing multithreaded programs restrict the kind of communication between processes [17,25], or compute over-approximations of the set of reachable states [6].

All these works have in common that they restrict to the analysis of an already existing system. A fundamentally different approach would be to synthesize a concurrent recursive program from a requirements specification, preferably automatically, so that the inferred system can be considered "correct by construction". The general idea of synthesizing programs from specifications goes back

* Partially supported by ARCUS, DOTS (ANR-06-SETIN-003), and P2R MODISTE-COVER/RNP Timed-DISCOVERI.

to [11]. The particular case of non-recursive distributed systems is, e.g., dealt with in [7,8,18].

In this paper, we address the synthesis problem for finite-state concurrent *recursive* programs that communicate via shared actions. More precisely, we are interested in transforming a given global specification in terms of a context-sensitive language into a design model of a distributed implementation thereof. The first step is to provide an automata model that captures both asynchronous procedure calls and shared-variable communication. To this aim, we combine visibly pushdown automata [2] and asynchronous automata [27], which, seen individually, constitute robust automata classes with desirable closure properties and decidable verification problems.

Merging visibly pushdown automata and asynchronous automata, we obtain *concurrent visibly pushdown automata* (Cvpa), which are a special case of multi-stack visibly pushdown automata (Mvpa) [16]. For Mvpa, the reachability problem is again undecidable. To counteract this drawback, La Torre et al. restrict the domain of possible inputs to *k-phase words*. A k-phase word can be decomposed into k subwords where all processes are able to evolve in a subword but only one process can return from a procedure [16]. Note that this is less restrictive than the notion of bounded context switches that we mentioned above. When we restrict to k-phase words, Mvpa actually have a decidable emptiness problem and lead to a language class that is closed under boolean operations.

Let us turn to the main contributions of our paper. We consider Cvpa as a model of concurrent recursive programs and Mvpa as specifications. Thus, we are interested in transforming an Mvpa into an equivalent Cvpa, if possible. Indeed, one can lift Zielonka's Theorem to the recursive setting: For every Mvpa language L that is closed under permutation rewriting of independent actions, there is a Cvpa $\mathcal{A}$ such that $L(\mathcal{A}) = L$. Unfortunately, it is in general undecidable if L is closed in this way. In the context of k-phase words, however, we can provide decidable sufficient criteria that guarantee that the closure of the specification can be recognized by a Cvpa. We will actually show that the closure of an Mvpa language that is *represented* (in a sense that will be made clear) by its k-phase executions can be realized as a Cvpa. The problem with Mvpa as specifications is that they do not necessarily possess the closure property that Cvpa naturally have. We therefore propose to use MSO logic as a specification language. Formulas from that logic are interpreted over *nested traces*, which are Mazurkiewicz traces equipped with multiple nesting relations. Under the assumption of a k-phase restriction, any MSO formula can be effectively transformed into a Cvpa. This constitutes an extension of the classical connection between asynchronous automata and MSO logic [26].

Organization. Section 2 provides basic definitions and introduces Mvpa and Cvpa. Section 3 considers the task of synthesizing a distributed system in terms of a Cvpa from an Mvpa specification. In doing so, we give two extensions of Zielonka's Theorem to concurrent recursive programs. In Section 4, we provide a logical characterization of Cvpa in terms of MSO logic. We conclude with Section 5, in which we suggest several directions for future work.

2 Definitions

The set $\{1, 2, \ldots\}$ of positive natural numbers is denoted by $\mathbb{N}$. We call any finite set an *alphabet*. Its elements are called *letters* or *actions*. For an alphabet Σ, Σ^* is the set of finite words over Σ; the empty word is denoted by ε. The concatenation uv of words $u, v \in \Sigma^*$ is denoted by $u \cdot v$. For a set X, we let $|X|$ denote its size and 2^X its powerset.

Concurrent Pushdown Alphabets. The architecture of a system is constituted by a *concurrent (visibly) pushdown alphabet*. To define it formally, we fix a nonempty finite set *Proc* of *process names* or, simply, *processes*. Now consider a collection $\widetilde{\Sigma} = ((\Sigma_p^c, \Sigma_p^r, \Sigma_p^{int}))_{p \in Proc}$ of alphabets. The triple $(\Sigma_p^c, \Sigma_p^r, \Sigma_p^{int})$ associated with process p contains the supplies of actions that can be executed by p. More precisely, the alphabets contain its call, return, and internal actions, respectively. We call $\widetilde{\Sigma}$ a *concurrent pushdown alphabet* (over *Proc*) if

– for every $p \in Proc$, the sets Σ_p^c, Σ_p^r, and Σ_p^{int} are pairwise disjoint, and
– for every $p, q \in Proc$ with $p \neq q$, $(\Sigma_p^c \cup \Sigma_p^r) \cap (\Sigma_q^c \cup \Sigma_q^r) = \emptyset$.

For $p \in Proc$, let Σ_p refer to $\Sigma_p^c \cup \Sigma_p^r \cup \Sigma_p^{int}$, the set of actions that are available to p. Thus, $\Sigma = \bigcup_{p \in Proc} \Sigma_p$ is the set of all the actions. Furthermore, for $a \in \Sigma$, let $proc(a) = \{p \in Proc \mid a \in \Sigma_p\}$. The intuition behind a concurrent pushdown alphabet is as follows: An action $a \in \Sigma$ is executed simultaneously by every process from $proc(a)$. In doing so, a process $p \in proc(a)$ can access the current state of any other process from $proc(a)$. The only restriction is that p can access and modify only its own stack, provided $a \in \Sigma_p^c \cup \Sigma_p^r$. However, in that case, the stack operation can be "observed" by some other process q if $a \in \Sigma_q^{int}$.

We introduce further useful abbreviations and let $\Sigma^c = \bigcup_{p \in Proc} \Sigma_p^c$, $\Sigma^r = \bigcup_{p \in Proc} \Sigma_p^r$, and $\Sigma^{int} = (\bigcup_{p \in Proc} \Sigma_p^{int}) \setminus (\Sigma^c \cup \Sigma^r)$.

Example 1. Let $Proc = \{p, q\}$ and let $\widetilde{\Sigma} = ((\{a\}, \{\overline{a}\}, \{b\}), (\{b\}, \{\overline{b}\}, \emptyset))$ be a concurrent pushdown alphabet where the triple $(\{a\}, \{\overline{a}\}, \{b\})$ refers to process p and $(\{b\}, \{\overline{b}\}, \emptyset)$ belongs to process q. Thus, $\Sigma = \{a, \overline{a}, b, \overline{b}\}$, $\Sigma^c = \{a, b\}$, $\Sigma^r = \{\overline{a}, \overline{b}\}$, and $\Sigma^{int} = \emptyset$. Note also that $proc(a) = \{p\}$ and $proc(b) = \{p, q\}$.

If not stated otherwise, $\widetilde{\Sigma}$ will henceforth be any concurrent pushdown alphabet.

Multi-Stack Visibly Pushdown Automata. Before we introduce our new automata model, we recall multi-stack visibly pushdown automata, as recently introduced by La Torre, Madhusudan, and Parlato [16]. Though this model will be parametrized by $\widetilde{\Sigma}$, it is not distributed yet. The concurrent pushdown alphabet only determines the number of stacks (which equals $|Proc|$) and the actions operating on them. In the next subsection, an element $p \in Proc$ will then actually play the role of a process.

Definition 2. *A* multi-stack visibly pushdown automaton (MVPA) *over $\widetilde{\Sigma}$ is a tuple $\mathcal{A} = (S, \Gamma, \delta, \iota, F)$ where S is its finite set of* states, *$\iota \in S$ is the* initial state, *$F \subseteq S$ is the set of* final states, *Γ is the finite* stack alphabet *containing a special symbol $\bot$, and $\delta \subseteq S \times \Sigma \times \Gamma \times S$ is the set of* transitions.

Consider a transition $(s, a, A, s') \in \delta$. If $a \in \Sigma_p^c$, then we deal with a push-transition meaning that, being in state s, the automaton can read a, push the symbol $A \in \Gamma \setminus \{\bot\}$ onto the p-stack, and go over to state s'. Transitions $(s, a, A, s') \in \delta$ with $a \in \Sigma^c$ and $A = \bot$ are discarded. If $a \in \Sigma_p^r$, then the transition allows us to pop $A \neq \bot$ from the p-stack when reading a, while the control changes from state s to state s'; if, however, $A = \bot$, then the a can be executed provided the stack of p is empty, i.e., $\bot$ is never popped. Finally, if $a \in \Sigma^{int}$, then an internal action is applied, which does not involve a stack operation. In that case, the symbol A is simply ignored.

Let us formalize the behavior of the Mvpa $\mathcal{A}$. A *stack content* is a nonempty finite sequence from $Cont = (\Gamma \setminus \{\bot\})^* \cdot \{\bot\}$. The leftmost symbol is thus the top symbol of the stack content. A configuration of $\mathcal{A}$ consists of a state and a stack content for each process. Hence, it is an element of $S \times Cont^{Proc}$. Consider a word $w = a_1 \dots a_n \in \Sigma^*$. A *run* of $\mathcal{A}$ on w is a sequence $\rho = (s_0, (\sigma_p^0)_{p \in Proc}) \dots (s_n, (\sigma_p^n)_{p \in Proc}) \in (S \times Cont^{Proc})^*$ such that $s_0 = \iota$, $\sigma_p^0 = \bot$ for all $p \in Proc$, and, for all $i \in \{1, \dots, n\}$, the following hold:

[Push]. If $a_i \in \Sigma_p^c$ for $p \in Proc$, then there is a stack symbol $A \in \Gamma \setminus \{\bot\}$ such that $(s_{i-1}, a_i, A, s_i) \in \delta$, $\sigma_p^i = A \cdot \sigma_p^{i-1}$, and $\sigma_q^i = \sigma_q^{i-1}$ for all $q \in Proc \setminus \{p\}$.

[Pop]. If $a_i \in \Sigma_p^r$ for $p \in Proc$, then there is a stack symbol $A \in \Gamma$ such that $(s_{i-1}, a_i, A, s_i) \in \delta$, $\sigma_q^i = \sigma_q^{i-1}$ for all $q \in Proc \setminus \{p\}$, and either $A \neq \bot$ and $\sigma_p^{i-1} = A \cdot \sigma_p^i$, or $A = \bot$ and $\sigma_p^i = \sigma_p^{i-1} = \bot$.

[Internal]. If $a_i \in \Sigma^{int}$, then there is $A \in \Gamma$ such that $(s_{i-1}, a_i, A, s_i) \in \delta$ and $\sigma_p^i = \sigma_p^{i-1}$ for every $p \in Proc$.

The run ρ is *accepting* if $s_n \in F$. The *language* of $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set of words $w \in \Sigma^*$ such that there is an accepting run of $\mathcal{A}$ on w. In the following, we denote by $|\mathcal{A}|$ the size $|S|$ of the set of states of $\mathcal{A}$.

Clearly, the emptiness problem for Mvpa is undecidable. Moreover, it was shown that Mvpa can in general not be complemented [5]. We can remedy this situation by restricting our domain to k-phase words [16]. Let $k \in \mathbb{N}$. A word $w \in \Sigma^*$ is called a *k-phase word* over $\widetilde{\Sigma}$ if it can be written as $w_1 \cdot \ldots \cdot w_k$ where, for all $i \in \{1, \dots, k\}$, we have $w_i \in (\Sigma^c \cup \Sigma^{int} \cup \Sigma_p^r)^*$ for some $p \in Proc$. The set of k-phase words over $\widetilde{\Sigma}$ is denoted by $\mathrm{W}_k(\widetilde{\Sigma})$. Note that $\mathrm{W}_k(\widetilde{\Sigma})$ is regular. The language of the Mvpa $\mathcal{A}$ relative to k-phase words, denoted by $L_k(\mathcal{A})$, is defined to be $L(\mathcal{A}) \cap \mathrm{W}_k(\widetilde{\Sigma})$. Even if we restrict to k-phase words, a deterministic variant of Mvpa is strictly weaker, unless we have $\Sigma = \Sigma^{int}$ [16,27].

In this paper, we will exploit the following two theorems concerning Mvpa:

Theorem 3 (La Torre-Madhusudan-Parlato [16]). *The following problem is decidable in doubly exponential time wrt. $|\mathcal{A}|$, $|Proc|$, and k:*

Input*: Concurrent pushdown alphabet $\widetilde{\Sigma}$; $k \in \mathbb{N}$;* Mvpa *$\mathcal{A}$ over $\widetilde{\Sigma}$.*
Question*: Does $L_k(\mathcal{A}) \neq \emptyset$ hold?*

Theorem 4 (La Torre-Madhusudan-Parlato [16]). *Let $k \in \mathbb{N}$ and let $\mathcal{A}$ be an* Mvpa *over $\widetilde{\Sigma}$. One can effectively construct an* Mvpa *$\mathcal{A}'$ over $\widetilde{\Sigma}$ such that $L(\mathcal{A}') = \overline{L_k(\mathcal{A})}$, where $\overline{L_k(\mathcal{A})}$ is defined to be $\Sigma^* \setminus L_k(\mathcal{A})$.*

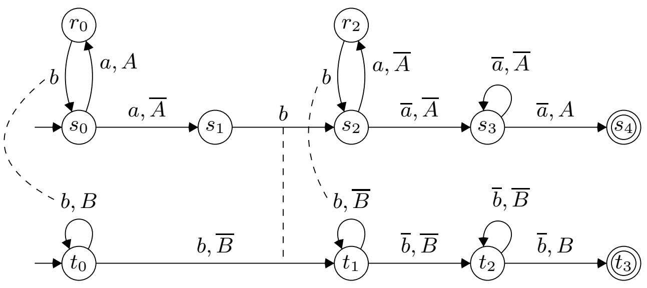


Fig. 1. A concurrent visibly pushdown automaton

Concurrent Visibly Pushdown Automata. We let $I_{\widetilde{\Sigma}} = \{(a, b) \in \Sigma \times \Sigma \mid proc(a) \cap proc(b) = \emptyset\}$ contain the pairs of actions that are considered *independent*. Moreover, $\sim_{\widetilde{\Sigma}} \subseteq \Sigma^* \times \Sigma^*$ shall be the least congruence that satisfies $ab \sim_{\widetilde{\Sigma}} ba$ for all $(a, b) \in I_{\widetilde{\Sigma}}$. The equivalence class of a representative $w \in \Sigma^*$ wrt. $\sim_{\widetilde{\Sigma}}$ is denoted by $[w]_{\sim_{\widetilde{\Sigma}}}$. We canonically extend $[.]_{\sim_{\widetilde{\Sigma}}}$ to sets $L \subseteq \Sigma^*$ and let $[L]_{\sim_{\widetilde{\Sigma}}} = \{w \in \Sigma^* \mid w \sim_{\widetilde{\Sigma}} w' \text{ for some } w' \in L\}$.

Based on Definition 2, we now introduce our model of a concurrent recursive program, which will indeed produce languages that are closed under $\sim_{\widetilde{\Sigma}}$.

Definition 5. *A* concurrent visibly pushdown automaton (CVPA) *over $\widetilde{\Sigma}$ is an* MVPA $(S, \Gamma, \delta, \iota, F)$ *over $\widetilde{\Sigma}$ such that there exist a family $(S_p)_{p \in Proc}$ of sets of* local states *and relations $\delta_a \subseteq \left(\prod_{p \in proc(a)} S_p\right) \times \Gamma \times \left(\prod_{p \in proc(a)} S_p\right)$ for $a \in \Sigma$ satisfying the following properties:*

- *$S = \prod_{p \in Proc} S_p$ and*
- *for every $s, s' \in S$, $a \in \Sigma$, and $A \in \Gamma$, we have $(s, a, A, s') \in \delta$ iff*
  - *$((s_p)_{p \in proc(a)}, A, (s'_p)_{p \in proc(a)}) \in \delta_a$ and*
  - *$s_p = s'_p$ for every $p \in Proc \setminus proc(a)$*

  where s_p denotes the p-component of state s.

To make local states and their transition relations explicit, we may consider a CVPA to be a structure $((S_p)_{p \in Proc}, \Gamma, (\delta_a)_{a \in \Sigma}, \iota, F)$.

Note that, if $\Sigma = \Sigma^{int}$ (i.e., $\widetilde{\Sigma} = ((\emptyset, \emptyset, \Sigma_p))_{p \in Proc}$), then a CVPA can be seen as a simple asynchronous automaton [8,27]. It is easy to show that the language $L(\mathcal{C})$ of a CVPA $\mathcal{C}$ is $\sim_{\widetilde{\Sigma}}$-closed meaning that $L(\mathcal{C}) = [L(\mathcal{C})]_{\sim_{\widetilde{\Sigma}}}$.

Example 6. Consider the concurrent pushdown alphabet $\widetilde{\Sigma}$ from Example 1. Assume $\mathcal{C} = (S, \Gamma, \delta, \iota, F)$ to be the CVPA depicted in Figure 1 where S is the cartesian product of $S_p = \{s_0, \ldots, s_4, r_0, r_2\}$ and $S_q = \{t_0, \ldots, t_3\}$. Actions from $\{a, \overline{a}, \overline{b}\}$ are exclusive to a single process so that corresponding transitions are local. For example, the relation δ_a, as required in Definition 5, is given by $\{(s_0, A, r_0), (s_0, \overline{A}, s_1), (s_2, \overline{A}, r_2)\}$. Thus, $((s_0, t_i), a, A, (r_0, t_i)) \in \delta$ for all $i \in \{0, \ldots, 3\}$. In contrast, executing b involves both processes, which is indicated by the dashed lines depicting δ_b. For example, $((s_1, t_0), \overline{B}, (s_2, t_1)) \in \delta_b$. Furthermore, $((r_0, t_0), b, B, (s_0, t_0))$, $((s_1, t_0), b, \overline{B}, (s_2, t_1))$, and $((r_2, t_1), b, \overline{B}, (s_2, t_1))$

are the global b-transitions contained in δ. Note that $L_1(\mathcal{C}) = \emptyset$, since at least two phases are needed to reach the final state (s_4, t_3). Moreover,

– $L_2(\mathcal{C}) = \{(ab)^n w \mid n \geq 2,\ w \in \{\overline{a}^m\overline{b}^m, \overline{b}^m\overline{a}^m\}$ for some $m \in \{2, \ldots, n\}\}$ and
– $L(\mathcal{C}) = \{(ab)^n w \mid n \geq 2,\ w \in \{\overline{a}, \overline{b}\}^*,\ |w|_{\overline{a}} = |w|_{\overline{b}} \in \{2, \ldots, n\}\} = [L_2(\mathcal{C})]_{\sim_{\widetilde{\Sigma}}}$

where $|w|_{\overline{a}}$ and $|w|_{\overline{b}}$ denote the number of occurrences of $\overline{a}$ and $\overline{b}$ in w. Note that $L_2(\mathcal{C})$ can be viewed as an incomplete description or representation of $L(\mathcal{C})$.

3 Realizability of Concurrent Recursive Programs

From now on, we consider an MVPA $\mathcal{A}$ to be a specification of a system, and we are looking for a *realization* or *implementation* of $\mathcal{A}$, which is provided by a CVPA $\mathcal{C}$ such that $L(\mathcal{C}) = L(\mathcal{A})$. Actually, specifications often have a "global" view of the system, and the difficult task is to *distribute* the state space onto the processes, which henceforth communicate in a restricted manner that conforms to the predefined system architecture $\widetilde{\Sigma}$. If, on the other hand, $\mathcal{A}$ is not closed under $\sim_{\widetilde{\Sigma}}$, it might yet be considered as an incomplete specification so that we ask for a CVPA $\mathcal{C}$ such that $L(\mathcal{C}) = [L(\mathcal{A})]_{\sim_{\widetilde{\Sigma}}}$.

We now recall two well-known theorems from Mazurkiewicz trace theory. The first one, Zielonka's celebrated theorem, applies to simple concurrent pushdown alphabets. It will later be lifted to general concurrent pushdown alphabets.

Theorem 7 (Zielonka [27]). *Suppose $\Sigma = \Sigma^{int}$. For every regular language $L \subseteq \Sigma^*$ that is $\sim_{\widetilde{\Sigma}}$-closed, there is a* CVPA *$\mathcal{C}$ over $\widetilde{\Sigma}$ such that $L(\mathcal{C}) = L$.*

We fix a strict total order $<_{\text{lex}}$ on Σ. It naturally induces a (strict) lexicographic order on Σ^*, which we denote by $<_{\text{lex}}$ as well. We say that $w \in \Sigma^*$ is in *(lexicographic) normal form* wrt. $<_{\text{lex}}$ if it is minimal wrt. $<_{\text{lex}}$ among all words in $[w]_{\sim_{\widetilde{\Sigma}}}$. There is exactly one word in $[w]_{\sim_{\widetilde{\Sigma}}}$ that is in normal form. For $L \subseteq \Sigma^*$, we write $\text{nf}_{<_{\text{lex}}}(L)$ to denote the set of words from L that are in normal form wrt. $<_{\text{lex}}$. In particular, $w \in \Sigma^*$ is in normal form iff $w \in \text{nf}_{<_{\text{lex}}}([w]_{\sim_{\widetilde{\Sigma}}})$.

Theorem 8 (Ochmański [20]). *If $L \subseteq \Sigma^*$ is a regular set of words in lexicographic normal form wrt. $<_{\text{lex}}$, then $[L]_{\sim_{\widetilde{\Sigma}}}$ is regular.*

It will turn out to be useful to consider an MVPA $\mathcal{A} = (S, \Gamma, \delta, \iota, F)$ over $\widetilde{\Sigma}$ as a finite automaton reading letters over the alphabet $\Sigma \times \Gamma$. Recall that δ is a subset of $S \times \Sigma \times \Gamma \times S$. We will now simply interpret a transition $(s, a, A, s') \in \delta$ as the transition $(s, (a, A), s')$ of a finite automaton with state space S, reading the single letter $(a, A) \in \Sigma \times \Gamma$. In this manner, we obtain from $\mathcal{A}$ a finite automaton, denoted by $\mathcal{F}_{\mathcal{A}}$, which recognizes a regular word language $L(\mathcal{F}_{\mathcal{A}})$ over $\Sigma \times \Gamma$. Though $L(\mathcal{A})$ is in general not even context-free, we can provide a link between $L(\mathcal{A})$ and $L(\mathcal{F}_{\mathcal{A}})$. Indeed, $L(\mathcal{A})$ contains the projections of words from $L(\mathcal{F}_{\mathcal{A}})$ onto their first component if we restrict to *well-formed* words.

In a well-formed word, we take into account that the stack symbols from Γ must obey a pushdown-stack policy. Towards the definition of a well-formed word, we first call a word from Σ^* *p-well-matched* (wrt. $\widetilde{\Sigma}$), for some process $p \in \mathit{Proc}$, if it is generated by the grammar $\mathrm{N} ::= a\,\mathrm{N}\,b \mid \mathrm{N}\,\mathrm{N} \mid \varepsilon \mid c$ where a ranges over Σ_p^c, b over Σ_p^r, and c over $\Sigma \setminus (\Sigma_p^c \cup \Sigma_p^r)$. Now suppose $w = a_1 \dots a_n \in \Sigma^*$. For $i, j \in \{1, \dots, n\}$, we call (i, j) a *matching pair* in w if $i < j$ and there is $p \in \mathit{Proc}$ such that $a_i \in \Sigma_p^c$, $a_j \in \Sigma_p^r$, and $a_{i+1} \dots a_{j-1}$ is p-well-matched. A position $i \in \{1, \dots, n\}$ is called *unmatched* in w if, for every $j \in \{1, \dots, n\}$, neither (i, j) nor (j, i) is a matching pair. We call a word $(a_1, A_1) \dots (a_n, A_n) \in (\Sigma \times \Gamma)^*$ well-formed if (i) for each matching pair (i, j) in $a_1 \dots a_n$, we have $A_i = A_j$, (ii) for all $i \in \{1, \dots, n\}$ such that $a_i \in \Sigma^c$, we have $A_i \neq \bot$, and (iii) for all $i \in \{1, \dots, n\}$ such that $a_i \in \Sigma^r$ and i is unmatched in $a_1 \dots a_n$, we have $A_i = \bot$. We provide a projection mapping $\pi : 2^{(\Sigma \times \Gamma)^*} \to 2^{\Sigma^*}$, which filters from an argument $L \subseteq (\Sigma \times \Gamma)^*$ all the well-formed words and then abstracts away the symbols from Γ. Formally, $\pi(L) = \{w \mid (w, W) \in L \text{ is well-formed}\}$ (here and in the following, we may write a word $(a_1, A_1) \dots (a_n, A_n) \in (\Sigma \times \Gamma)^*$ as the pair $(a_1 \dots a_n, A_1 \dots A_n)$). Though the notion of a well-formed word and the map π actually depend on a given MVPA, we will omit a corresponding index.

Next, we establish a link between an MVPA and its finite automaton. The subsequent lemma then extends Theorem 8 to our recursive setting.

Proposition 9. *For every* MVPA $\mathcal{A}$ *over* $\widetilde{\Sigma}$*, we have* $L(\mathcal{A}) = \pi(L(\mathcal{F}_\mathcal{A}))$*.*

Lemma 10. *Let* $\mathcal{A}$ *be an* MVPA *over* $\widetilde{\Sigma}$ *satisfying* $\mathrm{nf}_{<_{\mathrm{lex}}}([L(\mathcal{A})]_{\sim_{\widetilde{\Sigma}}}) \subseteq L(\mathcal{A})$*. There is a* CVPA $\mathcal{C} = ((S_p)_{p \in \mathit{Proc}}, \Gamma, (\delta_a)_{a \in \Sigma}, \iota, F)$ *over* $\widetilde{\Sigma}$ *such that* $L(\mathcal{C}) = [L(\mathcal{A})]_{\sim_{\widetilde{\Sigma}}}$*. For all* $p \in \mathit{Proc}$*,* $|S_p|$ *is doubly exponential in* $|\mathcal{A}|$ *and triply exponential in* $|\Sigma|$*.*

Proof. We will basically interpret a given MVPA over $\widetilde{\Sigma}$ as an MVPA over a simplified concurrent pushdown alphabet so that Theorems 7 and 8 can be applied. In turn, the resulting automaton will be considered as a CVPA over $\widetilde{\Sigma}$ and will indeed have the desired property.

So let $\mathcal{A} = (S, \Gamma, \delta, \iota, F)$ be an MVPA over $\widetilde{\Sigma}$ such that $\mathrm{nf}_{<_{\mathrm{lex}}}([L(\mathcal{A})]_{\sim_{\widetilde{\Sigma}}}) \subseteq L(\mathcal{A})$. We define a concurrent pushdown alphabet $\widetilde{\Omega} = ((\emptyset, \emptyset, \Sigma_p \times \Gamma))_{p \in \mathit{Proc}}$. In particular, we have $\Omega = \Sigma \times \Gamma$. Note that, for every $(a, A), (b, B) \in \Omega$, $((a, A), (b, B)) \in I_{\widetilde{\Omega}}$ iff $(a, b) \in I_{\widetilde{\Sigma}}$. Now consider any lexicographic order $<'_{\mathrm{lex}} \subseteq \Omega^* \times \Omega^*$ such that, for every $(a, A), (b, B) \in \Omega$, $a <_{\mathrm{lex}} b$ implies $(a, A) <'_{\mathrm{lex}} (b, B)$. Let NF denote the set of all words $x \in \Omega^*$ that are in lexicographic normal form wrt. $<'_{\mathrm{lex}}$, i.e., such that $x \in \mathrm{nf}_{<'_{\mathrm{lex}}}([x]_{\sim_{\widetilde{\Omega}}})$. This set forms a regular word language (cf. [15]) so that the intersection $L(\mathcal{F}_\mathcal{A}) \cap \mathrm{NF}$ is regular, too.

According to Theorem 8, $[L(\mathcal{F}_\mathcal{A}) \cap \mathrm{NF}]_{\sim_{\widetilde{\Omega}}}$ is regular, and Theorem 7 tells us that there is a CVPA $\mathcal{C}$ over the concurrent pushdown alphabet $\widetilde{\Omega}$ such that $L(\mathcal{C}) = [L(\mathcal{F}_\mathcal{A}) \cap \mathrm{NF}]_{\sim_{\widetilde{\Omega}}}$. From $\mathcal{C}$, we obtain an MVPA $\mathcal{C}'$ over $\widetilde{\Sigma}$ with stack alphabet Γ by transforming a transition $(s, (a, A), \mathrm{B}, s')$ into a transition (s, a, A, s') (recall that (a, A) is necessarily contained in Ω^{int} so that B can indeed be neglected). Observe that $\mathcal{C}'$ is actually a CVPA. As $L(\mathcal{F}_{\mathcal{C}'}) = L(\mathcal{C})$ and, by Proposition 9, $\pi(L(\mathcal{F}_{\mathcal{C}'})) = L(\mathcal{C}')$, we deduce $L(\mathcal{C}') = \pi(L(\mathcal{C}))$. So it remains to show that $[L(\mathcal{A})]_{\sim_{\widetilde{\Sigma}}} = \pi(L(\mathcal{C}))$.

Suppose $w \in [L(\mathcal{A})]_{\sim_{\widetilde{\Sigma}}}$. We chose the word $w' \in [w]_{\sim_{\widetilde{\Sigma}}}$ that is in lexicographic normal form wrt. $<_{\mathrm{lex}}$. As $\mathrm{nf}_{<_{\mathrm{lex}}}([L(\mathcal{A})]_{\sim_{\widetilde{\Sigma}}}) \subseteq L(\mathcal{A})$, we have $w' \in L(\mathcal{A})$. Thus, there must be $W' \in \Gamma^*$ such that (w', W') is well-formed and contained in $L(\mathcal{F}_{\mathcal{A}})$ (Proposition 9). As w' is in lexicographic normal form wrt. $<_{\mathrm{lex}}$ and as $<'_{\mathrm{lex}}$ is an extension of $<_{\mathrm{lex}}$, (w', W') is in lexicographic normal form wrt. $<'_{\mathrm{lex}}$ so that $(w', W') \in \mathrm{NF}$. We can now reorder (w', W') in such a way that its first component becomes w. Formally, there is $W \in \Gamma^*$ such that $(w, W) \sim_{\widetilde{\Omega}} (w', W')$. As every word from $[(w', W')]_{\sim_{\widetilde{\Omega}}}$ is well-formed, so is (w, W), and we conclude $w \in \pi([L(\mathcal{F}_{\mathcal{A}}) \cap \mathrm{NF}]_{\sim_{\widetilde{\Omega}}})$.

Now suppose $w \in \pi([L(\mathcal{F}_{\mathcal{A}}) \cap \mathrm{NF}]_{\sim_{\widetilde{\Omega}}})$. We can find an extension $W \in \Gamma^*$ of w such that (w, W) is well-formed and contained in $[L(\mathcal{F}_{\mathcal{A}}) \cap \mathrm{NF}]_{\sim_{\widetilde{\Omega}}}$. Thus, there is $(w', W') \in L(\mathcal{F}_{\mathcal{A}}) \cap \mathrm{NF}$ such that $(w', W') \sim_{\widetilde{\Omega}} (w, W)$. The latter implies $w' \sim_{\widetilde{\Sigma}} w$. Note that (w', W') is well-formed, too, so that, by Proposition 9, $w' \in L(\mathcal{A})$. We conclude $w \in [L(\mathcal{A})]_{\sim_{\widetilde{\Sigma}}}$.

Let us analyze the size of $\mathcal{C}'$. For this, we need to introduce two notions concerning finite automata over Ω. A finite automaton is called *loop-connected* if, for every nonempty word $\alpha_1 \ldots \alpha_n \in \Omega^*$ labeling a path from a state s back to state s, the graph (V, E) is connected, where $V = \{\alpha_i \mid i \in \{1, \ldots, n\}\}$ and $E = (V \times V) \setminus I_{\sim_{\widetilde{\Omega}}}$. It is said to be *$I$-diamond* if, for all pairs $(\alpha, \beta) \in I_{\sim_{\widetilde{\Omega}}}$ and all transitions $r \xrightarrow{\alpha} s \xrightarrow{\beta} t$, we have transitions $r \xrightarrow{\beta} s' \xrightarrow{\alpha} t$ for some state s'. From [15], we know that there is a deterministic loop-connected finite automaton $\mathcal{B}_1$ over Ω with $(|\Sigma| + 1)!$ many states that recognizes the set NF. The set of states of $\mathcal{F}_{\mathcal{A}}$ is the same as that of $\mathcal{A}$ so that we obtain, as the product of $\mathcal{F}_{\mathcal{A}}$ and $\mathcal{B}_1$, a finite automaton $\mathcal{B}_2$ of size $n := |\mathcal{A}| \cdot (|\Sigma| + 1)!$ recognizing $L(\mathcal{F}_{\mathcal{A}}) \cap \mathrm{NF}$. As $\mathcal{B}_1$ is loop-connected, so is $\mathcal{B}_2$. According to [15,19], there is an I-diamond finite automaton $\mathcal{B}_3$ over Ω with at most $N := (n^2 \cdot 2^{|\Sigma|})^{(n-1)(|\Sigma|+1)+1}$ many states that recognizes $[L(\mathcal{B}_2)]_{\sim_{\widetilde{\Omega}}}$. In the next step, we constructed, from $\mathcal{B}_3$, a CVPA $\mathcal{C} = ((S'_p)_{p \in Proc}, \Gamma', (\delta'_\alpha)_{\alpha \in \Omega}, \iota', F')$ over $\widetilde{\Omega}$ such that $L(\mathcal{C}) = L(\mathcal{B}_3)$. From [13], we know that, for all $p \in Proc$, $|S'_p|$ can be bounded by $2^{N^2 \cdot (|\Sigma|^2 + |\Sigma|) + 2|\Sigma|^4}$. As $\mathcal{C}'$ and $\mathcal{C}$ have the same local states, we conclude that the number of local states of $\mathcal{C}'$ is doubly exponential in $|\mathcal{A}|$ and triply exponential in $|\Sigma|$.

Alternatively, we can apply the construction from [4] to the I-diamond finite automaton $\mathcal{B}_3$. Then, $\mathcal{C}'$ has more nondeterminism, and its number of local states is exponential in $|S|$ and doubly exponential in $|\Sigma|$ and $|\Gamma|$. $\square$

Since $L = [L]_{\sim_{\widetilde{\Sigma}}}$ implies $\mathrm{nf}_{<_{\mathrm{lex}}}([L]_{\sim_{\widetilde{\Sigma}}}) \subseteq L$, we obtain, by Lemma 10, the following extension of Zielonka's Theorem.

Theorem 11. *Let $\mathcal{A}$ be an* MVPA *over $\widetilde{\Sigma}$ such that $L(\mathcal{A})$ is $\sim_{\widetilde{\Sigma}}$-closed. There is a* CVPA *$\mathcal{C} = ((S_p)_{p \in Proc}, \Gamma, (\delta_a)_{a \in \Sigma}, \iota, F)$ over $\widetilde{\Sigma}$ satisfying $L(\mathcal{C}) = L(\mathcal{A})$. For all $p \in Proc$, $|S_p|$ is doubly exponential in $|\mathcal{A}|$ and triply exponential in $|\Sigma|$.*

This result demonstrates that MVPA recognizing a $\sim_{\widetilde{\Sigma}}$-closed language are suitable specifications for CVPA. Unfortunately, it is in general undecidable if an MVPA has this property, which can be easily shown by a reduction from the undecidable emptiness problem. However, a restriction to k-phase words allows us to define decidable sufficient criteria for the transformation of an Mvpa into a Cvpa. We will state a Zielonka-like theorem that is tailored to this restriction. There, we require that an Mvpa *represents* the k-phase words of a system, while the final implementation can produce non-k-phase executions.

Definition 12. *For $k \in \mathbb{N}$, we call a language $L \subseteq \mathrm{W}_k(\widetilde{\Sigma})$ a k-*phase representation *if, for all $u, v \in \Sigma^*$ and $(a,b) \in I_{\widetilde{\Sigma}}$ with $\{uabv, ubav\} \subseteq \mathrm{W}_k(\widetilde{\Sigma})$, we have $uabv \in L$ iff $ubav \in L$.*

Next, we show that the closure of a k-phase representation that is given by an Mvpa can be realized as a Cvpa.

Theorem 13. *Let $k \in \mathbb{N}$ and let $\mathcal{A}$ be an* Mvpa *over $\widetilde{\Sigma}$ such that $L_k(\mathcal{A})$ is a k-phase representation. There is a* Cvpa *$\mathcal{C} = ((S_p)_{p \in Proc}, \Gamma, (\delta_a)_{a \in \Sigma}, \iota, F)$ over $\widetilde{\Sigma}$ such that $L(\mathcal{C}) = [L_k(\mathcal{A})]_{\sim_{\widetilde{\Sigma}}}$. For all $p \in Proc$, $|S_p|$ is doubly exponential in $|\mathcal{A}|$ and k, and triply exponential in $|\Sigma|$.*

Proof. Again, we exploit Lemma 10. Unlike in Theorem 11, we cannot apply it directly, as it is in general impossible to define the lexicographic order $<_{\mathrm{lex}}$ in such a way that $\mathrm{nf}_{<_{\mathrm{lex}}}([L_k(\mathcal{A})]_{\sim_{\widetilde{\Sigma}}}) \subseteq L_k(\mathcal{A})$ if $L_k(\mathcal{A})$ is a k-phase representation. Our trick is to extend $\widetilde{\Sigma}$ by a component that indicates the current phase of a letter. An appropriate definition of a normal form over this extended alphabet will then allow us to apply Lemma 10.

So let $k \in \mathbb{N}$ and let $\mathcal{A} = (S, \Gamma, \delta, \iota, F)$ be an Mvpa over $\widetilde{\Sigma}$ such that $L_k(\mathcal{A})$ is a k-phase representation. Based on $\widetilde{\Sigma}$, we define a new concurrent pushdown alphabet $\widetilde{\Omega}$ by $\Omega_p^c = \Sigma_p^c \times \{1, \ldots, k\}$, $\Omega_p^r = \Sigma_p^r \times \{1, \ldots, k\}$, and $\Omega_p^{int} = \Sigma_p^{int} \times \{1, \ldots, k\}$ for all $p \in Proc$. From $\mathcal{A}$, one can construct an Mvpa $\mathcal{B}$ over $\widetilde{\Omega}$ accepting the words $(a_1, ph_1) \ldots (a_n, ph_n)$ such that both $a_1 \ldots a_n \in L_k(\mathcal{A})$ and, for all $i \in \{1, \ldots, n\}$, $ph_i = \min\{j \in \{1, \ldots, k\} \mid a_1 \ldots a_i$ is a j-phase word$\}$. Intuitively, the additional components ph_i give rise to a unique *tight* factorization of $a_1 \ldots a_n$ into phases (cf. [16]). Now consider any lexicographic order $<'_{\mathrm{lex}} \subseteq \Omega^* \times \Omega^*$ such that $i < j$ implies $(a, i) <'_{\mathrm{lex}} (b, j)$ and, moreover, $a <_{\mathrm{lex}} b$ implies $(a, i) <'_{\mathrm{lex}} (b, i)$. We claim that $L(\mathcal{B})$ contains, for every word $x \in L(\mathcal{B})$, the normal form of x wrt. $<'_{\mathrm{lex}}$. Indeed $x \in L(\mathcal{B})$ can be written as a concatenation $x_1 \cdot \ldots \cdot x_k$ with $x_i \in (\Sigma \times \{i\})^*$ for all $i \in \{1, \ldots, k\}$. I.e., for two letters α and β occurring in x_i and, respectively, x_j with $i < j$, we have $\alpha <'_{\mathrm{lex}} \beta$. In particular, the normal form of x can be obtained by reordering letters within the factors x_i, i.e., $\mathrm{nf}_{<'_{\mathrm{lex}}}([x]_{\sim_{\widetilde{\Omega}}}) \subseteq \mathrm{nf}_{<'_{\mathrm{lex}}}([x_1]_{\sim_{\widetilde{\Omega}}}) \cdot \ldots \cdot \mathrm{nf}_{<'_{\mathrm{lex}}}([x_k]_{\sim_{\widetilde{\Omega}}})$. Note that the reordering does not increase the number of phases. As $L_k(\mathcal{A})$ is a k-phase representation, the reordering also preserves containment in $L(\mathcal{B})$ and we have $\mathrm{nf}_{<'_{\mathrm{lex}}}([x]_{\sim_{\widetilde{\Omega}}}) \subseteq L(\mathcal{B})$. By Lemma 10, there is a Cvpa $\mathcal{C}$ over $\widetilde{\Omega}$ with $L(\mathcal{C}) = [L(\mathcal{B})]_{\sim_{\widetilde{\Omega}}}$. Consider the projection from $\widetilde{\Omega}$ to $\widetilde{\Sigma}$ that is induced by the function $f : \Omega \to \Sigma$ given by $f((a, i)) = a$. It is easy to see that applying the projection to a Cvpa language over $\widetilde{\Omega}$ yields a Cvpa language over $\widetilde{\Sigma}$ (this was shown for Mvpa in [16]). Thus, there is a Cvpa $\mathcal{C}'$ over $\widetilde{\Sigma}$ such that $L(\mathcal{C}') = f([L(\mathcal{B})]_{\sim_{\widetilde{\Omega}}})$ (where f is canonically extended to words and, then, to languages). As $f([L(\mathcal{B})]_{\sim_{\widetilde{\Omega}}}) = [f(L(\mathcal{B}))]_{\sim_{\widetilde{\Sigma}}} = [L_k(\mathcal{A})]_{\sim_{\widetilde{\Sigma}}}$, we are done.

To establish the number of local states, observe that $|\mathcal{B}|$ can be bounded by $|\mathcal{A}| \cdot |\Sigma| \cdot (k+1)$. The rest of the construction follows that from the proof of Lemma 10. □

Remark 14. The transformations in the proofs of Lemma 10 and Theorems 11 and 13 are effective. In particular, one can explicitly give a decomposition of states and transitions of the Cvpa, as required in Definition 5.

When we restrict to k-phase words, it is actually decidable whether the previous theorems are applicable to a given Mvpa:

Theorem 15. *The following problems are decidable in elementary time:*
Input*: Concurrent pushdown alphabet* $\widetilde{\Sigma}$*;* $k \in \mathbb{N}$*;* Mvpa $\mathcal{A}$ *over* $\widetilde{\Sigma}$*.*
Question 1*: Is* $L_k(\mathcal{A})$ $\sim_{\widetilde{\Sigma}}$*-closed?*
Question 2*: Is* $L_k(\mathcal{A})$ *a* k*-phase representation?*

Proof. Our proof is inspired by [21] where similar problems are addressed in the finite setting. The main difficulty, however, arises from the presence of stacks.

We first show decidability of Question 1. Let $k \in \mathbb{N}$ and let furthermore $\mathcal{A}_1 = (S_1, \Gamma_1, \delta_1, \iota_1, F_1)$ be the Mvpa over $\widetilde{\Sigma}$ in question. By Theorem 4, one can obtain from $\mathcal{A}_1$ a further Mvpa $\mathcal{A}_2 = (S_2, \Gamma_2, \delta_2, \iota_2, F_2)$ over $\widetilde{\Sigma}$ such that $L(\mathcal{A}_2) = \overline{L_k(\mathcal{A}_1)}$. We will now construct an Mvpa $\mathcal{A}$ over $\widetilde{\Sigma}$ recognizing words of the form $uabv$ with $u, v \in \Sigma^*$, $(a,b) \in I_{\widetilde{\Sigma}}$, and both $uabv \in L(\mathcal{A}_1)$ and $ubav \in L(\mathcal{A}_2)$. Thus, if $L(\mathcal{A})$ contains a k-phase word $uabv$, then $uabv$ is contained in $L_k(\mathcal{A}_1)$ and $ubav$ (which is a $(k+2)$-phase word) is equivalent to $uabv$, but not contained in $L_k(\mathcal{A}_1)$. Indeed, $L_k(\mathcal{A}_1) \neq [L_k(\mathcal{A}_1)]_{\sim_{\widetilde{\Sigma}}}$ iff $L_k(\mathcal{A}) \neq \emptyset$. The latter question is decidable (Theorem 3).

The set of states of $\mathcal{A}$ is $S_1 \times S_2 \times (\{0,1\} \cup (I_{\widetilde{\Sigma}} \times \Gamma_2 \times \Gamma_2))$. The first component of a state is used to simulate $\mathcal{A}_1$, while the second component simulates $\mathcal{A}_2$. The third component starts in 0. In states of the form $(s_1, s_2, 0)$, both automata proceed synchronously: Reading $a \in \Sigma$, $\mathcal{A}$ applies a-transitions $(s_1, a, A_1, s_1') \in \delta_1$ and $(s_2, a, A_2, s_2') \in \delta_2$ to the first and the second component, respectively, resulting in a global step $((s_1, s_2, 0), a, (A_1, A_2), (s_1', s_2', 0))$. The stack alphabet is extended to $\Gamma_1 \times \Gamma_2$ to take into account that A_1 and A_2 can be different.

When reading an input word, $\mathcal{A}_1$ should eventually perform an action sequence ab with $(a,b) \in I_{\widetilde{\Sigma}}$, while $\mathcal{A}_2$ executes ba. So suppose $\mathcal{A}$ is about to simulate transitions (s_1, a, A_1, s_1') followed by (s_1', b, B_1, s_1'') in $\mathcal{A}_1$ and (s_2, b, B_2, s_2') followed by (s_2', a, A_2, s_2'') in $\mathcal{A}_2$. The global automaton $\mathcal{A}$ will produce this transition sequence "crosswise". It will first read the a and apply the transition involving $A_1 \in \Gamma_1$ to the first component. At the same time, the second component only changes its local state into s_2'. As the stack symbol B_2 cannot be applied directly, it is stored in the third component of the subsequent global state of $\mathcal{A}$, which is of the form $(s_1', s_2', ((a,b), B_2, A_2))$. Observe that A_2, which is associated to executing a in $\mathcal{A}_2$, must be applied together with reading a so that (A_1, A_2) acts as the stack symbol. Since a corresponding local transition (s_2', a, A_2, s_2'') has to follow in $\mathcal{A}_2$, the stack symbol A_2 needs to be stored as well. The formal description of this step can be found below (2). Now, being in the global state $(s_1', s_2', ((a,b), B_2, A_2))$, $\mathcal{A}$ will, according to the local transition (s_1', b, B_1, s_1''), perform a b and apply (B_1, B_2) to the designated stack. Again, $\mathcal{A}_2$ will only change its local state into s_2''. However, the local transition has to conform to the symbol A_2 that had been stored. This step corresponds to rule (3) below. We are now in a global state of the form $(s_1'', s_2'', 1)$. In states with 1 in the third position, $\mathcal{A}_1$ and $\mathcal{A}_2$ again act simultaneously (rule (1)).

Formally, $\mathcal{A} = (S, \Gamma, \delta, \iota, F)$ is given by $S = S_1 \times S_2 \times (\{0,1\} \cup (I_{\widetilde{\Sigma}} \times \Gamma_2 \times \Gamma_2))$, $\Gamma = \Gamma_1 \times \Gamma_2$, $\iota = (\iota_1, \iota_2, 0)$, and $F = F_1 \times F_2 \times \{1\}$. Let $(s_1, s_2, \sigma), (s_1', s_2', \sigma') \in S$, $a \in \Sigma$, and $(A_1, A_2) \in \Gamma$. Then, $((s_1, s_2, \sigma), a, (A_1, A_2), (s_1', s_2', \sigma')) \in \delta$ if there are $(B_1, B_2) \in \Gamma$ and $b \in \Sigma$ such that one of the following holds:

(1) $(\sigma = \sigma' = 0$ or $\sigma = \sigma' = 1)$, $(s_1, a, A_1, s_1') \in \delta_1$, and $(s_2, a, A_2, s_2') \in \delta_2$, or
(2) $\sigma = 0$, $\sigma' = ((a,b), B_2, A_2)$, $(s_1, a, A_1, s_1') \in \delta_1$, and $(s_2, b, B_2, s_2') \in \delta_2$, or
(3) $\sigma' = 1$, $\sigma = ((b,a), A_2, B_2)$, $(s_1, a, A_1, s_1') \in \delta_1$, and $(s_2, b, B_2, s_2') \in \delta_2$.

The only difference in the decision procedure for Question 2 is that $\mathcal{A}_2$ is such that $L(\mathcal{A}_2) = L_k(\mathcal{A}_2) = \overline{L_k(\mathcal{A}_1)} \cap \mathrm{W}_k(\widetilde{\Sigma})$.

An inspection of the constructions from [16] tells us that the size of $\mathcal{A}_2$ is in both cases triply exponential in $|\mathcal{A}_1|$, k, and $|Proc|$. As emptiness of MVPA wrt. k-phase words is decidable in doubly exponential time, we obtain elementary decision procedures for Question 1 and Question 2. $\square$

4 Specifying Programs in MSO Logic

In Section 3, we considered the language L of an MVPA to be a specification, and our aim was to find a CVPA $\mathcal{C}$ such that $L(\mathcal{C}) = [L]_{\sim_{\widetilde{\Sigma}}}$. Unfortunately, one cannot always find such a CVPA (consider, e.g., $L = (ab)^*$ with $(a,b) \in I_{\widetilde{\Sigma}}$). We now present a specification language that operates directly on equivalence classes of $\sim_{\widetilde{\Sigma}}$ so that, provided that we restrict to k-phase executions, any specification can be realized as a CVPA. In doing so, we extend the classical connection between monadic second-order (MSO) logic and finite automata. The study of relations between logical formalisms that may serve as a specification language and automata has had many generalizations, including MVPA [16].

Actually, we present an MSO logic that is interpreted over partial orders, which arise naturally from words in the presence of a concurrent pushdown alphabet and the induced independence relation. Any such partial order represents one equivalence class of words so that a formula defines a set of equivalence classes or, in other words, a set of words that is $\sim_{\widetilde{\Sigma}}$-closed.

Let $w = a_1 \ldots a_n \in \Sigma^*$. To w, we associate the labeled structure $T_{\widetilde{\Sigma}}(w) = (E, \preceq, \mu, \lambda)$, where $E = \{1, \ldots, n\}$ is the set of *events*, $\lambda : E \to \Sigma$ assigns to any event $i \in E$ the action $\lambda(i) = a_i$ it executes, and $\mu \subseteq E \times E$ contains the matching pairs in w (i.e., $(i,j) \in \mu$ iff (i,j) is a matching pair). Finally, $\preceq \subseteq E \times E$ is a partial-order relation (i.e., it is reflexive, transitive, and antisymmetric), which is defined to be the transitive closure of $\{(i,j) \in E \times E \mid i \leq j$ and $(a_i, a_j) \notin I_{\widetilde{\Sigma}}\}$.

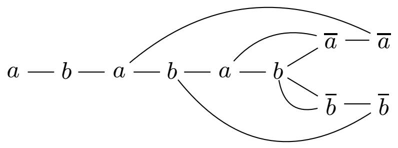


Fig. 2. A nested trace

We call the structure $T_{\widetilde{\Sigma}}(w)$ that arises from a word $w \in \Sigma^*$ a *nested trace* over $\widetilde{\Sigma}$. The set of nested traces over $\widetilde{\Sigma}$ is denoted by $\mathrm{Tr}(\widetilde{\Sigma})$. It is standard to prove that $T_{\widetilde{\Sigma}}(w) = T_{\widetilde{\Sigma}}(w')$ iff $w \sim_{\widetilde{\Sigma}} w'$ where we consider equality of nested traces up to isomorphism. In other words, there is a one-to-one correspondence between nested traces and equivalence classes of $\sim_{\widetilde{\Sigma}}$. We remark that nested traces are a merge of Mazurkiewicz traces [10] and *nested words* [3], which, in turn, generalize themselves the notion of a word.

Example 16. Figure 2 depicts $T = T_{\widetilde{\Sigma}}(a\,b\,a\,b\,a\,b\,\overline{a}\,\overline{a}\,\overline{b}\,\overline{b}) = T_{\widetilde{\Sigma}}(a\,b\,a\,b\,a\,b\,\overline{b}\,\overline{b}\,\overline{a}\,\overline{a})$ where $\widetilde{\Sigma}$ is taken from Example 1. Hereby, the straight edges form the cover relation $\prec \setminus \prec^2$ of the underlying partial-order relation $\preceq$, and the curved edges represent μ, i.e., the matching pairs. There are two unmatched events in T.

Fixing supplies of first-order variables $x, y, \ldots$ and second-order variables $X, Y, \ldots$, the syntax of our MSO logic complies with the signature of a nested trace. Formally, formulas from $\mathrm{MSO}(\widetilde{\Sigma})$ are given by the grammar

$$\varphi ::= x \preceq y \;\mid\; (x, y) \in \mu \;\mid\; \lambda(x) = a \;\mid\; x \in X \;\mid\; \neg\varphi \;\mid\; \varphi_1 \vee \varphi_2 \;\mid\; \exists x\varphi \;\mid\; \exists X\varphi$$

where x and y are first-order variables, X is a second-order variable, and $a \in \Sigma$. Moreover, one may use the usual abbreviations such as $\varphi_1 \wedge \varphi_2$, $\varphi_1 \rightarrow \varphi_2$, and $\forall x\varphi$. To determine the semantics, let $T = (E, \preceq, \mu, \lambda)$ be a nested trace over $\widetilde{\Sigma}$ and $\mathbb{I}$ be an interpretation function, which assigns to a first-order variable an element from E and to a second-order variable a subset of E. Let us define when $T, \mathbb{I} \models \varphi$ for $\varphi \in \mathrm{MSO}(\widetilde{\Sigma})$. Namely, $T, \mathbb{I} \models x \preceq y$ if $\mathbb{I}(x) \preceq \mathbb{I}(y)$, $T, \mathbb{I} \models (x, y) \in \mu$ if $(\mathbb{I}(x), \mathbb{I}(y)) \in \mu$, and $T, \mathbb{I} \models \lambda(x) = a$ if $\lambda(\mathbb{I}(x)) = a$. The rest of the semantics is classical for MSO logics. If φ is a sentence, i.e., a formula without free variables, we can write $T \models \varphi$ if $T, \mathbb{I} \models \varphi$ for some interpretation function $\mathbb{I}$. Now, given a sentence $\varphi \in \mathrm{MSO}(\widetilde{\Sigma})$, we set $\mathscr{L}(\varphi) = \{T \in \mathrm{Tr}(\widetilde{\Sigma}) \mid T \models \varphi\}$.

As the language of a Cvpa $\mathcal{C}$ is closed under $\sim_{\widetilde{\Sigma}}$, it is legitimate to assign to $\mathcal{C}$ a set of nested traces, too, letting $\mathscr{L}(\mathcal{C}) = \{T_{\widetilde{\Sigma}}(w) \mid w \in L(\mathcal{C})\}$.

Example 17. Suppose T to be the nested trace given in Figure 2 and consider the sentences $\varphi_1 = \forall x\,((\lambda(x) = a \vee \lambda(x) = b) \rightarrow \exists y\,(x, y) \in \mu)$ expressing that there is no pending call, and $\varphi_2 = \forall x\,((\lambda(x) = \overline{a} \vee \lambda(x) = \overline{b}) \rightarrow \exists y\,(y, x) \in \mu)$, which expresses that there is no pending return. We have $T \notin \mathscr{L}(\varphi_1)$ but $T \in \mathscr{L}(\varphi_2)$. Note also that $T \in \mathscr{L}(\mathcal{C})$ for the Cvpa $\mathcal{C}$ from Example 6 (Figure 1).

Before we look at a logical characterization of general Cvpa, let us recall a result that has already been found in the context of asynchronous automata, i.e., of Cvpa over simple concurrent pushdown alphabets.

Theorem 18 (Thomas [26]). *Suppose $\Sigma = \Sigma^{int}$ and let $\mathscr{L} \subseteq \mathrm{Tr}(\widetilde{\Sigma})$. Then, $\mathscr{L} = \mathscr{L}(\mathcal{C})$ for some* Cvpa *$\mathcal{C}$ over $\widetilde{\Sigma}$ iff $\mathscr{L} = \mathscr{L}(\varphi)$ for some $\varphi \in \mathrm{MSO}(\widetilde{\Sigma})$.*

Now let us turn towards Cvpa over general concurrent pushdown alphabets. It has been shown in [5] that MSO logic is in general strictly more expressive than Cvpa. We will therefore extend the notion of k-phase words to nested traces. For $k \in \mathbb{N}$, a nested trace $T \in \mathrm{Tr}(\widetilde{\Sigma})$ is called a *k-phase trace* if there is $w \in \mathrm{W}_k(\widetilde{\Sigma})$ such that $T_{\widetilde{\Sigma}}(w) = T$. The set of k-phase traces over $\widetilde{\Sigma}$ is denoted by $\mathrm{Tr}_k(\widetilde{\Sigma})$. For example, the nested trace T from Figure 2 is a 2-phase trace, even though we have $T = T_{\widetilde{\Sigma}}(w)$ for $w = a\,b\,a\,b\,a\,b\,\overline{a}\,\overline{b}\,\overline{a}\,\overline{b} \notin \mathrm{W}_2(\widetilde{\Sigma})$. The domain of k-phase traces is particularly interesting, because it is decidable whether $\mathscr{L}(\mathcal{C}) \cap \mathrm{Tr}_k(\widetilde{\Sigma}) \neq \emptyset$ holds for a Cvpa $\mathcal{C}$. To see this, observe that the latter holds iff $L(\mathcal{C}) \cap \mathrm{W}_k(\widetilde{\Sigma}) \neq \emptyset$, which is decidable according to Theorem 3.

For a logical characterization of Cvpa, we will need the following lemma.

Lemma 19. *Let $k \in \mathbb{N}$ and let $\mathcal{C}$ be a* Cvpa *over $\widetilde{\Sigma}$ such that $\mathscr{L}(\mathcal{C}) \subseteq \mathrm{Tr}_k(\widetilde{\Sigma})$. There is a* Cvpa *$\mathcal{C}'$ over $\widetilde{\Sigma}$ such that $\mathscr{L}(\mathcal{C}') = \overline{\mathscr{L}(\mathcal{C})} \cap \mathrm{Tr}_k(\widetilde{\Sigma})$, where $\overline{\mathscr{L}(\mathcal{C})} = \mathrm{Tr}(\widetilde{\Sigma}) \setminus \mathscr{L}(\mathcal{C})$.*

Proof. Let $k \in \mathbb{N}$ and let $\mathcal{C}$ be a Cvpa over $\widetilde{\Sigma}$ satisfying $\mathscr{L}(\mathcal{C}) \subseteq \mathrm{Tr}_k(\widetilde{\Sigma})$. Due to Theorem 4, there is an Mvpa $\mathcal{A}$ over $\widetilde{\Sigma}$ such that $L_k(\mathcal{A}) = \overline{L_k(\mathcal{C})} \cap \mathrm{W}_k(\widetilde{\Sigma})$. Observe that $L_k(\mathcal{A})$ is a k-phase representation. Thus, by Theorem 13, there is a Cvpa $\mathcal{C}'$ over $\widetilde{\Sigma}$ such that $L(\mathcal{C}') = [L_k(\mathcal{A})]_{\sim_{\widetilde{\Sigma}}}$. One easily verifies that we actually have $\mathscr{L}(\mathcal{C}') = \overline{\mathscr{L}(\mathcal{C})} \cap \mathrm{Tr}_k(\widetilde{\Sigma})$. ☐

As a corollary, we obtain that, for every $k \in \mathbb{N}$, there is a Cvpa $\mathcal{C}$ with $\mathscr{L}(\mathcal{C}) = \mathrm{Tr}_k(\widetilde{\Sigma})$. This is an important fact in the proof of Theorem 21. Indeed, the following two theorems constitute a logical characterization of Cvpa (restricted to k-phase traces). Both transformations are effective. Hereby, Theorem 20 has a standard proof (see [26] for a similar instance of that problem).

Theorem 20. *For every* Cvpa *$\mathcal{C}$ over $\widetilde{\Sigma}$, there is a sentence $\varphi \in \mathrm{MSO}(\widetilde{\Sigma})$ such that $\mathscr{L}(\varphi) = \mathscr{L}(\mathcal{C})$.*

Theorem 21. *Let $k \in \mathbb{N}$. For every sentence $\varphi \in \mathrm{MSO}(\widetilde{\Sigma})$, there is a* Cvpa *$\mathcal{C}$ over $\widetilde{\Sigma}$ such that $\mathscr{L}(\mathcal{C}) = \mathscr{L}(\varphi) \cap \mathrm{Tr}_k(\widetilde{\Sigma})$.*

Proof (sketch). As usual, one proceeds by induction on the structure of an MSO formula. However, treating negation is less obvious than in classical settings such as words and trees. To get a Cvpa for $\neg\varphi$, let $k \in \mathbb{N}$ and suppose that we already have a Cvpa $\mathcal{C}$ over $\widetilde{\Sigma}$ such that $\mathscr{L}(\mathcal{C}) = \mathscr{L}(\varphi) \cap \mathrm{Tr}_k(\widetilde{\Sigma})$ (actually, we need to consider extended concurrent pushdown alphabets to cope with free variables during the inductive translation). By Lemma 19, there is a Cvpa $\mathcal{C}'$ such that $\mathscr{L}(\mathcal{C}') = \overline{\mathscr{L}(\mathcal{C})} \cap \mathrm{Tr}_k(\widetilde{\Sigma})$. The latter equals $\mathscr{L}(\neg\varphi) \cap \mathrm{Tr}_k(\widetilde{\Sigma})$ so that we are done. The translations of atomic formulas, disjunction, and existential quantification exploit the fact that $\mathrm{Tr}_k(\widetilde{\Sigma})$ is recognizable by some Cvpa and that Cvpa are closed under union, intersection, and projection. ☐

5 Future Directions

Though the results in this paper are of rather theoretical nature, due to the high complexity of our constructions, Cvpa and the related notion of a nested trace may open a new line of research in Mazurkiewciz trace theory and the analysis of multithreaded recursive programs. We mention here some future directions:

We excluded an important question from our study: For $k \in \mathbb{N}$ and an Mvpa $\mathcal{A}$, when can we decide whether $[L_k(\mathcal{A})]_{\sim_{\widetilde{\Sigma}}}$ is the language of some Mvpa and, hence, of some Cvpa? If $\Sigma = \Sigma^{int}$, we know that this is the case iff $I_{\widetilde{\Sigma}} \cup \mathrm{id}_{\Sigma}$ is transitive [24]. In the general setting, the question remains open.

Given an Mvpa $\mathcal{A}$, one may ask if $\mathcal{A}$ is already a Cvpa such that its local state spaces and transition relations can be computed effectively. Those questions are addressed and answered positively in [18,8] for asynchronous automata.

In Cvpa, processes communicate via shared memory. It will be interesting to study extensions of communicating finite-state machines (CFMs), where processes communicate via first-in first-out channels, by visibly pushdown stacks. While Cvpa recognize sets of nested traces, a *visibly pushdown CFM* would give rise to the notion of a *nested message sequence chart*. Interestingly, there are theorems for CFMs that constitute counterparts of Zielonka's Theorem [12,14].

For both Mazurkiewicz traces [9] and nested words [1], temporal logics were studied. We raise the question if these logics can be combined towards specification formalisms with decidable satisfiability and model-checking problems.

In a distributed setting, deadlock-free systems are particularly important. The paper [8] addresses the problem of synthesizing deadlock-free asynchronous automata from regular specifications. It remains to define a notion of deadlock-freeness for our setting and to study if the ideas from [8] can be adopted.

References

1. Alur, R., Arenas, M., Barceló, P., Etessami, K., Immerman, N., Libkin, L.: First-order and temporal logics for nested words. Logical Methods in Computer Science 4(4:11), 1–44 (2008)
2. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: STOC 2004, pp. 202–211. ACM Press, New York (2004)
3. Alur, R., Madhusudan, P.: Adding nesting structure to words. In: H. Ibarra, O., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 1–13. Springer, Heidelberg (2006)
4. Baudru, N., Morin, R.: Unfolding synthesis of asynchronous automata. In: Grigoriev, D., Harrison, J., Hirsch, E.A. (eds.) CSR 2006. LNCS, vol. 3967, pp. 46–57. Springer, Heidelberg (2006)
5. Bollig, B.: On the expressive power of 2-stack visibly pushdown automata. Logical Methods in Computer Science 4(4:16), 1–35 (2008)
6. Bouajjani, A., Esparza, J., Touili, T.: A generic approach to the static analysis of concurrent programs with procedures. International Journal on Foundations of Computer Science 14(4), 551–582 (2003)
7. Castellani, I., Mukund, M., Thiagarajan, P.S.: Synthesizing distributed transition systems from global specifications. In: Pandu Rangan, C., Raman, V., Ramanujam, R. (eds.) FST TCS 1999. LNCS, vol. 1738, pp. 219–231. Springer, Heidelberg (1999)

8. Ştefănescu, A., Esparza, J., Muscholl, A.: Synthesis of distributed algorithms using asynchronous automata. In: Amadio, R., Lugiez, D. (eds.) CONCUR 2003. LNCS, vol. 2761, pp. 27–41. Springer, Heidelberg (2003)
9. Diekert, V., Gastin, P.: LTL is expressively complete for Mazurkiewicz traces. Journal of Computer and System Sciences 64(2), 396–418 (2002)
10. Diekert, V., Rozenberg, G. (eds.): The Book of Traces. World Scientific, Singapore (1995)
11. Emerson, E.A., Clarke, E.M.: Using branching time temporal logic to synthesize synchronization skeletons. Science of Computer Programming 2, 241–266 (1982)
12. Genest, B., Kuske, D., Muscholl, A.: A Kleene theorem and model checking algorithms for existentially bounded communicating automata. Information and Computation 204(6), 920–956 (2006)
13. Genest, B., Muscholl, A.: Constructing Exponential-Size Deterministic Zielonka Automata. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 565–576. Springer, Heidelberg (2006)
14. Henriksen, J.G., Mukund, M., Narayan Kumar, K., Sohoni, M., Thiagarajan, P.S.: A theory of regular MSC languages. Information and Computation 202(1), 1–38 (2005)
15. Kuske, D.: Weighted asynchronous cellular automata. Theoretical Computer Science 374(1-3), 127–148 (2007)
16. La Torre, S., Madhusudan, P., Parlato, G.: A robust class of context-sensitive languages. In: LICS 2007, pp. 161–170. IEEE Computer Society Press, Los Alamitos (2007)
17. La Torre, S., Madhusudan, P., Parlato, G.: Context-bounded analysis of concurrent queue systems. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 299–314. Springer, Heidelberg (2008)
18. Morin, R.: Decompositions of asynchronous systems. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 549–564. Springer, Heidelberg (1998)
19. Muscholl, A., Peled, D.: Message sequence graphs and decision problems on Mazurkiewicz traces. In: Kutyłowski, M., Wierzbicki, T., Pacholski, L. (eds.) MFCS 1999. LNCS, vol. 1672, pp. 81–91. Springer, Heidelberg (1999)
20. Ochmański, E.: Regular behaviour of concurrent systems. Bulletin of the European Association for Theoretical Computer Science (EATCS) 27, 56–67 (1985)
21. Peled, D., Wilke, T., Wolper, P.: An algorithmic approach for checking closure properties of temporal logic specifications and omega-regular languages. Theoretical Computer Science 195(2), 183–203 (1998)
22. Qadeer, S., Rehof, J.: Context-bounded model checking of concurrent software. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 93–107. Springer, Heidelberg (2005)
23. Ramalingam, G.: Context-sensitive synchronization-sensitive analysis is undecidable. ACM Trans. Prog. Lang. Syst. 22(2), 416–430 (2000)
24. Sakarovitch, J.: The "last" decision problem for rational trace languages. In: Simon, I. (ed.) LATIN 1992. LNCS, vol. 583, pp. 460–473. Springer, Heidelberg (1992)
25. Sen, K., Viswanathan, M.: Model checking multithreaded programs with asynchronous atomic methods. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 300–314. Springer, Heidelberg (2006)
26. Thomas, W.: On logical definability of trace languages. In: Proceedings of Algebraic and Syntactic Methods in Computer Science (ASMICS), Report TUM-I9002, Technical University of Munich, pp. 172–182 (1990)
27. Zielonka, W.: Notes on finite asynchronous automata. R.A.I.R.O. — Informatique Théorique et Applications 21, 99–135 (1987)

Beyond Shapes: Lists with Ordered Data

Kshitij Bansal[1,⋆], Rémi Brochenin[2,⋆⋆], and Etienne Lozes[2]

[1] Chennai Mathematical Institute
kshitij@cmi.ac.in
[2] LSV, ENS Cachan, CNRS
{brocheni,lozes}@lsv.ens-cachan.fr

Abstract. Standard analysis on recursive data structures restrict their attention to shape properties (for instance, a program that manipulates a list returns a list), excluding properties that deal with the actual content of these structures. For instance, these analysis would not establish that the result of merging two ordered lists is an ordered list. Separation logic, one of the prominent framework for these kind of analysis, proposed a heap model that could represent data, but, to our knowledge, no predicate dealing with data has ever been integrated to the logic while preserving decidability. We establish decidability for (first-order) separation logic with a predicate that allows to compare two successive data in a list. We then consider the extension where two data in arbitrary positions may be compared, and establish the undecidability in general. We define a guarded fragment that turns out to be both decidable and sufficiently expressive to prove the preservation of the loop invariant of a standard program merging ordered lists. We finally consider the extension with the magic-wand and prove that, by constrast with the data-free case, even a very restricted use of the magic wand already introduces undecidability.

1 Introduction

Data-ordering and shape analysis. Providing automatic methods for faults detection in programs manipulating recursive mutable data structures is a long-standing problem. Shape analysis is a well established approach that may detect faults due to in-depth properties of the heap, like creating a cycle in an acyclic list. Prominent logics that integrate such an analysis are separation logic [1], pointer assertion logic PAL [9], TVLA [10], LRP (logic of reachable patterns) [16], or alias logic [4], to quote a few examples. A common feature in these analyses is that they completely forget the data held in the recursive structures, focusing on the shape of the structure. As a consequence, ordering properties are out of the scope of these analyses: for instance, one cannot check or even specify that the reverse of a sorted list is a sorted list. Extensions of shape analysis have been proposed for ordering properties, stability properties,

⋆ Supported by P2R MODISTE-COVER/Timed-DISCOVERI, a project under the Indo-French Networking Programme.
⋆⋆ Supported by a DGA/CNRS fellowship.

and size properties, in shape graphs [3], in the TVLA approach [11], and in the separation logic approach [12] to cite a few. This paper studies the rather more theoretical issue of the decidability of the satisfiability problem. It proposes a general approach for reducing the shapes handling ordering properties to pure shapes, and stress some natural limitations we should put on the data properties we would like to check automatically.

Data-ordering in separation logic. Our approach lies in the framework of separation logic [14]. In essence, separation logic extends first order logic with two substructural connectives: the *separation* connective ($*$) and its adjoint (the separating implication $-\!\!*$, also known as the *magic wand*). These connectives are convenient to express pre and post conditions of all standard heap-manipulating instructions. For instance, the strongest post condition $\mathbf{Post}(\mathsf{x} := \mathsf{new}, A)$ of a memory allocation instruction can be expressed by $\mathbf{x}\mapsto - * \exists x.A\{^x/_\mathbf{x}\}$. This formula involves two more ingredients : the use of first-order logical variables, that here quantify over the memory location of $\mathbf{x}$ before allocation, and the points-to predicate $.\hookrightarrow.$ (or its precise version in this example). We extend the logic with the predicate $\mathsf{val}(x) \leq \mathsf{val}(y)$ that asserts that the value stored at the location x is smaller than the one stored at y, which in particular allows to define the predicate

$$x \overset{\leq}{\hookrightarrow} y \quad \overset{\text{def}}{\equiv} \quad x \hookrightarrow y \ \wedge \ \mathsf{val}(x) \leq \mathsf{val}(y)$$

and $x \overset{\geq}{\hookrightarrow} y$ accordingly. We call these predicates *short-distance comparisons*, and by contrast $\mathsf{val}(x) \leq \mathsf{val}(y)$ is called *long-distance comparison.* We moreover say that such a long-distance comparison is *guarded* if x or y is an open variable.

Separation logic's decidability. The decidability of the satisfiability problem for separation logic has been intensively studied so far: first-order separation logic over heap models with at least two selectors (record fields) is known to be undecidable [7] by containment of finite satisfiability for classical predicate logic with one binary relation [15] (even with no separating connectives). On the other hand, first-order separation logic over heaps with one selector has been proved to be decidable when the magic-wand is dropped [6], by reduction to monadic second order logic over functional graphs, but becomes undecidable in presence of magic wand. To our knowledge, nothing was known about first-order separation logic with data. The following table summarizes the results we present in this paper:

Undecidable	long distance comparison without $-\!\!*$ short distance comparison with (restricted) $-\!\!*$
Decidable	short + guarded long distance comparisons without $-\!\!*$

The decidability result comes from a reduction to monadic second order logic over functional graphs. The translation is strongly inspired by the one for separation logic over lists without data [6], but involves some non-trivial complications for ensuring the coherence of data abstraction. The undecidability results are obtained by reduction to first-order logic over (finite) data words, which was proved undecidable [2,8].

Case study. In order to illustrate the practical relevance of our results, we consider a very standard merge-sort program. Checking that *any* formula is a correct loop invariant requires in general to deal with the magic-wand connective, which leads to our undecidable fragments. However, for the loop invariant one may think about (that is, all working lists are ordered lists) the magic wand can be eliminated, and the formula considered falls into the decidable fragment.

Outline of the paper. Section 2 introduces our separation logic over lists with data. In Section 3, we illustrate on the merge program how the logic can deal with relevant loop invariants. In Section 4, we establish the decidability of the short distance comparison. Section 5 deals with the case of guarded and non-guarded long-distance comparison, whereas Section 6 explains the undecidability of the logic in the presence of the magic wand.

2 Preliminaries

In this section, we introduce first the separation logic with data considered in this work, then the monadic second order logic to which we reduce to. These logics are based on different classes of models: our separation logic deals with lists with data, whereas the monadic second order logic deals with shapes, e.g. lists without data.

2.1 A Separation Logic for Lists with Ordered Data

Memory model. We assume an infinite, totally ordered set $(\mathtt{Dat}, \leq)$ of data, and range over a particular datum with α, β. We moreover assume an infinite set $\mathtt{Loc}$ of locations, ranged over with l, l' etc. and an infinite set $\mathtt{Var}$ of variables, ranged over with either x, y, z or $\mathbf{x}, \mathbf{y}, \mathbf{z}$ etc. Variables can be interpreted as both variables from the programs or logical variables quantifying over locations; the main difference between both is that program variables are never quantified in the formula. We safely identify them and will use the font convention $\mathbf{x}$, $\mathbf{y}$ to emphasize that a variable should be understood as a program variable. In the latter, we may use the standard notation $A\{^y/_x\}$ for the formula A in which x replaces y.

Following the standard semantics of separation logic, we define a memory state as a pair of a store s and a heap h such that:

- $s : \mathtt{Var} \to \mathtt{Loc}$,
- $h : \mathtt{Loc} \rightharpoonup (\mathtt{Loc} \times \mathtt{Dat})$ is a partial function with finite domain.

We write $\mathtt{dom}(h)$ to denote the domain of h and $\mathtt{ran}(h)$ to denote its range. For $\mathcal{Z} \subseteq \mathtt{dom}(h)$, We write $h_{|\mathcal{Z}}$ to denote the restriction of h to $\mathcal{Z}$. We write fst and snd to denote the first and second projection on a product set. We write $h \perp h'$ if $\mathtt{dom}(h) \cap \mathtt{dom}(h') = \emptyset$, and the heap composition $h * h'$ is defined as $h \cup h'$ when $h \perp h'$.

Example 1 (Ordered lists). Programs manipulating ordered lists of integers can be modeled choosing $\mathtt{Dat} = \mathbb{Z}$ with the standard order. The same holds for lists of reals, lists of naturals, and so on.

Example 2 (Fine-grained concurrent lists). `Dat` could be thought as the state of a lock at the current node, that is the identifier of the thread holding the node (or some constant for an available lock). Here, the ordering on data is not relevant, but the equality between data is. For such a model, one may want to express, for instance, that every thread holds the locks of at most two nodes of a list, and that these nodes are necessarily consecutive.

Separation logic. We now define our assertion language $\mathtt{SL}_<$ by extending the standard separation logic with a comparison predicate. We assume a set `DVar` of data variables, ranged over with v, w, etc. A valuation interpreting data variables is a function $\rho : \mathtt{DVar} \to \mathtt{Dat}$.

Formulae of $\mathtt{SL}_<$ are defined by the grammar below.

$$\phi ::= \neg\phi \mid \phi \wedge \phi \mid \exists x.\phi \mid \exists v.\phi \mid x \hookrightarrow y \mid \mathsf{val}(x) \leq v \mid \mathsf{val}(x) \geq v \mid x = y \mid \phi * \phi \mid \phi \mathrel{-\!\!*} \phi$$

The semantics of the formulae is defined as usual, with the expected definition for the predicates $\mathsf{val}(x) \leq v$ and $\mathsf{val}(x) \geq v$.

$$\begin{array}{lll}
(s,h),\rho \models_{\mathtt{SL}} \phi \wedge \psi & \text{iff} & (s,h),\rho \models_{\mathtt{SL}} \phi \text{ and } (s,h),\rho \models_{\mathtt{SL}} \psi \\
(s,h),\rho \models_{\mathtt{SL}} \neg\phi & \text{iff} & \text{not } (s,h),\rho \models_{\mathtt{SL}} \phi \\
(s,h),\rho \models_{\mathtt{SL}} \exists x.\ \phi & \text{iff} & \text{there is } l \in \mathtt{Loc} \text{ such that } (s[x \mapsto l], h),\rho \models_{\mathtt{SL}} \phi \\
(s,h),\rho \models_{\mathtt{SL}} \exists v.\ \phi & \text{iff} & \text{there is } \alpha \in \mathtt{Dat} \text{ such that } (s,h),\rho[v \mapsto \alpha] \models_{\mathtt{SL}} \phi \\
(s,h),\rho \models_{\mathtt{SL}} x \hookrightarrow y & \text{iff} & \text{there is } \alpha \in \mathtt{Dat} \text{ such that } h(s(x)) = (s(y),\alpha) \\
(s,h),\rho \models_{\mathtt{SL}} \mathsf{val}(x) \leq v & \text{iff} & \text{there is } \alpha \in \mathtt{Dat} \text{ and } l \in \mathtt{Loc} \text{ such that} \\
 & & h(s(x)) = (l,\alpha), \text{ and } \alpha \leq \rho(v) \\
(s,h),\rho \models_{\mathtt{SL}} \mathsf{val}(x) \geq v & \text{iff} & \text{there is } \alpha \in \mathtt{Dat} \text{ and } l \in \mathtt{Loc} \text{ such that} \\
 & & h(s(x)) = (l,\alpha), \text{ and } \alpha \geq \rho(v) \\
(s,h),\rho \models_{\mathtt{SL}} x = y & \text{iff} & s(x) = s(y) \\
(s,h),\rho \models_{\mathtt{SL}} \phi_1 * \phi_2 & \text{iff} & \text{there are two heaps } h_1, h_2 \text{ such that} \\
 & & h = h_1 * h_2 \text{ and } (s,h_i),\rho \models_{\mathtt{SL}} \phi_i,\ i = 1,2 \\
(s,h),\rho \models_{\mathtt{SL}} \phi_1 \mathrel{-\!\!*} \phi_2 & \text{iff} & \text{for all } h', \text{if } h \perp h' \text{ and } (s,h'),\rho \models_{\mathtt{SL}} \phi_1, \\
 & & \ \text{then } (s, h * h'),\rho \models_{\mathtt{SL}} \phi_2
\end{array}$$

Note that, due to our memory model, the natural semantics of $\mathsf{val}(x) \leq v$ implies in particular $\exists z.x \hookrightarrow z$.

Derived formulae. We use standard notations $\vee, \forall, \Rightarrow$, and write $\mathsf{val}(x) = v$, $\mathsf{val}(x) \leq \mathsf{val}(y)$,... for the obvious combinations of comparison predicates. We write $\mathsf{precisely}(A)$ to denote $A \wedge \neg(A * \exists x, y.x \hookrightarrow y)$. We also abbreviate $\phi \mathrel{-\!\!\circledast} \psi$ for the sometimes called septraction connective defined by $\neg(\phi \mathrel{-\!\!*} \neg\psi)$. We use the wildcard notation, e.g. $x \hookrightarrow -$ for $\exists y.x \hookrightarrow y$, the so-called precise predicates $\mapsto$

(e.g. $x \mapsto y$ abbreviates $\mathsf{precisely}(x \hookrightarrow y)$), and equality over vectors $(x_1, .., x_n) = (y_1, .., y_n)$. We will also use the following shorthands:

$$\begin{array}{rl}
x \overset{\leq}{\hookrightarrow} y & \text{for } x \hookrightarrow y \wedge \mathsf{val}(x) \leq \mathsf{val}(y), \text{ and } x \overset{\geq}{\hookrightarrow} y \text{ accordingly,} \\
x \mapsto (y, v) & \text{for } x \mapsto y \wedge \mathsf{val}(x) = v \\
x \hookrightarrow^* y & \text{for } x = y \vee \Big(\top * \big(\ (x \hookrightarrow -) \wedge (- \hookrightarrow y) \wedge \neg(- \hookrightarrow x) \wedge \neg(y \hookrightarrow -) \\
 & \qquad\qquad\qquad \wedge \forall z \notin \{x, y\}.\ \big((z \hookrightarrow -) \Leftrightarrow (- \hookrightarrow z)\big)\big)\Big) \\
x \hookrightarrow^+ y & \text{for } \exists z. x \hookrightarrow z \wedge z \hookrightarrow^* y, \\
\mathsf{decls}(x, y) & \text{for } \quad \mathsf{precisely}(x = y) \quad \vee \quad x \mapsto y \\
 & \qquad \vee\ \mathsf{precisely}\big(\exists y'.\ x \hookrightarrow^+ y' \ \wedge\ y' \hookrightarrow y \wedge \forall z.(z \hookrightarrow^+ y') \Rightarrow (z \overset{\geq}{\hookrightarrow} -)\big)
\end{array}$$

We christen the $\overset{\leq}{\hookrightarrow}$ predicate *short-distance comparison*, and by contrast refer to $\mathsf{val}(x) \leq \mathsf{val}(y)$ as *long-distance comparison*. The binary predicate $x \hookrightarrow^* y$ is the accessibility relation (see [6]); it asserts that $(\mathsf{fst} \circ h)^n(s(x)) = s(y)$ for some $n \geq 0$.

The binary predicate $\mathsf{decls}(x, y)$ characterises a heap composed of a single list segment with data sorted in the decreasing order.

2.2 A Monadic Second Order Logic over Memory Shapes

Memory shapes. We define memory shapes as the abstraction of a memory heap forgetting the whole data component of all cells, while retaining the graphical aspect. A *memory shape* is hence a pair composed of a *store* and a *heap shape*, $(\mathfrak{s}, \mathfrak{h})$ such that

– $\mathfrak{s}$ is a variable valuation of the form $\mathfrak{s} : \mathtt{Var} \to \mathtt{Loc}$,
– $\mathfrak{h}$ is a partial function $\mathfrak{h} : \mathtt{Loc} \rightharpoonup \mathtt{Loc}$ with finite domain.

We will use the typographic convention to differentiate a memory state (s, h) from a memory shape $(\mathfrak{s}, \mathfrak{h})$. Note that concrete stores can be safely identified to abstract stores. We will write $\mathsf{shape}(.)$ for the obvious map from concrete heaps to heap shapes:

$$\mathsf{shape}(h) \ \overset{\text{def}}{\equiv} \ \begin{array}{l} \mathsf{Loc} \rightharpoonup \mathsf{Loc} \\ \quad l \ \ \mapsto \mathsf{fst}(h(l)) \end{array} \quad \text{with} \ \ \mathtt{dom}(\mathsf{shape}(h)) = \mathtt{dom}(h)$$

MSO over memory shapes. We assume a set $\mathtt{VAR}$ of monadic second-order variables, denoted by $\mathtt{P}, \mathtt{Q}, \mathtt{R}, \ldots$. An *environment* $\mathcal{E}$ is a map $\mathcal{E} : \mathtt{VAR} \to \mathcal{P}_{fin}(\mathtt{Loc})$ that associates to every second order variable a finite set of locations. Since we require finiteness of models, the version of monadic second-order logic we shall consider is usually called *weak*.

Formulae of (weak) monadic second-order logic ($\mathtt{MSO}$) are defined by the grammar below:

$$\phi := \neg\phi \,|\, \phi \wedge \phi \,|\, \exists x.\phi \,|\, x \hookrightarrow y \,|\, x = y \,|\, \exists \mathtt{P}.\phi \,|\, \mathtt{P}(x)$$

and are interpreted with the expected semantics:

$$\begin{array}{lll}
(\mathfrak{s},\mathfrak{h}),\mathcal{E} \models_{\mathtt{MSO}} \neg\phi & \text{iff not } (\mathfrak{s},\mathfrak{h}),\mathcal{E} \models_{\mathtt{MSO}} \phi \\
(\mathfrak{s},\mathfrak{h}),\mathcal{E} \models_{\mathtt{MSO}} \phi \wedge \psi & \text{iff } (\mathfrak{s},\mathfrak{h}),\mathcal{E} \models_{\mathtt{MSO}} \phi \text{ and } (\mathfrak{s},\mathfrak{h}),\mathcal{E} \models_{\mathtt{MSO}} \psi \\
(\mathfrak{s},\mathfrak{h}),\mathcal{E} \models_{\mathtt{MSO}} \exists x.\ \phi & \text{iff there is } l \in \mathtt{Loc} \text{ such that } (\mathfrak{s}[x \mapsto l],\mathfrak{h}),\mathcal{E} \models_{\mathtt{MSO}} \phi \\
(\mathfrak{s},\mathfrak{h}),\mathcal{E} \models_{\mathtt{MSO}} x \hookrightarrow y & \text{iff } \mathfrak{h}(\mathfrak{s}(x)) = \mathfrak{s}(y) \\
(\mathfrak{s},\mathfrak{h}),\mathcal{E} \models_{\mathtt{MSO}} x = y & \text{iff } \mathfrak{s}(x) = \mathfrak{s}(y) \\
(\mathfrak{s},\mathfrak{h}),\mathcal{E} \models_{\mathtt{MSO}} \exists \mathtt{P}.\ \phi & \text{iff there is a finite subset } \mathcal{P} \text{ of } \mathtt{Loc}, \\
 & \quad\text{such that } (\mathfrak{s},\mathfrak{h}),\mathcal{E}[\mathtt{P} \mapsto \mathcal{P}] \models_{\mathtt{MSO}} \phi \\
(\mathfrak{s},\mathfrak{h}),\mathcal{E} \models_{\mathtt{MSO}} \mathtt{P}(x) & \text{iff } \mathfrak{s}(x) \in \mathcal{E}(\mathtt{P})
\end{array}$$

As usual, we will write $\mathtt{P} \subseteq \mathtt{Q}$ for $\forall x.\mathtt{P}(x) \Rightarrow \mathtt{Q}(x)$, $\mathtt{P} \subsetneq \mathtt{Q}$ for $\mathtt{P} \subseteq \mathtt{Q} \wedge \exists x.\mathtt{P}(x) \wedge \neg\mathtt{Q}(x)$, and all set operators $\mathtt{P} \cap \mathtt{Q}, \mathtt{P} \cup \mathtt{Q}$, etc.

The following result is an almost straightforward consequence of the decidability of monadic second-order logic over structures with one function symbol [13] (see also [6] for details):

Theorem 1. *The satisfiability of MSO formulae interpreted over memory shapes is a decidable problem.*

3 Motivations

The merge function, that builds an ordered list from two ordered lists, will be our running example in this section. We consider the following C-like code :

```
struct cell {
  int val;
  struct cell *next;
};

function merge(cell *x, cell *y){
  cell *z, *head;
  if (x==NULL) return y;
  if (y==NULL) return x;
  if ((x->val) >= (y->val)){
    head = x; x=x->next;
  else {
    head = y; y=y->next;
  }
  z= head;
```

```
while((x!=NULL)&&(y!=NULL)) {
  /* MAIN LOOP P */
  if ((x->val) >= (y->val)) {
    z->next = x;
    x = x->next;
  } else {
    z->next = y;
    y = y->next;
  }
  z = z->next;
  /* END OF LOOP P*/
}
[...]
}
```

Let P denote the instruction block of the while loop. In order to prove the merge program, one usually needs at some point to provide a loop invariant A. This invariant may be automatically found, using some acceleration techniques, or might be provided by the user. In both cases, proving that the invariant is preserved is equivalent to showing that

$$\mathbf{Post}(P, A \wedge \mathbf{x} \neq \mathsf{null} \wedge \mathbf{y} \neq \mathsf{null}) \;\Rightarrow\; A \qquad (1)$$

is a valid formula, where **Post** denotes the strongest postcondition. There are several ways to compute the strongest postcondition of a loop-free sequence of instructions. We sketch here two approaches: the original one in separation logic theory [14], and the one followed by the tool SMALLFOOT [1].

The original approach does not make any assumption on the invariant A, and fully exploits the magic wand connective. To give an idea, in our case, the post-condition of the loop P of the merge program would look like

$$\begin{aligned}
&\mathbf{Post}(P,B) \;=\; \exists x',y',z'.\ z'\hookrightarrow\mathbf{z}\\
&\wedge\mathsf{val}(x')\geq\mathsf{val}(y')\Rightarrow\Big(\ \ x'\hookrightarrow\mathbf{x}\ \wedge\ y'=\mathbf{y}\\
&\qquad\qquad\wedge\Big(\exists v.\ \ z'\mapsto(x',v)*\big(z'\mapsto(-,v)\multimap B\{{}^{x',z'}/_{\mathbf{x},\mathbf{z}}\}\big)\Big)\Big)\\
&\wedge\mathsf{val}(x')<\mathsf{val}(y')\Rightarrow\Big(\ \ y'\hookrightarrow\mathbf{y}\ \wedge\ x'=\mathbf{x}\\
&\qquad\qquad\wedge\Big(\exists v.\ \ z'\mapsto(y',v)*\big(z'\mapsto(-,v)\multimap B\{{}^{x',z'}/_{\mathbf{x},\mathbf{z}}\}\big)\Big)\Big).
\end{aligned}$$

where primed variables quantify over the value of the corresponding program variable before the execution of the loop. What should be underlined concerning this approach is that automatically checking (1) would involve to solve the satisfiability of the logic in presence of magic wand, which is known to be undecidable even with only one selector [6].

In the SMALLFOOT *approach,* on the contrary, the symbolic computation is not parametric in the invariant. Usually, symbolic memory states are represented by (disjunctions of) formulae of the form $\exists x_1,\ldots,x_n.\Pi\wedge\Sigma$, where Σ (the "spatial" part) is a $*$-conjunct of the elementary list segments present in memory, and Π (the "pure" part) handles all other informations that are not properly adressed by local reasoning. For instance, a reasonable loop invariant following this format could be:

$$A \;\overset{\text{def}}{\equiv}\; \exists z_1,z_1' \begin{pmatrix} (\ z_1=\mathbf{x}\ \vee\ z_1=\mathbf{y}\ )\\ \wedge\ z_1'\overset{\geq}{\hookrightarrow}\mathbf{z}\wedge\mathsf{val}(\mathbf{z})\geq\mathsf{val}(\mathbf{x})\wedge\mathsf{val}(\mathbf{z})\geq\mathsf{val}(\mathbf{y})\end{pmatrix} \\ \wedge\ \mathsf{decls}(\mathtt{head},\mathbf{z})*\mathbf{z}\mapsto z_1*\mathsf{decls}(\mathbf{x},\mathsf{null})*\mathsf{decls}(\mathbf{y},\mathsf{null}) \tag{2}$$

Symbolic computation over lists with values has not been defined in SMALLFOOT, but taking inspiration from it, we may consider that for such an invariant the result of the symbolic computation would look like:

$$\begin{aligned}
&\mathbf{Post}(P,A\wedge\mathbf{x}\neq\mathsf{null}\wedge\mathbf{y}\neq\mathsf{null})\ \overset{\text{def}}{\equiv}\ \exists z_1',z_2,z_3,z_4,x',y',z'.\\
&\begin{pmatrix}((z_2,z_3,z_4,y')=(x',\mathbf{x},\mathbf{y},\mathbf{y})\vee(z_2,z_3,z_4,x')=(y',\mathbf{y},\mathbf{x},\mathbf{x}))\\ \wedge\ \mathbf{z}=z_2\ \wedge\ z_1'\overset{\geq}{\hookrightarrow}z'\ \wedge\ z_2\overset{\geq}{\hookrightarrow}z_3\\ \wedge\ \mathsf{val}(z')\geq\mathsf{val}(x')\wedge\mathsf{val}(z')\geq\mathsf{val}(y')\wedge\mathsf{val}(z_2)\geq\mathsf{val}(z_4)\end{pmatrix}\\
&\wedge\ \ \mathsf{decls}(\mathtt{head},z')*z'\mapsto z_2*z_2\mapsto z_3*\mathsf{decls}(\mathbf{x},\mathsf{null})*\mathsf{decls}(\mathbf{y},\mathsf{null})
\end{aligned}\tag{3}$$

where again primed variables should be thought as the past values of the corresponding program variables. We may underline that, unlike for (2), there are long-distance comparisons in the pure part of (3) that involve two quantified variables. As we will see in Section 5, this formula belongs to a fragment for which we obtain an undecidability result. However, looking more carefully to it, one may notice that, out of z' and z'_1, all quantified variables are aliased to program variables, which allows to rewrite the formula so that each long-distance comparison involves at least one open variable. Up to that, one may then use our decidability result of Section 5 to automatically check (1).

4 Decidability of Short-Distance Comparisons

In this section, we establish the decidability of the short-distance fragment of $\mathtt{SL}_<$. This fragment is defined by the following grammar:

$$\phi ::= \neg\phi \mid \phi \wedge \phi \mid \exists x.\phi \mid x \hookrightarrow y \mid x \overset{\leq}{\hookrightarrow} y \mid x \overset{\geq}{\hookrightarrow} y \mid x = y \mid \phi * \phi$$
(short-distance fragment)

The decidability of satisfiability for this fragment is obtained by reduction to the satisfiability of MSO over shapes.

Colored shapes. We hence have to abstract the values taking care of their local comparisons. To do so, we use a colored shape, with three colors on the edges[3]: '<', '>', and '='. In logical terms, these colors will be defined by two second order variables, noted X and Y, and we will observe the color '=' if both X and Y holds for the source location of the edge, '<' if X holds but not Y, and '>' if Y holds but not X. The case where neither X nor Y holds is irrelevant since we assumed a total order on data values, so we should constrain the possible choices for X and Y to avoid this situation. Moreover, some extra constraints will be involved by the necessity to manipulate only colored shapes for which it is possible to assign data respecting the colors (for instance, a cycle of '<' cannot be assigned data).

The graph of constraints. Given a shape $(\mathfrak{s}, \mathfrak{h})$, and the interpretations $\mathcal{X}, \mathcal{Y} \subseteq \mathtt{dom}(\mathfrak{h})$ of the second-order variables mentioned before, we define the associated graph of constraints $G = (V, E)$ where:

- V is $\mathtt{dom}(\mathfrak{h})$ quotiented by the equivalence $l \sim l'$ relating locations connected by a non oriented, '='-labeled path in the colored shape. Note that each $\sim$-equivalence class contains at most one location l whose image under $\mathfrak{h}$ lies outside the equivalence class of l. In such a situation, $[l]$ denotes this equivalence class.
- E is the set of pairs of equivalence classes $([l], [l'])$ such that
  - either $h(l) = l'$ and the color on l is '>'
  - or $h(l') = l$ and the color on l' is '<'

[3] Formally, on vertices, but each edge can be non-ambiguously identified to its source vertex in a shape.

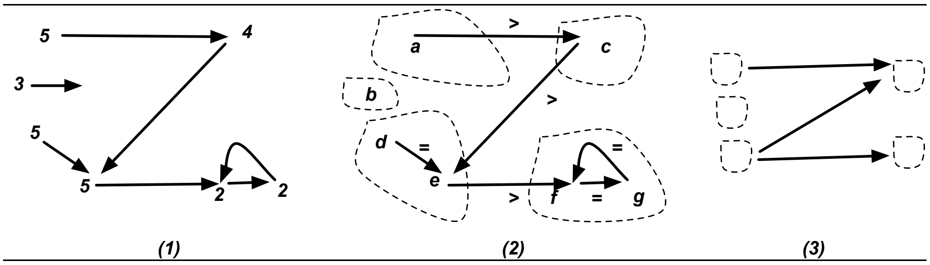


Fig. 1. A concrete heap (1), its colored abstraction (2), and the associated graph of constraints (3). Here $\mathcal{X} = \{c, d, f, g\}$ and $\mathcal{Y} = \{a, d, e, f, g\}$.

Figure 1 gives an example of a colored shape and its associated graph of constraints. Note that an edge towards a dangling pointer cannot be colored, and this is in fact the unique situation in which one allows $\neg X \wedge \neg Y$. The graph of constraints helps us to decide whether or not it is possible to assign values to a colored shape: indeed, this problem is equivalent to defining a topological order on the graph of constraints, which is known to be equivalent to this graph being acyclic. What remains to be explained now is: (1) how to define the graph of constraints in MSO, (2) how to express acyclicity, (3) how to treat separating conjunction.

The reduction. The reduction from $\mathtt{SL}_<$ over memory states to $\mathtt{MSO}$ over shapes is defined by $\mathsf{rd}_{\mathtt{SL}_< \to \mathtt{MSO}}(\phi) = \exists X.\exists Y.\exists Z.\mathrm{Cons}(X, Y, Z) \wedge \mathsf{rd}(\phi, X, Y, Z)$ where:

- Z is an extra second-order variable that is needed to define the current focus, that is the subheap of the original heap on which the (sub)formula is currently evaluated.
- rd is an auxiliary reduction that works assuming that X, Y and Z have been correctly guessed, updating these parameters appropriately when $*$ is translated.
- Cons are constraints imposed on X, Y and Z to guarantee that the first guess is a valid one: $\mathcal{Z}$ is the domain of the heap, and $\mathcal{X}$ and $\mathcal{Y}$ define a colored shape to which one may assign values.

Constraints. We impose three contraints : $\mathrm{Cons}(X, Y, Z) \stackrel{\text{def}}{\equiv} \mathrm{Cons1}(X, Y, Z) \wedge \mathrm{Cons2}(X, Y, Z) \wedge \mathrm{Cons3}(X, Y, Z)$

1. the only admitted color on a monochromatic cycle is '=' (this is indeed equivalent to the acyclicity condition on the graph of constraints):

$$\mathrm{Cons1}(X, Y, Z) \stackrel{\text{def}}{\equiv} \forall U \subseteq Z.\ \mathrm{Loop}(U) \Rightarrow (U \subseteq X \Leftrightarrow U \subseteq Y)$$

   where $\mathrm{Loop}(U)$ is defined as $\mathrm{SetOfLoops}(U) \wedge \forall V \subsetneq U. \neg\mathrm{SetOfLoops}(V)$ and $\mathrm{SetOfLoops}(U)$ is $\forall x.U(x) \Rightarrow \exists y.U(y) \wedge y \hookrightarrow x$
2. every edge that should be colored is colored with '<, '>' or '='

$$\mathrm{Cons2}(X, Y, Z) \stackrel{\text{def}}{\equiv} \forall x.\quad (Z(x) \wedge (\exists y.Z(y) \wedge x \hookrightarrow y)) \Leftrightarrow (X(x) \vee Y(x))$$

3. $\mathcal{Z}$ is the domain of the heap: $\mathrm{Cons3}(X, Y, Z) \stackrel{\text{def}}{\equiv} \forall x.(x \hookrightarrow -) \Leftrightarrow Z(x)$.

Let us now state the results we may derive from these definitions. We say that a location l is an increasing (resp. decreasing) node if there are $l', l'' \in \texttt{Loc}$ and $\alpha, \beta \in \texttt{Dat}$ such that $h(l) = (l', \alpha)$, $h(l') = (l'', \beta)$, and $\alpha \leq \beta$ (resp. $\alpha \geq \beta$). We write $\texttt{dom}^+(h)$ (resp. $\texttt{dom}^-(h)$) to denote the set of increasing (resp. decreasing) nodes of h, and $\mathcal{E}_h$ denotes $[X \mapsto \texttt{dom}^+(h),\ Y \mapsto \texttt{dom}^-(h),\ Z \mapsto \texttt{dom}(h)]$.

Lemma 1 (Constraints soundness). *If* $(\mathfrak{s}, \mathfrak{h}), \mathcal{E} \models_{\texttt{MSO}} \mathrm{Cons}(X, Y, Z)$ *then there is a* $h : \texttt{Loc} \rightharpoonup \texttt{Loc} \times \texttt{Dat}$ *such that* $\mathsf{shape}(h) = \mathfrak{h}$, $\mathcal{E}(Z) = \texttt{dom}(h)$, $\mathcal{E}(X) = \texttt{dom}^+(h)$ *and* $\mathcal{E}(Y) = \texttt{dom}^-(h)$.

Lemma 2 (Constraints completeness). *For all models with data* (s, h)*:*

$$(s, \mathsf{shape}(h)), \mathcal{E}_h \models_{\texttt{MSO}} \mathrm{Cons}(X, Y, Z).$$

Auxiliary recursive translation. The auxiliary recursive translation rd is defined as follows: (1) it is isomorphic on the cases of $\phi \wedge \psi$, $\neg\phi$, $\exists x.\phi$, and $x = y$, and (2) for other connectives, parameters X, Y, Z come into play:

$$\begin{aligned}
&\mathsf{rd}(x \hookrightarrow y, X, Y, Z) \overset{\text{def}}{\equiv} Z(x) \wedge x \hookrightarrow y\\
&\mathsf{rd}(x \overset{\leq}{\hookrightarrow} y, X, Y, Z) \overset{\text{def}}{\equiv} Z(x) \wedge Z(y) \wedge X(x) \wedge x \hookrightarrow y\\
&\mathsf{rd}(x \overset{\geq}{\hookrightarrow} y, X, Y, Z) \overset{\text{def}}{\equiv} Z(x) \wedge Z(y) \wedge Y(x) \wedge x \hookrightarrow y\\
&\mathsf{rd}(\phi_1 * \phi_2, X, Y, Z) \overset{\text{def}}{\equiv} \exists Z_1, Z_2.\\
&\qquad \mathsf{rd}(\phi_1, X, Y, Z_1) \ \wedge\ \mathsf{rd}(\phi_2, X, Y, Z_2) \ \wedge\ Z = Z_1 \cup Z_2 \ \wedge\ Z_1 \cap Z_2 = \emptyset
\end{aligned}$$

Lemma 3 (Reduction Lemma). *For all* s, h, *for all* $\mathcal{Z} \subseteq \texttt{dom}(h)$,

$$(s, \mathsf{shape}(h)), \mathcal{E}_h[Z \mapsto \mathcal{Z}] \models_{\texttt{MSO}} \mathsf{rd}(\phi, X, Y, Z) \text{ if and only if } (s, h_{|\mathcal{Z}}) \models_{\texttt{SL}} \phi.$$

Theorem 2. *For all formulae* ϕ *of* $\texttt{SL}_<$, *there exists* (s, h) *such that* $(s, h) \models_{\texttt{SL}} \phi$ *if and only if there exists* $(\mathfrak{s}, \mathfrak{h})$ *such that* $(\mathfrak{s}, \mathfrak{h}) \models_{\texttt{MSO}} \mathsf{rd}_{\texttt{SL}_< \to \texttt{MSO}}(\phi)$

Thanks to Theorem 2 and Theorem 1, we have established the announced result:

Corollary 1. *The satisfiability problem for the fragment of* $\texttt{SL}_<$ *with short-distance comparisons is decidable.*

5 Long-Distance Comparisons

5.1 An Undecidablity Result

We consider now the fragment of $\texttt{SL}_<$ where magic wand is still dropped, but long-distance comparison is considered:

$$\phi ::= \neg\phi \mid \phi \wedge \phi \mid \exists x.\phi \mid \exists v.\phi \mid x \hookrightarrow y \mid \mathsf{val}(x) \leq v \mid \mathsf{val}(x) \geq v \mid x = y \mid \phi * \phi.$$
(long-distance fragment)

We show that, without any further restriction, long-distance comparisons yield undecidability, even for a simpler fragment:

$$\phi ::= \neg\phi \mid \phi \wedge \phi \mid \exists x.\phi \mid x \hookrightarrow y \mid \mathsf{val}(x) = \mathsf{val}(y) \mid x = y \mid \phi * \phi.$$
(equality long-distance fragment)

Theorem 3. *The satisfiability problem for the equality long-distance fragment is undecidable.*

The proof goes by reduction to the satisfiability problem of first-order formulae over data words. Before giving the intuition of the reduction, we first recall this logic.

First-order logic over data words. We assume a finite set Σ. A finite *data word* is a sequence $w = w_1..w_n$, where $w_i = (a_i, \alpha_i) \in \Sigma \times \mathtt{Dat}$; we write $|\, w\,|$ to denote the length $n \in \mathbb{N}$ of w. Note that, so far, we assumed a total order on Dat, but this aspect is not essential for this reduction, and one may think of Dat as any arbitrary infinite set. The first-order formulae we will evaluate over these models are defined by the following grammar:

$$\text{(FO over data words)} \quad \phi ::= \neg\phi \,|\, \phi \wedge \phi \,|\, \exists x.\phi \,|\, a(x) \,|\, x = y + 1 \,|\, x \sim_{\mathtt{Dat}} y$$

where $a \in \Sigma$. Variables are interpreted as positions in the word through a valuation $\sigma : \mathtt{Var} \rightarrow \{1..|\, w\,|\}$, $+1$ is the standard addition over $\mathbb{N}$, and $\sim_{\mathtt{Dat}}$ relates positions holding the same datum. More formally

$$\begin{array}{ll} w, \sigma & \models_{FO} \exists x.\phi \text{ if there is } n \in \{1..|\, w\,|\} \text{ s.t. } w, \sigma[x \mapsto n] \models_{FO} \phi \\ w, \sigma & \models_{FO} a(x) \text{ if } a_{\sigma(x)} = a \\ w, \sigma & \models_{FO} x = y + 1 \text{ if } \sigma(x) = \sigma(y) + 1 \\ w, \sigma & \models_{FO} \ x \sim_{\mathtt{Dat}} y \text{ if } \alpha_{\sigma(x)} = \alpha_{\sigma(y)} \end{array}$$

Theorem 4 (see [2], Prop. 27). *The satisfiability problem for a closed sentence of FO over data words is undecidable.*

The reduction. To prove Theorem 3, we define a translation from FO to the long-distance fragment such that a formula ϕ admits a data word model if and only if its translation admits a memory state model. A data word of length n is encoded as a list segment of length $2n$, placing the sequence of letters of Σ in the even positions, and the data sequence in odd positions. Then $x = y + 1$ can be encoded by $y \hookrightarrow^2 x$, and $x \sim_{\mathtt{Dat}} y$ can be encoded by $\mathsf{val}(x) = \mathsf{val}(y)$.

5.2 Decidability of Guarded Long-Distance Comparisons

We now consider the fragment of formulae where every quantification over values is restricted to values stored in the cells that are pointed by the program variables:

$$\begin{array}{l} \phi ::= \neg\phi \,|\, \phi \wedge \phi \,|\, \exists x.\phi \,|\, \exists v.\mathsf{val}(\mathbf{x}) = v \wedge \phi \\ \qquad |\, x \overset{\leq}{\hookrightarrow} y \,|\, x \overset{\geq}{\hookrightarrow} y \,|\, x \hookrightarrow y \,|\, \mathsf{val}(x) \leq v \,|\, \mathsf{val}(x) \geq v \,|\, x = y \,|\, \phi * \phi. \\ \text{(guarded long-distance fragment)} \end{array}$$

Note that guarded long-distance comparisons are quite weak, and we need to add short-distance comparisons as basic predicates if we still want to use them.

Theorem 5. *The satisfiability problem for the guarded long-distance fragment is decidable.*

Proof of Theorem 5. We only sketch the proof. We adapt the proof of Theorem 2 by extending the notions of colored shapes and graphs of constraints. Let $\texttt{ProgVar} = \{\mathbf{x}_1, .., \mathbf{x}_n\} \subsetneq \texttt{Var}$ be a finite set of variables such that every formula to be translated will have all its open variables in $\texttt{ProgVar}$. To every variable $\mathbf{x} \in \texttt{ProgVar}$, we associate two second-order variable $X_{\mathbf{x}}, Y_{\mathbf{x}}$. A colored shape is then a tuple

$$CS \ = \ \Big(\ (\mathfrak{s}, \mathfrak{h}) \ , \ \mathcal{X}, \mathcal{Y}, \ \mathcal{X}_{\mathbf{x}_1}, \mathcal{Y}_{\mathbf{x}_1}, \ldots, \mathcal{X}_{\mathbf{x}_n}, \mathcal{Y}_{\mathbf{x}_n} \ \Big)$$

where $\mathcal{X}_{\mathbf{x}}, \mathcal{Y}_{\mathbf{x}}$ are finite sets of locations; it is *well defined* if $\mathcal{X} \cup \mathcal{Y} = \texttt{dom}(h) \cap h^{-1}(\texttt{dom}(h))$ and $\mathcal{X}_{\mathbf{x}} \cup \mathcal{Y}_{\mathbf{x}} = \texttt{dom}(h)$ for every program variable $\mathbf{x}$ such that $\mathfrak{s}(\mathbf{x}) \in \texttt{dom}(h)$. Let $(\mathfrak{s}, \mathfrak{h})$ be a fixed shape. We define the relation $\sim$ on $\texttt{dom}(\mathfrak{h})$ as the smallest equivalence relation such that:

- if $l \in \mathcal{X}_{\mathbf{x}} \cap \mathcal{Y}_{\mathbf{x}}$ and $\mathfrak{s}(\mathbf{x}) \in \texttt{dom}(\mathfrak{h})$, then $\mathfrak{s}(\mathbf{x}) \sim l$;
- if $\mathfrak{h}(l) = l'$, and $l \in \mathcal{X} \cap \mathcal{Y}$, then $l \sim l'$.

The graph of constraints associated to CS is the pair (V, E) where the vertex set V is the quotient of $\texttt{dom}(\mathfrak{h})$ by $\sim$, and there is an edge from the equivalence class c_1 to c_2 if at least one of the following conditions holds:

- either there is $\mathfrak{s}(\mathbf{x}) \in c_1$ and $l \in c_2$ such that $l \in \mathcal{Y}_{\mathbf{x}} - \mathcal{X}_{\mathbf{x}}$;
- or there is $\mathfrak{s}(\mathbf{x}) \in c_2$ and $l \in c_1$ such that $l \in \mathcal{X}_{\mathbf{x}} - \mathcal{Y}_{\mathbf{x}}$;
- or there is $l \in c_1, l' \in c_2$ such that $\mathfrak{h}(l) = l'$ and $l \in \mathcal{Y} - \mathcal{X}$;
- or there is $l \in c_1, l' \in c_2$ such that $\mathfrak{h}(l') = l$ and $l' \in \mathcal{X} - \mathcal{Y}$.

It is possible to check that the graph of constraints and the acyclicity condition on it are MSO definable. We may then adapt the reduction of Section 4: we guess the $\mathcal{X}_{\mathbf{x}}$s and $\mathcal{Y}_{\mathbf{x}}$s at start and check we made a valid guess, and we extend the recursive translation $\mathsf{rd}(\phi)$ with the following cases:

$$\begin{aligned}
&\mathsf{rd}(\exists v.\mathsf{val}(\mathbf{x}) = v \wedge \phi) \stackrel{\text{def}}{\equiv} \mathsf{rd}(\phi\{^{\mathsf{val}(\mathbf{x})}/_{v}\}) \\
&\mathsf{rd}(\mathsf{val}(x) \leq \mathsf{val}(\mathbf{x})) \quad \stackrel{\text{def}}{\equiv} Z(x) \wedge Z(\mathbf{x}) \wedge Y_{\mathbf{x}}(x) \wedge \neg X_{\mathbf{x}}(x) \\
&\mathsf{rd}(\mathsf{val}(x) \geq \mathsf{val}(\mathbf{x})) \quad \stackrel{\text{def}}{\equiv} Z(x) \wedge Z(\mathbf{x}) \wedge X_{\mathbf{x}}(x) \wedge \neg Y_{\mathbf{x}}(x)
\end{aligned}$$

Perspectives. We expect this decidability result to extend to more complex data structures that would have a decidable MSO theory (trees, doubly-linked lists, lists of lists, and more generally tree-width bounded structures), and to more complex short-distance comparisons (n-th successor, brothers,...). Moreover, such restrictions may be sufficient to handle other interesting applications, for instance search-trees. In this sense, we claim that the graph of constraints is the "right" general concept for logics dealing with sorted data structures.

6 Magic Wand and Restricted Magic Wand

Even without data, the logic with the operator $-\!\!*$ is proved to be undecidable in [6]. In the technical report [5] corresponding to the paper, a decidable separation logic with a restricted magic wand is presented. Let us write the definition of this binary operator, $-\!\!*_n$ (for n an integer). Unlike the plain operator $-\!\!*$, the quantification on disjoint heaps of $-\!\!*_n$ considers only heaps for which the cardinality of the domain is bounded by n. More formally, we define that $(\mathfrak{s}, \mathfrak{h}) \models \phi_1 -\!\!*_n \phi_2$ if and only if for all $\mathfrak{h}'$ such that $\mathfrak{h}' \perp \mathfrak{h}$ and $|\mathtt{dom}(\mathfrak{h}')| \leq n$, if $(\mathfrak{s}, \mathfrak{h}') \models \phi_1$ then $(\mathfrak{s}, \mathfrak{h} * \mathfrak{h}') \models \phi_2$. It can be seen as an abbreviation of $(\phi_1 \wedge \neg \exists x_1, \ldots, x_{n+1} . \bigwedge_{i \neq j} x_i \neq x_j \wedge \bigwedge_i \exists y . x_i \hookrightarrow y) -\!\!* \phi_2$. In the sequel, we will prove that, in the context of heaps with data, $-\!\!*_1$ is sufficient to obtain undecidability.

Let R denote an arbitrary binary relation on $\mathtt{Dat}$. Let us call $\sim_R$ the equivalence relation defined as $\alpha \sim_R \alpha'$ iff $\{\beta, \beta RR\alpha\} = \{\beta, \beta RR\alpha'\}$. We consider the atomic formula $\mathsf{val}(x) R \mathsf{val}(y)$ stating that values stored in x and y compare through R. Formally, $(s, h) \models_{\mathtt{SL}} \mathsf{val}(x) R \mathsf{val}(y)$ iff there are $\alpha, \beta \in \mathtt{Val}$ and $l, l' \in \mathtt{Loc}$ such that $h(s(x)) = (l, \alpha)$, $h(s(y)) = (l', \beta)$, and $\alpha R \beta$.

We now introduce the relation $x \overset{R}{\hookrightarrow} y$ for $x \hookrightarrow y \wedge \mathsf{val}(x) R \mathsf{val}(y)$, and define the logic $\mathtt{SL}(R, -\!\!*_1)$ with the grammar:

$$\phi ::= \neg \phi \mid \phi \wedge \phi \mid \exists x . \phi \mid x \hookrightarrow y \mid x \overset{R}{\hookrightarrow} y \mid x = y \mid \phi * \phi \mid \phi -\!\!*_1 \phi .$$

We are going to prove that satisfiability and validity problems are undecidable for $\mathtt{SL}(R, -\!\!*_1)$, for any $R \in \{\leq, \geq, =, <, >\}$. We will rely on the previous section, especially Theorem 3, by simulating a long-distance equality. We first need the following fact:

Lemma 4. *Let $R \in \{\leq, \geq, =, <, >\}$. Then $\sim_R$ has an infinite number of equivalence classes.*

Let $\sim$ be an equivalence relation on $\mathtt{Dat}$ with infinitely many equivalence classes. Let us define the following fragment:

$$\phi ::= \neg \phi \mid \phi \wedge \phi \mid \exists x . \phi \mid x \hookrightarrow y \mid \mathsf{val}(x) \sim \mathsf{val}(y) \mid x = y \mid \phi * \phi .$$
(equivalence long-distance fragment)

Then the following lemma, a slight variation of Theorem 3, also holds in this generalised framework:

Lemma 5. *The satisfiability problem for the equivalence long-distance fragment is undecidable.*

Proof. By the same encoding as the one of Theorem 3, one may reduce a satisfiability problem of an FO sentence over data words, where data taken from the infinite quotient set $\mathtt{Dat}/\sim_{\mathtt{R}}$, to the satisfiability problem for the equivalence long-distance fragment. □

Lemma 6. *There is a formula $\phi_R(x, x') \in \mathtt{SL}(R, -\!\!*_1)$ such that for all (s, h) with $\{s(x), s(x')\} \subseteq \mathtt{dom}(h)$:*

$$(s, h) \models \phi_R(x, x') \textit{ iff } (s, h) \models \mathsf{val}(x) \sim_R \mathsf{val}(x')$$

We only sketch the proof. $\phi \mathrel{-\!\!\circledast}_1 \psi$ will abbreviate $\neg(\phi \mathrel{-\!\!*}_1 \neg\psi)$. Then $(s,h) \models \phi \mathrel{-\!\!\circledast}_1 \psi$ iff there is h' such that $(s,h') \models \phi$, $(s, h * h') \models \psi$ and $|\,\mathtt{dom}(\mathfrak{h}')\,| \leq 1$. The operators $\mathrel{-\!\!*}_1$ and $\mathrel{-\!\!\circledast}_1$ will be used to simulate restricted quantifications over $\mathtt{Dat}$, respectively universal and existential. Consider the formula ϕ

$$\begin{gathered}\exists x_1.\exists x_2.\big(\neg\exists x_3.x_1 \hookrightarrow x_3 \vee x_2 \hookrightarrow x_3\big)\\ \wedge(x_1 \hookrightarrow x_2) \mathrel{-\!\!*}_1 \big((\mathsf{val}(x_1)\ RR\ \mathsf{val}(x)) \quad \Leftrightarrow \quad (\mathsf{val}(x_1)\ RR\ \mathsf{val}(x'))\big)\end{gathered}$$

where $\mathsf{val}(x_1) RR \mathsf{val}(x)$ abbreviates $(x_2 \hookrightarrow x) \mathrel{-\!\!\circledast}_1 [(x_1 \overset{R}{\hookrightarrow} x_2) \wedge x_2 \overset{R}{\hookrightarrow} x]$. The formula ϕ expresses that for all α, there is β such that $\alpha R \beta R \mathsf{snd}(h(s(x)))$ if and only if there is β such that $\alpha R \beta R \mathsf{snd}(h(s(x')))$, that is $\mathsf{val}(x) \sim_R \mathsf{val}(x')$. As a consequence:

Theorem 6. *For any $R \in \{\leq, \geq, <, >, =\}$, the validity and satisfiability problems for $\mathtt{SL}(R, \mathrel{-\!\!*}_1)$ are undecidable.*

7 Conclusion

Our results give a wide picture of the decidability status of the satisfiability problem for separation logic dealing with data.

With the ability to describe lists and quantify over locations, allowing long-distance comparisons brings undecidability, and so does allowing the operator $\mathrel{-\!\!*}$, even strongly restricted. Yet, there is a very positive result: dropping these two features makes the satisfiability problem decidable, still being able to do local reasoning and express properties about ordered recursive structures. The decidability even holds when a finite set of references can be compared to all the rest of the memory.

Some ways to restrict the full language are still unexplored, for instance bounding the amount of quantified variables. With the same hope to obtain decidability for satisfiability problems, one may look at extension of our decidable fragment. For instance, our results are general for any totally ordered infinite set, and questions remain open about partially ordered sets.

References

1. Berdine, J., Calcagno, C., O'Hearn, P.: Smallfoot: Modular automatic assertion checking with separation logic. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2005. LNCS, vol. 4111, pp. 115–137. Springer, Heidelberg (2006)
2. Bojanczyk, M., Muscholl, A., Schwentick, T., Segoufin, L., David, C.: Two-variable logic on words with data. In: Proceedings of 21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, pp. 7–16 (2006)
3. Bouajjani, A., Bozga, M., Habermehl, P., Iosif, R., Moro, P., Vojnar, T.: Programs with lists are counter automata. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 517–531. Springer, Heidelberg (2006)

4. Bozga, M., Iosif, R., Lakhnech, Y.: On logics of aliasing. In: Giacobazzi, R. (ed.) SAS 2004. LNCS, vol. 3148, pp. 344–360. Springer, Heidelberg (2004)
5. Brochenin, R., Demri, S., Lozes, E.: On the almighty wand. Technical report, LSV, ENS de Cachan (2008)
6. Brochenin, R., Demri, S., Lozes, É.: On the almighty wand (To appear). In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 323–338. Springer, Heidelberg (2008)
7. Calcagno, C., Yang, H., O'Hearn, P.: Computability and complexity results for a spatial assertion language for data structures. In: Hariharan, R., Mukund, M., Vinay, V. (eds.) FSTTCS 2001. LNCS, vol. 2245, pp. 108–119. Springer, Heidelberg (2001)
8. Demri, S., Lazić, R., Nowak, D.: On the freeze quantifier in constraint LTL: Decidability and complexity. In: Proceedings of the 12th International Symposium on Temporal Representation and Reasoning (TIME 2005), Burlington, Vermont, USA, pp. 113–121. IEEE Computer Society Press, Los Alamitos (2005)
9. Jensen, J., Jorgensen, M., Klarlund, N., Schwartzbach, M.: Automatic verification of pointer programs using monadic second-order logic. In: PLDI 1997, pp. 226–236. ACM, New York (1997)
10. Lev-Ami, T., Sagiv, M.: TVLA: A system for implementing static analyses. In: Palsberg, J. (ed.) SAS 2000. LNCS, vol. 1824, pp. 280–302. Springer, Heidelberg (2000)
11. Loginov, A., Reps, T., Sagiv, M.: Refinement-based verification for possibly-cyclic lists. In: Reps, T., Sagiv, M., Bauer, J. (eds.) Wilhelm Festschrift. LNCS, vol. 4444, pp. 247–272. Springer, Heidelberg (2007)
12. Nguyen, H.H., David, C., Qin, S.C., Chin, W.-N.: Automated verification of shape and size properties via separation logic. In: Cook, B., Podelski, A. (eds.) VMCAI 2007. LNCS, vol. 4349, pp. 251–266. Springer, Heidelberg (2007)
13. Rabin, M.: Decidability of second-order theories and automata on infinite trees. Transactions of the American Mathematical Society 41, 1–35 (1969)
14. Reynolds, J.C.: Separation logic: a logic for shared mutable data structures. In: LICS 2002, pp. 55–74. IEEE, Los Alamitos (2002)
15. Trakhtenbrot, B.A.: The impossibility of an algorithm for the decision problem for finite models. Dokl. Akad. Nauk SSSR 70, 572–596 (1950); English translation in: AMS Transl. Ser. 2, 23(1063), 1–6
16. Yorsh, G., Rabinovich, A.M., Sagiv, M., Meyer, A., Bouajjani, A.: A logic of reachable patterns in linked data structures. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 94–110. Springer, Heidelberg (2005)

Interprocedural Dataflow Analysis over Weight Domains with Infinite Descending Chains*

Morten Kühnrich[2], Stefan Schwoon[1], Jiří Srba[2], and Stefan Kiefer[1]

[1] Technische Universität München
Boltzmannstr. 3, 85748 Garching, Germany
{kiefer,schwoon}@in.tum.de
[2] Department of Computer Science, Aalborg University
Selma Lagerlöfs Vej 300, 9220 Aalborg East, Denmark
{mokyhn,srba}@cs.aau.dk

Abstract. We study generalized fixed-point equations over idempotent semirings and provide an efficient algorithm for the detection whether a sequence of Kleene's iterations stabilizes after a finite number of steps. Previously known approaches considered only bounded semirings where there are no infinite descending chains. The main novelty of our work is that we deal with semirings without the boundedness restriction. Our study is motivated by several applications from interprocedural dataflow analysis. We demonstrate how the reachability problem for weighted pushdown automata can be reduced to solving equations in the framework mentioned above and we describe a few applications to demonstrate its usability.

1 Introduction

Weighted pushdown systems [20] are a suitable model for analyzing programs with procedures. They have been used successfully in a number of applications, e.g. BDD-based model checking [23,7], trust-management systems [10], path optimization [14], and interprocedural dataflow analysis (see [19] for a survey).

The main idea is that the transitions of a pushdown system are labelled with values from a given data domain (e.g. natural numbers). These values can be composed when executed in sequence (e.g. using the addition on natural numbers) and one is then interested in a number of verification questions like reachability of a given configuration with the combined value over all paths leading into this configuration (e.g. by taking the minimum value over all such paths). It has been shown that there are efficient polynomial time algorithms for answering these questions [20].

In this paper, we contribute to the research in this area. We first draw a connection between reachability in weighted pushdown systems (WPDS) over an

* The second and fourth authors are supported in part by the DFG project *Algorithms for Software Model Checking*. The third author is supported in part by Institute for Theoretical Computer Science, project No. 1M0545.

idempotent semiring and solving fixed-point equations over the same semiring. Unlike related work, we allow for infinite descending chains in our semirings (our approach e.g. includes the integer semiring). Due to this reason, the system of equations constructed from a WPDS may not have a solution. We therefore provide an efficient algorithm that either determines the solution or detects the presence of an infinite descending chain. In the latter case, we output some component (variable) of the system affected by the problem. So on one hand we treat domains with infinite descending chains but on the other hand, two restrictions are necessary to make this possible. However, as argued in Section 4, the framework still includes a number of interesting applications. A full version of the paper is available as [12].

1.1 Dataflow Analysis and Fixed-Point Equations

Static analysis gathers information about a program without executing it. Dataflow analysis is an instance of static analysis: it reasons about run-time values of variables or expressions. More to the point, we desire to establish facts that hold at some control point whenever an execution reaches it.

Most approaches to dataflow analysis reduce the problem (explicitly or implicitly) to solving a system of fixed-point equations over some algebraic structure, e.g. a lattice or a semiring. They map the control-flow graph of a program to an equation system $\boldsymbol{X} = \boldsymbol{f}(\boldsymbol{X})$, where the vector $\boldsymbol{X} = (X_1, \ldots, X_n)$ stands for the nodes in the control flow graph, and takes values from some dataflow domain. The vector $\boldsymbol{f} = (f_1, \ldots, f_n)$ stands for the edges in the graph, i.e., the *transfer function* $f_i(\boldsymbol{X})$ describes the effect of the program on X_i in terms of the other dataflow values. Under certain conditions (e.g., the functions f_i are distributive) the desired dataflow information is precisely the greatest solution of the system $\boldsymbol{X} = \boldsymbol{f}(\boldsymbol{X})$, i.e., the greatest fixed point $gfp(\boldsymbol{f})$ of $\boldsymbol{f}$ [18,22].

There is a large body of literature dealing with dataflow analysis along these lines. Of particular interest to us are interprocedural analyses. The seminal work of Sharir and Pnueli [22] shows how to set up an equation system that captures only the *interprocedurally valid* paths, i.e. those paths in which all return statements lead back to the site of the most recent call. However, [22] computes only one dataflow value for each program point, merging together all the paths that reach it, regardless of the calling context. In [20] a generalization was provided, where the solution of the equations computes a solution for each *configuration*, where configuration denotes a program point together with its calling context. Thus, [20] allows to distinguish dataflow values for different, arbitrary calling contexts. (The merged information can still be obtained as a special case.) The results of [20] were phrased in terms of weighted pushdown systems (WPDS), and we will adopt this notion in our paper.

If the dataflow domain satisfies the so-called *descending chain condition* (i.e. each infinite descending chain eventually becomes stationary), $gfp(\boldsymbol{f})$ can be obtained by *Kleene's iteration*: Let $\overline{0}$ be the greatest domain element, and $\overline{\boldsymbol{0}} = (\overline{0}, \ldots, \overline{0})$. Then Kleene's fixed-point theorem guarantees that the sequence

$\overline{\mathbf{0}}, \boldsymbol{f}(\overline{\mathbf{0}}), \boldsymbol{f}(\boldsymbol{f}(\overline{\mathbf{0}})), \ldots$ reaches $gfp(\boldsymbol{f})$ after finitely many steps. Both [22] and [20] require the descending chain condition.

However, the descending chain condition does not always hold. For example, the lattice of non-positive integers with $\sqcap = \min$ and $\sqcup = \max$ does not satisfy the condition because of the infinite descending chain $0, -1, -2, \ldots$. In fact, this chain arises when doing Kleene's iteration on the equation $X = f(X)$ where $f(X) = \min(X, X - 1)$. More to the point, Kleene's iteration on f would fail to terminate. We will show how to overcome this problem.

Previous work (e.g., [20]) has shown that many important analysis problems can be phrased as equation systems, where $\boldsymbol{f}(\boldsymbol{X})$ contains polynomials over *idempotent semirings*. By polynomial, we mean an expression that is built up from variables, constant elements, and the semiring operations '$\oplus$' (combine) and '$\otimes$' (extend).

Recently, fixed-point equations over idempotent semirings have been studied intensively. While the classical solution is to use Kleene's iteration or chaotic iteration, recent work has proposed faster algorithms and better convergence results based on Newton's method [9,5,4,6]. In these works, the boundedness condition is dropped, but replaced by another condition called ω-continuity, requiring that the infimum of every infinite set exists, thus ensuring that a greatest fixed point can always be found. Our work does not require this condition, and a greatest fixed point is not always guaranteed to exist (but our algorithm detects such a case and reports it). The penalty for this is that a different kind of restriction has to be introduced: we require that semirings are totally ordered and that "extend preserves inequality", i.e., $a \otimes c \neq b \otimes c$ for $a \neq b$ and $a, b, c \neq \overline{0}$.

Our algorithm executes Kleene's iteration, and if the iteration terminates, it outputs the greatest fixed point. If Kleene's iteration fails to terminate, our algorithm will detect this and still terminate, indicating a responsible variable (a so-called *witness component*).

The work closest to ours is the one by Gawlitza and Seidl [8], who consider systems of equations over the integer semiring. Our algorithm can be seen as a generalization of one of their algorithms to totally ordered semirings where extend preserves inequality and to equations over arbitrary polynomials. Moreover, we provide a direct and self-contained proof of the result. Another related work is by Leroux and Sutre [15]. They present an algorithm for computing least fixed-points for monotone *bounded-increasing* functions over integers. On one hand they consider more general functions like e.g. factorials, on the other hand the minimum and maximum functions are not bounded-increasing according to their definition. As a result, their algorithm is not applicable in our setting of weighted pushdown systems.

We proceed as follows: In Section 2, we provide a new algorithm for solving fixed-point equations. Using this result, we design an algorithm for interprocedural dataflow analysis in Section 3, which is based on WPDS [20] and still requires a polynomial number of semiring operations. Like previous work on WPDS, the algorithm allows to compute dataflow information for each configuration (if desired). Due to the properties of the systems we handle, our algorithm either returns a solution (if it exists) or reports that none exists (usually indicating an error in the program). We provide several applications of our theory in Section 4.

2 Fixed-Point Equations over Idempotent Semirings

In this section we shall study fixed-point equations over idempotent semirings and Kleene's iterations over vectors of polynomials.

Definition 1 (Idempotent Semiring). *An* idempotent semiring *is a 5-tuple* $\mathcal{S} = (D, \oplus, \otimes, \overline{0}, \overline{1})$ *where* D *is a set called the* domain, $\overline{0}, \overline{1} \in D$, *and the binary operators* combine *'$\oplus$' and* extend *'$\otimes$' on* D *satisfy:*

1. *$(D, \oplus)$ is a commutative monoid with $\overline{0}$ as its neutral element and $(D, \otimes)$ is a monoid with $\overline{1}$ as its neutral element,*
2. *extend distributes over combine, i.e., $\forall a, b, c \in D : a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$ and $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$,*
3. *$\overline{0}$ is an annihilator for extend, i.e., $\forall a \in D : a \otimes \overline{0} = \overline{0} \otimes a = \overline{0}$, and*
4. *every $a \in D$ is idempotent w.r.t. combine, i.e., $\forall a \in D : a \oplus a = a$.*

Definition 2 (Ordering). *We write $a \sqsubseteq b$ for $a, b \in D$ whenever $a \oplus b = a$.*

As we are mainly interested in algorithmic verification approaches, we shall implicitly consider only *computable* semirings where the elements from the domain are effectively representable, operations combine and extend are algorithmically computable and the test on equality is decidable. We will use the big-O-notation for complexity upper-bounds, though it should be always interpreted relative to the complexity of the semiring operations. In the semirings considered in our applications, we can assume that all operations can be performed in $O(1)$ time. Hence the big-O-notation for the semirings mentioned in this paper corresponds to the standard asymptotic complexity.

Lemma 1. *(i) For all $a, b \in D$ it holds that $a \oplus b \sqsubseteq a$. (ii) For all $a, b, c \in D$ it holds that if $a \sqsubseteq b$ then $a \otimes c \sqsubseteq b \otimes c$.*

The proof of Lemma 1 is straightforward. We shall now define an additional condition on the extend operator that will be used later on in this section.

Definition 3 (Extend Preserves Inequality). *Given an idempotent semiring we say that* extend preserves inequality *if $a \neq b$ implies that $a \otimes c \neq b \otimes c$ for any $a, b, c \in D \setminus \{\overline{0}\}$.*

Example 1. The tuple $\mathcal{S}_{int} = (\mathbb{Z}_\infty, \min, +, \infty, 0)$ is an idempotent semiring. The domain are the integers extended with infinity $\mathbb{Z}_\infty = \mathbb{Z} \cup \{\infty\}$ where $\min(\infty, a) = \min(a, \infty) = a$ and $a + \infty = \infty + a = \infty$ for all $a \in \mathbb{Z}_\infty$. Combine is the minimum and extend is the usual addition on integers. It is easy to see that $\mathcal{S}_{int}$ meets the requirements of Definition 1. It moreover preserves inequality because the addition does so, and $\sqsubseteq$ is a total order.

Another example of an idempotent semirings is $\mathcal{S}_{rat} = (\mathbb{Q}[0,1], \max, *, 0, 1)$ which is the semiring defined over the rationals in the interval from 0 to 1. Here combine is the maximum and extend is the multiplication on rationals. This semiring $\mathcal{S}_{rat}$ also meets the requirements of Definition 1, extend preserves inequality and $\sqsubseteq$ is a total order. $\square$

In what follows we fix an idempotent semiring $\mathcal{S} = (D, \oplus, \otimes, \overline{0}, \overline{1})$. We often omit the $\otimes$ sign in "products", i.e., we write ab for $a \otimes b$. We also fix a set $\mathcal{X} = \{X_1, \ldots, X_n\}$ of variables. Now we define vectors of polynomials over $\mathcal{S}$ and their fixed points following [4].

Let $V = D^n$ denote the set of *vectors* over $\mathcal{S}$. We use bold letters to denote vectors, e.g., $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n)$. We also write $\boldsymbol{X} = (X_1, \ldots, X_n)$ to arrange the variables from $\mathcal{X}$ in a vector. We extend $\sqsubseteq$ to vectors by setting $\boldsymbol{u} \sqsubseteq \boldsymbol{v}$ if $\boldsymbol{u}_i \sqsubseteq \boldsymbol{v}_i$ for all $1 \leq i \leq n$.

A *monomial* is a finite expression $a_1 X_{i_1} a_2 X_{i_2} \cdots a_s X_{i_s} a_{s+1}$ where $s \geq 0$, $a_1, \ldots, a_{s+1} \in D$ and $X_{i_1}, \ldots, X_{i_s} \in \mathcal{X}$. A *polynomial* is an expression of the form $m_1 \oplus \cdots \oplus m_s$ where $s \geq 0$ and $m_1, \ldots, m_s$ are monomials. The value of a monomial $m = a_1 X_{i_1} a_2 \cdots a_s X_{i_s} a_{s+1}$ at $\boldsymbol{v}$ is $m(\boldsymbol{v}) = a_1 \boldsymbol{v}_{i_1} a_2 \cdots a_s \boldsymbol{v}_{i_s} a_{s+1} \in D$. The value of a polynomial $f = m_1 \oplus \cdots \oplus m_s$ at $\boldsymbol{v}$ is $f(\boldsymbol{v}) = m_1(\boldsymbol{v}) \oplus \cdots \oplus m_s(\boldsymbol{v})$. A polynomial induces a mapping from V to D that assigns to $\boldsymbol{v}$ the element $f(\boldsymbol{v})$. A vector of polynomials $\boldsymbol{f} = (\boldsymbol{f}_1, \ldots, \boldsymbol{f}_n)$ is an n-tuple of polynomials; it induces a mapping from V to V that assigns to a vector $\boldsymbol{v}$ the vector $\boldsymbol{f}(\boldsymbol{v}) = (\boldsymbol{f}_1(\boldsymbol{v}), \ldots, \boldsymbol{f}_n(\boldsymbol{v}))$. A *fixed point of* $\boldsymbol{f}$ is a vector $\boldsymbol{v}$ that satisfies $\boldsymbol{v} = \boldsymbol{f}(\boldsymbol{v})$. A greatest fixed point of $\boldsymbol{f}$ is a fixed point $\boldsymbol{v}$ such that $\boldsymbol{v}' \sqsubseteq \boldsymbol{v}$ holds for all other fixed points $\boldsymbol{v}'$. The size $K(\boldsymbol{f})$ of a vector of polynomials $\boldsymbol{f}$ is the total number of $\oplus$ and $\otimes$ operators in $\boldsymbol{f}$. In particular, given a vector $\boldsymbol{v}$, it takes $O(K(\boldsymbol{f}))$ time to compute $\boldsymbol{f}(\boldsymbol{v})$.

Example 2. Consider the semiring $\mathcal{S}_{int}$ from Example 1. Let $\mathcal{X} = \{X_1, X_2, X_3\}$. Then $\boldsymbol{f} = (-2 \oplus X_2 \otimes X_3,\ X_3 \otimes 1,\ X_1 \oplus X_2)$ is a vector of polynomials over $\mathcal{S}_{int}$. It can be rewritten as $\boldsymbol{f} = (\min\{-2, X_2 + X_3\},\ X_3 + 1,\ \min\{X_1, X_2\})$. The size $K(\boldsymbol{f})$ equals 4. $\square$

It is easy to see that polynomials are monotone and continuous mappings w.r.t. $\sqsubseteq$, see Lemma 1. Kleene's theorem can then be applied (see e.g. [13]), which leads to the following proposition.

Proposition 1. *Let* $\boldsymbol{f}$ *be a vector of polynomials. Let the* Kleene sequence $(\boldsymbol{\kappa}^{(k)})_{k \in \mathbb{N}}$ *be defined by* $\boldsymbol{\kappa}^{(0)} = \overline{\boldsymbol{0}}$ *and* $\boldsymbol{\kappa}^{(k+1)} = \boldsymbol{f}(\boldsymbol{\kappa}^{(k)})$.

(a) We have $\boldsymbol{\kappa}^{(k+1)} \sqsubseteq \boldsymbol{\kappa}^{(k)}$ *for all* $k \in \mathbb{N}$.
(b) If a greatest fixed point exists then it is the infimum of $\{\boldsymbol{\kappa}^{(k)} \mid k \in \mathbb{N}\}$.
(c) If the infimum of $\{\boldsymbol{\kappa}^{(k)} \mid k \in \mathbb{N}\}$ *exists then it is the greatest fixed point.*

Proposition 1 is the mathematical basis for the classical fixed-point iteration: apply $\boldsymbol{f}$ until a fixed point is reached, which is, by Proposition 1 (c), the greatest fixed point of $\boldsymbol{f}$. We call this method *Kleene's iteration.* In general, Kleene's iteration does not always reach a fixed point. Some equations, like $X = X \otimes (-1)$

over $\mathcal{S}_{int}$, do not have any (greatest) fixed point, other equations might have a greatest fixed point but it is not achievable in a finite number of Kleene's iterations (consider for example the above equation but over the semiring $\mathcal{S}_{int}$ extended with the element $-\infty$). It is not a priori clear how to detect whether Kleene's iteration terminates, i.e., computes the greatest fixed point in a finite number of iterations.

Algorithm 1 (called "safe Kleene's iteration") solves this problem. If Kleene's iteration reaches the greatest fixed point, then the algorithm computes it. Otherwise it outputs a *witness component* where Kleene's iteration does not terminate. Formally, a witness component is defined as follows.

Definition 4 (Witness Component). *Let $\boldsymbol{f}$ be a vector of polynomials over an idempotent semiring. A component i $(1 \le i \le n)$ is a* witness component *if $\{\boldsymbol{\kappa}_i^{(k)} \mid k \ge 0\}$ is an infinite set.*

In our applications, the presence of a witness component pinpoints a problem of the analyzed model which the user may want to fix. More details are given in Section 4.

Algorithm 1 is based on the generalized Bellman-Ford algorithm of [8] for $\mathcal{S}_{int}$ and generalizes it further to totally ordered semirings where extend preserves inequality and to equations over arbitrary polynomials.

Algorithm 1. Safe Kleene's iteration

```
Input: A vector of polynomials $\boldsymbol{f} = (\boldsymbol{f}_1, \dots, \boldsymbol{f}_n)$ over an idempotent semiring $\mathcal{S} =$
    $(D, \oplus, \otimes, \overline{0}, \overline{1})$ s.t. $\sqsubseteq$ is a total order and where extend preserves inequality.
Output: Greatest fixed point of $\boldsymbol{f}$ or a witness component.
1: $\boldsymbol{\kappa}^{(0)} := \overline{\mathbf{0}}$
2: for $k := 1$ to $n+1$ do
3:     $\boldsymbol{\kappa}^{(k)} := \boldsymbol{f}(\boldsymbol{\kappa}^{(k-1)})$
4: end for
5: if $\exists i$ with $1 \le i \le n$ such that $\boldsymbol{\kappa}_i^{(n+1)} \ne \boldsymbol{\kappa}_i^{(n)}$ then
6:     return "Kleene's iteration does not terminate. Component $i$ is a witness."
7: else
8:     return "The vector $\boldsymbol{\kappa}^{(n)}$ is the greatest fixed point."
9: end if
```

Theorem 1. *Algorithm 1 is correct and terminates in time $O(n \cdot K(\boldsymbol{f}))$.*

Algorithm 1 on its own is very straightforward, and its proof for polynomials of degree only 1 would directly mimic the proof of Bellman-Ford algorithm. Our contribution is that we prove that it works also for polynomials of higher degrees where more involved technical treatment is necessary. Details are presented in [12].

Remark 1. In the integer semiring $\mathcal{S}_{int}$, Algorithm 1 can be extended such that it computes *all* witness components and for the remaining terminating components returns the exact value. This is done as follows. The main loop on lines 2–4 is run once again, but the components that still change are assigned a new semiring element "$-\infty$" on which the operators "+" and "min" act as expected. Thus, $-\infty$ may be propagated through the components during the repetition of the main loop. At the end, all components that are not $-\infty$ have reached their final value, all others can be reported as witness components. For details see [8].

Example 3. Consider again the vector of polynomials from Example 2:

$$\boldsymbol{f} = (\min\{-2, X_2 + X_3\},\ X_3 + 1,\ \min\{X_1, X_2\})\ .$$

Kleene's iteration produces the following Kleene sequence: $\boldsymbol{\kappa}^{(0)} = (\infty, \infty, \infty)$, $\boldsymbol{\kappa}^{(1)} = (-2, \infty, \infty)$, $\boldsymbol{\kappa}^{(2)} = (-2, \infty, -2)$, $\boldsymbol{\kappa}^{(3)} = (-2, -1, -2)$, $\boldsymbol{\kappa}^{(4)} = (-3, -1, -2)$. As $\boldsymbol{\kappa}_1^{(3)} = -2 \neq -3 = \boldsymbol{\kappa}_1^{(4)}$, Alg. 1 returns the first component as a witness. □

Notice that Algorithm 1 merely indicates whether a greatest fixed point can be found *using Kleene's iteration* or not. Even if Algorithm 1 outputs a witness component, a greatest fixed point may still exist (and be found by other means). An example is a semiring over the reals which can admit the sequence $1/2^n$ for some variable. This sequence converges to 0, but Kleene's iteration fails to detect this. Nevertheless, for some semirings like $\mathcal{S}_{int}$ used in our applications, we can make the following stronger statement.

Corollary 1. *Algorithm 1 applied to polynomials over the semiring $\mathcal{S}_{int}$ finds the greatest fixed point iff it exists. If it does not exist, all witness components can be explicitly marked.*

Proof. In $\mathcal{S}_{int}$ a component is a witness component iff Kleene's iteration does not terminate in that component. The rest follows from Definition 4, Proposition 1 and Remark 1. □

3 Weighted Pushdown Systems

In this section we will use the fixed-point equations studied in the previous section for reasoning about properties of weighted pushdown systems (WPDS) [20]. We are interested in applying Theorem 1 to weighted pushdown systems; therefore we implicitly consider only semirings that are totally ordered, and where extend preserves inequality.

Definition 5 (Weighted Pushdown System). *A weighted pushdown system is a 4-tuple $\mathcal{W} = (P, \Gamma, \Delta, \mathcal{S})$, where P is a finite set of* control states*, Γ is a finite* stack alphabet*, $\Delta \subseteq (P \times \Gamma) \times D \times (P \times \Gamma^*)$ is a finite set of* rules*, and $\mathcal{S} = (D, \oplus, \otimes, \overline{0}, \overline{1})$ is an idempotent semiring.*

We write $pX \overset{d}{\hookrightarrow} q\alpha$ whenever $r = (p, X, d, q, \alpha) \in \Delta$ and call d the *weight* of r, denoted by d_r. We consider only rules where $|\alpha| \leq 2$. (It is well-known that every WPDS can be translated into a one that obeys this restriction and is larger by only a constant factor, see, e.g., [21]. The reduction preserves reachability.) We let the symbols X, Y, Z range over Γ and α, β, γ range over Γ^*.

Example 4. As a running example in this section, we consider a weighted pushdown system over the semiring with both positive and negative integers as weights, i.e. $\mathcal{W}_{ex} = (\{p, q\}, \{X, Y\}, \Delta_{ex}, \mathcal{S}_{int})$, where $\Delta_{ex} = \{pX \overset{1}{\hookrightarrow} qY,\ \ pX \overset{1}{\hookrightarrow} pXY,\ \ pY \overset{1}{\hookrightarrow} p,\ \ qY \overset{-2}{\hookrightarrow} q\}$. $\square$

A *configuration* of a weighted pushdown system $\mathcal{W}$ is a pair $p\gamma$ where $p \in P$ and $\gamma \in \Gamma^*$. A transition relation $\Rightarrow$ on configurations is defined by $pX\gamma \overset{r}{\Rightarrow} q\alpha\gamma$ iff $\gamma \in \Gamma^*$ and there exists $r \in \Delta$, where $r = (pX \overset{d}{\hookrightarrow} q\alpha)$. We annotate $\Rightarrow$ with the rule $r \in \Delta$ which was used to derive the conclusion. If there exists a sequence of configurations $c_0, \dots, c_n$ and rules $r_1, \dots, r_n$ such that $c_{i-1} \overset{r_i}{\Rightarrow} c_i$ for all $i = 1, \dots, n$, then we write $c_0 \overset{\sigma}{\Rightarrow} c_n$, where $\sigma := r_1 \dots r_n$. The weight of σ is defined as $v(\sigma) = d_{r_1} \otimes \cdots \otimes d_{r_n}$. By definition $v(\epsilon) = \overline{1}$.

Let c, c' be two configurations and $\sigma \in \Delta^*$ such that $c \overset{\sigma}{\Rightarrow} c'$. We call c a *predecessor* of c' and c' a *successor* of c. In the following, we will consider the problem of computing the set of all predecessors $pre^*(c_f)$ and successors $post^*(c_f)$ for a given configuration c_f. Due to space limitations we provide the full treatment only for the predecessors; the computation of successors is analogous and it is provided in [12].

Let us fix a WPDS $\mathcal{W}$ and a *target configuration* c_f, where $c_f = p_f\epsilon$ for some control state p_f. For any configuration c of $\mathcal{W}$, we want to know the minimal weight of a path from c to c_f. If a path of minimal weight does not exist for every c, we want to detect such a case. In our applications (see Section 4), this situation usually indicates the existence of an error.

Remark 2. In the literature, it is more common to consider a *regular* set C of target configurations. This problem, however, reduces to the one with only a single target configuration c_f. The reduction can be achieved by extending $\mathcal{W}$ with additional 'pop' rules that simulate a finite automaton for C; the 'pop' rules will succeed in reducing the stack to c_f iff they begin with a configuration in C. For details, see [20], Section 3.1.1.

At an abstract level, we are interested in solutions for the following equation system, in which each configuration c is represented by a variable $[c]$. Intuitively, the greatest solution (if it exists) for the variable $[c]$ will correspond to the minimum (w.r.t. the combine operator) of accumulated weights over all paths leading from the configuration c to c_f.

$$[c] = I(c) \oplus \bigoplus_{c \overset{r}{\Rightarrow} c'} (d_r \otimes [c']), \qquad \text{where } I(c) := \begin{cases} \overline{1} & \text{if } c = c_f \\ \overline{0} & \text{otherwise} \end{cases} \tag{1}$$

Let us consider the Kleene sequence $(\boldsymbol{\kappa}^{(k)})_{k \in \mathbb{N}}$ for (1). By $\boldsymbol{\kappa}^{(k)}_{[c]}$ we denote the entry for configuration c in the k-th iteration of the Kleene sequence.

Lemma 2. *For $k \geq 1$ and any configuration c, the following holds*

$$\boldsymbol{\kappa}^{(k)}_{[c]} = \bigoplus \{\, v(\sigma) \mid c \overset{\sigma}{\Rightarrow} c_f,\ |\sigma| < k \,\} \ .$$

Thus, $[c]$ is a witness component of (1) iff no path of minimal weight exists, because it is possible to construct longer and longer paths with smaller and smaller weights. On the other hand, if (1) has a greatest fixed point, then the fixed point at $[c]$ gives the combine of the weights of all sequences leading from c to c_f, commonly known as the *meet-over-all-paths.* However, (1) defines an infinite system of equations, which we cannot handle directly. In the following, we shall derive a *finite* system of equations, from which we can determine the greatest fixed point of (1) or the existence of a witness component.

Definition 6 (Pop Sequence). *Let p, q be control states and X be a stack symbol. A* pop sequence *for p, X, q is any sequence $\sigma \in \Delta^*$ such that $pX \overset{\sigma}{\Rightarrow} q\epsilon$.*

Let us consider the following polynomial equation system, in which the variables are triples $[pXq]$, where p, q are control states and X a stack symbol:

$$[pXq] = \bigoplus_{(pX \overset{d}{\hookrightarrow} q\epsilon) \in \Delta} d \;\oplus\; \bigoplus_{(pX \overset{d}{\hookrightarrow} rY) \in \Delta} \big(d \otimes [rYq]\big) \;\oplus\; \bigoplus_{(pX \overset{d}{\hookrightarrow} rYZ) \in \Delta} \left(d \otimes \bigoplus_{s \in P} \big([rYs] \otimes [sZq]\big)\right) . \quad (2)$$

Intuitively, Equation (2) lists all the possible ways in which a pop sequence for p, X, q can be generated and computes the values accumulated along each of them.

Example 5. Let us consider the WPDS $\mathcal{W}_{ex}$ from Example 4. Here, the scheme presented in (2) yields a system with eight variables and equations, four of which are reproduced below.

$$\begin{aligned} [pXp] &= \min\{1 + [qYp],\ 1 + [pXp] + [pYp],\ 1 + [pXq] + [qYp]\} & [pYp] &= 1 \\ [pXq] &= \min\{1 + [qYq],\ 1 + [pXp] + [pYq],\ 1 + [pXq] + [qYq]\} & [qYq] &= -2 \end{aligned}$$

Notice that the other four variables would be simply assigned to the $\overline{0}$ element, in this case ∞. $\square$

We now examine the Kleene sequence $(\boldsymbol{\kappa}^{(k)})_{k \in \mathbb{N}}$ for (2).

Lemma 3. *For any $k \geq 1$, control states p, q, and stack symbol X,*

$$\bigoplus \{\, v(\sigma) \mid c \overset{\sigma}{\Rightarrow} c_f,\ |\sigma| \leq 2^{k-1} \,\} \sqsubseteq \boldsymbol{\kappa}^{(k)}_{[pXq]} \sqsubseteq \bigoplus \{\, v(\sigma) \mid c \overset{\sigma}{\Rightarrow} c_f,\ |\sigma| \leq k-1 \,\} .$$

Thus, $[pXq]$ is a witness component of (2) iff no minimal-weight pop sequence exists for p, X, q. On the other hand, if no witness component exists, then the value of $[pXq]$ in the greatest fixed point denotes the combine of the weights of all pop sequences for p, X, q.

We now show how (2) can be used to derive statements about (1). Let a configuration $c = pX_1 \ldots X_n$ be a predecessor of c_f. Then any sequence σ leading from c to c_f can be subdivided into subsequences $\sigma_1, \ldots, \sigma_n$ and there exist states $p =: p_0, p_1, \ldots, p_{n-1}, p_n := p_f$ such that σ_i is a pop sequence for p_{i-1}, X_i, p_i, for all $i = 1, \ldots, n$. As a consequence, we can obtain a solution for (1) from a solution for (2): suppose that $\boldsymbol{\lambda}$ is the greatest fixed point of (2), and let $\boldsymbol{\mu}$ be a vector of configurations as follows:

$$\boldsymbol{\mu}_{[c]} = \bigoplus_{p_1,\dots,p_{n-1}} \left(\boldsymbol{\lambda}_{[pX_1p_1]} \otimes \cdots \otimes \boldsymbol{\lambda}_{[p_{n-1}X_np_f]}\right), \qquad \text{for } c = pX_1 \dots X_n\ . \tag{3}$$

It is easy to see that (3) "sums up" all possible paths from c to c_f, and therefore yields the meet-over-all-paths for c. Thus, $\boldsymbol{\mu}$ is a solution (greatest fixed point) of (1). On the other hand, if (1) has a witness component, then (2) must also have one.

Theorem 2. *Applying Algorithm 1 to (2) either yields a witness component or, via (3), the greatest fixed point of (1).*

Example 6. Once again, consider $\mathcal{W}_{ex}$ from Example 4 and the equation system from Example 5. Here, the Kleene sequence quickly converges to the values 1 for $[pYp]$, -2 for $[qYq]$, and ∞ for all other variables except $[pXq]$, which turns out to be a witness component of (2). Indeed, one can construct a series of pop sequences for p, X, q with smaller and smaller weights, e.g. $pX \overset{1}{\Rightarrow} qY \overset{-2}{\Rightarrow} q\epsilon$, and $pX \overset{1}{\Rightarrow} pXY \overset{1}{\Rightarrow} qYY \overset{-2}{\Rightarrow} qY \overset{-2}{\Rightarrow} q\epsilon$, and etc. with weights -1, -2 etc. If $c_f = q\epsilon$, this implies that, e.g., pX is a witness component of (1). On the other hand, qY or qYY would not be a witness components, because their values in (3), would not be affected by the variable $[pXq]$ and evaluate to -2 and -4, respectively. $\square$

Remark 3. The size of the equation system (2) is polynomial in $\mathcal{W}$. Notice that it makes sense to generate equations only for such triples p, X, q in which pX occurs on the left-hand side or right-hand side of some rule. Under this assumption, the number of equations in (2) is $\mathcal{O}(|P| \cdot |\Delta|)$, and its overall size is $\mathcal{O}(|P|^2 \cdot |\Delta|)$, the same complexity as in the algorithms for computing predecessors in [3]. According to Theorem 1, Algorithm 1 therefore runs in $\mathcal{O}(|P|^3 \cdot |\Delta|^2)$ time on (2). For any configuration c of interest, the value $\boldsymbol{\mu}_c$ in (3) can be easily obtained from the result of Algorithm 1. See also the $\mathcal{W}$-automaton technique in the subsection to follow. A similar conclusion about the complexity of the algorithm for computing successors can be drawn thanks to the (linear) connection between forward and backward reachability analysis described in [12].

3.1 Weighted Automata

For (unweighted) pushdown systems, it is well-known that reachability preserves regularity; in other words, given a regular set of configurations, the set of all predecessors resp. successors is regular. Moreover, given a finite automaton recognizing a set of configurations, automata recognizing the predecessors or successors can be constructed in polynomial time (see, e.g., [3]).

It is also known that the results carry over to weighted pushdown systems provided that the semiring is *bounded*, i.e., there are no infinite descending chains w.r.t. $\sqsubseteq$ [20]. For this purpose, so-called *weighted automata* are employed.

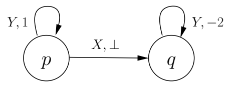
Definition 7 (Weighted $\mathcal{W}$-Automaton). *Let $\mathcal{W} = (P, \Gamma, \Delta, \mathcal{S})$ be a pushdown system over a bounded semiring $\mathcal{S}$. A $\mathcal{W}$-*automaton *is a 5-tuple $\mathcal{A} = (Q, \Gamma, \rightarrow, P, F)$ where Q is a finite set of* states*, $\rightarrow \subseteq Q \times \Gamma \times D \times Q$ is a finite set of* transitions*, $P \subseteq Q$, i.e. the control states of $\mathcal{W}$, are the set of* initial states *and $F \subseteq Q$ is a set of* final (accepting) states*.*

Let $\pi = t_1 \dots t_n$ be a path in $\mathcal{A}$, where $t_i = (q_i, X_i, d_i, q_{i+1})$ for all $1 \leq i \leq n$. The weight of π is defined as $v(\pi) := d_1 \otimes \dots \otimes d_n$. If $q_1 \in P$ and $q_{n+1} \in F$, then we say that π accepts *the configuration $q_1 X_1 \dots X_n$. Moreover, if c is a configuration, we define $v_{\mathcal{A}}(c)$ as the combine of all $v(\pi)$ such that π accepts c. In this case, we also say that $\mathcal{A}$* accepts c with weight $v_{\mathcal{A}}(c)$*.*

In [20] the following problem is considered for the case of bounded semirings: compute a $\mathcal{W}$-automaton $\mathcal{A}$ such that $v_A(c)$ equals the meet-over-all-paths (or equivalently the greatest fixed point of (1), which always exists for bounded semirings) from c to c_f, for every configuration c.

We extend this solution to the case of unbounded semirings, using Theorem 2. We first apply Algorithm 1 to the equation system (2). If the algorithm yields the greatest fixed point, then we construct a $\mathcal{W}$-automaton $\mathcal{A} = (P, \Gamma, \rightarrow, P, \{c_f\})$, with $(p, X, d, q) \in \rightarrow$ for all p, X, q such that d is the value of $[pXq]$ in the greatest fixed point computed by Algorithm 1. Given a configuration c, it is easy to see that $v_{\mathcal{A}}(c)$ yields the same result as in (3).

Example 7. The automaton arising from Example 6 is depicted below where the witness component is marked by $\perp$ and transitions with the value ∞ are omitted completely.



□

The problem of computing successors is also considered in [20], i.e., computing a $\mathcal{W}$-automaton $\mathcal{A}$ where $v_{\mathcal{A}}(c)$ is the meet-over-all-paths from an initial configuration c_0 to c. Using our technique, this result can also be extended to unbounded semirings; [12] shows an equation system for this problem, which can be converted into a $\mathcal{W}$-automaton for $post^*(c_0)$ in analogous fashion.

4 Applications

Here we outline some applications of the theory developed in this paper. Unless stated otherwise, we will consider the semiring $\mathcal{S}_{int}$ as described in Example 1. Following Remark 1 and Corollary 1, we assume that all nonterminating components can be detected in this semiring and the corresponding transitions in the $\mathcal{W}$-automaton will be assigned the value $\perp$. The terminating components resp. the corresponding transitions in the $\mathcal{W}$-automaton take the computed value.

Note that the previously known approaches to reachability in weighted pushdown automata are not applicable to any of the below presented cases because they required the semiring to be bounded (no infinite descending chains). Boundedness is, however, not satisfied in any of our applications. Our first two applications are new and we are not aware of any other algorithms that could achieve the same results. Our third application deals with shape-balancedness of context-free languages, a problem for which an algorithm was recently described in [24].

Memory Allocations in Linux Kernel. Correct memory allocation and deallocation is crucial for the proper functionality of an operating system. In Linux the library `linux/gfp.h` is used for allocation and deallocation of kernel memory pages via the functions `alloc_pages` and `__free_pages` respectively. The functions which are argumented with a number n (also called the *order*) allocate or deallocate 2^n memory pages. Citing [16, page 187]: "You must be careful to free only pages you allocate. Passing the wrong `struct page` or address, or the incorrect *order*, can result in corruption." This means that a basic safety requirement is: never free more pages than what are allocated.

As most questions about real programs are in general undecidable, several techniques have been suggested to provide more tractable models. For example so-called boolean programs [2] have recently been used to provide a suitable abstraction via pushdown systems. Assume a given pushdown system abstraction resulting from the program code. The transitions in the pushdown system are labelled with the programming primitives, among others the ones for allocation and deallocation of memory pages. If a given pushdown transition allocates 2^n memory pages, we assign it the weight 2^n; if it deallocates 2^n pages, we assign it the weight -2^n; in all other cases the weight is set to 0.

Now the pushdown abstraction corrupts the memory iff a configuration is reachable from the given initial configuration pX with negative weight. As shown in Section 3, we can in polynomial time (w.r.t. to the input pushdown system $\mathcal{W}$) construct a $\mathcal{W}$-automaton $\mathcal{A}$ for $post^*(\{pX\})$. For technical convenience, we first replace all occurrences of $\perp$ in $\mathcal{A}$ with $-\infty$. From all initial control-states of $\mathcal{A}$ we now run e.g. the Bellman-Ford shortest path algorithm (which can detect negative cycles and assign the weight to $-\infty$ should there be such) to check whether there is a path going to some accept state with an accumulated negative weight. This is doable in polynomial time. If a negative weight path is found this means that the corresponding configuration is reachable with a negative weight, hence there is a memory corruption (at least in the pushdown abstraction). Otherwise, the system is safe. All together our technique gives a polynomial time algorithm for checking memory corruption with respect to the size of the abstracted pushdown system. Also depending on whether under- or over-approximation is used in the abstraction step, our technique can be used for detecting errors or showing the absence of them, respectively.

Correspondence Assertions. In [25] Woo and Lam analyze protocols using the so-called *correspondences* between protocol points. A correspondence property relates the occurrence of a transition to an earlier occurrence of some other transition. In sequential programs (modelled as pushdown systems) assume that assertions of the form `begin` ℓ and `end` ℓ (where ℓ is a label taken from a finite set of labels) are inserted by the programmer into the code. The program is *safe* if for each reachable `end` ℓ there is a unique corresponding `begin` ℓ at an earlier execution point of the program. Verifying safety via correspondence assertions can be done using a similar technique as before. For each label ℓ we create a WPDS based on the initially given boolean program abstraction where every instruction `begin` ℓ has the weight $+1$, every instruction `end` ℓ the weight -1, and all other instructions have the weight 0. Now the pushdown system is safe if and only if every reachable configuration has nonnegative accumulated weight. This can be verified in polynomial time as outlined above.

Shape-Balancedness of Context-Free Languages. In static analysis of programs generating XML strings and in other XML-related questions, the balancedness problem has been recently studied (see e.g. [1,11,17]). The problem is, given a context-free language with a paired alphabet of opening and closing tags, to determine whether every word in the language is properly balanced (i.e. whether every opening tag has a corresponding closing tag and vice versa). Tozawa and Minamide [24] recently suggested a polynomial time algorithm for the problem. Their involved algorithm consists of two stages. In the first stage they test for the *shape-balancedness* property, i.e., if all opening and all closing tags are treated as of the same sort, is every accepted word balanced? Assume a given pushdown automaton accepting (by final control-states) the given context-free language. If we label all opening tags with weight $+1$ and all closing tags with weight -1, the shape-balancedness question is equivalent to checking (i) whether every accepted word has the weight equal to 0 and (ii) whether all configurations on every path to some final control-state have nonnegative accumulated weights. Our generic technique provides polynomial time algorithms to answer these questions.

To verify property (i), we first consider the semiring $\mathcal{S}_{int} = (\mathbb{Z}_\infty, \min, +, \infty, 0)$. We now construct in polynomial time for the given initial configuration pX a weighted $post^*(\{pX\})$ $\mathcal{W}$-automaton $\mathcal{A}$, replace all labels $\perp$ with $-\infty$, and for each final control-state q (of the pushdown automaton) we find in $\mathcal{A}$ a shortest path from q to every accept state of $\mathcal{A}$. This can be done in polynomial time using e.g. the Bellman-Ford shortest path algorithm, which can moreover detect negative cycles and set the respective shortest path to $-\infty$. If any of the shortest paths are different from 0, we terminate because the shape-balancedness property is broken. If the system passes the first test, we run the same procedure once more but this time with the semiring $(\mathbb{Z} \cup \{-\infty\}, \max, +, -\infty, 0)$ and where $\perp$ is replaced with ∞, i.e., we are searching for the longest path in the automaton $\mathcal{A}$. Again if at least one of those paths has the accumulated weight different from 0, we terminate with a negative answer. If the pushdown system passes both our tests, this means that any configuration in the set $post^*(\{pX\})$ starting with some final control-state (of the pushdown automaton) is reachable only with the accumulated weight 0 and we can proceed to verify property (ii).

For (ii), we construct the weighted $post^*(\{pX\})$ $\mathcal{W}$-automaton for the integer semiring $\mathcal{S}_{int}$. Now we restrict the automaton to contain only those configurations that can really involve into some accepting configuration by simply intersecting it (by the usual product construction) with the unweighted $\mathcal{W}$-automaton (of polynomial size) representing $pre^*((q_1 + \cdots + q_n)\Gamma^*)$ where $q_1, \ldots, q_n$ are all final control-states and Γ is the stack alphabet. Property (ii) now reduces to checking whether the product automaton accepts some configuration with negative weight, which can be answered in polynomial time using the technique described in our first application.

Unfortunately, [24] provides no complexity analysis other than the statement that the algorithm is polynomial. Our general-purpose algorithm, on the other hand, immediately provides a precise complexity bound. Consider a given context-free grammar of size n over some paired alphabet. It can be (by the standard textbook construction) translated into a (weighted) pushdown automaton of size $O(n)$ and moreover with a constant number of states. As mentioned in Section 3, this automaton can be normalized in linear time and we can then build a weighted $post^*(\{pX\})$ $\mathcal{W}$-automaton, of size $O(n^2)$ with $O(n)$ states and in time $O(n^4)$. Details can be found in [12]. Now running the Bellman-Ford algorithm twice in order to verify property (i) takes only $O(n^3)$ time. In property (ii) the Bellman-Ford algorithm is run on a product of the weighted $post^*$ automaton and an unweighted pre^* automaton, which has only a constant number states. Hence the size of the product is still $O(n^2)$ and Bellman-Ford algorithm will run in time $O(n^3)$ as before. This gives the total running time of $O(n^4)$.

5 Conclusion

We presented a unified framework how to deal with interprocedural dataflow analysis on weighted pushdown automata where the weight domains might contain infinite descending chains. The problem was solved by reformulating it via generalized fixed-point equations which required polynomials of degree two. To the best of our knowledge this is the first approach that enables to handle this kind of domains. On the other hand, we do not consider completely general idempotent semirings as we require that the elements in the domain are totally ordered and that extend preserves inequality. Nevertheless, we showed that our theory is still applicable. Already the reachability analysis of weighted pushdown automata over the integer semiring, one particular instance of our general framework, was not known before and we provided several examples of its potential use in verification.

Regarding the two restrictions we introduced, we claim that the first condition of total ordering can be relaxed to orderings of bounded width, where the maximum number of incomparable elements is bounded by some a priori given constant c. By running the main loop in Algorithm 1 $cn + 1$ times, we should be able to detect nontermination also in this case. The motivation for introducing bounded width comes from the fact that this will allow us to combine (via the product construction) one unbounded domain, like e.g. the integer semiring, with a fixed number of finite domains in order to observe additional properties along the computations. Whether also the second restriction (extend preserves inequality) can be relaxed remains open and is a part of our future work.

References

1. Berstel, J., Boasson, L.: Formal properties of XML grammars and languages. Acta Informatica 38(9), 649–671 (2002)
2. Bouajjani, A., Esparza, J.: Rewriting models of boolean programs. In: Pfenning, F. (ed.) RTA 2006. LNCS, vol. 4098, pp. 136–150. Springer, Heidelberg (2006)
3. Esparza, J., Hansel, D., Rossmanith, P., Schwoon, S.: Efficient algorithms for model checking pushdown systems. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 232–247. Springer, Heidelberg (2000)
4. Esparza, J., Kiefer, S., Luttenberger, M.: An extension of Newton's method to ω-continuous semirings. In: Harju, T., Karhumäki, J., Lepistö, A. (eds.) DLT 2007. LNCS, vol. 4588, pp. 157–168. Springer, Heidelberg (2007)
5. Esparza, J., Kiefer, S., Luttenberger, M.: On fixed point equations over commutative semirings. In: Stephanidis, C., Pieper, M. (eds.) ERCIM Ws UI4ALL 2006. LNCS, vol. 4397, pp. 296–307. Springer, Heidelberg (2007)
6. Esparza, J., Kiefer, S., Luttenberger, M.: Newton's method for ω-continuous semirings. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 14–26. Springer, Heidelberg (2008)
7. Esparza, J., Kiefer, S., Schwoon, S.: Abstraction refinement with Craig interpolation and symbolic pushdown systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 489–503. Springer, Heidelberg (2006)
8. Gawlitza, T., Seidl, H.: Precise fixpoint computation through strategy iteration. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 300–315. Springer, Heidelberg (2007)
9. Hopkins, M.W., Kozen, D.: Parikh's theorem in commutative Kleene algebra. In: Proc. LICS, pp. 394–401. IEEE, Los Alamitos (1999)
10. Jha, S., Schwoon, S., Wang, H., Reps, T.: Weighted pushdown systems and trust-management systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 1–26. Springer, Heidelberg (2006)
11. Kirkegaard, C., Møller, A.: Static Analysis for Java Servlets and JSP. In: Yi, K. (ed.) SAS 2006. LNCS, vol. 4134, pp. 336–352. Springer, Heidelberg (2006)
12. Kühnrich, M., Schwoon, S., Srba, J., Kiefer, S.: Interprocedural dataflow analysis over weight domains with infinite descending chains. Technical report (2009), `http://arxiv.org/abs/0901.0501`
13. Kuich, W.: Semirings and Formal Power Series: Their Relevance to Formal Languages and Automata. In: Handbook of Formal Languages, ch.9, vol. 1, pp. 609–677. Springer, Heidelberg (1997)
14. Lal, A., Lim, J., Polishchuk, M., Liblit, B.: Path optimization in programs and its application to debugging. In: Sestoft, P. (ed.) ESOP 2006. LNCS, vol. 3924, pp. 246–263. Springer, Heidelberg (2006)
15. Leroux, J., Sutre, G.: Accelerated data-flow analysis. In: Riis Nielson, H., Filé, G. (eds.) SAS 2007. LNCS, vol. 4634, pp. 184–199. Springer, Heidelberg (2007)
16. Love, R.: Linux Kernel Development, 2nd edn. Novell Press (2005)
17. Minamide, Y., Tozawa, A.: XML validation for context-free grammars. In: Kobayashi, N. (ed.) APLAS 2006. LNCS, vol. 4279, pp. 357–373. Springer, Heidelberg (2006)
18. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer, Heidelberg (1999)

19. Reps, T., Lal, A., Kidd, N.: Program analysis using weighted pushdown systems. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 23–51. Springer, Heidelberg (2007)
20. Reps, T., Schwoon, S., Jha, S., Melski, D.: Weighted pushdown systems and their application to interprocedural dataflow analysis. SCP 58(1–2), 206–263 (2005)
21. Schwoon, S.: Model-Checking Pushdown Systems. PhD thesis, TU Munich (2002)
22. Sharir, M., Pnueli, A.: Two Approaches to Interprocedural Data Flow Analysis. In: Program Flow Analysis: Theory and Applications, ch.7, pp. 189–233. Prentice-Hall, Englewood Cliffs (1981)
23. Suwimonteerabuth, D., Berger, F., Schwoon, S., Esparza, J.: jMoped: A test environment for Java programs. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 164–167. Springer, Heidelberg (2007)
24. Tozawa, A., Minamide, Y.: Complexity results on balanced context-free languages. In: Seidl, H. (ed.) FOSSACS 2007. LNCS, vol. 4423, pp. 346–360. Springer, Heidelberg (2007)
25. Woo, T.Y.C., Lam, S.S.: A semantic model for authentication protocols. In: Proc. SP, pp. 112–118. IEEE, Los Alamitos (1993)

Realizability Semantics of Parametric Polymorphism, General References, and Recursive Types

Lars Birkedal, Kristian Støvring, and Jacob Thamsborg

IT University of Copenhagen, Rued Langgaards Vej 7,
2300 Copenhagen S, Denmark

Abstract. We present a realizability model for a call-by-value, higher-order programming language with parametric polymorphism, general first-class references, and recursive types. The main novelty is a relational interpretation of open types (as needed for parametricity reasoning) that include general reference types. The interpretation uses a new approach to modeling references.

The universe of semantic types consists of world-indexed families of logical relations over a universal predomain. In order to model general reference types, worlds are finite maps from locations to semantic types: this introduces a circularity between semantic types and worlds that precludes a direct definition of either. Our solution is to solve a recursive equation in an appropriate category of metric spaces. In effect, types are interpreted using a Kripke logical relation over a recursively defined set of worlds.

We illustrate how the model can be used to prove simple equivalences between different implementations of imperative abstract data types.

1 Introduction

In this article we develop a semantic model of a call-by-value programming language with impredicative and parametric polymorphism, general first-class references, and recursive types. Motivations for conducting this study include:

- Extending the approach to reasoning about abstract data types via relational parametricity from pure languages to more realistic languages with effects, here general references. We discussed this point of view extensively earlier [8].
- Investigating what semantic structures are needed in general models for effects. Indeed, we see the present work as a pilot study for studying general type theories and models of effects (e.g., [12, 19]), in which we identify key ingredients needed for semantic modeling of general first-class references.
- Paving the way for developing models of separation logic for ML-like languages with reference types. Earlier such models of separation logic [16] only treat so-called strong references, where the type on the contents of a reference cell can vary: therefore proof rules cannot take advantage of the strong invariants provided by ML-style reference types.

We now give an overview of the conceptual development of the paper. The development is centered around three recursively defined structures, defined in three stages. In slogan form, there is one recursively defined structure for each of the type constructors $\forall$, ref, and μ alluded to in the title.

First, since the language involves impredicative polymorphism, the semantic model is based on a realizability interpretation [4] over a certain recursively defined predomain V. Using this predomain we can give a denotational semantics of an untyped version of the language. This part is mostly standard, except for the fact that we model locations as pairs (l, n), with l a natural number corresponding to a standard location and $n \in \mathbb{N} \cup \{\infty\}$ indicating the "approximation stage" of the location [8]. These pairs, called semantic locations, are needed for modeling reference types in stage three.[1]

Second, to account for dynamic allocation of typed reference cells, we follow earlier work on modeling simple integer references [7] and use a Kripke-style possible worlds model. Here, however, the set of worlds needs to be recursively defined since we treat general references. Semantically, a world maps locations to semantic types, which, following the general realizability idea, are certain world-indexed families of relations on V: this introduces a circularity between semantic types and worlds that precludes a direct definition of either. Thus we need to solve recursive equations of approximately the following form

$$\mathcal{W} = \mathbb{N}_0 \rightharpoonup_{fin} \mathcal{T}$$
$$\mathcal{T} = \mathcal{W} \to \mathit{CURel}(V)$$

even in order to define the space in which types will be modeled. We formally define the recursive equations in certain ultrametric spaces and show how to solve them using known results from metric-space based semantics. The employed metric on relations on V is well-known from work on interpreting recursive types and impredicative polymorphism [1, 4, 5, 10, 13]; here we extend its use to reference types (combined with these two other features).

Third, having now defined the space in which types should be modeled, the actual semantics of types can be defined. For recursive types, that also involves a recursive definition. Since the space $\mathcal{T}$ of semantic types is a metric space we can employ Banach's fixed point theorem to find a solution as the fixed point of a contractive operator on $\mathcal{T}$.[2] This involves interpreting the various type constructors of the language as non-expansive operators. For most type constructors doing so is straightforward, but for the reference-type constructor

[1] Intuitively, the problem with modeling locations using a flat cpo of natural numbers is that such "flat" locations contain no approximation information that can be used to define relations by induction. (See page 466.)

[2] We remark that the fixed point could also be found using the technique of Pitts [18]; the proof techniques are very similar because of the particular way the requisite metrics are defined. In this article we do in any case need the metric-space formulation, but not the extra separation of positive and negative arguments in recursive definitions of relations, and hence we define the meaning of recursive types via Banach's fixed point theorem.

it is not. That is the reason for introducing the semantic locations mentioned above: using these, we can define a semantic reference-type operator (and show that it is non-expansive).

Finally, having now defined semantics of types using a family of world-indexed logical relations, we define the typed meaning of terms by proving the fundamental theorem of logical relations wrt. the untyped semantics of terms.

In this article we do not consider operational semantics but focus on presenting the model outlined above. We have earlier shown a computational-adequacy result for a semantics similar to the untyped semantics defined in stage one [8]: we expect that result to carry over to the present setup. Also, the model does not validate standard equivalences involving local state,[3] although we expect that it can be extended to do so (see Section 6). Here we rather aim to present the fundamental ideas behind Kripke logical relations over recursively defined sets of worlds.

The remainder of the article is organized as follows. Section 2 sketches the language we consider. In Section 3 we present the untyped semantics, corresponding to stage one in the outline above. In Section 4 we present the typed semantics, corresponding to the last two stages. In Section 5 we present a few examples of reasoning using the model. Related work is discussed in Section 6.

Because of space limitations, some definitions and most proofs have been omitted from this article. They can be found in the long version, available from the authors' web pages.[4]

2 Language

We consider a standard call-by-value language with universal types, iso-recursive types, ML-style reference types, and a ground type of integers. The language is sketched in Figure 1. Terms are not intrinsically typed; this allows us to give a denotational semantics of untyped terms. The typing rules are standard [17]. In the figure, Ξ and Γ range over contexts of type variables and term variables, respectively. As we do not consider operational semantics in this article, there is no need for location constants, and hence no need for store typings.

3 Untyped Semantics

We now give a denotational semantics for the untyped term language above. As usual for models of untyped languages, the semantics is given by means of a "universal" complete partial order (cpo) in which one can inject integers, pairs, functions, etc. This universal cpo is obtained by solving a recursive predomain equation. The only non-standard aspect of the semantics is the treatment of store locations: locations are modeled as elements of the cpo $Loc = \mathbb{N}_0 \times \overline{\omega}$ where $\overline{\omega}$ is the "vertical natural numbers" cpo: $1 \sqsubset 2 \sqsubset \cdots \sqsubset n \sqsubset \cdots \sqsubset \infty$. (For notational reasons it is convenient to call the least element 1 rather than 0.) The

[3] The model can only equate computations that allocate references "in lockstep".

[4] Currently: `http://www.itu.dk/people/kss/papers/poly-ref-rec.pdf`

$$\text{Types:} \qquad \tau ::= \mathsf{int} \mid 1 \mid \tau_1 \times \tau_2 \mid \tau_1 + \tau_2 \mid \tau_1 \to \tau_2 \mid \mu\alpha.\tau \mid \forall\alpha.\tau \mid \alpha \mid \mathsf{ref}\,\tau$$

$$\begin{aligned}\text{Terms:} \qquad t ::= {} & x \mid \overline{n} \mid \mathsf{ifz}\,t_0\,t_1\,t_2 \mid t_1 + t_2 \mid t_1 - t_2 \mid () \mid (t_1,\, t_2) \mid \mathsf{fst}\,t \mid \mathsf{snd}\,t \\ & \mid \mathsf{inl}\,t \mid \mathsf{inr}\,t \mid \mathsf{case}\,t_0\;x_1.t_1\;x_2.t_2 \mid \lambda x.t \mid \mathsf{fix}\,f.\lambda x.t \mid t_1\;t_2 \\ & \mid \mathsf{fold}\;t \mid \mathsf{unfold}\;t \mid \Lambda\alpha.t \mid t\,[\tau] \mid \mathsf{ref}\;t \mid {!t} \mid t_1 := t_2\end{aligned}$$

Sample typing rules:

$$\frac{\Xi \mid \Gamma \vdash t : \tau[\mu\alpha.\tau/\alpha]}{\Xi \mid \Gamma \vdash \mathsf{fold}\,t : \mu\alpha.\tau} \qquad \frac{\Xi \mid \Gamma \vdash t : \mu\alpha.\tau}{\Xi \mid \Gamma \vdash \mathsf{unfold}\,t : \tau[\mu\alpha.\tau/\alpha]}$$

$$\frac{\Xi, \alpha \mid \Gamma \vdash t : \tau}{\Xi \mid \Gamma \vdash \Lambda\alpha.t : \forall\alpha.\tau}\;(\Xi \vdash \Gamma) \qquad \frac{\Xi \mid \Gamma \vdash t : \forall\alpha.\tau_0}{\Xi \mid \Gamma \vdash t\,[\tau_1] : \tau_0[\tau_1/\alpha]}\;(\Xi \vdash \tau_1)$$

$$\frac{\Xi \mid \Gamma \vdash t : \tau}{\Xi \mid \Gamma \vdash \mathsf{ref}\;t : \mathsf{ref}\,\tau} \qquad \frac{\Xi \mid \Gamma \vdash t : \mathsf{ref}\,\tau}{\Xi \mid \Gamma \vdash {!t} : \tau}$$

$$\frac{\Xi \mid \Gamma \vdash t_1 : \mathsf{ref}\,\tau \qquad \Xi \mid \Gamma \vdash t_2 : \tau}{\Xi \mid \Gamma \vdash t_1 := t_2 : 1}$$

Fig. 1. Programming language

intuitive idea is that locations can be approximated: the element $(l, \infty) \in \mathit{Loc}$ is the "ideal" location numbered l, while the elements of the form (l, n) for $n \neq \infty$ are its approximations. It is essential for the construction of the typed semantics (in the next section) that these "approximate locations" (l, n) are included.

Let in the following Cpo be the category of ω-cpos and ω-continuous functions. We use the standard notation for products, sums, lifting, and function spaces in Cpo. Injections into binary sums are written ι_1 and ι_2. For any set M and any cpo A, the cpo $M \rightharpoonup_{\mathit{fin}} A$ has maps from finite subsets of M to A as elements, and is ordered as follows: $f \sqsubseteq f'$ if and only if f and f' has the same domain M_0 and $f(m) \sqsubseteq f'(m)$ for all $m \in M_0$. The Kleisli composition $g \mathbin{\overline{\circ}} f$ of two continuous functions $f : A \to B_\bot$ and $g : B \to C_\bot$ is given by $(g \mathbin{\overline{\circ}} f)(a) = g\,b$ if $f\,a = \lfloor b \rfloor$ for some b, and $(g \mathbin{\overline{\circ}} f)(a) = \bot$ otherwise. A complete, pointed partial order (cppo) is a cpo containing a least element. The least fixed-point of a continuous function $f : D \to D$ from a cppo D to itself is written $\mathit{fix}\,f$.

The semantics below is presented in monadic style [15], i.e., structured using a monad that models the effects of the language. It is most convenient to define this monad by means of a Kleisli triple: for every cpo S and every cppo Ans, the continuation-and-state monad $T_{S,\mathit{Ans}} : \mathsf{Cpo} \to \mathsf{Cpo}$ over S and Ans is given by $T_{S,\mathit{Ans}}\,A = (A \to S \to \mathit{Ans}) \to S \to \mathit{Ans}$, $\eta_A\,a = \lambda k.\lambda s.\,k\,a\,s$, and $c \star_{A,B} f = \lambda k.\lambda s.\,c\,(\lambda a.\lambda s'.f\;a\;k\;s')\;s$ (with $\eta_A : A \to T_{S,\mathit{Ans}}A$ and $\star_{A,B} : T_{S,\mathit{Ans}}A \to (A \to T_{S,\mathit{Ans}}B) \to T_{S,\mathit{Ans}}B$.) In the following we omit the type subscripts on η and $\star$.[5]

[5] Continuations are included for a technical reason, namely to ensure chain-completeness of the relations that will be used to model computations.

The standard methods for solving recursive (pre)domain equations give solutions that satisfy certain induction principles [18, 21]. One aspect of these induction principles is that, loosely speaking, one obtains as a solution not only a cpo A, but also a family of "projection" functions ϖ_n on A (one function for each $n \in \omega$) such that each element a of A is the limit of its projections $\varpi_0(a)$, $\varpi_1(a)$, etc. These functions therefore provide a handle for proving properties about A by induction on n.

Definition 1. *A* uniform cpo $(A, (\varpi_n)_{n\in\omega})$ *is a cpo A together with a family $(\varpi_n)_{n\in\omega}$ of continuous functions from A to $A_\perp$, satisfying*

$$\begin{aligned} \varpi_0 &= \lambda e.\,\perp \\ \varpi_0 &\sqsubseteq \varpi_1 \sqsubseteq \cdots \sqsubseteq \varpi_n \sqsubseteq \ldots \\ \bigsqcup_{n\in\omega} \varpi_n &= \lambda a.\lfloor a \rfloor \\ \varpi_m \mathbin{\bar{\circ}} \varpi_n = \varpi_n \mathbin{\bar{\circ}} \varpi_m &= \varpi_{\min(m,n)}\,. \end{aligned}$$

We are now ready to construct a uniform cpo $(V, (\pi_n)_{n\in\omega})$ such that V is a suitable "universal" cpo. The functions π_n will be used in the definition of the untyped semantics. Intuitively, if one for example looks up the approximate location $(l,\, n{+}1)$ in a store s, one only obtains the approximate element $\pi_n(s(l))$ as a result.

Proposition 2. *There exists a uniform cpo $(V, (\pi_n)_{n\in\omega})$ satisfying the following two properties:*

1. *The following isomorphism holds in* Cpo*:*

$$V \;\cong\; \mathbb{Z} + \mathit{Loc} + 1 + (V \times V) + (V + V) + (V \to T_{S,\mathit{Ans}}V) \\ + V + T_{S,\mathit{Ans}}V \qquad (1)$$

   where $T_{S,\mathit{Ans}}V = (V \to S \to \mathit{Ans}) \to S \to \mathit{Ans}$, $S = \mathbb{N}_0 \rightharpoonup_{\mathit{fin}} V$, $\mathit{Ans} = (\mathbb{Z} + \mathit{Err})_\perp$, $\mathit{Loc} = \mathbb{N}_0 \times \overline{\omega}$, and $\mathit{Err} = 1$.
2. *Abbreviate $TV = T_{S,\mathit{Ans}}V$ and $K = V \to S \to \mathit{Ans}$. Define the following injection functions corresponding to the summands on the right-hand side of the isomorphism (1):*

$$\begin{aligned} &\mathit{in}_{\mathbb{Z}} : \mathbb{Z} \to V && \mathit{in}_{+} : V + V \to V \\ &\mathit{in}_{\mathit{Loc}} : \mathit{Loc} \to V && \mathit{in}_{\to} : (V \to TV) \to V \\ &\mathit{in}_{1} : 1 \to V && \mathit{in}_{\mu} : V \to V \\ &\mathit{in}_{\times} : V \times V \to V && \mathit{in}_{\forall} : TV \to V \end{aligned}$$

   With that notation, the functions $\pi_n : V \to V_\perp$ satisfy (and are determined by) the equations shown in Figure 2.

These two properties determine V uniquely, up to isomorphism in Cpo.

$$\begin{aligned}
\pi_0 &= \lambda v.\bot \\
\pi_{n+1}(in_{\mathbb{Z}}(m)) &= \lfloor in_{\mathbb{Z}}(m) \rfloor \\
\pi_{n+1}(in_1(*)) &= \lfloor in_1(*) \rfloor \\
\pi_{n+1}(in_{Loc}(l,\infty)) &= \lfloor in_{Loc}(l, n+1) \rfloor \\
\pi_{n+1}(in_{Loc}(l,m)) &= \lfloor in_{Loc}(l, \min(n+1,m)) \rfloor \\
\pi_{n+1}(in_{\times}(v_1,v_2)) &= \begin{cases} \lfloor in_{\times}(v_1',v_2') \rfloor & \text{if } \pi_n\, v_1 = \lfloor v_1' \rfloor \text{ and } \pi_n\, v_2 = \lfloor v_2' \rfloor \\ \bot & \text{otherwise} \end{cases} \\
\pi_{n+1}(in_{+}(\iota_i\, v)) &= \begin{cases} \lfloor in_{+}(\iota_i\, v') \rfloor & \text{if } \pi_n\, v = \lfloor v' \rfloor \\ \bot & \text{otherwise} \end{cases} \qquad (i = 1,2) \\
\pi_{n+1}(in_{\mu}\, v) &= \begin{cases} \lfloor in_{\mu}\, v' \rfloor & \text{if } \pi_n v = \lfloor v' \rfloor \\ \bot & \text{otherwise} \end{cases} \\
\pi_{n+1}(in_{\forall}\, c) &= \lfloor in_{\forall}(\pi^T_{n+1}\, c) \rfloor \\
\pi_{n+1}(in_{\rightarrow}\, f) &= \left\lfloor in_{\rightarrow} \left(\lambda v. \begin{cases} \pi^T_{n+1}\, (f\, v') & \text{if } \pi_n\, v = \lfloor v' \rfloor \\ \bot & \text{otherwise} \end{cases} \right) \right\rfloor
\end{aligned}$$

Here the functions $\pi^S_n : S \to S_\bot$ and $\pi^K_n : K \to K$ and $\pi^T_n : TV \to TV$ are defined as follows:

$$\pi^S_0 = \lambda s.\bot \qquad \pi^K_0 = \lambda k.\bot \qquad \pi^T_0 = \lambda c.\bot$$

$$\begin{aligned}
\pi^S_{n+1}(s) &= \begin{cases} \lfloor s' \rfloor & \text{if } \pi_n \circ s = \lambda l.\lfloor s'(l) \rfloor \\ \bot & \text{otherwise} \end{cases} \\
\pi^K_{n+1}(k) &= \lambda v.\lambda s. \begin{cases} k\, v'\, s' & \text{if } \pi_n\, v = \lfloor v' \rfloor \text{ and } \pi^S_{n+1}\, s = \lfloor s' \rfloor \\ \bot & \text{otherwise} \end{cases} \\
\pi^T_{n+1}(c) &= \lambda k.\lambda s. \begin{cases} c\, (\pi^K_{n+1}\, k)\, s' & \text{if } \pi^S_{n+1}\, s = \lfloor s' \rfloor \\ \bot & \text{otherwise}\,. \end{cases}
\end{aligned}$$

Fig. 2. Characterization of the projection functions $\pi_n : V \to V_\bot$

Proof (sketch). By modifying the usual projection functions, obtained from a minimal-invariant solution of (1), on arguments corresponding to locations. □

From here on, let V and $(\pi_n)_{n\in\omega}$ be as in the proposition above. We furthermore use the abbreviations, notation for injections, etc. introduced in the proposition; in particular, $TV = (V \to S \to Ans) \to S \to Ans$. Additionally, abbreviate $\lambda_l = in_{Loc}(l,\infty)$ and $\lambda^n_l = in_{Loc}(l,n)$. Let $error_{Ans} = \lfloor \iota_2 * \rfloor \in Ans$ be the "error answer" and let $error = \lambda k.\lambda s.\, error_{Ans} \in TV$ be the "error computation".

In order to model the three operations of the untyped language that involve references, we define the three functions $alloc : V \to TV$, $lookup : V \to TV$, and $assign : V \to V \to TV$. The first two of these functions are shown in the lower part of Figure 3. The third function, *assign*, is similar to *lookup*: the idea is that when one assigns a value to an approximate location, only an approximate value is inserted in the store. Notice that it would not suffice to define, e.g., $lookup(\lambda^{n+1}_l)(k)(s) = \bot$ for $l \in \text{dom}(s)$, and hence avoid mentioning the projection functions: *lookup* would then not be continuous.

We are now ready to define the untyped semantics.

For every t with $\mathrm{FV}(t) \subseteq X$, define the continuous $[\![t]\!]_X : V^X \to TV$ by induction on t:

$$
\begin{aligned}
[\![x]\!]_X\,\rho &= \eta(\rho(x)) \\
[\![\lambda x.t]\!]_X\,\rho &= \eta(\mathit{in}_{\to}(\lambda v.\ [\![t]\!]_{X,x}\,(\rho[x \mapsto v]))) \\
[\![t_1\ t_2]\!]_X\,\rho &= [\![t_1]\!]_X\,\rho \star \lambda v_1.\ [\![t_2]\!]_X\,\rho \star \lambda v_2. \begin{cases} f\ v_2 & \text{if } v_1 = \mathit{in}_{\to}\ f \\ \mathit{error} & \text{otherwise} \end{cases} \\
[\![\Lambda\alpha.t]\!]_X\,\rho &= \eta(\mathit{in}_{\forall}\ ([\![t]\!]_X\,\rho)) \\
[\![t\ [\tau]]\!]_X\,\rho &= [\![t]\!]_X\,\rho \star \lambda v. \begin{cases} c & \text{if } v = \mathit{in}_{\forall}\ c \\ \mathit{error} & \text{otherwise} \end{cases} \\
[\![\mathsf{ref}\ t]\!]_X\,\rho &= [\![t]\!]_X\,\rho \star \lambda v.\ \mathit{alloc}\,v \\
[\![!\,t]\!]_X\,\rho &= [\![t]\!]_X\,\rho \star \lambda v.\ \mathit{lookup}\ v \\
[\![t_1 := t_2]\!]_X\,\rho &= [\![t_1]\!]_X\,\rho \star \lambda v_1.\ [\![t_2]\!]_X\,\rho \star \lambda v_2.\ \mathit{assign}\ v_1\ v_2 \\
&\cdots
\end{aligned}
$$

$$
\begin{aligned}
\mathit{alloc}\,v &= \lambda k\,\lambda s.\,k\ (\lambda_{\mathit{free}(s)})\ (s[\mathit{free}(s) \mapsto v]) \\
&\quad (\text{where } \mathit{free}(s) = \min\{n \in \mathbb{N}_0 \mid n \notin \mathrm{dom}(s)\}) \\
\mathit{lookup}\ v &= \lambda k\,\lambda s. \begin{cases} k\ s(l)\ s & \text{if } v = \lambda_l \text{ and } l \in \mathrm{dom}(s) \\ k\ v'\ s & \text{if } v = \lambda_l^{n+1},\ l \in \mathrm{dom}(s), \text{ and } \pi_n(s(l)) = \lfloor v' \rfloor \\ \bot_{\mathit{Ans}} & \text{if } v = \lambda_l^{n+1},\ l \in \mathrm{dom}(s), \text{ and } \pi_n(s(l)) = \bot \\ \mathit{error}_{\mathit{Ans}} & \text{otherwise} \end{cases}
\end{aligned}
$$

Fig. 3. Untyped semantics of terms (sample cases)

Definition 3. *Let t be a term and let X be a set of term variables such that $\mathrm{FV}(t) \subseteq X$. The* untyped semantics of t with respect to X *is the continuous function $[\![t]\!]_X : V^X \to TV$ defined by induction on t in Figure 3.*

The semantics of a complete program is defined by supplying an initial continuation and the empty store:

Definition 4. *Let t be a term with no free term variables or type variables. The* program semantics of t *is the element $[\![t]\!]^{\mathrm{P}}$ of Ans defined by $[\![t]\!]^{\mathrm{P}} = [\![t]\!]_{\emptyset}\,\emptyset\,k_{\mathit{init}}\,s_{\mathit{init}}$ where $s_{\mathit{init}} \in S$ is the empty store and*

$$
k_{\mathit{init}} = \lambda v.\lambda s. \begin{cases} \lfloor \iota_1\ m \rfloor & \text{if } v = \mathit{in}_{\mathbb{Z}}(m) \\ \mathit{error}_{\mathit{Ans}} & \text{otherwise.} \end{cases}
$$

We emphasize that even though the above semantics is slightly non-standard because of the use of the projection functions in lookup and assignment, we can still use it to reason about operational behaviour: as mentioned in the Introduction an earlier adequacy proof [8] should carry over to the present setting.

4 Typed Semantics

In this section we present a "typed semantics", i.e., an interpretation of types and typed terms. As described in the introduction, types will be interpreted as world-indexed families of binary relations on the universal cpo V. Since worlds depend on semantic types, the space of semantic types is obtained by solving a recursive metric-space equation, i.e., by finding a fixed-point of a functor on metric spaces.

4.1 Ultrametric Spaces

Recall that a metric space (X, d) is 1-*bounded* if $d(x, y) \leq 1$ for all x and y in X. Let CBUlt be the category with complete, 1-bounded ultrametric spaces as objects and non-expansive (i.e., non-distance-increasing) functions as morphisms [6]. This category is cartesian closed [22]; here one needs the ultrametric inequality. The exponential $(X_1, d_1) \to (X_2, d_2)$ has the set of non-expansive functions from (X_1, d_1) to (X_2, d_2) as the underlying set, and the "sup"-metric $d_{X_1 \to X_2}$ as distance function: $d_{X_1 \to X_2}(f, g) = \sup\{d_2(f(x), g(x)) \mid x \in X_1\}$.

For a given non-empty, complete metric space, consider the function *fix* that maps every contractive operator to its unique fixed-point (which exists by Banach's fixed-point theorem). On complete ultrametric spaces, *fix* is non-expansive in the following sense.

Proposition 5. *Let $(X, d) \in \mathsf{CBUlt}$ be non-empty. For all contractive functions f and g from (X, d) to itself, $d(\mathit{fix}\ f, \mathit{fix}\ g) \leq d(f, g)$.*

4.2 The Space of Semantic Types

We now turn to constructing the space of semantic types. First, some standard definitions. For every cpo A, let $\mathit{Rel}(A)$ be the set of binary relations $R \subseteq A \times A$ on A. A relation $R \in \mathit{Rel}(A)$ is *complete* if for all chains $(a_n)_{n \in \omega}$ and $(a'_n)_{n \in \omega}$ such that $(a_n, a'_n) \in R$ for all n, also $(\sqcup_{n \in \omega} a_n, \sqcup_{n \in \omega} a'_n) \in R$. Let $\mathit{CRel}(A)$ be the set of complete relations on A. For every cpo A and every relation $R \in \mathit{Rel}(A)$, define the relation $R_\perp \in \mathit{Rel}(A_\perp)$ by $R_\perp = \{\, (\perp, \perp) \,\} \cup \{\, (\lfloor a \rfloor, \lfloor a' \rfloor) \mid (a, a') \in R \,\}$. For $R \in \mathit{Rel}(A)$ and $S \in \mathit{Rel}(B)$, let $R \to S$ be the set of continuous functions f from A to B satisfying that for all $(a, a') \in R$, $(f\ a, f\ a') \in\ S$.

On uniform cpos, we furthermore define *uniform* binary relations [1, 4]:

Definition 6. *Let $(A, (\varpi_n)_{n \in \omega})$ be a uniform cpo. A relation $R \in \mathit{Rel}(A)$ is* uniform with respect to $(\varpi_n)_{n \in \omega}$ *if $\varpi_n \in R \to R_\perp$ for all n. Let $\mathit{CURel}(A, (\varpi_n)_{n \in \omega})$ be the set of binary relations on A that are complete and uniform with respect to $(\varpi_n)_{n \in \omega}$.*

Below we define a number of metric spaces that will be used in the interpretation of types. After defining one of these metric spaces (X, d), the "distance function" d will be fixed, so we usually omit it and call X itself a metric space.

Let in the following $(A, (\varpi_n)_{n \in \omega})$ be a uniform cpo and let $\mathit{CURel}(A) = \mathit{CURel}(A, (\varpi_n)_{n \in \omega})$. First, as in Amadio [4], we obtain:

Proposition 7. *The set $\mathit{CURel}(A)$ is a complete, 1-bounded ultrametric space with the distance function given by*

$$d(R, S) = \begin{cases} 2^{-\max\{\, n \in \omega \ \mid\ \varpi_n \in R \to S_\perp \ \wedge\ \varpi_n \in S \to R_\perp \,\}} & \text{if } R \neq S \\ 0 & \text{if } R = S. \end{cases}$$

We also need metrics on "worlds" and monotone functions from worlds:

Proposition 8. *Let $(X, d) \in \mathsf{CBUlt}$. The set $\mathbb{N}_0 \rightharpoonup_{\mathit{fin}} X$ of finite maps from natural numbers to elements of X is a complete, 1-bounded ultrametric space with the distance function given by*

$$d'(\Delta, \Delta') = \begin{cases} \max\{d(\Delta(l), \Delta'(l)) \mid l \in \mathrm{dom}(\Delta)\} & \textit{if } \mathrm{dom}(\Delta) = \mathrm{dom}(\Delta') \\ 1 & \textit{otherwise.} \end{cases}$$

Definition 9. *For every $(X, d) \in \mathsf{CBUlt}$, define an "extension" ordering $\leq$ on $\mathbb{N}_0 \rightharpoonup_{\mathit{fin}} X$ by $\Delta \leq \Delta' \iff \mathrm{dom}(\Delta) \subseteq \mathrm{dom}(\Delta') \wedge \forall l \in \mathrm{dom}(\Delta).\, \Delta(l) = \Delta'(l)$.*

Proposition 10. *Let $(X, d) \in \mathsf{CBUlt}$, and let $(\mathbb{N}_0 \rightharpoonup_{\mathit{fin}} X) \to_{\mathit{mon}} \mathit{CURel}(A)$ be the set of functions ν from $\mathbb{N}_0 \rightharpoonup_{\mathit{fin}} X$ to $\mathit{CURel}(A)$ that are both non-expansive and monotone in the sense that $\Delta \leq \Delta'$ implies $\nu(\Delta) \subseteq \nu(\Delta')$. This set is a complete, 1-bounded ultrametric space with the "sup"-metric, given by*

$$d'(\nu, \nu') = \sup\{d(\nu(\Delta), \nu'(\Delta)) \mid \Delta \in \mathbb{N}_0 \rightharpoonup_{\mathit{fin}} X\}.$$

Proof (sketch). It suffices to show that the set of monotone and non-expansive functions is a closed subset of the (complete) metric space of all non-expansive functions. To that end, one needs the following property: if $R, S \in \mathit{CURel}(A)$ satisfy that $\varpi_n \in R \to S_\perp$ for all n, then $R \subseteq S$. □

In the rest of this section we do not need the extra generality of uniform cpos: recall that V is the cpo obtained from Proposition 2 and abbreviate $\mathit{CURel}(V) = \mathit{CURel}(V, (\pi_n)_{n \in \omega})$.

Proposition 11. *The operation mapping each $(X, d) \in \mathsf{CBUlt}$ to the metric space $(\mathbb{N}_0 \rightharpoonup_{\mathit{fin}} X) \to_{\mathit{mon}} \mathit{CURel}(V)$ (as given by the previous proposition) can be extended to a functor $F : \mathsf{CBUlt}^{\mathrm{op}} \to \mathsf{CBUlt}$ in the natural way.*

Given $(X, d) \in \mathsf{CBUlt}$ and $0 < \delta < 1$ one defines $\delta \cdot (X, d) \in \mathsf{CBUlt}$ with the same underlying set X but with all distances multiplied by δ.

Theorem 12. *There exists a unique (up to isomorphism) complete, 1-bounded ultrametric space $\widehat{\mathcal{T}}$ such that the following isomorphism holds in CBUlt:*

$$\widehat{\mathcal{T}} \cong \tfrac{1}{2}((\mathbb{N}_0 \rightharpoonup_{\mathit{fin}} \widehat{\mathcal{T}}) \to_{\mathit{mon}} \mathit{CURel}(V)). \tag{2}$$

Proof (sketch). By a well-known adaptation of the inverse-limit method [6, 20, 22] one can show that so-called locally contractive mixed-variance functors on CBUlt have unique fixed-points up to isomorphism. The functor F defined in the previous proposition is only locally non-expansive (i.e., non-expansive as a function on each hom-set) so we use the standard method of multiplying F with the "shrinking factor" $\delta = 1/2$, thus obtaining a locally contractive functor. □

4.3 Interpretation of Types

Let in the following $\widehat{\mathcal{T}}$ be a complete, 1-bounded ultrametric space satisfying (2), and let $App : \widehat{\mathcal{T}} \to \frac{1}{2}((\mathbb{N}_0 \rightharpoonup_{fin} \widehat{\mathcal{T}}) \to_{mon} CURel(V))$ be an isomorphism. For convenience, we use the following abbreviations (where the names $\mathcal{W}$ and $\mathcal{T}$ are intended to indicate "worlds" and "types", respectively):

$$\begin{aligned} \mathcal{W} &= \mathbb{N}_0 \rightharpoonup_{fin} \widehat{\mathcal{T}} \\ \mathcal{T} &= \mathcal{W} \to_{mon} CURel(V)\,. \end{aligned}$$

With that notation, (2) expresses that $\widehat{\mathcal{T}}$ is isomorphic to $\frac{1}{2}\mathcal{T}$. We choose $\mathcal{T}$ as our space of semantic types.

We additionally define families of relations on "states" (elements of S), "continuations" (elements of $K = V \to S \to Ans$), and "computations" (elements of TV). First, $(S, (\pi_n^S)_{n\in\omega})$ is a uniform cpo; abbreviate $CURel(S) = CURel(S, (\pi_n^S)_{n\in\omega})$. We then define

$$\mathcal{T}_S = \mathcal{W} \to CURel(S)$$

as the element of CBUlt obtained from Proposition 7 and the exponential in CBUlt. (The elements of $\mathcal{T}_S$ are non-expansive but not necessarily monotone functions.) As for continuations and computations, one observes that $(K, (\pi_n^K)_{n\in\omega})$ and $(TV, (\pi_n^T)_{n\in\omega})$ are "uniform cppos", i.e., satisfy conditions similar to those in Definition 1, but in the category of cppos and strict continuous functions (see the long version of this article for the details). Using analogues of Definition 6 and Propositions 7 and 10 we obtain $CURel(K) = CURel(K, (\pi_n^K)_{n\in\omega})$ and $CURel(TV) = CURel(TV, (\pi_n^T)_{n\in\omega})$ in CBUlt and define

$$\begin{aligned} \mathcal{T}_K &= \mathcal{W} \to_{mon} CURel(K) \\ \mathcal{T}_T &= \mathcal{W} \to_{mon} CURel(TV)\,. \end{aligned}$$

In all the ultrametric spaces we consider here, all non-zero distances have the form 2^{-m} for some m. For such ultrametric spaces, there is a useful notion of n-approximated equality of elements: the notation $x =_n y$ means that $d(x,y) \leq 2^{-n}$. The ultrametric inequality then amounts to the fact that each relation $=_n$ is transitive, and therefore an equivalence relation. The factor $1/2$ in (2) implies that worlds that are "$(n+1)$-equal" only contain "n-equal" semantic types.

To interpret types of the language as elements of $\mathcal{T}$, it remains to define a number of operators on $\mathcal{T}$ (and $\mathcal{T}_T$ and $\mathcal{T}_K$) that will be used to interpret the various type constructors of the language; these operators are shown in Figure 4. Notice that the operator *ref* is defined in terms of n-approximated equality $=_n$ on $CURel(V)$. In order to interpret the fragment of the language without recursive types, it suffices to verify that these operators are well-defined (e.g., *ref* actually maps elements of $\mathcal{T}$ into $\mathcal{T}$.) In order to interpret recursive types, however, we furthermore need to verify that the operators are non-expansive.

Lemma 13. *The operators $\times$, $+$, ref, $\to$, cont, and comp shown in the lower part of Figure 4 are well-defined and non-expansive.*

For every $\Xi \vdash \tau$, define the non-expansive $[\![\tau]\!]_\Xi : \mathcal{T}^\Xi \to \mathcal{T}$ by induction on τ:

$$\begin{aligned}
[\![\alpha]\!]_\Xi\,\varphi &= \varphi(\alpha)\\
[\![\mathsf{int}]\!]_\Xi\,\varphi &= \lambda\Delta.\,\{\,(in_{\mathbb{Z}}\,m, in_{\mathbb{Z}}\,m) \mid m \in \mathbb{Z}\,\}\\
[\![1]\!]_\Xi\,\varphi &= \lambda\Delta.\,\{\,(in_1\,*, in_1\,*)\,\}\\
[\![\tau_1 \times \tau_2]\!]_\Xi\,\varphi &= [\![\tau_1]\!]_\Xi\,\varphi \times [\![\tau_2]\!]_\Xi\,\varphi\\
[\![\tau_1 + \tau_2]\!]_\Xi\,\varphi &= [\![\tau_1]\!]_\Xi\,\varphi + [\![\tau_2]\!]_\Xi\,\varphi\\
[\![\mathsf{ref}\,\tau]\!]_\Xi\,\varphi &= \mathit{ref}([\![\tau]\!]_\Xi\,\varphi)\\
[\![\forall\alpha.\tau]\!]_\Xi\,\varphi &= \lambda\Delta.\,\{\,(in_\forall\,c, in_\forall\,c') \mid \forall\nu \in \mathcal{T}.\,(c,c') \in \mathit{comp}([\![\tau]\!]_{\Xi,\alpha}\,\varphi[\alpha \mapsto \nu])(\Delta)\,\}\\
[\![\mu\alpha.\tau]\!]_\Xi\,\varphi &= \mathit{fix}\left(\lambda\nu.\lambda\Delta.\{(in_\mu\,v, in_\mu\,v') \mid (v,v') \in [\![\tau]\!]_{\Xi,\alpha}\,\varphi[\alpha\mapsto\nu](\Delta)\}\right) \quad \text{(see Def. 14)}\\
[\![\tau_1 \to \tau_2]\!]_\Xi\,\varphi &= ([\![\tau_1]\!]_\Xi\,\varphi) \to (\mathit{comp}([\![\tau_2]\!]_\Xi\,\varphi))
\end{aligned}$$

The following operators and elements are used above:

$$\begin{aligned}
&\times : \mathcal{T} \times \mathcal{T} \to \mathcal{T} && \mathit{comp} : \mathcal{T} \to \mathcal{T}_T\\
&+ : \mathcal{T} \times \mathcal{T} \to \mathcal{T} && \mathit{cont} : \mathcal{T} \to \mathcal{T}_K\\
&\mathit{ref} : \mathcal{T} \to \mathcal{T} && \mathit{states} \in \mathcal{T}_S\\
&\to : \mathcal{T} \times \mathcal{T}_T \to \mathcal{T} && R_{Ans} \in \mathit{CRel}(Ans)
\end{aligned}$$

$$(\nu_1 \times \nu_2)(\Delta) = \{\,(in_\times(v_1,v_2), in_\times(v_1',v_2')) \mid (v_1,v_1') \in \nu_1(\Delta) \wedge (v_2,v_2') \in \nu_2(\Delta)\,\}$$

$$\begin{aligned}(\nu_1 + \nu_2)(\Delta) = {}&\{\,(in_+(\iota_1\,v_1), in_+(\iota_1\,v_1')) \mid (v_1,v_1') \in \nu_1(\Delta)\,\}\\ &\cup \{\,(in_+(\iota_2\,v_2), in_+(\iota_2\,v_2')) \mid (v_2,v_2') \in \nu_2(\Delta)\,\}\end{aligned}$$

$$\begin{aligned}\mathit{ref}(\nu)(\Delta) = {}&\{\,(\lambda_l,\lambda_l) \mid l \in \mathrm{dom}(\Delta) \wedge \forall\Delta_1 \geq \Delta.\ \mathit{App}\,(\Delta(l))\,(\Delta_1) = \nu(\Delta_1)\,\}\\ &\cup \{\,(\lambda_l^{n+1},\lambda_l^{n+1}) \mid l \in \mathrm{dom}(\Delta) \wedge \forall\Delta_1 \geq \Delta.\ \mathit{App}\,(\Delta(l))\,(\Delta_1) =_n \nu(\Delta_1)\,\}\end{aligned}$$

$$(\nu \to \xi)(\Delta) = \{\,(in_\to\,f, in_\to\,f') \mid \forall\Delta_1 \geq \Delta.\forall(v,v') \in \nu(\Delta_1)\,.(f\,v, f'\,v') \in \xi(\Delta_1)\,\}$$

$$\mathit{cont}(\nu)(\Delta) = \{(k,k') \mid \forall\Delta_1 \geq \Delta.\forall(v,v') \in \nu(\Delta_1).\forall(s,s') \in \mathit{states}(\Delta_1).(k\,v\,s, k'\,v'\,s') \in R_{Ans}\}$$

$$\begin{aligned}\mathit{comp}(\nu)(\Delta) = \{\,(c,c') \mid {}&\forall\Delta_1 \geq \Delta.\,\forall(k,k') \in \mathit{cont}(\nu)(\Delta_1).\\ &\forall(s,s') \in \mathit{states}(\Delta_1).\,(c\,k\,s, c'\,k'\,s') \in R_{Ans}\,\}\end{aligned}$$

$$\begin{aligned}\mathit{states}(\Delta) = \{\,(s,s') \mid {}&\mathrm{dom}(s) = \mathrm{dom}(s') = \mathrm{dom}(\Delta)\\ &\wedge\ \forall l \in \mathrm{dom}(\Delta).\,(s(l), s'(l)) \in \mathit{App}\,(\Delta(l))\,(\Delta)\,\}\end{aligned}$$

$$R_{Ans} = \{\,(\bot,\bot)\,\} \cup \{\,(\lfloor\iota_1\,m\rfloor, \lfloor\iota_1\,m\rfloor) \mid m \in \mathbb{Z}\,\}$$

Fig. 4. Interpretation of types

It is here, in order to show that *ref* is well-defined (and non-expansive), that we need the approximate locations λ_l^n. Suppose for the sake of argument that locations were modeled simply using a flat cpo of natural numbers, i.e., suppose that $Loc = \mathbb{N}_0$ and that $\pi_1(in_{Loc}\,l) = \lfloor in_{Loc}\,l\rfloor$ for all $l \in \mathbb{N}_0$. The definition of *ref* would then have the form $\mathit{ref}(\nu)(\Delta) = \{(in_{Loc}\,l, in_{Loc}\,l) \mid l \in \mathrm{dom}(\Delta) \wedge \dots\}$. The function $\mathit{ref}(\nu)$ from worlds to relations must be non-expansive. But assume then that $\Delta =_1 \Delta'$; then $\mathit{ref}(\nu)(\Delta) =_1 \mathit{ref}(\nu)(\Delta')$ by non-expansiveness, and hence $\mathit{ref}(\nu)(\Delta) = \mathit{ref}(\nu)(\Delta')$ since π_1 is the (lifted) identity on locations. In other words, $\mathit{ref}(\nu)$ would only depend on the "first approximation" of its argument world Δ: this can never be right, no matter what the particular definition of *ref* is.[6] This observation generalizes to variants where $\pi_n(in_{Loc}\, l) = \lfloor in_{Loc}\, l \rfloor)$ for some arbitrary finite n.

For any finite set Ξ of type variables, the set $\mathcal{T}^{\Xi}$ of functions from Ξ to $\mathcal{T}$ is a metric space with the product metric: $d'(\varphi, \varphi') = \max\{\, d(\varphi(\alpha), \varphi'(\alpha)) \mid \alpha \in \Xi \,\}$.

Definition 14. *Let τ be a type and let Ξ be a type environment such that $\Xi \vdash \tau$. The* relational interpretation of τ with respect to Ξ *is the non-expansive function $[\![\tau]\!]_{\Xi} : \mathcal{T}^{\Xi} \to \mathcal{T}$ defined by induction on τ in Figure 4. The interpretation of recursive types is by appeal to Banach's fixed-point theorem (see below).*

In more detail, the use of Banach's fixed point theorem in the interpretation of recursive types means that well-definedness of $[\![\tau]\!]_{\Xi}$ must be argued together with non-expansiveness, by induction on τ.[7] This is similar to the more familiar situation with the untyped semantics of terms presented in Section 3: there, well-definedness must be argued together with continuity because of the use of Kleene's fixed-point theorem in the interpretation of $\mathsf{fix}\, f.\lambda x.t$. The proof that $[\![\mu\alpha.\tau]\!]_{\Xi}$ is non-expansive uses Proposition 5.

We need the following substitution lemma, easily proved by induction on τ:

Lemma 15. *Let τ and τ' be types such that $\Xi, \alpha \vdash \tau$ and $\Xi \vdash \tau'$. For all $\varphi \in \mathcal{T}^{\Xi}$, $[\![\tau[\tau'/\alpha]]\!]_{\Xi}\, \varphi = [\![\tau]\!]_{\Xi,\alpha} (\varphi[\alpha \mapsto [\![\tau']\!]_{\Xi}\, \varphi])$.*

Corollary 16. $[\![\mu\alpha.\tau]\!]_{\Xi}\, \varphi = \lambda\Delta.\, \{\, (in_{\mu}\, v, in_{\mu}\, v') \mid (v, v') \in [\![\tau[\mu\alpha.\tau/\alpha]]\!]_{\Xi}\, \varphi\, \Delta \,\}$.

4.4 Interpretation of Terms

As for the interpretation of terms, we must show that the untyped meaning of a typed term is related to itself at the appropriate type. We first show that *comp* respects the operations of the monad T.

Definition 17. *For $\nu \in \mathcal{T}$ and $\xi \in \mathcal{T}_T$ and $\Delta \in \mathcal{W}$, let $\nu \overset{\Delta}{\to} \xi$ be the binary relation on functions $V \to TV$ defined by*

$$\nu \overset{\Delta}{\to} \xi = \{\, (f, f') \mid \forall \Delta_1 \geq \Delta. \forall (v, v') \in \nu(\Delta_1).\, (f\, v, f'\, v') \in \xi(\Delta_1) \,\}\,.$$

Proposition 18. *Let $\nu, \nu_1, \nu_2 \in \mathcal{T}$ and $\Delta \in \mathcal{W}$. (1) If $(v, v') \in \nu(\Delta)$, then $(\eta\, v, \eta\, v') \in comp(\nu)(\Delta)$. (2) If $(c, c') \in comp(\nu_1)(\Delta)$ and $(f, f') \in \nu_1 \overset{\Delta}{\to} comp(\nu_2)$, then $(c \star f,\, c' \star f') \in comp(\nu_2)(\Delta)$.*

Definition 19. *For every term environment $\Xi \vdash \Gamma$, every $\varphi \in \mathcal{T}^{\Xi}$, and every $\Delta \in \mathcal{W}$, let $[\![\Gamma]\!]_{\Xi}\, \varphi\, \Delta$ be the binary relation on $V^{\mathrm{dom}(\Gamma)}$ defined by*

$$[\![\Gamma]\!]_{\Xi}\, \varphi\, \Delta = \{\, (\rho, \rho') \mid \forall x \in \mathrm{dom}(\Gamma).\, (\rho(x), \rho'(x)) \in [\![\Gamma(x)]\!]_{\Xi}\, \varphi\, \Delta \,\}\,.$$

[6] In particular, the obvious definition of *ref* as $ref(\nu)(\Delta) = \{(in_{Loc}\, l, in_{Loc}\, l) \mid l \in \mathrm{dom}(\Delta) \wedge \forall \Delta_1 \geq \Delta.\ App\ (\Delta(l))\ (\Delta_1) = \nu(\Delta_1)\}$ would *not* be well-defined, since it would not be non-expansive in Δ.

[7] Non-expansiveness of $[\![\tau]\!]_{\Xi,\alpha}$ implies contractiveness of $\lambda\nu.\, \lambda\Delta.\, \{\, (in_{\mu}\, v, in_{\mu}\, v') \mid (v, v') \in [\![\tau]\!]_{\Xi,\alpha}\, \varphi[\alpha \mapsto \nu]\, (\Delta) \,\}$, as needed in the definition of $[\![\mu\alpha.\tau]\!]_{\Xi}\, \varphi$.

Definition 20. *Two typed terms $\Xi \mid \Gamma \vdash t : \tau$ and $\Xi \mid \Gamma \vdash t' : \tau$ of the same type are* semantically related*, written $\Xi \mid \Gamma \models t \sim t' : \tau$, if for all $\varphi \in \mathcal{T}^{\Xi}$, all $\Delta \in \mathcal{W}$, and all $(\rho, \rho') \in [\![\Gamma]\!]_{\Xi}\, \varphi\, \Delta$,*

$$\left([\![t]\!]_{\mathrm{dom}(\Gamma)}\, \rho, [\![t']\!]_{\mathrm{dom}(\Gamma)}\, \rho'\right) \in \mathit{comp}([\![\tau]\!]_{\Xi}\, \varphi)(\Delta)\,.$$

Theorem 21 (Fundamental Theorem). *Every typed term is semantically related to itself: if $\Xi \mid \Gamma \vdash t : \tau$, then $\Xi \mid \Gamma \models t \sim t : \tau$.*

Proof (sketch). By showing the stronger property that semantic relatedness is preserved by all the term constructs. The proof uses Proposition 18. □

Corollary 22 (Type soundness)

1. *If $\emptyset \mid \emptyset \vdash t : \tau$ is a closed term of type τ, then $[\![t]\!]_{\emptyset}\, \emptyset \neq \mathit{error}$.*
2. *If $\emptyset \mid \emptyset \vdash t : \mathsf{int}$ is a closed term of type* int*, then $[\![t]\!]^{\mathrm{p}} \neq \mathit{error}_{Ans}$.*

5 Examples

The model can be used to prove the equivalences in Section 5 of our earlier work [8]. More specifically, one can use the model to prove that some equivalences between different *functional* implementations of abstract data types are still valid in the presence of general references, and also prove some simple equivalences involving *imperative* abstract data types. (See Section 6 for more about extending the model to account properly for local state.) Here we only sketch one of these examples, as well as a "non-example": an equivalence that cannot be shown because of the existence of approximated locations in the model.

Example 23. We use the usual encoding of existential types by means of universal types [11]: $\exists\alpha.\tau = \forall\beta.(\forall\alpha.\,\tau \to \beta) \to \beta$. The type $\tau_{\mathrm{m}} = \exists\alpha.\,(1 \to \alpha) \times (\alpha \to 1) \times (\alpha \to \mathtt{int})$ can then be used to model imperative counter modules: the idea as that a value of type τ_{m} consists of some hidden type α, used to represent imperative counters, as well as three operations for creating a new counter, incrementing a counter, and reading the value of a counter, respectively.

Consider the following two module implementations, i.e., closed terms of type τ_{m}: $J = \Lambda\beta.\lambda c.\, c[\mathtt{ref\ int}]I$ and $J' = \Lambda\beta.\lambda c.\, c[\mathtt{ref\ int}]I'$ where

$$I = (\lambda x.\,\mathtt{ref}(0),\ \lambda x.\, x := {!x} + 1,\ \lambda x.\, {!x})$$
$$I' = (\lambda x.\,\mathtt{ref}(0),\ \lambda x.\, x := {!x} - 1,\ \lambda x.\, -({!x}))\,.$$

By parametricity reasoning, i.e., by exploiting the universal quantification in the interpretation of universal types, one can show that $\emptyset \mid \emptyset \models J \sim J' : \tau_{\mathrm{m}}$.

Example 24. Abbreviate $0 = \mu\alpha.\alpha$. One can show that 0 is an empty type: there are no closed values of type 0 and furthermore $[\![0]\!]_{\Xi}\, \varphi = \lambda\Delta.\emptyset$. Consider now the two terms $K = \lambda x.2$ and $K' = \lambda x.3$ of type $\mathsf{ref}\, 0 \to \mathsf{int}$. Given a standard operational semantics for the language, a simple bisimulation-style argument should suffice to show that K and K' are contextually equivalent: no reference cell can ever contain a value of type 0, and therefore neither function can ever be applied. However, the equivalence $\emptyset \mid \emptyset \models K \sim K' : \mathsf{ref}\, 0 \to \mathsf{int}$ does not hold. Briefly, the reason is the existence of approximated locations in the model.

6 Related Work

As already mentioned, the metric-space structure on uniform relations over universal domains is well-known [1, 4, 5, 10, 13]. The inverse-limit method for solving recursive domain equations was first adapted to metric spaces by America and Rutten [6]; see also Rutten [20]. For a unified account covering both domains and metric spaces, see Wagner [22]. Kripke logical relations are covered in Mitchell [14, Section 8.6] and in the references there.

Semantic (or "approximated") locations were first introduced in our earlier work [8]. That work contains an adequacy proof with respect to an operational semantics and an entirely different, quasi-syntactic interpretation of open types. Here we instead present an in some ways more natural interpretation that results from solving a recursive metric-space equation, thus obtaining a proper universe of semantic types. Open types are then interpreted in the expected way, i.e., as maps from environments of semantic types to semantic types.

The fundamental circularity between worlds and types in realizability-style possible-worlds models of polymorphism and general references was observed by Ahmed [2, p. 62] in the setting of operational semantics (and for unary relations). Rather than solve a recursive equation, her solution is to stratify worlds and types into different levels, represented by natural numbers. So-called step-indexing is used in the definition to ensure that a stratified variant of the fundamental theorem holds. These stratified worlds and types are somewhat analogous to the approximants of recursive-equation solutions that are employed in the inverse-limit method. The main advantage in "going to the limit" of the approximations and working with an actual solution (as we do here) is that approximation information is then not ubiquitous in definitions and proofs; by analogy, the only "approximation information" in our model is in the interpretation of references and in the requirement that user-supplied relations are uniform.[8]

Ahmed et al. [3] have recently (and independently) proposed a step-indexed model of a language very similar to ours, but in which worlds are defined in a more complicated way: this allows for proofs of much more advanced equivalences involving local state. We believe that our approach extends to this style of worlds and plan to investigate this further in future work: one potential advantage would be the removal of "approximation information" in definitions and equivalence proofs. We also plan to investigate local-state parameters [9]. In this article, we instead hope to have presented the fundamental ideas behind Kripke logical relations over recursively defined sets of worlds as needed for semantic modeling of parametric polymorphism, recursive types, and general references.

References

[1] Abadi, M., Plotkin, G.D.: A per model of polymorphism and recursive types. In: Proceedings of LICS, pp. 355–365 (1990)

[2] Ahmed, A.: Semantics of Types for Mutable State. PhD thesis, Princeton University (2004)

[8] In future work we plan to perform a more formal comparison.

[3] Ahmed, A., Dreyer, D., Rossberg, A.: State-dependent representation independence. In: Proceedings of POPL (to appear, 2009)

[4] Amadio, R.M.: Recursion over realizability structures. Information and Computation 91(1), 55–85 (1991)

[5] Amadio, R.M., Curien, P.-L.: Domains and Lambda-Calculi. Cambridge University Press, Cambridge (1998)

[6] America, P., Rutten, J.J.M.M.: Solving reflexive domain equations in a category of complete metric spaces. J. Comput. Syst. Sci. 39(3), 343–375 (1989)

[7] Benton, N., Leperchey, B.: Relational reasoning in a nominal semantics for storage. In: Urzyczyn, P. (ed.) TLCA 2005. LNCS, vol. 3461, pp. 86–101. Springer, Heidelberg (2005)

[8] Birkedal, L., Støvring, K., Thamsborg, J.: Relational parametricity for references and recursive types. In: Proceedings of TLDI (to appear, 2009)

[9] Bohr, N., Birkedal, L.: Relational reasoning for recursive types and references. In: Kobayashi, N. (ed.) APLAS 2006. LNCS, vol. 4279, pp. 79–96. Springer, Heidelberg (2006)

[10] Cardone, F.: Relational semantics for recursive types and bounded quantification. In: Ronchi Della Rocca, S., Ausiello, G., Dezani-Ciancaglini, M. (eds.) ICALP 1989. LNCS, vol. 372, pp. 164–178. Springer, Heidelberg (1989)

[11] Crary, K., Harper, R.: Syntactic logical relations for polymorphic and recursive types. Electronic Notes in Theoretical Computer Science 172, 259–299 (2007)

[12] Levy, P.B.: Call-by-push-value: Decomposing call-by-value and call-by-name. Higher-Order and Symbolic Computation 19(4), 377–414 (2006)

[13] MacQueen, D.B., Plotkin, G.D., Sethi, R.: An ideal model for recursive polymorphic types. Information and Control 71(1/2), 95–130 (1986)

[14] Mitchell, J.C.: Foundations for Programming Languages. MIT Press, Cambridge (1996)

[15] Moggi, E.: Notions of computation and monads. Information and Computation 93, 55–92 (1991)

[16] Petersen, R.L., Birkedal, L., Nanevski, A., Morrisett, G.: A realizability model for impredicative hoare type theory. In: Drossopoulou, S. (ed.) ESOP 2008. LNCS, vol. 4960, pp. 337–352. Springer, Heidelberg (2008)

[17] Pierce, B.C.: Types and Programming Languages. MIT Press, Cambridge (2002)

[18] Pitts, A.M.: Relational properties of domains. Information and Computation 127, 66–90 (1996)

[19] Plotkin, G.D., Power, J.: Computational effects and operations: An overview. Electronic Notes in Theoretical Computer Science 73, 149–163 (2004)

[20] Rutten, J.J.M.M.: Elements of generalized ultrametric domain theory. Theoretical Computer Science 170(1-2), 349–381 (1996)

[21] Smyth, M.B., Plotkin, G.D.: The category-theoretic solution of recursive domain equations. SIAM Journal on Computing 11(4), 761–783 (1982)

[22] Wagner, K.R.: Solving Recursive Domain Equations with Enriched Categories. PhD thesis, Carnegie Mellon University (1994)

Author Index