Luke Ong (Ed.)

LNCS 6014

ARCoSS

Foundations of Software Science and Computational Structures

13th International Conference, FOSSACS 2010
Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010
Paphos, Cyprus, March 2010, Proceedings

European Joint Conferences on Theory and Practice of Software

2010

Springer

Luke Ong (Ed.)

Foundations of Software Science and Computational Structures

13th International Conference, FOSSACS 2010
Held as Part of the Joint European Conferences
on Theory and Practice of Software, ETAPS 2010
Paphos, Cyprus, March 20-28, 2010. Proceedings

Springer

Volume Editor

Luke Ong
Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD, UK
E-mail: lo@comlab.ox.ac.uk

Library of Congress Control Number: 2010921919

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 3-642-12031-8 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-12031-2 Springer Berlin Heidelberg New York

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

Foreword

ETAPS 2010 was the 13th 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 the usual five sister conferences (CC, ESOP, FASE, FOSSACS, TACAS), 19 satellite workshops (ACCAT, ARSPA-WITS, Bytecode, CMCS, COCV, DCC, DICE, FBTC, FESCA, FOSS-AMA, GaLoP, GT-VMT, LDTA, MBT, PLACES, QAPL, SafeCert, WGT, and WRLA) and seven invited lectures (excluding those that were specific to the satellite events). The five main conferences this year received 497 submissions (including 31 tool demonstration papers), 130 of which were accepted (10 tool demos), giving an overall acceptance rate of 26%, with most of the conferences at around 24%. 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 to make of 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 toward 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 2010 was organised by the University of Cyprus 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 the Cyprus Tourism Organisation.

The organising team comprised:

General Chairs:	Tiziana Margaria and Anna Philippou
Local Chair:	George Papadopoulos
Secretariat:	Maria Kittira
Administration:	Petros Stratis
Satellite Events:	Anna Philippou
Website:	Konstantinos Kakousis.

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

Vladimiro Sassone (Southampton, Chair), Parosh Abdulla (Uppsala), Luca de Alfaro (Santa Cruz), Gilles Barthe (IMDEA-Software), Giuseppe Castagna (CNRS Paris), Marsha Chechik (Toronto), Sophia Drossopoulou (Imperial College London), Javier Esparza (TU Munich), Dimitra Giannakopoulou (CMU/NASA Ames), Andrew D. Gordon (MSR Cambridge), Rajiv Gupta (UC Riverside), Chris Hankin (Imperial College London), Holger Hermanns (Saarbrücken), Mike Hinchey (Lero, the Irish Software Engineering Research Centre), Martin Hofmann (LM Munich), Joost-Pieter Katoen (Aachen), Paul Klint (Amsterdam), Jens Knoop (Vienna), Shriram Krishnamurthi (Brown), Kim Larsen (Aalborg), Rustan Leino (MSR Redmond), Gerald Luettgen (Bamberg), Rupak Majumdar (Los Angeles), Tiziana Margaria (Potsdam), Ugo Montanari (Pisa), Oege de Moor (Oxford), Luke Ong (Oxford), Fernando Orejas (Barcelona) Catuscia Palamidessi (INRIA Paris), George Papadopoulos (Cyprus), David Rosenblum (UCL), Don Sannella (Edinburgh), João Saraiva (Minho), Michael Schwartzbach (Aarhus), Perdita Stevens (Edinburgh), Gabriele Taentzer (Marburg), and Martin Wirsing (LM Munich).

I would like to express my sincere gratitude to all of these people and organisations, the Programme Committee Chairs and members of the ETAPS conferences, the organisers of the satellite events, the speakers themselves, the many reviewers, all the participants, and Springer for agreeing to publish the ETAPS proceedings in the ARCoSS subline.

Finally, I would like to thank the organising Chair of ETAPS 2010, George Papadopoulos, for arranging for us to have ETAPS in the most beautiful surroundings of Paphos.

January 2010 Vladimiro Sassone

Preface

This volume contains the proceedings of the 13th International Conference on the Foundations of Software Science and Computational Structures (FoSSaCS) 2010, held in Paphos, Cyprus, 20–28 March 2010. FoSSaCS is an event of the Joint European Conferences on Theory and Practice of Software (ETAPS). The previous 12 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), Budapest (2008) and York (2009).

FoSSaCS presents original papers on the foundations of software science. The Programme Committee (PC) invited submissions on theories and methods to support analysis, synthesis, transformation and verification of programs and software systems.

We received 110 abstracts and 86 full paper submissions; of these, 25 were selected for presentation at FoSSaCS and inclusion in the proceedings. Also included is the abstract of a lecture, "Introduction to Decidability of Higher-Order Matching" by Colin Stirling, the FoSSaCS 2010 invited speaker. The PC members, and the external experts they consulted, wrote a total of over 460 paper reviews, and the discussion phase of the meeting involved several hundred messages. The competition was extremely keen; unfortunately many good papers could not be accepted.

I thank all the authors of papers submitted to FoSSaCS 2010. I thank also members of the PC for their sterling work, as well as the external reviewers for the expert help and reviews they provided. Throughout 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 his team. Last but not least, we would like to thank the ETAPS 2010 Local Organizing Committee (chaired by George A. Papadopoulos) and the ETAPS Steering Committee (chaired by Vladimiro Sassone) for their efficient coordination of all the activities leading up to FoSSaCS 2010.

January 2010 Luke Ong

Conference Organization

Programme Chair

Luke Ong	University of Oxford, UK

Programme Committee

Andreas Abel	Ludwig Maximilians University, Munich, Germany
Christel Baier	University of Dresden, Germany
Patrick Baillot	ENS Lyon, France
Mikołaj Bojańczyk	University of Warsaw, Poland
Patricia Bouyer	ENS Cachan, France
Krishnendu Chatterjee	Institute of Science and Technology, Austria
Hubert Comon	ENS Cachan, France
Thierry Coquand	Göteborg University, Sweden
Herman Geuvers	Radboud University, Nijmegen, The Netherlands
Masahito Hasegawa	Kyoto University, Japan
Ranko Lazic	University of Warwick, UK
John Longley	University of Edinburgh, UK
Carsten Lutz	University of Bremen, Germany
Guy McCusker	University of Bath, UK
Angelo Montanari	University of Udine, Italy
Markus Müller-Olm	University of Münster, Germany
Rocco De Nicola	University of Florence, Italy
Luke Ong (Chair)	University of Oxford, UK
Jens Palsberg	University of California at Los Angeles, USA
Dusko Pavlovic	Kestrel Institute, USA
Benjamin Pierce	University of Pennsylvania, USA
Alexander Rabinovich	University of Tel Aviv, Israel
Jan Rutten	CWI and Free University Amsterdam, The Netherlands
Makoto Tatsuta	National Institute of Informatics, Tokyo, Japan

External Reviewers

Lucia Acciai
Luca Aceto
Klaus Aehlig
Jonathan Aldrich
Wladimir Araujo
Robert Atkey
Roland Axelsson
Lennart Beringer
Marco Bernardo
Nathalie Bertrand
Laura Bocchi
Chiara Bodei
Aaron Bohannon
Udi Boker

Benedikt Bollig
Filippo Bonchi
Marcello Bonsangue
Michele Boreale
Tomas Brazdil
Davide Bresolin
Thomas Brihaye
Christopher Broadbent
Dumitru Potop Butucaru
Luis Caires
Venanzio Capretta
Alberto Casagrande
Bor-Yuh Evan Chang
Adam Chlipala
Frank Ciesinski
Corina Cirstea
Dave Clarke
Matthew Collinson
Adriana Compagnoni
Pierluigi Crescenzi
Giovanna D'Agostino
Morten Dahl
Stéphanie Delaune
Giorgio Delzanno
Pierre-Malo Deniélou
josee Desharnais
Mariangiola Dezani
Laurent Doyen
Peter Dybjer
Rachid Echahed
Thomas Ehrhard
Alain Finkel
Marco Faella
Xinyu Feng
Wan Fokkink
Vojteich Forejt
Luca Fossati
Cédric Fournet
Massimo Franceschet
Oliver Friedmann
Andrew Gacek
Fabio Gadducci
Didier Galmiche
Pierre Ganty
Wouter Gelade
Raffaella Gentilini
Elena Giachino
Hugo Gimbert
Valentin Goranko
Daniele Gorla
Marcus Groesser
Stefan Haar
Matthew Hague
Ichiro Hasuo
Tobias Heindel
Angela Hennessy
Holger Hermanns
Thomas Hildebrandt
Florian Horn
Tomasz Idziaszek
Bart Jacobs
Petr Jancar
Alan Jeffrey
Barbara Jobstmann
Tomasz Jurdzinski
Antonios Kalampakas
Shin-ya Katsumata
Joachim Klein
Bartek Klin
Sascha Klueppelholz
Eryk Kopczynski
Stephane Kreutzer
Clemens Kupke
Alexander Kurz
Barbara König
Anna Labella
Jim Laird
Peter Lammich
Martin Lange
Francois Laroussinie
Slawomir Lasota
Olivier Laurent
Marina Lenisa
Paul Levy
Hans-Wolfgang Loidl
William Lovas
Etienne Lozes
Christoph Lüth
Christof Löding
Radu Mardare

Nicolas Markey
Mieke Massink
Ralph Matthes
James McKinna
Catherine Meadows
Paul-Andre Mellies
Massimo Merro
Stephan Merz
Jakub Michaliszyn
Marino Miculan
Michael Mislove
Larry Moss
Mohammad Reza Mousavi
Aniello Murano
Andrzej Murawski
Keiko Nakata
Koji Nakazawa
Monica Nesi
Uwe Nestmann
Martin R. Neuhußer
Milad Niqui
Damian Niwinski
David Nowak
Vincent van Oostrom
Karol Ostrovsky
Joel Ouaknine
Damien Pous
Alessandra Palmigiano
Prakash Panangaden
Mimmo Parente
Pawel Parys
Dirk Pattinson
Levy Paul
Jan Peleska
Detlef Plump
Arnd Poetzsch-Heffter
John Power
Vinayak Prabhu
Gabriele Puppis
Femke van Raamsdonk
Sasa Radomirovic
Vishwanath Raman
Jason Reed
Alejandro Russo
Pietro Sala
Davide Sangiorgi
Arnaud Sangnier
Mor Savas
Zdenek Sawa
Sylvain Schmitz
Lutz Schrder
Jakob Grue Simonsen
Alex Simpson
Geoffrey Smith
Pawel Sobocinski
Ana Sokolova
Marielle Stoelinga
Pierre-Yves Strub
Aaron Stump
Grégoire Sutre
Luca Tesei
Salvatore La Torre
Ashutosh Trivedi
Josef Urban
Lionel Vaux
Nicola Vitacolonna
Dimitrios Vytiniotis
Daniel Wagner
Richard Warburton
Alexander Wenner
Freek Wiedijk
James Worrell
Hans Zantema
Marc Zeitoun

Table of Contents

Introduction to Decidability of Higher-Order Matching

Colin Stirling

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

Abstract. Higher-order unification is the problem given an equation $t = u$ containing free variables is there a solution substitution θ such that $t\theta$ and $u\theta$ have the same normal form? The terms t and u are from the simply typed lambda calculus and the same normal form is with respect to $\beta\eta$-equivalence. Higher-order matching is the particular instance when the term u is closed; can t be pattern matched to u? Although higher-order unification is undecidable, higher-order matching was conjectured to be decidable by Huet [2]. Decidability was shown in [7] via a game-theoretic analysis of β-reduction when component terms are in η-long normal form.

In the talk we outline the proof of decidability. Besides the use of games to understand β-reduction, we also emphasize how tree automata can recognize terms of simply typed lambda calculus as developed in [1, 3–6].

References

1. 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)
2. Huet, G.: Rèsolution d'èquations dans les langages d'ordre 1, 2, ... ω. Thèse de doctorat d'ètat, Universitè Paris VII (1976)
3. Ong, C.-H.L.: On model-checking trees generated by higher-order recursion schemes. In: Procs. LICS 2006, pp. 81–90 (2006)
4. Ong, C.-H.L., Tzevelekos: Functional reachability. In: Procs. LICS 2009, pp. 286–295 (2009)
5. Stirling, C.: Higher-order matching, games and automata. In: Procs. LICS 2007, pp. 326–335 (2007)
6. Stirling, C.: Dependency tree automata. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 92–106. Springer, Heidelberg (2009)
7. Stirling, C.: Decidability of higher-order matching. Logical Methods in Computer Science 5(3:2), 1–52 (2009)

A Semantic Foundation for Hidden State

Jan Schwinghammer[1], Hongseok Yang[2], Lars Birkedal[3], François Pottier[4], and Bernhard Reus[5]

[1] Saarland Univ.
[2] Queen Mary Univ. of London
[3] IT Univ. of Copenhagen
[4] INRIA
[5] Univ. of Sussex

Abstract. We present the first complete soundness proof of the anti-frame rule, a recently proposed proof rule for capturing information hiding in the presence of higher-order store. Our proof involves solving a non-trivial recursive domain equation, and it helps identify some of the key ingredients for soundness.

1 Introduction

Information hiding, or *hidden state*, is one of the key design principles used by programmers in order to control the complexity of large-scale software systems. Being able to exploit this principle in the formal setting of a program logic could represent an important step towards the development of modular, scalable program verification techniques.

The idea is that an object (or function, or module) need not reveal in its interface the fact that it owns and maintains a private, mutable data structure. Hiding this internal invariant from the client has several beneficial effects. First, the complexity of the object's specification is slightly decreased. More importantly, the client is relieved from the need to thread the object's invariant through its own code. In particular, when an object has multiple clients, they are freed from the need to cooperate with one another in threading this invariant. Last, by hiding its internal state, the object escapes the restrictions on aliasing and ownership that are normally imposed to objects with mutable state.

It is worth emphasizing that *hiding* and *abstraction* (as studied, for instance, in separation logic [1–3]) are distinct mechanisms, which may co-exist within a single program logic.[1]

The recently proposed *anti-frame* proof rule [4] enables hiding in the presence of higher-order store (i.e., memory cells containing procedures or code fragments). In this paper, we study the semantic foundation of the anti-frame rule,

[1] Abstraction is often implemented in terms of assertion variables (called abstract predicates by Parkinson) that describe the private data structures of an object. These variables are exposed to a client, but their definitions are not, so that the object's internals are presented to the client in an abstract form. Hiding, on the other hand, conceals the object's internals completely.

and give the first complete soundness proof for it. Our proof involves the solution of an intricate recursive domain equation, and it helps identify some of the key ingredients for soundness.

Information hiding with frame and anti-frame rules. Our results push the frontier of recent logic-based approaches to information hiding. These approaches adopt a standard semantics of the programming language, and deal with information hiding on a logical basis, by extending a Hoare calculus with special proof rules. These usually take the form of *frame rules* that allow the implementation of an object to ignore (hence implicitly preserve) some of the invariants provided by the context, and of *anti-frame rules*, which allow an object to hide its internal invariant from the context [4–7].

In its simplest form, the frame rule [5] states that invariants R can be added to valid triples: if $\{P\}C\{Q\}$ is valid, then so is $\{P * R\}C\{Q * R\}$, where the separating conjunction $P * R$ indicates that P and R govern disjoint regions of the heap. In subsequent developments, the rule was extended to handle higher-order procedures [6, 7] and higher-order store [8, 9]. Moreover, it was argued that both extensions of the rule support information hiding: they allow one to hide the invariant of a module [6] and to prove properties of clients, as long as the module is understood in continuation-passing style.

Thorough semantic analyses were required to determine the conditions under which these extensions of the frame rule are sound. Indeed, the soundness of these rules raises subtle issues. For instance, the frame rule for higher-order procedures turns out to be inconsistent with the conjunction rule, a standard rule of Hoare logic [6, 7]. Furthermore, seemingly innocent variants of the frame rule for higher-order store have been shown unsound [9, 10].

In the most recent development in this line of research, Pottier [4] proposed an anti-frame rule, which expresses the information hiding aspect of an object directly, instead of in continuation-passing style. Besides giving several extensive examples of how the anti-frame rule supports hidden state, Pottier argued that the anti-frame rule is sound by *sketching* a plausible syntactic argument. This argument, however, relied on several non-trivial assumptions about the existence of certain recursively defined types and recursively defined operations over types.

In this paper, we systematically study the semantic foundation of hidden state, as captured by frame and anti-frame rules, in the presence of higher-order store. In particular, we describe our soundness proof of a program logic that has both frame and anti-frame rules.

Overview of the technical development. The anti-frame rule was originally proposed in an expressive type system for an ML-like language [4]. In the context of separation logic, it becomes an inference rule for deriving Hoare triples. A slightly simplified version of it takes the following form:

$$\frac{\{P \otimes R\}C\{(Q \otimes R) * R\}}{\{P\}C\{Q\}}$$

Recall that the separating conjunction $P' * Q'$ holds of a heap when the heap can be split into two sub-heaps that respectively satisfy P' and Q'. In order to specify properties of stored code, we allow assertions to contain nested triples [9], and introduce a $\otimes$ operator, whose meaning is roughly the following: $P' \otimes R'$ denotes a version of P' where R' has been $*$-conjoined with the pre- and post-conditions of every triple, including deeply nested triples.

In the anti-frame rule above, the code C can be thought of as allocating and initializing an object. The assertion R describes an internal invariant of this object, which one wishes to hide. The conjunct $- * R$ in the post-condition of the premise ensures that the invariant R is established by C, so R holds initially. The two occurrences of $- \otimes R$ in the premise guarantee that every triple that appears in the premise has R as pre- and post-conditions, so every interaction between the object and the outside preserves R. The invariant R does not appear in the conclusion of the rule, so one reasons about the rest of the program just as if the object had no internal state.

Our soundness proof of the anti-frame rule is based on two key components. The first is a new interpretation of Hoare triples, which explicates the universal and existential quantifications that are implicit in the anti-frame rule. Let $P \circ R$ abbreviate $(P \otimes R) * R$. Roughly speaking, in our interpretation, a triple $\{P\} C \{Q\}$ is valid if, for all invariants R, the triple

$$\{P \circ R\}\, C\, \{\exists R'.\ Q \circ (R \circ R')\} \tag{1}$$

holds in the standard interpretation of triples. Pottier [4] showed how the anti-frame rule allows encoding ML-like weak references in terms of strong references. Readers who are familiar with models of ML references (see, e.g., [11]) may thus find the above interpretation natural. Roughly speaking, the code C has a function type $P \to Q$, whose interpretation is: “for all worlds R, if C is given an argument of type P in world R, then, for some future world $R \circ R'$ (an extension of R), C returns a result of type Q in world $R \circ R'$.”

The second element in our soundness proof is a formalization of the above intuition. Our interpretation of assertions is parameterized by a set W of *worlds*, or *invariants*, so that, semantically, an *assertion* is a function $W \to \mathcal{P}(\mathit{Heap})$ from worlds to (certain) sets of heaps. Corresponding to $R \circ R'$ in (1), there is a semantic operation $\circ$ on W that lets us combine two invariants. This operation induces a preorder on invariants, whereby $R \circ R'$ is greater than (i.e., a future world of) R.

In order to prove that the anti-frame rule is sound, we require assertions P to be monotonic with respect to the preorder on invariants: that is, $P(R)$ must be a subset of $P(R \circ R')$, for all P, R and R'. This lets us relate an assertion P at two different invariants R_0 and R_1, by first exhibiting an upper bound R_2 of R_0 and R_1 and then using the monotonicity to conclude $P(R_i) \subseteq P(R_2)$ for $i \in \{0, 1\}$. This forms an important step of our soundness proof.

In order to present our soundness proof as abstractly and elegantly as we can, we begin with an axiomatization of worlds and world composition, that is, we state a number of requirements that worlds should satisfy (Section 2). This allows us to define the interpretation of triples and establish some of its key properties in an abstract setting (Section 3).

In Sections 4 and 5, we move to a more concrete setting and present a small imperative programming language that features higher-order store, in the form of storable commands. We equip it with a proof system, which features nested Hoare triples, frame rules, and anti-frame rules. As in Pottier's original setting, in this system, it is desirable that every assertion R be allowed to play the role of an invariant. As a consequence, in this concrete instance, the set of invariants W should be isomorphic to the semantic domain of assertions.

In summary, for this instance the requirements that we have described above amount to the following non-standard recursive domain equation:

$$Assert \;\cong\; Assert \to_m \mathcal{P}(Heap). \tag{2}$$

The subscript $-_m$ indicates that we consider only the subset of monotonic functions. By restricting the codomain to subsets of *Heap* that satisfy particular conditions, and by further restricting the function space, we can find a solution to (a variant of) equation (2) in a category of complete metric spaces. However, because monotonicity is defined in terms of the ordering over assertions, which itself is defined in terms of the operation of composition $\circ$ on W (recall that W and *Assert* are isomorphic), we cannot do this using "off-the-shelf" techniques for solving recursive domain equations, like those of Rutten [12] or Birkedal et al. [13]. Instead, we obtain a solution by explicitly constructing the inverse limit of an appropriately chosen sequence of approximations to (2). We discuss these challenges and our solution in more detail in Section 4.

Contributions. In summary, our main contributions are the following:

– We highlight which semantic ingredients are critical in establishing the validity of the frame and anti-frame rules. We hope that this will lead to increased understanding, and expect that these ingredients can be used as building blocks in the soundness proofs of future logics with information hiding principles.
– We give a proof of the soundness of a program logic that includes frame and anti-frame rules for higher-order store.

For space reasons, many proofs are omitted; some can be found in the full version of this paper [14].

2 Semantic Setup

In this section, we describe semantic ingredients that can be used to validate (certain types of) anti-frame and frame rules.

Programming language. Our assumptions on the semantics of the programming language are fairly standard. We assume that there is a (pointed, chain-complete partially ordered) set of heaps, *Heap*, and that commands either diverge, terminate successfully, or fault, i.e., that $Com = Heap \multimap (Heap \oplus \{error\}_\perp)$ is the set of (strict continuous) functions into *Heap* with an error element adjoined. We also assume that there exists a family of projection functions $(\pi_k : \mathit{Heap} \multimap \mathit{Heap})_{k\in\mathbb{N}}$. The images of these projection functions must contain only finite elements[2] and the projection functions must satisfy the following conditions:

- $\bot = \pi_0(h) \sqsubseteq \ldots \sqsubseteq \pi_k(h) \sqsubseteq \pi_{k+1}(h) \sqsubseteq \ldots \sqsubseteq h$ for all $h \in \mathit{Heap}$, i.e., the π_k's form an increasing chain of approximations of the identity on *Heap*;
- $\pi_j \circ \pi_k = \pi_{\min\{j,k\}}$ for all j, k; in particular, every π_k is idempotent;
- $\bigsqcup_k \pi_k(h) = h$, i.e., every heap is the limit of its approximations.

For example, these conditions hold if *Heap* is an SFP domain (e.g., [15]) with a particular choice of projections. Finally, we assume a partial commutative associative operation $h \cdot h'$, which intuitively lets us combine heaps with disjoint locations and is compatible with the projections: $\pi_k(h \cdot h') = \pi_k(h) \cdot \pi_k(h')$.

We write $\mathbb{N}_\infty$ for the natural numbers extended with ∞, with $\infty + k = k + \infty = \infty$. We define $\pi_\infty = \mathit{id}$ and $2^{-\infty} = 0$. The *rank of* h, written $\mathit{rnk}(h)$, is the least $k \in \mathbb{N}_\infty$ such that $\pi_k(h) = h$.

Uniformity and distance. Our program logics concern properties of heaps that are closed under the projection functions π_k. We write *UAdm* for the set of admissible[3] subsets $p \subseteq \mathit{Heap}$ that are *uniform*: for any $k \in \mathbb{N}$, if $h \in p$ then $\pi_k(h) \in p$. Using the heap combination operation, we define separating conjunction $p * q$ for $p, q \in \mathit{UAdm}$ in the usual way: $h \in p * q$ iff $h = h_1 \cdot h_2$ for some $h_1 \in p$ and $h_2 \in q$. We assume that $p * q$ is in *UAdm*.[4]

Uniformity gives rise to a notion of distance between (or, similarity of) properties of heaps. More precisely, writing $\pi_k(p)$ for the image of p under the projection π_k, the function $d(p, q) = 2^{-\sup\{k\in\mathbb{N}_\infty \mid \pi_k(p)=\pi_k(q)\}}$ defines a notion of distance on *UAdm*. This function satisfies the requirements of a *1-bounded ultrametric*: that is, d takes real values in the interval $[0, 1]$, is symmetric, is such that $d(p, q) = 0$ holds iff $p = q$, and satisfies $d(p, q) \leq \max\{d(p, r), d(r, q)\}$ for all $p, q, r \in \mathit{UAdm}$. With respect to this metric, *UAdm* is *complete* in the usual sense that every Cauchy sequence has a limit. The metric and this completeness result of *UAdm* make it possible to model recursively defined assertions using the Banach fixed point theorem.

Worlds and assertions. Our semantics of assertions is defined in the category *CBUlt* of complete 1-bounded ultrametric spaces and non-expansive functions. This means that every semantic domain involved in the semantics has an appropriate notion of distance and that every function is non-expansive, i.e., the distance between two outputs is no greater than the distance between the two corresponding inputs.

The main ingredients necessary for validating forms of anti-frame and frame rules are a set of possible worlds, and an interpretation of the worlds as assertions. Thus, we require:

[2] An element d in a cpo D is finite iff for all chains $\{d_n\}_{n\in\omega}$ in D, $d \sqsubseteq \bigsqcup_{n\in\omega} d_n$ implies that $d \sqsubseteq d_n$ for some n.

[3] The admissibility of p means that p is closed under limits of chains and contains $\bot$.

[4] This assumption holds when *Heap* is constructed in a standard way in terms of finite partial functions or records, just like our model in Sections 4 and 5.

1. A monoid $(W, e, \circ)$ of *worlds,* or *invariants,* where W is an object in *CBUlt* and the operation $\circ$ is non-expansive with respect to this metric. The monoid structure induces a preorder $\sqsubseteq$ on W, by $w \sqsubseteq w' \Leftrightarrow \exists w_0 \in W. w' = w \circ w_0$. Note that $\circ$ is in general not commutative, and that w' is obtained by extending w on the *right.* Using this preorder, we define a domain
$$Assert \stackrel{def}{=} (\tfrac{1}{2} \cdot W) \to_m UAdm.$$
for assertions. Here, $\frac{1}{2} \cdot W$ denotes the scaling of the distance function on W by $1/2$, and the function space consists of the non-expansive *monotone* functions. Assertions are thus parameterized by worlds, and this parameterization satisfies two conditions. The first condition is *contractiveness*, meaning that the distance between worlds gets reduced when they are used in an assertion: $d(p(w_0), p(w_1)) \le d(w_0, w_1)/2$ for all $p \in Assert$ and all $w_0, w_1 \in W$. In the definition, contractiveness is formalized in two steps, first by scaling down the distance of worlds by $1/2$ and then by stipulating that assertions should preserve this scaled-down distance (i.e., be non-expansive).

   The second condition is monotonicity: $p(w) \subseteq p(w \circ w_0)$ holds for all $p \in Assert$ and all $w, w_0 \in W$. Here, w_0 can be thought of as an invariant that is hidden in world w and revealed in world $w \circ w_0$. If some heap h satisfies the assertion p while w_0 is hidden, then h still satisfies p after w_0 is revealed. Intuitively, because the commands stored in the heap h do not know about the invariant w_0, they must preserve it.
2. A non-expansive *coercion* function $i : W \to Assert$, which offers a way of interpreting worlds as assertions.

   We do not in general require $W \cong Assert$: a one-way coercion is sufficient for our purposes. In fact, it is possible to define instances of our framework where W is strictly "smaller" than *Assert*. Such a restriction, where not every assertion can play the role of a hidden invariant, can be exploited to establish the soundness of stronger versions of anti-frame and frame rules than the ones in the literature [4, 9]. For details, see Appendix D of the full version of this paper [14].

For the moment, we simply assume that the above ingredients are provided. In Section 4, we actually construct a particular set of worlds, together with an appropriate coercion function from worlds to assertions. In this particular case, $W \cong Assert$ holds.

The parameterization of assertions by a monoid W has the interesting consequence that the following $\otimes$ operator, from $Assert \times W$ to $Assert$:
$$(p \otimes w) \stackrel{def}{=} \lambda w_0 . p\,(w \circ w_0)$$
is an action of this monoid over assertions, that is, it satisfies $p \otimes e = p$ and $(p \otimes w) \otimes w_0 = p \otimes (w \circ w_0)$. These are the semantic analogues of two of the distribution axioms in Pottier's type system [4], and are also included in the logic of Section 5.

Healthiness conditions. The monoid of worlds and the coercion function must satisfy two further compatibility conditions. To express these, we use the abbreviation $p \circ w \overset{def}{=} p \otimes w * i(w)$, where $*$ denotes the pointwise lifting of the separating conjunction on $UAdm$ to $Assert$. The first condition is:

Condition 1. Coercions preserve $- \circ w_0$: $\forall w, w_0 \in W.\ i(w \circ w_0) = i(w) \circ w_0$.

This condition lets us explain the extension of one invariant w by a second invariant w_0 in terms of assertions. By unfolding the definition, we see that $i(w \circ w_0)$ is the assertion obtained by $*$-conjoining the assertion $i(w_0)$ to $i(w)$, and additionally ensuring that all computations described by the latter also preserve the invariant w_0.

The asymmetric nature of Condition 1 indicates that we cannot in general expect the monoid to be commutative. Instead, we require a weaker property: the existence of commutative pairs.

Definition 1 (Commutative pair). *Let w_0, w_1, a_0 and a_1 be worlds. The pair (a_0, a_1) is a* commutative pair *for (w_0, w_1) iff (1) $w_0 \circ a_1 = w_1 \circ a_0$, (2) $i(w_0) \otimes a_1 = i(a_0)$ and (3) $i(w_1) \otimes a_0 = i(a_1)$.*

If (a_0, a_1) is a commutative pair for (w_0, w_1) then $w_0 \circ a_1 = w_1 \circ a_0$ provides an upper bound of w_0 and w_1 with respect to the extension order $\sqsubseteq$. Intuitively, we can "merge" two invariants, ensuring that all computations described in the first invariant preserve the second invariant, and vice versa.

$$\begin{tikzcd}
 & \cdot & \\
\cdot \arrow[ur, "a_1"] & & \cdot \arrow[ul, "a_0"'] \\
 & \cdot \arrow[ul, "w_0"] \arrow[ur, "w_1"'] &
\end{tikzcd}$$

Condition 2. Every pair (w_0, w_1) of worlds has a commutative pair.

Pottier's *revelation lemma* [4], which forms the core of his sketch of a syntactic soundness argument for his anti-frame rule, assumes the existence of commutative pairs. Commutative pairs play a similarly important role in the semantic soundness proofs below. The model described in Section 4 gives a rigorous justification for their existence.

As a consequence of Condition 1 and of the fact that $\otimes$ is a monoid action, we have the following lemma:

Lemma 2. *For all $p \in Assert$ and all $w, w_0 \in W$, $(p \circ w) \circ w_0 = p \circ (w \circ w_0)$.*

3 Semantic Triples, Anti-frame Rule and Frame Rules

In this section we consider the soundness of specific versions of anti-frame and frame rules, based on the semantic setting described above. For a command $c \in Com$, let $\pi_k(c)$ be the command defined by $\pi_k(c)(h) = error$ if $c(\pi_k(h)) = error$, and by $\pi_k(c)(h) = \pi_k(c(\pi_k(h)))$ if $c(\pi_k(h)) \in Heap$. Note that $\pi_\infty(c) = c$.

Definition 3. *Let tri be the ternary predicate on $Assert \times Com \times Assert$ such that $tri(p, c, q)$ holds iff*

$$\forall u \in UAdm. \forall h \in p(e) * u.\ \ c(h) \in \mathrm{Ad}(\textstyle\bigcup_w (q \circ w)(e) * u),$$

where $\mathrm{Ad}(-)$ *is the admissible downward closure.*

$$\textsc{Anti-frame}\ \frac{\models \{p \otimes w_0\} c \{q \circ w_0\}}{\models \{p\} c \{q\}} \qquad \textsc{Deep-frame}\ \frac{\models \{p\} c \{q\}}{\models \{p \circ w_0\} c \{q \circ w_0\}} \qquad \textsc{Shallow-frame}\ \frac{}{\{p\} c \{q\} \models \{p * i(w_0)\} c \{q * i(w_0)\}}$$

Fig. 1. Semantic versions of a basic form of anti-frame and frame rules

This definition deserves some explanation. First, the universal quantification over $u \in \mathit{UAdm}$ in the definition of $\mathit{tri}(p, c, q)$ "bakes in" the first-order frame rule, i.e., $\mathit{tri}(p, c, q)$ is only true of commands c that validate the first-order frame rule. Next, the existential quantification (union) over worlds $w \in W$ achieves the hiding of state from the post-condition of triples, as expressed by anti-frame rules. Because uniform admissible sets are not closed under arbitrary unions, we take the admissible downward closure. Technically, this makes sense because we assume commands are continuous and because we consider partial correctness only. We view the post-condition $\mathrm{Ad}(\bigcup_w (q \circ w)(e) * u) \subseteq \mathit{Heap}$ as a subset of $\mathit{Heap} \oplus \{\mathit{error}\}_\perp$ in the evident way. In particular, this means that $\mathit{tri}(p, c, q)$ is only true of commands c that do not fault for states in the pre-condition. This definition does not "bake in" monotonicity w.r.t. invariants (worlds): p and $q \circ w$ are "closed" just by applying them to the empty world. This is rectified in the following definition of validity:

Definition 4 (Validity). *A (semantic) triple* $\{p\}c\{q\}$ holds with respect to w and $k \in \mathbb{N}_\infty$, *which we write* $w \models_k \{p\}c\{q\}$, *iff* $\mathit{tri}(p \circ (w \circ w_0),\ \pi_k(c),\ q \circ (w \circ w_0))$ *holds for all* $w_0 \in W$. *We sometimes omit the index when* $k = \infty$.

As a consequence of the quantification over worlds w_0 in this definition, the validity of a triple is monotonic: if $w \models_k \{p\}c\{q\}$ and $w \sqsubseteq w'$ then $w' \models_k \{p\}c\{q\}$. The approximate validity (i.e., the case where $k \neq \infty$) is used when considering *nested* triples. Recall that assertions are the *contractive* monotone functions from W to *UAdm*.[5] Because nested triples will be interpreted as elements in *Assert*, they must be contractive. The approximations will allow us to satisfy this requirement. We write $\models \{p\}c\{q\}$ to mean that $w \models_k \{p\}c\{q\}$ for all k, w. Also, we write $\{p\}c\{q\} \models \{p'\}c'\{d'\}$ to mean that $w \models_k \{p\}c\{q\} \Rightarrow w \models_k \{p'\}c'\{q'\}$ holds for all w, k.

We are now ready to describe semantic versions of examples of anti-frame and frame rules and to prove their soundness. The semantic rules are given in Fig. 1, where the first two rules should be understood as the implication from the premise to the conclusion.

Our first lemma is a consequence of the monotonicity of assertions.

Lemma 5. *For all* w, *we have that (1)* $\mathit{tri}(p \otimes w, c, q) \Rightarrow \mathit{tri}(p, c, q)$ *and (2)* $\mathit{tri}(p, c, q \circ w) \Rightarrow \mathit{tri}(p, c, q)$.

Proof. For the first implication, suppose that $u \in \mathit{UAdm}$ and $h \in p(e) * u$. By the monotonicity of p and by $e \sqsubseteq w$ we obtain $p(e) \subseteq p(w) = p(w \circ e) = (p \otimes w)(e)$.

[5] More precisely, they are the non-expansive monotone functions from $\frac{1}{2} \cdot W$ to *UAdm*.

Therefore, $h \in (p \otimes w)(e) * u$. The result now follows from the assumption that $\mathit{tri}(p \otimes w, c, q)$ holds.

For the second implication, suppose again that $u \in \mathit{UAdm}$ and $h \in p(e) * u$. We must show that $c(h) \in \mathrm{Ad}(\bigcup_{w'}(q \circ w')(e) * u)$. From the assumption that $\mathit{tri}(p, c, q \circ w)$ holds we obtain $c(h) \in \mathrm{Ad}(\bigcup_{w''}((q \circ w) \circ w'')(e) * u)$. By Lemma 2,

$$\textstyle\bigcup_{w''}((q \circ w) \circ w'')(e) \;=\; \bigcup_{w''}(q \circ (w \circ w''))(e) \;\subseteq\; \bigcup_{w'}(q \circ w')(e).$$

The result then follows from the monotonicity of $*$ and the monotonicity of the closure operation $\mathrm{Ad}(\cdot)$. □

The next lemma amounts to gluing two commutative pair diagrams together (along a_0 there). Its proof involves the associativity of $\circ$ as well as Condition 1.

Lemma 6. *If (a_0, a_1) is a commutative pair for (w_0, w_1) and (b_0, a_2) is a commutative pair for (a_0, w_2) then $(b_0, a_1 \circ a_2)$ is a commutative pair for $(w_0, w_1 \circ w_2)$.*

The following proposition combines Lemmas 5 and 6, and relates the validity of two triples in our anti-frame rule.

Proposition 7. *For all worlds (w_0, w), if (a_0, a) is a commutative pair for (w_0, w), then $a \models_k \{p \otimes w_0\}c\{q \circ w_0\}$ implies $w \models_k \{p\}c\{q\}$.*

Proof. We need to show that $w \models_k \{p\}c\{q\}$, which by definition means that for all w_1, letting $w_2 = w \circ w_1$ and $d = \pi_k(c)$,

$$\mathit{tri}(p \circ w_2,\, d,\, q \circ w_2). \tag{3}$$

By Condition 2, there exists a commutative pair (b_0, a_1) for (a_0, w_1). Let $b_2 = a \circ a_1$. By Lemma 6, (b_0, b_2) is a commutative pair for $(w_0, w \circ w_1) = (w_0, w_2)$. In particular, we have $w_0 \circ b_2 = w_2 \circ b_0$ and $i(b_2) = i(w_2) \otimes b_0$. The assumed triple implies $\mathit{tri}((p \otimes w_0) \circ b_2,\, d,\, (q \circ w_0) \circ b_2)$. Thus,

$$\mathit{tri}((p \circ w_2) \otimes b_0,\, d,\, (q \circ w_2) \circ b_0) \tag{4}$$

follows, using the following equalities:

$$\begin{aligned}
(p \otimes w_0) \circ b_2 \;&=\; p \otimes (w_0 \circ b_2) * i(b_2) \;=\; p \otimes (w_2 \circ b_0) * i(w_2) \otimes b_0 \\
&=\; (p \otimes w_2) \otimes b_0 * i(w_2) \otimes b_0 \;=\; (p \otimes w_2 * i(w_2)) \otimes b_0 \\
&=\; (p \circ w_2) \otimes b_0, \\
(q \circ w_0) \circ b_2 \;&=\; q \circ (w_0 \circ b_2) \;=\; q \circ (w_2 \circ b_0) \;=\; (q \circ w_2) \circ b_0.
\end{aligned}$$

From (4), we derive the desired triple (3) as shown below:

$$\begin{aligned}
\mathit{tri}((p \circ w_2) \otimes b_0, d, (q \circ w_2) \circ b_0) &\Rightarrow \mathit{tri}(p \circ w_2, d, (q \circ w_2) \circ b_0) \\
&\Rightarrow \mathit{tri}(p \circ w_2, d, q \circ w_2).
\end{aligned}$$

Both implications hold because of Lemma 5. □

Corollary 8 (Anti-frame rule). *The anti-frame rule in Fig. 1 is sound.*

Proof. Pick w, k. Let p, c, q, w_0 be as in the anti-frame rule in Fig. 1. We must prove that $w \models_k \{p\}c\{q\}$. By Condition 2, there exists a commutative pair (a_0, a) for (w_0, w). By assumption, we have $\models \{p \otimes w_0\}c\{q \circ w_0\}$, so, in particular, $a \models_k \{p \otimes w_0\}c\{q \circ w_0\}$. By Proposition 7, this implies $w \models_k \{p\}c\{q\}$, as desired. □

Next, we move on to the soundness proof of the two frame rules in Fig. 1.

Lemma 9. *The following equivalence, which expresses a distribution axiom [9], holds:* $w_0 \circ w \models_k \{p\}c\{q\}$ *iff* $w \models_k \{p \circ w_0\}c\{q \circ w_0\}$.

Proof. Pick w_1. Let $w_1' = w \circ w_1$. By Definition 4 and the associativity of $\circ$, it suffices to prove the equivalence of $tri(p \circ (w_0 \circ w_1'),\ \pi_k(c),\ q \circ (w_0 \circ w_1'))$ and $tri((p \circ w_0) \circ w_1',\ \pi_k(c),\ (q \circ w_0) \circ w_1')$. This equivalence follows from Lemma 2. □

Corollary 10 (Frame rules). *The frame rules in Fig. 1 are sound.*

Proof. The soundness of the deep frame rule follows from Lemma 9. The shallow rule is sound thanks to the universal quantification over u in Definition 3. □

4 A Concrete Model with Recursively Defined Worlds

In this section, we consider a concrete instance of the general framework described in Sections 2 and 3 where W is isomorphic to $Assert$. The $Assert \to W$ direction of this isomorphism means that all assertions can be used as hidden invariants. This lets us define a semantic model of the program logic that is presented next (Section 5). The heap model in this particular case is given by the following recursively defined cpos:

$$Heap = Rec(Val) \quad Val = Int_\perp \oplus Com_\perp \quad Com = Heap \multimap Heap \oplus \{error\}_\perp \quad (5)$$

where $Rec(Val)$ denotes records with entries in Val labelled by positive natural numbers[6]. These labels serve as addresses or locations. The partial operation $h \cdot h'$ combines two heaps h and h' (i.e., takes the union) whenever the domains of h and h' are disjoint. When $h = \perp$ or $h' = \perp$, $h \cdot h'$ is $\perp$. The empty record provides a unit for heap combination, thus there is also a unit for the separating conjunction on $UAdm$. Finally, the solution of (5) in the category $\mathbf{Cppo}_\perp$ of pointed cpos and strict continuous functions comes equipped with a family of projections π_k that satisfy the requirements of Section 2.

The key result of this section is the following theorem:

Theorem 11. *There exists a monoid* $(W, e, \circ)$*, where* W *is an object in CBUlt with an isomorphism* ι *from* W *to* $(\frac{1}{2} \cdot W) \to_m UAdm$*. The operation* $\circ$ *satisfies*

$$\forall\, w_1, w_2, w \in W.\quad \iota(w_1 \circ w_2)(w) \;=\; \iota(w_1)(w_2 \circ w) \,*\, \iota(w_2)(w).$$

[6] Formally, $Rec(D) = (\Sigma_{N \subseteq_{fin} Nats^+}(N \to D_\downarrow))_\perp$ where $N \to D_\downarrow$ is the cpo of maps from the finite address set N to $D_\downarrow = D \setminus \{\perp\}$ of non-bottom elements of D.

The equation in this theorem is just Condition 1, where the coercion i is taken to be the isomorphism ι.

Construction of the worlds W in Theorem 11. In previous work [9] we gave a model of a separation logic with nested triples and higher-order frame rules, but no anti-frame rule. For this model we needed a solution W' to the following domain equation:[7]

$$W' \;\cong\; (\tfrac{1}{2} \cdot W') \to \mathit{UAdm}. \tag{6}$$

Note that (6) is almost the same domain equation as described in Theorem 11 above, except that there is no restriction to monotonic functions in the function space on the right. One can use a general existence theorem [12, 13] to obtain a solution W' for (6) in the category *CBUlt*. In a second step, using the complete metric on W', one can then define a monoid operation $\circ$ that satisfies the equation stated in Theorem 11.

In the present setup, this two-step approach to constructing a solution $W \cong \mathit{Assert}$ and a monoid operation $\circ$ cannot be applied, however, because of the added monotonicity requirement in Theorem 11: since the order on W is defined in terms of the operation $\circ$, one needs $\circ$ already in order to *express* this equation, i.e., it appears necessary to define the operation $\circ$ *at the same time* as W. Thus we construct $W \cong (\frac{1}{2} \cdot W) \to_m \mathit{UAdm}$ explicitly, as (inverse) limit

$$W = \left\{ x \in \textstyle\prod_{k \geq 0} W_k \mid \forall k \geq 0.\ \iota_k^\circ(x_{k+1}) = x_k \right\}$$

of a sequence of "approximations" W_k of W,

$$\begin{tikzcd} W_0 \arrow[r, shift left, "\iota_0"] & W_1 \arrow[l, shift left, "\iota_0^\circ"] \arrow[r, shift left, "\iota_1"] & W_2 \arrow[l, shift left, "\iota_1^\circ"] \arrow[r, shift left, "\iota_2"] & \cdots \arrow[l, shift left, "\iota_2^\circ"] \arrow[r, shift left, "\iota_k"] & W_{k+1} \arrow[l, shift left, "\iota_k^\circ"] \arrow[r, shift left, "\iota_{k+1}"] & \cdots \arrow[l, shift left, "\iota_{k+1}^\circ"] \end{tikzcd} \tag{7}$$

Each W_k is a complete 1-bounded ultrametric space equipped with a non-expansive operation $\circ_k : W_k \times W_k \to W_k$ and a preorder $\sqsubseteq_k$, so that $W_{k+1} = (\frac{1}{2} \cdot W_k) \to_m \mathit{UAdm}$ are the non-expansive and monotone functions with respect to $\sqsubseteq_k$. The maps ι_k and ι_k° are given by $\iota_{k+1}(w) = \pi_{k+2} \circ w \circ \iota_k^\circ$ and $\iota_{k+1}^\circ(w) = \pi_{k+1} \circ w \circ \iota_k$. The diagram (7) forms a *Cauchy tower* [13], in that $\sup_w d_{k+1}(w, (\iota_k \circ \iota_k^\circ)(w))$ and $\sup_w d_k(w, (\iota_k^\circ \circ \iota_k)(w))$ become arbitrarily small as k increases. The operation $\circ_{k+1}$ is defined in terms of $\circ_k$ and ι_k°:

$$(w_1 \circ_{k+1} w_2)(w) \;\stackrel{\mathit{def}}{=}\; w_1\,(\iota_k^\circ(w_2) \circ_k w) \,*\, w_2(w).$$

One technical inconvenience is that the $\circ_k$'s are not associative. However, associativity holds "up to approximation k," $d_k((x \circ_k y) \circ_k z, x \circ_k (y \circ_k z)) \leq 2^{-k}$, which yields associativity "in the limit" and thus a monoid structure on W by

$$(x_k)_{k\geq 0} \circ (y_k)_{k \geq 0} \stackrel{\mathit{def}}{=} \left(\lim_{j>k} \iota_k^\circ(\ldots(\iota_{j-1}^\circ(x_j \circ_j y_j))\right)_{k\geq 0}.$$

[7] Technically, we solved a different equation $W' \cong \frac{1}{2}(W' \to \mathit{UAdm})$. This difference is insignificant, since the solution of one equation leads to that of the other equation.

To finish the proof of Theorem 11 one shows that $\iota(w) \stackrel{def}{=} \lim_k(\lambda w'.\, w_{k+1}(w'_k))$ establishes an isomorphism ι between W and $(\frac{1}{2}\cdot W) \rightarrow_m UAdm$, which satisfies $\iota(w_1 \circ w_2) = \iota(w_1) \otimes w_2 \, * \, \iota(w_2)$ for $(p \otimes w) = \lambda w'.p(w \circ w')$.

The details of the proof are given in the full version of this paper [14].

Existence of commutative pairs. To show that W forms an instance of our semantic framework, we also need to prove the existence of commutative pairs. Given a pair (w_0, w_1) of worlds, we construct a commutative pair using properties of $\otimes$ and the coercion ι. Since the monoid operation $\otimes$ is contractive in its second argument, so is the function $f(a_0, a_1) = \big(\iota^{-1}(\iota(w_0) \otimes a_1),\ \iota^{-1}(\iota(w_1) \otimes a_0)\big)$ on $W \times W$. By the Banach fixed point theorem, there exists a unique pair $(a_0, a_1) = f(a_0, a_1)$. Thus $\iota(a_0) = \iota(w_0) \otimes a_1$ and $\iota(a_1) = \iota(w_1) \otimes a_0$. Since ι is injective, we can prove the remaining $w_0 \circ a_1 = w_1 \circ a_0$ as follows. For all $w \in W$,

$$\begin{aligned}
\iota(w_0 \circ a_1)(w) &= (\iota(w_0) \otimes a_1)(w) * \iota(a_1)(w) && \text{(by Theorem 11 and def. of } \otimes) \\
&= \iota(a_0)(w) * (\iota(w_1) \otimes a_0)(w) && \text{(by the above properties of } a_0,\, a_1) \\
&= \iota(a_0)(w) * \iota(w_1)(a_0 \circ w) && \text{(by def. of } \otimes) \\
&= \iota(w_1 \circ a_0)(w) && \text{(by Theorem 11).}
\end{aligned}$$

Theorem 12. *The monoid $(W, e, \circ)$ in Theorem 11 and the isomorphism ι : $W \rightarrow Assert$ form an instance of the framework in Section 2.*

5 Program Logic

We now give one application of our semantic development. We present a program logic for higher-order store, which includes anti-frame and frame rules. Using the results of Sections 3 and 4, we define the semantics of the logic and prove its soundness.

Programming language. Fig. 2 gives the syntax of a small imperative programming language equipped with operations for stored code and heap manipulation. The expressions in the language are integer expressions, variables, and the quote expression 'C' for representing an unevaluated command C. The integer or code value denoted by expression e_1 is stored in a heap cell e_0 using $[e_0]{:=}e_1$, and this stored value is later looked up and bound to the variable y by let $y{=}[e_0]$ in D. In the case that the value stored in cell e_0 is code 'C', we can run (or "evaluate") this code by executing eval $[e_0]$. As in ML, all variables x, y, z are *immutable.* The language does not include while loops: they can be expressed by stored code (using Landin's knot). The interpretation of commands in the cpo Com of (5) is straightforward [9]. The interpretation of the quote operation, 'C', uses the injection of Com into Val in Section 4.

Assertions and distribution axioms. As in previous work [9], our assertion language is first-order intuitionistic logic, extended with the separating connectives $\mathsf{e}, *$, and the points-to predicate $\mapsto$ [5]. The syntax of assertions appears in Fig. 2. The standard connectives are omitted.

The most distinguishing features of the assertion language are Hoare triples $\{P\}e\{Q\}$ and invariant extensions $P \otimes Q$. The fact that a triple is an assertion

$$\begin{array}{rcll}
e \in \mathit{Exp} & ::= & -1 \mid 1 \mid e_1{+}e_2 \mid \ldots \mid x \mid `C\text{'} & \text{integer expression, variable, quote} \\
C \in \mathit{Com} & ::= & [e_1]{:=}e_2 \mid \mathsf{let}\ y{=}[e]\ \mathsf{in}\ C \mid \mathsf{eval}\,[e] & \text{assignment, lookup, unquote} \\
 & \mid & \mathsf{let}\ x{=}\mathsf{new}\ (e_1,\ldots,e_n)\ \mathsf{in}\ C \mid \mathsf{free}\ e & \text{allocation, disposal} \\
 & \mid & \mathsf{skip} \mid C_1;C_2 \mid \mathsf{if}\ (e_1{=}e_2)\ C_1\ C_2 & \text{no op, sequencing, conditional} \\
P, Q \in \mathit{Assn} & ::= & e_1 \mapsto e_2 \mid \mathsf{e} \mid P * Q & \text{separating connectives} \\
 & \mid & \{P\}e\{Q\} \mid P \otimes Q & \text{Hoare triple, invariant extension} \\
 & \mid & X \mid \mu X.P \mid \ldots & \text{assertion variable, recursion}
\end{array}$$

Fig. 2. Syntax of expressions, commands and assertions

$$\begin{array}{rcll}
\{P\}e\{Q\} \otimes R & \Leftrightarrow & \{P \circ R\}e\{Q \circ R\} & \\
(\kappa x.P) \otimes R & \Leftrightarrow & \kappa x.(P \otimes R) & (\kappa \in \{\forall, \exists\}, x \notin \mathit{fv}(R)) \\
(P \otimes R) \otimes R' & \Leftrightarrow & P \otimes (R \circ R') & \\
(P \oplus Q) \otimes R & \Leftrightarrow & (P \otimes R) \oplus (Q \otimes R) & (\oplus \in \{\Rightarrow, \wedge, \vee, *\}) \\
P \otimes R & \Leftrightarrow & P & (R \text{ is } \mathsf{e} \text{ or } P \text{ is one of } \mathsf{true}, \mathsf{false}, \mathsf{e}, e \mapsto e', \ldots) \\
(\mu X.P) \otimes R & \Leftrightarrow & \mu X.(P \otimes R) & (X \notin \mathit{fv}(R))
\end{array}$$

Fig. 3. Axioms for distributing $- \otimes R$

means that triples can be nested. Intuitively, the assertion $P \otimes Q$ denotes a version of P where every (possibly deeply nested) triple receives a copy of Q as an extra $*$-conjunct in its pre- and post-conditions. More precisely, the behaviour of the $\otimes$ operator is described by the axioms in Fig. 3, which let us distribute $\otimes$ through all the constructs of the assertion language. These axioms use the abbreviation $Q \circ R$ for $(Q \otimes R) * R$.

Assertions include assertion variables $X \in \mathcal{X}$, and can be recursively defined: the construct $\mu X.P$ binds X in P and satisfies the axiom $\mu X.P \Leftrightarrow P[X := \mu X.P]$. Not every recursive assertion is permitted: for $\mu X.P$ to be well-formed, we require that P be *formally contractive in* X. In short, this means that every free occurrence of X within P must lie either within a triple or within the second argument of a $\otimes$ construct. (We omit the straightforward inductive definition of formal contractiveness.) Semantically, this requirement ensures that $\mu X.P$ is well-defined as the unique fixed point of P, viewed as a function of X. In particular, all assertions of the form $\mu X.P \otimes X$, where X does not appear in P, are formally contractive. Pottier's applications of the anti-frame rule [4] make extensive use of assertions of this form.

The interpretation of assertions uses W in Section 4. Given such W and an environment η that maps variables x to values $\eta(x) \in \mathit{Val}$, we interpret an assertion P as a non-expansive function $[\![P]\!]_\eta : \mathit{Assert}^{\mathcal{X}} \to \mathit{Assert}$. The uniform admissible sets in UAdm, partially ordered by inclusion, form a complete Heyting algebra with a monotone commutative monoid. The domain Assert, ordered pointwise, inherits this structure (see Appendix B of the full version of this

$$\begin{aligned}
[\![P \otimes R]\!]_{\eta,\xi} &= [\![P]\!]_{\eta,\xi} \otimes \iota^{-1}([\![R]\!]_{\eta,\xi}) \qquad\qquad [\![\mu X.P]\!]_{\eta,\xi} = \mathit{fix}(\lambda q.\ [\![P]\!]_{\eta,\xi[X:=q]}) \\
[\![\{P\}\text{`}C\text{'}\{Q\}]\!]_{\eta,\xi} &= \lambda w.\,\{h \mid \mathit{rnk}(h) > 0 \ \Rightarrow\ w \models_{\mathit{rnk}(h)-1} \{[\![P]\!]_{\eta,\xi}\}\, [\![C]\!]_{\eta}\, \{[\![Q]\!]_{\eta,\xi}\}\}
\end{aligned}$$

Fig. 4. Interpretation of assertions

$$\textsc{Anti-frame}\ \frac{\Gamma;\Xi \vdash \{P \otimes R\}e\{Q \circ R\}}{\Gamma;\Xi \vdash \{P\}e\{Q\}} \qquad \textsc{Deep-frame}\ \frac{\Gamma;\Xi \vdash \{P\}e\{Q\}}{\Gamma;\Xi \vdash \{P \circ R\}e\{Q \circ R\}} \qquad \textsc{Shallow-frame}\ \frac{}{\Gamma;\Xi \vdash \{P\}e\{Q\} \Rightarrow \{P * R\}e\{Q * R\}}$$

Fig. 5. Proof rules from separation logic

paper [14]). This is used to interpret the intuitionistic first-order fragment, $*$ and e of the assertion language. Fig. 4 shows three of the remaining cases. First, via the isomorphism ι^{-1}, we can turn any assertion $r \in \mathit{Assert}$ into an invariant $\iota^{-1}(r) \in W$ and thus interpret the invariant extension $P \otimes R$. Next, because P must be formally contractive in X, the map $q \mapsto [\![P]\!]_{\eta,\xi[X:=q]}$ on Assert is contractive in the metric sense: thus, by the Banach fixed point theorem, it has a unique fixed point. Finally, the interpretation of nested triples is in terms of semantic triples, and uses approximate validity to ensure non-expansiveness.

Proof rules. The logic derives judgements of the form $\Gamma;\Xi \vdash P$, where P is an assertion, and Γ and Ξ respectively bind variables and assertion variables. For instance, to prove that command C stores at cell 1 some code that writes 0 into cell 10, one would need to derive $\Gamma;\Xi \vdash \{1 \mapsto \_\}\text{`}C\text{'}\{1 \mapsto \{10 \mapsto \_\}\_\{10 \mapsto 0\}\}$.

The logic includes the standard proof rules for intuitionistic logic and the logic of bunched implications [16] as well as standard separation logic proof rules [9]. We do not repeat these rules here. Fig. 5 shows a version of the anti-frame rule and two versions of the frame rule: the deep frame rule (expressed in combination with the distribution axioms) and the first-order shallow frame rule (which takes the form of an axiom).

Theorem 13 (Soundness). *The interpretation of assertions is well-defined, and validates the distribution axioms of Fig. 3 and the inference rules of Fig. 5.*

6 Conclusion and Future Work

We have presented a semantic framework for studying the soundness of anti-frame and frame rules for languages with higher-order store. Moreover, we have presented a concrete instance of the framework, and used it to give the first rigorous proof of soundness of separation logic with anti-frame and frame rules for a language with higher-order store.

We are aware of other instantiations of the semantic framework, which can be used to show the soundness of stronger variants of the anti-frame and frame rules, provided the universe W of invariants is restricted. For space reasons, we have not included those instantiations in this extended abstract; they appear in the full version of the paper [14].

Future work includes lifting the results in this paper to Pottier's type-and-capability system as well as extending our soundness results to generalized versions of the anti-frame and frame rules where invariants evolve in more sophisticated ways over time [17, 18].

Acknowledgments. We would like to thank Kristian Støvring and Jacob Thamsborg for helpful discussions. Partial support has been provided by FNU project 272-07-0305 "Modular reasoning about software" and EPSRC.

References

1. Parkinson, M., Bierman, G.: Separation logic and abstraction. In: POPL, pp. 247–258 (2005)
2. Biering, B., Birkedal, L., Torp-Smith, N.: BI-hyperdoctrines, higher-order separation logic, and abstraction. TOPLAS 29(5) (2007)
3. Parkinson, M., Bierman, G.: Separation logic, abstraction and inheritance. In: POPL, pp. 75–86 (2008)
4. Pottier, F.: Hiding local state in direct style: a higher-order anti-frame rule. In: LICS, pp. 331–340 (2008)
5. Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: LICS, pp. 55–74 (2002)
6. O'Hearn, P.W., Yang, H., Reynolds, J.C.: Separation and information hiding. In: POPL, pp. 268–280 (2004)
7. Birkedal, L., Torp-Smith, N., Yang, H.: Semantics of separation-logic typing and higher-order frame rules for Algol-like languages. LMCS 2(5:1) (2006)
8. Birkedal, L., Reus, B., Schwinghammer, J., Yang, H.: A simple model of separation logic for higher-order store. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 348–360. Springer, Heidelberg (2008)
9. Schwinghammer, J., Birkedal, L., Reus, B., Yang, H.: Nested Hoare triples and frame rules for higher-order store. In: CSL, pp. 440–454 (2009)
10. Pottier, F.: Three comments on the anti-frame rule (July 2009) (unpublished note)
11. Levy, P.B.: Possible world semantics for general storage in call-by-value. In: CSL, pp. 232–246 (2002)
12. Rutten, J.J.M.M.: Elements of generalized ultrametric domain theory. TCS 170(1-2), 349–381 (1996)
13. Birkedal, L., Støvring, K., Thamsborg, J.: The category-theoretic solution of recursive metric-space equations. Technical Report ITU-2009-119, IT University of Copenhagen (2009)
14. Schwinghammer, J., Yang, H., Birkedal, L., Pottier, F., Reus, B.: A semantic foundation for hidden state (December 2009), http://www.dcs.qmul.ac.uk/~hyang/paper/fossacs10-full.pdf

15. Streicher, T.: Domain-theoretic Foundations of Functional Programming. World Scientific, Singapore (2006)
16. O'Hearn, P.W., Pym, D.J.: The logic of bunched implications. Bulletin of Symbolic Logic 5(2), 215–244 (1999)
17. Pilkiewicz, A., Pottier, F.: The essence of monotonic state (October 2009) (submitted)
18. Pottier, F.: Generalizing the higher-order frame and anti-frame rules (July 2009) (unpublished note)

Linearly-Used Continuations in the Enriched Effect Calculus

Jeff Egger[1,⋆], Rasmus Ejlers Møgelberg[2,⋆⋆], and Alex Simpson[1,∗]

[1] LFCS, School of Informatics, University of Edinburgh, Scotland, UK
[2] IT University of Copenhagen, Copenhagen, Denmark

Abstract. The *enriched effect calculus* is an extension of Moggi's computational metalanguage with a selection of primitives from linear logic. In this paper, we present an extended case study within the enriched effect calculus: the linear usage of continuations. We show that established call-by-value and call-by name linearly-used CPS translations are uniformly captured by a single generic translation of the enriched effect calculus into itself. As a main syntactic theorem, we prove that the generic translation is involutive up to isomorphism. As corollaries, we obtain full completeness results for the original call-by-value and call-by-name translations. The main syntactic theorem is proved using a category-theoretic semantics for the enriched effect calculus. We show that models are closed under a natural *dual model* construction. The canonical linearly-used CPS translation then arises as the unique (up to isomorphism) map from the syntactic initial model to its own dual. This map is an equivalence of models. Thus the initial model is self-dual.

1 Introduction

The *continuations monad* $((-) \to R) \to R$ is commonly used to model control effects. Following Moggi's idea of interpreting call-by-value programs in the Kleisli category [11,12], a call-by-value program from X to Y is interpreted as a *continuation transformer* $(Y \to R) \to (X \to R)$. In an influential paper [3], Berdine *et al.* observe that, in many programming situations, continuation transformers satisfy an additional property: their argument, the continuation $Y \to R$, is used *linearly*. Thus a call-by-value program can be more informatively modelled as a linear function $(Y \to R) \multimap (X \to R)$, corresponding to a Kleisli map for the *linearly-used continuations monad* $((-) \to R) \multimap R$.

One goal of the present paper is to address the question: what is the natural type-theoretic (and semantic) context for modelling linearly-used continuations? With the presence of both intuitionistic ($\to$) and linear ($\multimap$) arrows, *intuitionistic linear type theory (ILL)* (and its model theory) seems a natural answer. Indeed, ILL has been used as the basis of a systematic study of linearly-used

⋆ Research supported by EPSRC research grant "Linear Observations and Computational Effects".

⋆⋆ Research supported by the Danish Agency for Science, Technology and Innovation.

continuations by Hasegawa. In [6], he presents a continuation passing style (CPS) translation of Moggi's call-by-value computational λ-calculus into ILL, using the linearly-used continuations monad, and establishes a full completeness result for this. A follow-up paper [7] considers call-by-name.

In this paper, we use a more general type theory, the *enriched effect calculus (EEC)* [4], as a context for modelling linearly-used continuations. On the one hand, it can be seen as a fragment of ILL and, as such, its models strictly generalise models of ILL. On the other hand, it is a conservative extension of the standard calculi for modelling computational effects (Moggi's *computational metalanguage* [12], and Levy's *call-by-push-value (CBPV)* [10]) with a selection of constructs from linear logic. In fact, any *adjunction model* of CBPV (and hence any model of Moggi's computational metalanguage) expands to a model of EEC [4]. This provides an abundant supply of computationally interesting models of EEC that are not models of ILL.

The paper begins with a brief presentation of the enriched effect calculus in Section 2. This is followed, in Section 3, by the treatment of linearly-used continuations within EEC. The starting point is the observation that Hasegawa's call-by-value [6] and call-by-name [7] linearly-used CPS translations of simply-typed λ-calculus both fall in that fragment of ILL corresponding to EEC. The first contribution of the paper is to show that, using EEC, we can recover these translations in a particularly interesting way. This is achieved by identifying a single canonical linearly-used CPS-translation of the entire enriched effect calculus into itself. Hasegawa's call-by-value and call-by-name translations are derived from this by composing the canonical translation with the standard call-by-value and call-by-name encodings of typed λ-calculus into effect calculi (cf. Moggi [12], Filinski [5], Levy [10]).

The canonical linearly-used CPS-translation of EEC into itself possesses a remarkable property, unexpected in the context of CPS translations: it is involutive up to isomorphism. That is, the translation of a translated term equals the original term modulo type isomorphism. This property is the main syntactic theorem of the paper. As corollaries, we obtain full-completeness results for the call-by-value and call-by-name linearly-used CPS translations into EEC, mirroring Hasegawa's results for the translations into ILL.

The second half of the paper provides a semantic context for the first. Section 4 reviews the notion of EEC model given in [4]. Following this, Section 5 gives a semantic account of the canonical linearly-used CPS translation. We show that the linearly-used continuations monad forms the basis of a duality of EEC models. The *dual model* of a model can be viewed as a linearly-used continuations model constructed over the original. The "dual" terminology is justified by every model being isomorphic to its own double dual. Thus, surprisingly, every model of the enriched effect calculus arises as a linearly-used continuations model relative to some other model.

Finally, we specialise the dual model construction to the syntactic model of EEC given by typed terms modulo equality. By the universal property of this model [4], there is a unique (up to isomorphism) morphism of models from the

$$\frac{}{\Gamma, x\colon \mathsf{A} \mid - \vdash x\colon \mathsf{A}} \qquad \frac{}{\Gamma \mid z\colon \underline{\mathsf{A}} \vdash z\colon \underline{\mathsf{A}}} \qquad \frac{}{\Gamma \mid \Delta \vdash *\colon 1}$$

$$\frac{\Gamma \mid \Delta \vdash t\colon \mathsf{A} \quad \Gamma \mid \Delta \vdash u\colon \mathsf{B}}{\Gamma \mid \Delta \vdash \langle t, u\rangle\colon \mathsf{A} \times \mathsf{B}} \qquad \frac{\Gamma \mid \Delta \vdash t\colon \mathsf{A} \times \mathsf{B}}{\Gamma \mid \Delta \vdash \mathrm{fst}(t)\colon \mathsf{A}} \qquad \frac{\Gamma \mid \Delta \vdash t\colon \mathsf{A} \times \mathsf{B}}{\Gamma \mid \Delta \vdash \mathrm{snd}(t)\colon \mathsf{B}}$$

$$\frac{\Gamma, x\colon \mathsf{A} \mid \Delta \vdash t\colon \mathsf{B}}{\Gamma \mid \Delta \vdash \lambda x\colon \mathsf{A}.\, t\colon \mathsf{A} \to \mathsf{B}} \qquad \frac{\Gamma \mid \Delta \vdash s\colon \mathsf{A} \to \mathsf{B} \quad \Gamma \mid - \vdash t\colon \mathsf{A}}{\Gamma \mid \Delta \vdash s(t)\colon \mathsf{B}}$$

$$\frac{\Gamma \mid - \vdash t\colon \mathsf{A}}{\Gamma \mid - \vdash !t\colon !\mathsf{A}} \qquad \frac{\Gamma \mid \Delta \vdash t\colon !\mathsf{A} \quad \Gamma, x\colon \mathsf{A} \mid - \vdash u\colon \underline{\mathsf{B}}}{\Gamma \mid \Delta \vdash \mathrm{let}\, !x\, \mathrm{be}\, t\, \mathrm{in}\, u\colon \underline{\mathsf{B}}}$$

$$\frac{\Gamma \mid z\colon \underline{\mathsf{A}} \vdash t\colon \underline{\mathsf{B}}}{\Gamma \mid - \vdash \underline{\lambda} z\colon \underline{\mathsf{A}}.\, t\colon \underline{\mathsf{A}} \multimap \underline{\mathsf{B}}} \qquad \frac{\Gamma \mid - \vdash s\colon \underline{\mathsf{A}} \multimap \underline{\mathsf{B}} \quad \Gamma \mid \Delta \vdash t\colon \underline{\mathsf{A}}}{\Gamma \mid \Delta \vdash s[t]\colon \underline{\mathsf{B}}}$$

$$\frac{\Gamma \mid - \vdash t\colon \mathsf{A} \quad \Gamma \mid \Delta \vdash u\colon \underline{\mathsf{B}}}{\Gamma \mid \Delta \vdash !t \otimes u\colon !\mathsf{A} \otimes \underline{\mathsf{B}}} \qquad \frac{\Gamma \mid \Delta \vdash s\colon !\mathsf{A} \otimes \underline{\mathsf{B}} \quad \Gamma, x\colon \mathsf{A} \mid z\colon \underline{\mathsf{B}} \vdash t\colon \underline{\mathsf{C}}}{\Gamma \mid \Delta \vdash \mathrm{let}\, !x \otimes z\, \mathrm{be}\, s\, \mathrm{in}\, t\colon \underline{\mathsf{C}}}$$

$$\frac{\Gamma \mid \Delta \vdash t\colon \underline{0}}{\Gamma \mid \Delta \vdash \underline{\mathrm{image}}(t)\colon \underline{\mathsf{A}}} \qquad \frac{\Gamma \mid \Delta \vdash t\colon \underline{\mathsf{A}}}{\Gamma \mid \Delta \vdash \underline{\mathrm{inl}}(t)\colon \underline{\mathsf{A}} \oplus \underline{\mathsf{B}}} \qquad \frac{\Gamma \mid \Delta \vdash t\colon \underline{\mathsf{B}}}{\Gamma \mid \Delta \vdash \underline{\mathrm{inr}}(t)\colon \underline{\mathsf{A}} \oplus \underline{\mathsf{B}}}$$

$$\frac{\Gamma \mid \Delta \vdash s\colon \underline{\mathsf{A}} \oplus \underline{\mathsf{B}} \quad \Gamma \mid x\colon \underline{\mathsf{A}} \vdash t\colon \underline{\mathsf{C}} \quad \Gamma \mid y\colon \underline{\mathsf{B}} \vdash u\colon \underline{\mathsf{C}}}{\Gamma \mid \Delta \vdash \underline{\mathrm{case}}\, s\, \mathrm{of}\, (\underline{\mathrm{inl}}(x).\, t; \underline{\mathrm{inr}}(y).\, u)\colon \underline{\mathsf{C}}}$$

Fig. 1. Typing rules for the effect calculus

syntactic model to its dual. This morphism is an equivalence of models. Thus the syntactic model is *self-dual*. Furthermore, the morphism identified from the syntactic model to its dual is nothing other than the canonical linearly-used CPS translation of EEC into itself from Section 3, and it is the equivalence property of this morphism that proves the involutivity of the translation. Thus we obtain a semantic proof of the main syntactic theorem of Section 3.

2 The Enriched Effect Calculus

The *enriched effect calculus* [4] is an extension of Moggi's computational metalanguage [12] with constructors from linear type theory. Similar to Filinski's effect PCF [5] and Levy's CBPV [10], it has two notions of types: *value types* and *computation types*. We use $\alpha, \beta, \ldots$ to range over a set of *value type constants*, and $\underline{\alpha}, \underline{\beta}, \ldots$ to range over a disjoint set of *computation type constants*. We then use $\mathsf{A}, \mathsf{B}, \ldots$ to range over *value types*, and $\underline{\mathsf{A}}, \underline{\mathsf{B}}, \ldots$ to range over *computation types*, which are specified by the grammar below:

$$\begin{aligned}
\mathsf{A} &::= \alpha \mid \underline{\alpha} \mid 1 \mid \mathsf{A}\times\mathsf{B} \mid \mathsf{A}\to\mathsf{B} \mid {!}\,\mathsf{A} \mid \underline{\mathsf{A}} \multimap \underline{\mathsf{B}} \mid {!}\mathsf{A}\otimes\underline{\mathsf{B}} \mid \underline{0} \mid \underline{\mathsf{A}}\oplus\underline{\mathsf{B}}\\
\underline{\mathsf{A}} &::= \underline{\alpha} \mid 1 \mid \underline{\mathsf{A}}\times\underline{\mathsf{B}} \mid \mathsf{A}\to\underline{\mathsf{B}} \mid {!}\,\mathsf{A} \mid {!}\mathsf{A}\otimes\underline{\mathsf{B}} \mid \underline{0} \mid \underline{\mathsf{A}}\oplus\underline{\mathsf{B}}\ .
\end{aligned}$$

The type constructor ! plays the role of Moggi's monadic type constructor T and Levy's F. Our notation has been chosen to exhibit the enriched effect calculus as a fragment of intuitionistic linear logic, a view which will be enhanced by the typing rules below.

Note that computation types form a subset of the value types (which is not the case in CBPV [10]). The reader is referred to [4] for further comparisons with other calculi and explanations of the enriched effect calculus.

The enriched effect calculus has two typing judgements:

$$\text{(i)}\ \ \Gamma \mid - \vdash t\colon \mathsf{B} \qquad\qquad \text{(ii)}\ \ \Gamma \mid z\colon\underline{\mathsf{A}} \vdash t\colon \underline{\mathsf{B}}\ ,$$

where Γ is a context of value-type assignments to variables. On the right of Γ is a *stoup*, which may either be empty, as in the case of judgement (i), or may consist of a unique type assignment $x\colon\underline{\mathsf{A}}$, in which case the type on the right of the turnstyle is also required to be a computation type, as in (ii). The typing rules are given in Figure 1. In them, Δ ranges over an arbitrary (possibly empty) stoup, and the rules are only applicable in the case of typing judgements that conform to (i) or (ii) above.

$$\begin{array}{ll}
\Gamma\mid\Delta\vdash t = *\colon 1 & \text{if } \Gamma\mid\Delta\vdash t\colon 1\\
\Gamma\mid\Delta\vdash \mathrm{fst}(\langle t,u\rangle) = t\colon \mathsf{A} & \text{if } \Gamma\mid\Delta\vdash t\colon\mathsf{A} \text{ and } \Gamma\mid\Delta\vdash t\colon\mathsf{B}\\
\Gamma\mid\Delta\vdash \mathrm{snd}(\langle t,u\rangle) = u\colon \mathsf{B} & \text{if } \Gamma\mid\Delta\vdash t\colon\mathsf{A} \text{ and } \Gamma\mid\Delta\vdash t\colon\mathsf{B}\\
\Gamma\mid\Delta\vdash \langle\mathrm{fst}(t),\mathrm{snd}(t)\rangle = t\colon \mathsf{A}\times\mathsf{B} & \text{if } \Gamma\mid\Delta\vdash t\colon\mathsf{A}\times\mathsf{B}\\
\Gamma\mid\Delta\vdash (\lambda x\colon\mathsf{A}.\,t)(u) = t[u/x]\colon \mathsf{B} & \text{if } \Gamma, x\colon\mathsf{A}\mid\Delta\vdash t\colon\mathsf{B} \text{ and } \Gamma\mid-\vdash u\colon\mathsf{A}\\
\Gamma\mid\Delta\vdash \lambda x\colon\mathsf{A}.\,(t(x)) = t\colon \mathsf{A}\to\mathsf{B} & \text{if } \Gamma\mid\Delta\vdash t\colon\mathsf{A}\to\mathsf{B} \text{ and } x\notin\Gamma,\Delta\\
\Gamma\mid-\vdash \mathrm{let}\ {!}x\ \mathrm{be}\ {!}t\ \mathrm{in}\ u = u[t/x]\colon \underline{\mathsf{B}} & \text{if } \Gamma\mid-\vdash t\colon\mathsf{A} \text{ and } \Gamma, x\colon\mathsf{A}\mid-\vdash u\colon\underline{\mathsf{B}}\\
\Gamma\mid\Delta\vdash \mathrm{let}\ {!}x\ \mathrm{be}\ t\ \mathrm{in}\ u[!\,x/y] = u[t\,/\,y]\colon \underline{\mathsf{B}} & \text{if } \Gamma\mid\Delta\vdash t\colon{!}\mathsf{A} \text{ and } \Gamma\mid y\colon{!}\mathsf{A}\vdash u\colon\underline{\mathsf{B}}\\
\Gamma\mid\Delta\vdash (\underline{\lambda}x\colon\underline{\mathsf{A}}.\,t)[u] = t[u/x]\colon \underline{\mathsf{B}} & \text{if } \Gamma\mid x\colon\underline{\mathsf{A}}\vdash t\colon\underline{\mathsf{B}} \text{ and } \Gamma\mid\Delta\vdash u\colon\underline{\mathsf{A}}\\
\Gamma\mid-\vdash \underline{\lambda}x\colon\underline{\mathsf{A}}.\,(t[x]) = t\colon\ \underline{\mathsf{A}}\multimap\underline{\mathsf{B}} & \text{if } \Gamma\mid-\vdash t\colon\underline{\mathsf{A}}\multimap\underline{\mathsf{B}} \text{ and } x\notin\Gamma\\
\Gamma\mid\Delta\vdash \mathrm{let}\ {!}x\otimes y\ \mathrm{be}\ {!}t\otimes s\ \mathrm{in}\ u = u[t,s/x,y]\colon \underline{\mathsf{C}} & \text{if } \Gamma\mid-\vdash t\colon\mathsf{A} \text{ and } \Gamma\mid\Delta\vdash s\colon\underline{\mathsf{B}}\\
 & \quad\text{and } \Gamma, x\colon\mathsf{A}\mid y\colon\underline{\mathsf{B}}\vdash u\colon\underline{\mathsf{C}}\\
\Gamma\mid\Delta\vdash \mathrm{let}\ {!}x\otimes y\ \mathrm{be}\ t\ \mathrm{in}\ u[!x\otimes y/z] = u[t/z]\colon \underline{\mathsf{C}} & \text{if } \Gamma\mid\Delta\vdash t\colon{!}\mathsf{A}\otimes\underline{\mathsf{B}} \text{ and } \Gamma\mid z\colon{!}\mathsf{A}\otimes\underline{\mathsf{B}}\vdash u\colon\underline{\mathsf{C}}\\
\Gamma\mid x\colon\underline{0}\vdash t = \underline{\mathrm{image}}(x)\colon\underline{\mathsf{A}} & \text{if } \Gamma\mid x\colon\underline{0}\vdash t\colon\underline{\mathsf{A}}\\
\Gamma\mid\Delta\vdash \underline{\mathrm{case}}\,\underline{\mathrm{inl}}(t)\ \mathrm{of}\ (\underline{\mathrm{inl}}(x).\,u;\underline{\mathrm{inr}}(y).\,u') & \text{if } \Gamma\mid x\colon\underline{\mathsf{A}}\vdash u\colon\underline{\mathsf{C}} \text{ and } \Gamma\mid y\colon\underline{\mathsf{B}}\vdash u'\colon\underline{\mathsf{C}}\\
\qquad\quad = u[t/x]\ \colon\ \underline{\mathsf{C}} & \quad\text{and } \Gamma\mid\Delta\vdash t\colon\underline{\mathsf{A}}\\
\Gamma\mid\Delta\vdash \underline{\mathrm{case}}\,\underline{\mathrm{inr}}(t)\ \mathrm{of}\ (\underline{\mathrm{inl}}(x).\,u;\underline{\mathrm{inr}}(y).\,u') & \text{if } \Gamma\mid x\colon\underline{\mathsf{A}}\vdash u\colon\underline{\mathsf{C}} \text{ and } \Gamma\mid y\colon\underline{\mathsf{B}}\vdash u'\colon\underline{\mathsf{C}}\\
\qquad\quad = u'[t/y]\ \colon\ \underline{\mathsf{C}} & \quad\text{and } \Gamma\mid\Delta\vdash t\colon\underline{\mathsf{B}}\\
\Gamma\mid\Delta\vdash \underline{\mathrm{case}}\,t\ \mathrm{of}\ (\underline{\mathrm{inl}}(x).u[\underline{\mathrm{inl}}(x)/z];\underline{\mathrm{inr}}(y).\,u[\underline{\mathrm{inr}}(y)/z]) & \\
\qquad\quad = u[t/z]\ \colon\ \underline{\mathsf{C}} & \text{if } \Gamma\mid\Delta\vdash t\colon\underline{\mathsf{A}}\oplus\underline{\mathsf{B}} \text{ and } \Gamma\mid z\colon\underline{\mathsf{A}}\oplus\underline{\mathsf{B}}\vdash u\colon\underline{\mathsf{C}}
\end{array}$$

Fig. 2. Equality rules for the enriched effect calculus

Rules for equalities between typed terms are presented in Figure 2. They are to be considered in addition to the expected (typed) congruence and α-equivalence rules.

There is a standard call-by-value translation of typed λ-calculus into Moggi's computational metalanguage, Filinski's effect PCF [5], and Levy's CBPV [10]. Similarly, there is a standard call-by-name translation into the latter two, which exploits the existence of computation types. We recall these using the syntax of our effect calculus. As a source calculus, we use the simply-typed λ-calculus with types $\sigma, \tau, \ldots$ given by:

$$\sigma ::= \alpha \mid 1 \mid \sigma \times \tau \mid \sigma \to \tau \ .$$

The call-by-value interpretation translates a type σ into a value type σ^{cbv}, and the call-by-name interpretation into a computation type σ^{cbn}.

$$\begin{aligned} \alpha^{\mathrm{cbv}} &= \alpha & \alpha^{\mathrm{cbn}} &= \underline{\alpha} \\ 1^{\mathrm{cbv}} &= 1 & 1^{\mathrm{cbn}} &= 1 \\ (\sigma \times \tau)^{\mathrm{cbv}} &= \sigma^{\mathrm{cbv}} \times \tau^{\mathrm{cbv}} & (\sigma \times \tau)^{\mathrm{cbn}} &= \sigma^{\mathrm{cbn}} \times \tau^{\mathrm{cbn}} \\ (\sigma \to \tau)^{\mathrm{cbv}} &= \sigma^{\mathrm{cbv}} \to !\tau^{\mathrm{cbv}} & (\sigma \to \tau)^{\mathrm{cbn}} &= \sigma^{\mathrm{cbn}} \to \tau^{\mathrm{cbn}} \ . \end{aligned}$$

Here, we assume that each type constant α of the typed λ-calculus, is included as a value-type constant, and has an associated computation-type constant $\underline{\alpha}$.

On terms, the cbv translation maps a judgement $x_1 \colon \sigma_1, \ldots, x_n \colon \sigma_n \vdash t \colon \tau$ to

$$x_1 \colon \sigma_1^{\mathrm{cbv}}, \ldots, x_n \colon \sigma_n^{\mathrm{cbv}} \mid - \vdash t^{\mathrm{cbv}} \colon !\tau^{\mathrm{cbv}} \ .$$

We omit the (routine) details. The cbn translation rather trivially maps a judgement $x_1 \colon \sigma_1, \ldots, x_n \colon \sigma_n \vdash t \colon \tau$ to

$$x_1 \colon \sigma_1^{\mathrm{cbn}}, \ldots, x_n \colon \sigma_n^{\mathrm{cbn}} \mid - \vdash t^{\mathrm{cbn}} \colon \tau^{\mathrm{cbn}} \ .$$

3 Linearly-Used Continuations

In [6], Hasegawa studies a translation from typed λ-calculus into intuitionistic linear type theory (ILL) which implements a continuation passing semantics in which continuations are used linearly. In [7], he defines a corresponding call-by-name translation (actually for a variant of Parigot's $\lambda\mu$-calculus [13]). The call-by-value interpretation translates a type σ into a linear type $\sigma^{\mathrm{cbv}_{\underline{\mathsf{R}}}}$, and the call-by-name interpretation translates σ to $\sigma^{\mathrm{cbn}_{\underline{\mathsf{R}}}}$, as defined in Figure 3. Both translations are defined with respect to a distinguished type constant $\underline{\mathsf{R}}$, which acts as a result type for continuations. Whereas Hasegawa's translations are into ILL, we give them into the enriched effect calculus, for which it is crucial that $\underline{\mathsf{R}}$ is a computation type, hence the underlining. The important point is that Hasegawa's type translations fall into the EEC fragment of ILL.

For terms, Hasegawa [6,7] gives translations into ILL. Once again, these translations land in the fragment of ILL given by the enriched effect calculus, and we

$$\begin{aligned}
\alpha^{\mathrm{cbv}_{\underline{\mathsf{R}}}} &= \alpha & \alpha^{\mathrm{cbn}_{\underline{\mathsf{R}}}} &= \underline{\alpha} \\
1^{\mathrm{cbv}_{\underline{\mathsf{R}}}} &= 1 & 1^{\mathrm{cbn}_{\underline{\mathsf{R}}}} &= \underline{0} \\
(\sigma \times \tau)^{\mathrm{cbv}_{\underline{\mathsf{R}}}} &= \sigma^{\mathrm{cbv}_{\underline{\mathsf{R}}}} \times \tau^{\mathrm{cbv}_{\underline{\mathsf{R}}}} & (\sigma \times \tau)^{\mathrm{cbn}_{\underline{\mathsf{R}}}} &= \sigma^{\mathrm{cbn}_{\underline{\mathsf{R}}}} \oplus \tau^{\mathrm{cbn}_{\underline{\mathsf{R}}}} \\
(\sigma \to \tau)^{\mathrm{cbv}_{\underline{\mathsf{R}}}} &= \sigma^{\mathrm{cbv}_{\underline{\mathsf{R}}}} \to ((\tau^{\mathrm{cbv}_{\underline{\mathsf{R}}}} \to \underline{\mathsf{R}}) \multimap \underline{\mathsf{R}}) & (\sigma \to \tau)^{\mathrm{cbn}_{\underline{\mathsf{R}}}} &= !(\sigma^{\mathrm{cbn}_{\underline{\mathsf{R}}}} \multimap \underline{\mathsf{R}}) \otimes \tau^{\mathrm{cbn}_{\underline{\mathsf{R}}}} \; .
\end{aligned}$$

Fig. 3. Cbv and cbn linearly-used CPS translations of typed λ-calculus

shall just give them directly as translations into the latter. On terms, the cbv translation maps a judgement $x_1 \colon \sigma_1, \ldots, x_n \colon \sigma_n \vdash t \colon \tau$ to

$$x_1 \colon \sigma_1^{\mathrm{cbv}_{\underline{\mathsf{R}}}}, \ldots, x_n \colon \sigma_n^{\mathrm{cbv}_{\underline{\mathsf{R}}}} \,|\, - \vdash t^{\mathrm{cbv}_{\underline{\mathsf{R}}}} \colon (\tau^{\mathrm{cbv}_{\underline{\mathsf{R}}}} \to \underline{\mathsf{R}}) \multimap \underline{\mathsf{R}} \; .$$

The cbn translation maps a typing judgement $x_1 \colon \sigma_1, \ldots, x_n \colon \sigma_n \vdash t \colon \tau$ to

$$x_1 \colon \sigma_1^{\mathrm{cbn}_{\underline{\mathsf{R}}}} \multimap \underline{\mathsf{R}}, \ldots, x_n \colon \sigma_n^{\mathrm{cbn}_{\underline{\mathsf{R}}}} \multimap \underline{\mathsf{R}} \,|\, - \vdash t^{\mathrm{cbn}_{\underline{\mathsf{R}}}} \colon \tau^{\mathrm{cbn}_{\underline{\mathsf{R}}}} \multimap \underline{\mathsf{R}} \; .$$

For lack of space, we do not give the details of the term translations, see [6,7].

We now present the canonical linearly-used CPS translation of the entire EEC into itself. This translation maps any value type A to a value type $\mathsf{A}^{\mathcal{V}_{\underline{\mathsf{R}}}}$ and any computation type $\underline{\mathsf{B}}$ to a computation type $\underline{\mathsf{B}}^{\mathcal{C}_{\underline{\mathsf{R}}}}$, as defined in Figure 4. Note, that the result type $\underline{\mathsf{R}}$ is treated differently from the other computation-type constants, and $\underline{\alpha}$ ranges only over the latter. For each computation type $\underline{\mathsf{A}}$, we obtain an isomorphism $i_{\underline{\mathsf{A}}} \colon (\underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \multimap \underline{\mathsf{R}}) \to \underline{\mathsf{A}}^{\mathcal{V}_{\underline{\mathsf{R}}}}$, whose easy definition we omit.

Next, we define the translation of terms. We translate a typing judgement $\Gamma \,|\, - \vdash t \colon \mathsf{A}$ as:

$$\Gamma^{\mathcal{V}_{\underline{\mathsf{R}}}} \,|\, - \vdash t^{\mathcal{V}_{\underline{\mathsf{R}}}} \colon \mathsf{A}^{\mathcal{V}_{\underline{\mathsf{R}}}} \; ,$$

where $\Gamma^{\mathcal{V}_{\underline{\mathsf{R}}}}$ is the context obtained by applying $(-)^{\mathcal{V}_{\underline{\mathsf{R}}}}$ to every type in Γ. A typing judgement $\Gamma \,|\, z \colon \underline{\mathsf{A}} \vdash t \colon \underline{\mathsf{B}}$ is translated to:

$$\Gamma^{\mathcal{V}_{\underline{\mathsf{R}}}} \,|\, z \colon \underline{\mathsf{B}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \vdash t^{\mathcal{C}_{\underline{\mathsf{R}}}} \colon \underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \; .$$

The two translations are given in Figure 5. In this figure, each line corresponds to one of the typing rules in Figure 1. Observe that each typing rule that mentions Δ has two cases in Figure 5: one for empty stoup, and one for non-empty stoup. Also note that, in Figure 5, we always write z for the stoup variable.

We now establish the main properties of the translation. Since it is defined compositionally on the term structure, a straightforward induction proves:

Proposition 1 (Soundness)

1. *If* $\Gamma \,|\, - \vdash t = u \colon \mathsf{A}$ *then* $\Gamma^{\mathcal{V}_{\underline{\mathsf{R}}}} \,|\, - \vdash t^{\mathcal{V}_{\underline{\mathsf{R}}}} = u^{\mathcal{V}_{\underline{\mathsf{R}}}} \colon \mathsf{A}^{\mathcal{V}_{\underline{\mathsf{R}}}}$.
2. *If* $\Gamma \,|\, z \colon \underline{\mathsf{A}} \vdash t = u \colon \underline{\mathsf{B}}$ *then* $\Gamma^{\mathcal{V}_{\underline{\mathsf{R}}}} \,|\, z \colon \underline{\mathsf{B}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \vdash t^{\mathcal{C}_{\underline{\mathsf{R}}}} = u^{\mathcal{C}_{\underline{\mathsf{R}}}} \colon \underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}}$.

A fundamental property is that, for the entire enriched effect calculus, the linearly-used CPS translation is involutive up to type isomorphism.

$$\begin{array}{rclrcl}
\alpha^{\mathcal{V}_{\underline{\mathsf{R}}}} &=& \alpha & & & \\
\underline{\alpha}^{\mathcal{V}_{\underline{\mathsf{R}}}} &=& \underline{\alpha}^{\mathcal{C}_{\underline{\mathsf{R}}}} \multimap \underline{\mathsf{R}} & \underline{\alpha}^{\mathcal{C}_{\underline{\mathsf{R}}}} &=& \underline{\alpha} \\
1^{\mathcal{V}_{\underline{\mathsf{R}}}} &=& 1 & 1^{\mathcal{C}_{\underline{\mathsf{R}}}} &=& \underline{0} \\
(\mathsf{A} \times \mathsf{B})^{\mathcal{V}_{\underline{\mathsf{R}}}} &=& \mathsf{A}^{\mathcal{V}_{\underline{\mathsf{R}}}} \times \mathsf{B}^{\mathcal{V}_{\underline{\mathsf{R}}}} & (\underline{\mathsf{A}} \times \underline{\mathsf{B}})^{\mathcal{C}_{\underline{\mathsf{R}}}} &=& \underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \oplus \underline{\mathsf{B}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \\
(\mathsf{A} \to \mathsf{B})^{\mathcal{V}_{\underline{\mathsf{R}}}} &=& \mathsf{A}^{\mathcal{V}_{\underline{\mathsf{R}}}} \to \mathsf{B}^{\mathcal{V}_{\underline{\mathsf{R}}}} & (\mathsf{A} \to \underline{\mathsf{B}})^{\mathcal{C}_{\underline{\mathsf{R}}}} &=& !\,\mathsf{A}^{\mathcal{V}_{\underline{\mathsf{R}}}} \otimes \underline{\mathsf{B}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \\
(!\mathsf{A})^{\mathcal{V}_{\underline{\mathsf{R}}}} &=& (!\mathsf{A})^{\mathcal{C}_{\underline{\mathsf{R}}}} \multimap \underline{\mathsf{R}} & (!\mathsf{A})^{\mathcal{C}_{\underline{\mathsf{R}}}} &=& \mathsf{A}^{\mathcal{V}_{\underline{\mathsf{R}}}} \to \underline{\mathsf{R}} \\
(\underline{\mathsf{A}} \multimap \underline{\mathsf{B}})^{\mathcal{V}_{\underline{\mathsf{R}}}} &=& \underline{\mathsf{B}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \multimap \underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}} & & & \\
(!\mathsf{A} \otimes \underline{\mathsf{B}})^{\mathcal{V}_{\underline{\mathsf{R}}}} &=& (!\mathsf{A} \otimes \underline{\mathsf{B}})^{\mathcal{C}_{\underline{\mathsf{R}}}} \multimap \underline{\mathsf{R}} & (!\mathsf{A} \otimes \underline{\mathsf{B}})^{\mathcal{C}_{\underline{\mathsf{R}}}} &=& \mathsf{A}^{\mathcal{V}_{\underline{\mathsf{R}}}} \to \underline{\mathsf{B}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \\
\underline{0}^{\mathcal{V}_{\underline{\mathsf{R}}}} &=& \underline{0}^{\mathcal{C}_{\underline{\mathsf{R}}}} \multimap \underline{\mathsf{R}} & \underline{0}^{\mathcal{C}_{\underline{\mathsf{R}}}} &=& 1 \\
(\underline{\mathsf{A}} \oplus \underline{\mathsf{B}})^{\mathcal{V}_{\underline{\mathsf{R}}}} &=& (\underline{\mathsf{A}} \oplus \underline{\mathsf{B}})^{\mathcal{C}_{\underline{\mathsf{R}}}} \multimap \underline{\mathsf{R}} & (\underline{\mathsf{A}} \oplus \underline{\mathsf{B}})^{\mathcal{C}_{\underline{\mathsf{R}}}} &=& \underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \times \underline{\mathsf{B}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \\
\underline{\mathsf{R}}^{\mathcal{V}_{\underline{\mathsf{R}}}} &=& \underline{\mathsf{R}} & \underline{\mathsf{R}}^{\mathcal{C}_{\underline{\mathsf{R}}}} &=& !\,1
\end{array}$$

Fig. 4. Linearly-used CPS translation of enriched effect calculus types

Theorem 1 (Involution property). *For every value type* A*, there is an isomorphism* $j_{\mathsf{A}} \colon \mathsf{A}^{\mathcal{V}_{\underline{\mathsf{R}}}\mathcal{V}_{\underline{\mathsf{R}}}} \to \mathsf{A}$*, and, for every computation type* $\underline{\mathsf{A}}$*, there is a linear isomorphism* $k_{\underline{\mathsf{A}}} \colon \underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}\mathcal{C}_{\underline{\mathsf{R}}}} \multimap \underline{\mathsf{A}}$*, such that:*

1. *If* $\Gamma \mid - \vdash t \colon \mathsf{A}$ *then* $t = j_{\mathsf{A}}\,(t^{\mathcal{V}_{\underline{\mathsf{R}}}\mathcal{V}_{\underline{\mathsf{R}}}})\,[j^{-1}(\Gamma)]$*, where we write* $[j^{-1}(\Gamma)]$ *for the substitution* $[j^{-1}(x) \,/\, x]_{x \colon \mathsf{C} \in \Gamma}$*.*
2. *If* $\Gamma \mid z \colon \underline{\mathsf{A}} \vdash t \colon \underline{\mathsf{B}}$ *then* $t = k_{\underline{\mathsf{B}}}\,(t^{\mathcal{C}_{\underline{\mathsf{R}}}\mathcal{C}_{\underline{\mathsf{R}}}})\,[k_{\underline{\mathsf{A}}}^{-1}(z) \,/\, z]\,[j^{-1}(\Gamma)]$*.*

For the proof, suitable isomorphisms j_{A} and $k_{\underline{\mathsf{A}}}$ are easily defined by induction on the structure of types. Given these, it should, in principle, be a routine unwinding of definitions to verify the equalities in items 1 and 2. However, because of the intricacy of Figure 5, the terms $t^{\mathcal{V}_{\underline{\mathsf{R}}}\mathcal{V}_{\underline{\mathsf{R}}}}$ and $t^{\mathcal{C}_{\underline{\mathsf{R}}}\mathcal{C}_{\underline{\mathsf{R}}}}$ are extremely unwieldy, and such a direct verification seems impractical. Instead, we shall give a more manageable (as well as more conceptually appealing) proof of the theorem using categorical model theory, in Section 5.

Theorem 2 (Full-completeness). *The linearly-used CPS translation is full and faithful, i.e.:*

1. *If* $\Gamma \mid - \vdash t, u \colon \mathsf{A}$ *are two terms of the same type and* $t^{\mathcal{V}_{\underline{\mathsf{R}}}} = u^{\mathcal{V}_{\underline{\mathsf{R}}}}$ *(i.e., the equality is provable in context) then* $t = u$*.*
2. *If* $\Gamma \mid z \colon \underline{\mathsf{A}} \vdash t, u \colon \underline{\mathsf{B}}$ *and* $t^{\mathcal{C}_{\underline{\mathsf{R}}}} = u^{\mathcal{C}_{\underline{\mathsf{R}}}}$ *then* $t = u$*.*
3. *If* $\Gamma^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid - \vdash t \colon \mathsf{A}^{\mathcal{V}_{\underline{\mathsf{R}}}}$ *then there exists* $\Gamma \mid - \vdash u \colon \mathsf{A}$ *such that* $t = u^{\mathcal{V}_{\underline{\mathsf{R}}}}$*.*
4. *If* $\Gamma^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid z \colon \underline{\mathsf{B}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \vdash t \colon \underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}}$ *is a valid typing judgement, then there exists* $\Gamma \mid z \colon \underline{\mathsf{A}} \vdash u \colon \underline{\mathsf{B}}$ *such that* $t = u^{\mathcal{C}_{\underline{\mathsf{R}}}}$*.*

Proof. For statement 1, suppose $\Gamma \mid - \vdash t, u \colon \mathsf{A}$ and $t^{\mathcal{V}_{\underline{\mathsf{R}}}} = u^{\mathcal{V}_{\underline{\mathsf{R}}}}$. Then:

$$\begin{array}{rll}
t &= j_{\mathsf{A}}\,(t^{\mathcal{V}_{\underline{\mathsf{R}}}\mathcal{V}_{\underline{\mathsf{R}}}})\,[j_{\mathsf{C}}^{-1}(x) \,/\, x]_{x \colon \mathsf{C} \in \Gamma} & \text{(Theorem 1.1)} \\
&= j_{\mathsf{A}}\,(u^{\mathcal{V}_{\underline{\mathsf{R}}}\mathcal{V}_{\underline{\mathsf{R}}}})\,[j_{\mathsf{C}}^{-1}(x) \,/\, x]_{x \colon \mathsf{C} \in \Gamma} & \text{(Proposition 1)} \\
&= u & \text{(Theorem 1.1)} \ .
\end{array}$$

$$
\begin{aligned}
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}}, x\colon \mathsf{A}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid - \vdash x^{\mathcal{V}_{\underline{\mathsf{R}}}} &= x\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid z\colon \underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \vdash z^{\mathcal{C}_{\underline{\mathsf{R}}}} &= z\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid - \vdash *^{\mathcal{V}_{\underline{\mathsf{R}}}} &= *\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid z\colon \underline{0} \vdash *^{\mathcal{C}_{\underline{\mathsf{R}}}} &= \underline{\mathrm{image}}(z)\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid - \vdash \langle t, u\rangle^{\mathcal{V}_{\underline{\mathsf{R}}}} &= \langle t^{\mathcal{V}_{\underline{\mathsf{R}}}}, u^{\mathcal{V}_{\underline{\mathsf{R}}}}\rangle\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid z\colon \underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \oplus \underline{\mathsf{B}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \vdash \langle t, u\rangle^{\mathcal{C}_{\underline{\mathsf{R}}}} &= \underline{\mathrm{case}}\, z \,\mathrm{of}\, (\underline{\mathrm{inl}}(z).\, t^{\mathcal{C}_{\underline{\mathsf{R}}}}; \underline{\mathrm{inr}}(z).\, u^{\mathcal{C}_{\underline{\mathsf{R}}}})\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid - \vdash \mathrm{fst}(t)^{\mathcal{V}_{\underline{\mathsf{R}}}} &= \mathrm{fst}(t^{\mathcal{V}_{\underline{\mathsf{R}}}})\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid z\colon \underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \vdash \mathrm{fst}(t)^{\mathcal{C}_{\underline{\mathsf{R}}}} &= t^{\mathcal{C}_{\underline{\mathsf{R}}}}[\underline{\mathrm{inl}}(z)/z]\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid - \vdash \mathrm{snd}(t)^{\mathcal{V}_{\underline{\mathsf{R}}}} &= \mathrm{snd}(t^{\mathcal{V}_{\underline{\mathsf{R}}}})\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid z\colon \underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \vdash \mathrm{snd}(t)^{\mathcal{C}_{\underline{\mathsf{R}}}} &= t^{\mathcal{C}_{\underline{\mathsf{R}}}}[\underline{\mathrm{inr}}(z)/z]\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid - \vdash (\lambda x\colon \mathsf{A}.\, t)^{\mathcal{V}_{\underline{\mathsf{R}}}} &= \lambda x\colon \mathsf{A}^{\mathcal{V}_{\underline{\mathsf{R}}}}.\, t^{\mathcal{V}_{\underline{\mathsf{R}}}}\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid z\colon !\mathsf{A}^{\mathcal{V}_{\underline{\mathsf{R}}}} \otimes \underline{\mathsf{B}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \vdash (\lambda x\colon \mathsf{A}.\, t)^{\mathcal{C}_{\underline{\mathsf{R}}}} &= \mathrm{let}\, !x \otimes z \,\mathrm{be}\, z \,\mathrm{in}\, (t^{\mathcal{C}_{\underline{\mathsf{R}}}})\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid - \vdash (s(t))^{\mathcal{V}_{\underline{\mathsf{R}}}} &= s^{\mathcal{V}_{\underline{\mathsf{R}}}}(t^{\mathcal{V}_{\underline{\mathsf{R}}}})\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid z\colon \underline{\mathsf{B}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \vdash (s(t))^{\mathcal{C}_{\underline{\mathsf{R}}}} &= s^{\mathcal{C}_{\underline{\mathsf{R}}}}\,[!t^{\mathcal{V}_{\underline{\mathsf{R}}}} \otimes z \,/\, z]\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid - \vdash (!\, t)^{\mathcal{V}_{\underline{\mathsf{R}}}} &= \underline{\lambda} f\colon \mathsf{A}^{\mathcal{V}_{\underline{\mathsf{R}}}} \to \underline{\mathsf{R}}.\, f(t^{\mathcal{V}_{\underline{\mathsf{R}}}})\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid - \vdash (\mathrm{let}\, !\, x \,\mathrm{be}\, t \,\mathrm{in}\, u)^{\mathcal{V}_{\underline{\mathsf{R}}}} &= i_{\underline{\mathsf{B}}}(\underline{\lambda} z\colon \underline{\mathsf{B}}^{\mathcal{C}_{\underline{\mathsf{R}}}}.\, t^{\mathcal{V}_{\underline{\mathsf{R}}}}[\lambda x\colon A^{\mathcal{V}_{\underline{\mathsf{R}}}}.\, i_{\underline{\mathsf{B}}}^{-1}(u^{\mathcal{V}_{\underline{\mathsf{R}}}})[z]])\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid z\colon \underline{\mathsf{B}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \vdash (\mathrm{let}\, !\, x \,\mathrm{be}\, t \,\mathrm{in}\, u)^{\mathcal{C}_{\underline{\mathsf{R}}}} &= t^{\mathcal{C}_{\underline{\mathsf{R}}}}\,[(\lambda x\colon A^{\mathcal{V}_{\underline{\mathsf{R}}}}.\, i_{\underline{\mathsf{B}}}^{-1}(u^{\mathcal{V}_{\underline{\mathsf{R}}}})[z]) \,/ z]\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid - \vdash (\underline{\lambda} z\colon \underline{\mathsf{A}}.\, t)^{\mathcal{V}_{\underline{\mathsf{R}}}} &= \underline{\lambda} z\colon \underline{\mathsf{B}}^{\mathcal{C}_{\underline{\mathsf{R}}}}.\, t^{\mathcal{C}_{\underline{\mathsf{R}}}}\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid - \vdash (s[t])^{\mathcal{V}_{\underline{\mathsf{R}}}} &= i_{\underline{\mathsf{B}}}(\underline{\lambda} z\colon \underline{\mathsf{B}}^{\mathcal{C}_{\underline{\mathsf{R}}}}.\, i_{\underline{\mathsf{A}}}^{-1}(t^{\mathcal{V}_{\underline{\mathsf{R}}}})[s^{\mathcal{V}_{\underline{\mathsf{R}}}}[z]])\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid z\colon \underline{\mathsf{B}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \vdash (s[t])^{\mathcal{C}_{\underline{\mathsf{R}}}} &= t^{\mathcal{C}_{\underline{\mathsf{R}}}}\,[s^{\mathcal{V}_{\underline{\mathsf{R}}}}[z] \,/\, z]\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid - \vdash (!t \otimes u)^{\mathcal{V}_{\underline{\mathsf{R}}}} &= \underline{\lambda} z\colon \mathsf{A}^{\mathcal{V}_{\underline{\mathsf{R}}}} \to \underline{\mathsf{B}}^{\mathcal{C}_{\underline{\mathsf{R}}}}.\, i_{\underline{\mathsf{B}}}^{-1}(u^{\mathcal{V}_{\underline{\mathsf{R}}}})[z(t^{\mathcal{V}_{\underline{\mathsf{R}}}})]\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid z\colon \mathsf{A}^{\mathcal{V}_{\underline{\mathsf{R}}}} \to \underline{\mathsf{B}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \vdash (!t \otimes u)^{\mathcal{C}_{\underline{\mathsf{R}}}} &= u^{\mathcal{C}_{\underline{\mathsf{R}}}}\,[z(t^{\mathcal{V}_{\underline{\mathsf{R}}}}) \,/\, z]\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid - \vdash (\mathrm{let}\, !x \otimes z \,\mathrm{be}\, s \,\mathrm{in}\, t)^{\mathcal{V}_{\underline{\mathsf{R}}}} &= i_{\underline{\mathsf{C}}}(\underline{\lambda} z\colon \underline{\mathsf{C}}^{\mathcal{C}_{\underline{\mathsf{R}}}}.\, s^{\mathcal{V}_{\underline{\mathsf{R}}}}\,[\lambda x\colon A^{\mathcal{V}_{\underline{\mathsf{R}}}}.\, t^{\mathcal{C}_{\underline{\mathsf{R}}}}])\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid z\colon \underline{\mathsf{C}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \vdash (\mathrm{let}\, !x \otimes z \,\mathrm{be}\, s \,\mathrm{in}\, t)^{\mathcal{C}_{\underline{\mathsf{R}}}} &= s^{\mathcal{C}_{\underline{\mathsf{R}}}}\,[(\lambda x\colon \mathsf{A}^{\mathcal{V}_{\underline{\mathsf{R}}}}.\, t^{\mathcal{C}_{\underline{\mathsf{R}}}}) \,/\, z]\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid - \vdash (\underline{\mathrm{image}}(t))^{\mathcal{V}_{\underline{\mathsf{R}}}} &= i_{\underline{\mathsf{A}}}(\underline{\lambda} z\colon \underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}}.\, t^{\mathcal{V}_{\underline{\mathsf{R}}}}[*])\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid z\colon \underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \vdash (\underline{\mathrm{image}}(t))^{\mathcal{C}_{\underline{\mathsf{R}}}} &= t^{\mathcal{C}_{\underline{\mathsf{R}}}}\,[*/z]\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid - \vdash (\underline{\mathrm{inl}}(t))^{\mathcal{V}_{\underline{\mathsf{R}}}} &= \underline{\lambda} z\colon \underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \times \underline{\mathsf{B}}^{\mathcal{C}_{\underline{\mathsf{R}}}}.\ i_{\underline{\mathsf{A}}}^{-1}(t^{\mathcal{V}_{\underline{\mathsf{R}}}})[\mathrm{fst}(z)]\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid z\colon \underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \times \underline{\mathsf{B}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \vdash (\underline{\mathrm{inl}}(t))^{\mathcal{C}_{\underline{\mathsf{R}}}} &= t^{\mathcal{C}_{\underline{\mathsf{R}}}}\,[\mathrm{fst}(z) \,/\, z]\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid - \vdash (\underline{\mathrm{inr}}(t))^{\mathcal{V}_{\underline{\mathsf{R}}}} &= \underline{\lambda} z\colon \underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \times \underline{\mathsf{B}}^{\mathcal{C}_{\underline{\mathsf{R}}}}.\ i_{\underline{\mathsf{A}}}^{-1}(t^{\mathcal{V}_{\underline{\mathsf{R}}}})[\mathrm{snd}(z)]\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid z\colon \underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \times \underline{\mathsf{B}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \vdash (\underline{\mathrm{inr}}(t))^{\mathcal{C}_{\underline{\mathsf{R}}}} &= t^{\mathcal{C}_{\underline{\mathsf{R}}}}\,[\mathrm{snd}(z) \,/\, z]\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid - \vdash (\underline{\mathrm{case}}\, s \,\mathrm{of}\, (\underline{\mathrm{inl}}(x).\, t; \underline{\mathrm{inr}}(y).\, u))^{\mathcal{V}_{\underline{\mathsf{R}}}} &= i_{\underline{\mathsf{C}}}(\underline{\lambda} z\colon \underline{\mathsf{C}}^{\mathcal{C}_{\underline{\mathsf{R}}}}.\ s^{\mathcal{V}_{\underline{\mathsf{R}}}}\,[\langle t^{\mathcal{C}_{\underline{\mathsf{R}}}}, u^{\mathcal{C}_{\underline{\mathsf{R}}}}\rangle])\\
\mathit{\Gamma}^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid z\colon \underline{\mathsf{C}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \vdash (\underline{\mathrm{case}}\, s \,\mathrm{of}\, (\underline{\mathrm{inl}}(x).\, t; \underline{\mathrm{inr}}(y).\, u))^{\mathcal{C}_{\underline{\mathsf{R}}}} &= s^{\mathcal{C}_{\underline{\mathsf{R}}}}\,[\langle t^{\mathcal{C}_{\underline{\mathsf{R}}}}, u^{\mathcal{C}_{\underline{\mathsf{R}}}}\rangle \,/\, z]
\end{aligned}
$$

Fig. 5. Linearly-used CPS translation of terms

For statement 3, suppose $\Gamma^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid - \vdash t\colon \mathsf{A}^{\mathcal{V}_{\underline{\mathsf{R}}}}$. Define $u = j_{\mathsf{A}}(t^{\mathcal{V}_{\underline{\mathsf{R}}}})\,[j^{-1}(\Gamma)]$. Then:

$$\begin{aligned}
u^{\mathcal{V}_{\underline{\mathsf{R}}}} &= j_{\mathsf{A}^{\mathcal{V}_{\underline{\mathsf{R}}}}}(u^{\mathcal{V}_{\underline{\mathsf{R}}}\mathcal{V}_{\underline{\mathsf{R}}}\mathcal{V}_{\underline{\mathsf{R}}}})\,[j^{-1}(\Gamma^{\mathcal{V}_{\underline{\mathsf{R}}}})] && \text{(Theorem 1.1)}\\
&= j_{\mathsf{A}^{\mathcal{V}_{\underline{\mathsf{R}}}}}((j_{\underline{\mathsf{A}}}^{-1}(u)\,[j(\Gamma)])^{\mathcal{V}_{\underline{\mathsf{R}}}})\,[j^{-1}(\Gamma^{\mathcal{V}_{\underline{\mathsf{R}}}})] && \text{(Theorem 1.1)}\\
&= j_{\mathsf{A}^{\mathcal{V}_{\underline{\mathsf{R}}}}}(t^{\mathcal{V}_{\underline{\mathsf{R}}}\mathcal{V}_{\underline{\mathsf{R}}}})\,[j^{-1}(\Gamma^{\mathcal{V}_{\underline{\mathsf{R}}}})] && \text{(Definition } u\text{)}\\
&= t && \text{(Theorem 1.1)}\ .
\end{aligned}$$

The proofs of items 2 and 4 are similar. □

We end this section by showing how the linearly-used CPS translation of the enriched effect calculus into itself subsumes the call-by-value and call-by-name linearly-used CPS translations of Figure 3, from [7,6]. Indeed, these are obtained uniformly by precomposing the translation on the enriched effect calculus with the standard call-by-value and call-by-name translations.

Theorem 3 (Recovering $(\cdot)^{\mathrm{cbv}_{\underline{\mathsf{R}}}}$ and $(\cdot)^{\mathrm{cbn}_{\underline{\mathsf{R}}}}$).

1. *For every simple type σ, we have $\sigma^{\mathrm{cbv}_{\underline{\mathsf{R}}}} = (\sigma^{\mathrm{cbv}})^{\mathcal{V}_{\underline{\mathsf{R}}}}$; and, for every simply-typed term $\Theta \vdash t\colon \sigma$, we have $t^{\mathrm{cbv}_{\underline{\mathsf{R}}}} = (t^{\mathrm{cbv}})^{\mathcal{V}_{\underline{\mathsf{R}}}}$.*
2. *For every simple type σ, there is a linear isomorphism $r_\sigma\colon (\sigma^{\mathrm{cbn}})^{\mathcal{C}_{\underline{\mathsf{R}}}} \multimap \sigma^{\mathrm{cbn}_{\underline{\mathsf{R}}}}$ such that, for every $\Theta \vdash t\colon \sigma$, it holds that $t^{\mathrm{cbn}_{\underline{\mathsf{R}}}} = p^{-1}_{\tau^{\mathrm{cbn}}}((t^{\mathrm{cbn}})^{\mathcal{V}_{\underline{\mathsf{R}}}})[p(\Theta)]$, where p_σ is the isomorphism $i_{\sigma^{\mathrm{cbn}}} \circ (r_\sigma \multimap \underline{\mathsf{R}})\colon (\sigma^{\mathrm{cbn}_{\underline{\mathsf{R}}}} \multimap \underline{\mathsf{R}}) \to (\sigma^{\mathrm{cbn}})^{\mathcal{V}_{\underline{\mathsf{R}}}}$.*

That proofs are by induction on the structure of σ and t.

Statement 2 of the theorem is more complex than one would like because the types $(\sigma^{\mathrm{cbn}})^{\mathcal{C}_{\underline{\mathsf{R}}}}$ and $\sigma^{\mathrm{cbn}_{\underline{\mathsf{R}}}}$ are not syntactically identical. The difficulty derives from $((\sigma \to \tau)^{\mathrm{cbn}})^{\mathcal{C}_{\underline{\mathsf{R}}}} = !\,(\sigma^{\mathrm{cbn}})^{\mathcal{V}_{\underline{\mathsf{R}}}} \otimes (\tau^{\mathrm{cbn}})^{\mathcal{C}_{\underline{\mathsf{R}}}} \cong !\,((\sigma^{\mathrm{cbn}})^{\mathcal{C}_{\underline{\mathsf{R}}}} \multimap \underline{\mathsf{R}}) \otimes (\tau^{\mathrm{cbn}})^{\mathcal{C}_{\underline{\mathsf{R}}}}$. But identity does not hold because we do not have $\underline{\mathsf{A}}^{\mathcal{V}_{\underline{\mathsf{R}}}} = \underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \multimap \underline{\mathsf{R}}$. This technicality could be avoided by changing to a syntax for EEC with no overloading between value and computation types, as in Levy's CBPV [10]. However, then the syntax of EEC terms would become more complex.

Hasegawa [6] proves full-completeness for the call-by-value linearly-used CPS translation of Moggi's computational λ-calculus [11] into ILL. He also has a similar result for the call-by-name translation of [7] restricted to the simply-typed λ-calculus (private communication). The corollary below adapts this to our identical translations into EEC. In fact, our results follow from Hasegawa's (and not vice versa!). What is interesting is our method of proof, which applies the results we have established for our canonical translation. In the statement, we write λ_c for the equational theory of Moggi's computational λ-calculus.

Corollary 1 (Full completeness of $(\cdot)^{\mathrm{cbv}_{\underline{\mathsf{R}}}}$ and $(\cdot)^{\mathrm{cbn}_{\underline{\mathsf{R}}}}$).

1. *If $\Theta \vdash t, u\colon \tau$ and $\Theta^{\mathrm{cbv}_{\underline{\mathsf{R}}}} \mid - \vdash t^{\mathrm{cbv}_{\underline{\mathsf{R}}}} = u^{\mathrm{cbv}_{\underline{\mathsf{R}}}}\colon (\tau^{\mathrm{cbv}_{\underline{\mathsf{R}}}} \to \underline{\mathsf{R}}) \multimap \underline{\mathsf{R}}$ then $\Theta \vdash_{\lambda_c} t = u\colon \tau$.*
2. *If $\Theta^{\mathrm{cbv}_{\underline{\mathsf{R}}}} \mid - \vdash u\colon (\tau^{\mathrm{cbv}_{\underline{\mathsf{R}}}} \to \underline{\mathsf{R}}) \multimap \underline{\mathsf{R}}$ then there exists a simply-typed term $\Theta \vdash t\colon \tau$ such that $u = t^{\mathrm{cbv}_{\underline{\mathsf{R}}}}$.*

3. If $\Theta \vdash t, u \colon \tau$ and $\Theta^{\mathrm{cbn}_{\underline{\mathsf{R}}}} \mid - \,\vdash\, t^{\mathrm{cbn}_{\underline{\mathsf{R}}}} = u^{\mathrm{cbn}_{\underline{\mathsf{R}}}} \colon \tau^{\mathrm{cbn}_{\underline{\mathsf{R}}}}$ then $\Theta \vdash_{\beta\eta} t = u \colon \tau$.
4. If $\Theta^{\mathrm{cbn}_{\underline{\mathsf{R}}}} \mid - \,\vdash\, u \colon \tau^{\mathrm{cbn}_{\underline{\mathsf{R}}}}$ then there exists a simply-typed term $\Theta \vdash t \colon \tau$ such that $u = t^{\mathrm{cbn}_{\underline{\mathsf{R}}}}$.

Proof. We only give arguments for call-by-value. It can be shown that the mapping $(\cdot)^{\mathrm{cbv}}$ from the computational λ-calculus into EEC is full and faithful.[1] For statement 1, suppose $\Theta^{\mathrm{cbv}_{\underline{\mathsf{R}}}} \mid - \,\vdash\, t^{\mathrm{cbv}_{\underline{\mathsf{R}}}} = u^{\mathrm{cbv}_{\underline{\mathsf{R}}}} \colon (\tau^{\mathrm{cbv}_{\underline{\mathsf{R}}}} \to \underline{\mathsf{R}}) \multimap \underline{\mathsf{R}}$. Then, by Theorem 3.1, $(\Theta^{\mathrm{cbv}})^{\mathcal{V}_{\underline{\mathsf{R}}}} \mid - \,\vdash\, (t^{\mathrm{cbv}})^{\mathcal{V}_{\underline{\mathsf{R}}}} = (u^{\mathrm{cbv}})^{\mathcal{V}_{\underline{\mathsf{R}}}} \colon ((\tau^{\mathrm{cbv}})^{\mathcal{V}_{\underline{\mathsf{R}}}} \to \underline{\mathsf{R}}) \multimap \underline{\mathsf{R}}$. But we have $((\tau^{\mathrm{cbv}})^{\mathcal{V}_{\underline{\mathsf{R}}}} \to \underline{\mathsf{R}}) \multimap \underline{\mathsf{R}} \;=\; (!\tau^{\mathrm{cbv}})^{\mathcal{V}_{\underline{\mathsf{R}}}}$. So, by Theorem 2.1, $\Theta^{\mathrm{cbv}} \mid - \,\vdash\, t^{\mathrm{cbv}} = u^{\mathrm{cbv}} \colon !\,\tau^{\mathrm{cbv}}$ and statement 1 follows from the faithfulness of $(\cdot)^{\mathrm{cbv}}$. For statement 2, by Theorem 2.3 there exists u' such that $\Theta^{\mathrm{cbv}} \mid - \,\vdash\, u' \colon !\,\tau^{\mathrm{cbv}}$ and $(u')^{\mathcal{V}_{\underline{\mathsf{R}}}} = u$. Statement 2 now follows from the fullness of $(\cdot)^{\mathrm{cbv}}$. □

4 Models

We review the notion of *model* of the enriched effect calculus [4]. This is defined in terms of *enriched category theory* [8]. Recall that, given a chosen monoidal category $\mathbf{V}$, a $\mathbf{V}$*-enriched category* (or, $\mathbf{V}$*-category*) $\mathbf{C}$ is given by a collection of objects, with, for every pair of objects $A, B \in \mathbf{C}$, a specified *hom-object* $\mathbf{C}(A, B) \in \mathbf{V}$, together with families of morphisms in $\mathbf{V}$ supplying $\mathbf{C}$ with its identity maps and an associative composition. When $\mathbf{V}$ is the category of sets, a $\mathbf{V}$-category is just an ordinary (locally small) category. When modelling the enriched effect calculus, it is natural to ask for the category of linear maps between computation types to be enriched over the category of value types, since the value types $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ act as hom-objects. In the sequel, we shall assume some basic knowledge of enriched category theory; see [8] for a detailed account.

We shall consider enrichment only with respect to categories $\mathbf{V}$ that are cartesian closed (we write B^A or $[A \to B]$ for functions spaces). Any such category is self-enriched. We say that a $\mathbf{V}$-enriched category $\mathbf{C}$ has *(*$\mathbf{V}$*-)powers*[2] if, for all objects $A \in \mathbf{V}$ and $\underline{B} \in \mathbf{C}$, there exists an object $\underline{B}^A \in \mathbf{C}$ with isomorphisms

$$\mathbf{C}(\underline{C}, \underline{B}^A) \longrightarrow [A \to \mathbf{C}(\underline{C}, \underline{B})]$$

$\mathbf{V}$-natural in objects $\underline{C}$ of $\mathbf{C}$. The dual property is that of having *(*$\mathbf{V}$*-)copowers*: for all $A \in \mathbf{V}$ and $\underline{B} \in \mathbf{C}$, there must exist an object $A \cdot \underline{B}$ of $\mathbf{C}$ with isomorphisms

$$\mathbf{C}(A \cdot \underline{B}, \underline{C}) \longrightarrow [A \to \mathbf{C}(\underline{B}, \underline{C})]$$

$\mathbf{V}$-natural in $\underline{C}$. An *enriched* adjunction $F \dashv U$ between $\mathbf{V}$-functors $F \colon \mathbf{D} \to \mathbf{C}$ and $U \colon \mathbf{C} \to \mathbf{D}$ requires the existence of isomorphisms in $\mathbf{V}$

$$\mathbf{C}(F(A), \underline{B}) \longrightarrow [A \to U(\underline{B})]$$

which are $\mathbf{V}$-natural in A and $\underline{B}$.

[1] E.g., the term model of λ_c [11] fully embeds in a model of CBPV [10], and every model of CBPV fully embeds in a model of EEC [4] (the assumptions of sums [10] and cartesian closedness [4] are not needed for the embeddings).

[2] Kelly writes *cotensors* (resp., *tensors*) where we write *powers* (resp., *copowers*).

Definition 1. An *enriched-effect-calculus model* comprises: a cartesian closed category $\mathbf{V}$, with $\mathbf{V}$-enriched finite products, coproducts, powers, copowers, and a $\mathbf{V}$-adjunction $F \dashv U \colon \mathbf{C} \to \mathbf{V}$.

We shall loosely specify models as $F \dashv U \colon \mathbf{C} \to \mathbf{V}$, without making the other structure (which is, in any case, determined up to isomorphism) explicit.

To interpret the enriched effect calculus in a model, value types A are interpreted as objects $\mathbf{V}[\![\mathsf{A}]\!]$ of $\mathbf{V}$, and computation types $\underline{\mathsf{A}}$ are interpreted as pairs $(\mathbf{C}[\![\underline{\mathsf{A}}]\!], s_{\underline{\mathsf{A}}})$ where $\mathbf{C}[\![\underline{\mathsf{A}}]\!]$ is an object of $\mathbf{C}$, and $s_{\underline{\mathsf{A}}} \colon U(\mathbf{C}[\![\underline{\mathsf{A}}]\!]) \to \mathbf{V}[\![\underline{\mathsf{A}}]\!]$ is an isomorphism in $\mathbf{V}$. The reader is referred to [4] for further details.

In [4], the equational theory of the enriched effect calculus is shown to be sound and complete with respect to interpretations in models. Completeness is proved via a syntactic model construction. Since this model will play an important role in Section 5, we recall its definition.

The category $\mathbf{V}_{\mathrm{Syn}}$ has as objects value types and as morphisms from A to B terms of the form $x \colon \mathsf{A} \mid - \vdash t \colon \mathsf{B}$ identified up to the equality theory of EEC. Composition is given by substitution. The $\mathbf{V}_{\mathrm{Syn}}$-enriched category $\mathbf{C}_{\mathrm{Syn}}$ has as objects all computation types and as object of morphisms from $\underline{\mathsf{A}}$ to $\underline{\mathsf{B}}$ the value type $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$. Powers and copowers are given by the constructions $\mathsf{A} \to \underline{\mathsf{B}}$ and $!\mathsf{A} \otimes \underline{\mathsf{B}}$, respectively. The right adjoint is the forgetful functor from computation types to value types, with its action on morphisms defined by the coercion $(\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}) \to (\underline{\mathsf{A}} \to \underline{\mathsf{B}})$. The left adjoint is given by $!$.

The syntactic model is characterised by a universal property. To formulate this, one needs a notion of morphism of models. Essentially, a morphism from $F \dashv U \colon \mathbf{C} \to \mathbf{V}$ to $F' \dashv U' \colon \mathbf{C}' \to \mathbf{V}'$, is given by a pair of functors $S \colon \mathbf{V} \to \mathbf{V}'$ and $T \colon \mathbf{C} \to \mathbf{C}'$ (jointly) preserving the structure. There are, however, two complicating factors. First, morphisms need only preserve structure up to isomorphism rather than identity. (This choice is both mathematically natural and essential to the results of Section 5.) Second, $\mathbf{C}$ and $\mathbf{C}'$ are enriched over two different categories, which leads to subtle requirements regarding how the functor T enriches. These issues are treated in [4]. There is a notion of coherent natural isomorphism (henceforth, *cni*) between morphisms of models; hence, the category of models is naturally enriched in that of groupoids, **Grpd**.

Theorem 4 ([4, Theorem 3]). *Given an enriched-effect-calculus model $F \dashv U \colon \mathbf{C} \to \mathbf{V}$ and families of objects, $\mathbf{V}[\![\alpha]\!]$ in $\mathbf{V}$ and $\mathbf{C}[\![\underline{\alpha}]\!]$ in $\mathbf{C}$, indexed by type constants, there exists a morphism of models from the syntactic model $F_{\mathrm{Syn}} \dashv U_{\mathrm{Syn}} \colon \mathbf{C}_{\mathrm{Syn}} \to \mathbf{V}_{\mathrm{Syn}}$ to $F \dashv U \colon \mathbf{C} \to \mathbf{V}$ that extends the given interpretation of type constants, and this morphism is unique up to cni.*

We remark, that the proof of the theorem produces a morphism that preserves type constants up to equality. Nevertheless, uniqueness (up to isomorphism) holds relative to the wider class of morphisms that only preserve type constants up to isomorphism.

5 Dual Models

Given a model $F \dashv U\colon \mathbf{C} \to \mathbf{V}$ of the enriched effect calculus and an object $\underline{\mathsf{R}}$ of $\mathbf{C}$, one can define another model $F^{\underline{\mathsf{R}}} \dashv U^{\underline{\mathsf{R}}}\colon \mathbf{C}^{\mathrm{op}} \to \mathbf{V}$ by:

$$F^{\underline{\mathsf{R}}} := \underline{\mathsf{R}}^{(-)} \qquad\qquad U^{\underline{\mathsf{R}}} := \mathbf{C}(-,\underline{\mathsf{R}})\ .$$

Indeed, $\mathbf{C}^{\mathrm{op}}$ is trivially $\mathbf{V}$-enriched. Its enriched powers and copowers are given by copowers and powers in $\mathbf{C}$, respectively; similarly also products and coproducts. And it is standard that $\underline{\mathsf{R}}^{(-)} \dashv \mathbf{C}(-,\underline{\mathsf{R}})$ is an enriched adjunction. We call the constructed model the $\underline{\mathsf{R}}$-*dual* of $F \dashv U\colon \mathbf{C} \to \mathbf{V}$.

There are obvious similarities between the dual model construction and the linearly-used CPS translation on enriched effect calculus types, defined in Fig. 4. For example, $F^{\underline{\mathsf{R}}}$ corresponds to the $(-) \to \underline{\mathsf{R}}$ action of the translation of $!(-)$ to computation types. Similarly, $U^{\underline{\mathsf{R}}}$ implements the feature that the translation, $\underline{\mathsf{A}}^{\mathcal{V}_{\underline{\mathsf{R}}}}$, of $\underline{\mathsf{A}}$ as a value type is obtained (modulo the isomorphism $i_{\underline{\mathsf{A}}}$) as $\underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}} \multimap \underline{\mathsf{R}}$, where $\underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}}$ is the interpretation of $\underline{\mathsf{A}}$ as a computation type. Furthermore, the induced monad $\mathbf{C}(\underline{\mathsf{R}}^{(-)},\underline{\mathsf{R}})$ on $\mathbf{V}$ corresponds to the linearly-used continuations monad $((-) \to \underline{\mathsf{R}}) \multimap \underline{\mathsf{R}}$.

Monads of the form $\mathbf{C}(\underline{\mathsf{R}}^{(-)},\underline{\mathsf{R}})$ have been called *dual monads* by Lawvere [9], who considers them in the case that $U\colon \mathbf{C} \to \mathbf{V}$ exhibits $\mathbf{C}$ as a category of algebras for a monad over its enriching category. Our *dual model* construction performs the analogous operation on general enriched adjunctions, rather than on monads. As we shall see below (Theorem 5), in our setting, the “dual” terminology is particularly appropriate.

Because of the choice of object $\underline{\mathsf{R}}$, the natural context for considering the dual model construction is as an operation on *pointed models*, $(F \dashv U\colon \mathbf{C} \to \mathbf{V},\ \underline{\mathsf{R}})$, where $\underline{\mathsf{R}}$ is a chosen object of $\mathbf{C}$. A morphism of pointed models is a morphism of models (S,T) together with an isomorphism $T\underline{\mathsf{R}} \to \underline{\mathsf{R}}'$. For a pointed model $\mathcal{M} = (F \dashv U\colon \mathbf{C} \to \mathbf{V},\ \underline{\mathsf{R}})$, the *dual pointed model* $\mathcal{M}^{\perp}$ is defined to be $(F^{\underline{\mathsf{R}}} \dashv U^{\underline{\mathsf{R}}}\colon \mathbf{C}^{\mathrm{op}} \to \mathbf{V},\ F1)$. The choice of $F1$ may seem arbitrary here, but it is crucial to Theorem 5 below.

Proposition 2. *The dual model construction is a* **Grpd**-*enriched functor on the* **Grpd**-*category of pointed models.*

Proof (outline). A morphism (S,T) is mapped to $(S,\ T^{\mathrm{op}}\colon \mathbf{C}^{\mathrm{op}} \to (\mathbf{C}')^{\mathrm{op}})$, and a cni $(\alpha\colon S \Rightarrow S',\ \beta\colon T \Rightarrow T')$ is mapped to $(\alpha,\ \beta^{-1}\colon T^{\mathrm{op}} \Rightarrow (T')^{\mathrm{op}})$. □

Theorem 5. *For every pointed model $\mathcal{M}$, we have a natural isomorphism of pointed models between $\mathcal{M}$ and $\mathcal{M}^{\perp\perp}$.*

Proof (outline). We show that the pair of identity functors is the required isomorphism from $\mathcal{M} \to \mathcal{M}^{\perp\perp}$. The double dual $\mathcal{M}^{\perp\perp}$ turns out to be $(F^{\perp\perp} \dashv U^{\perp\perp}\colon \mathbf{C} \to \mathbf{V},\ F^{\underline{\mathsf{R}}}1)$, where $F^{\perp\perp} = (-)\cdot F1$ (this is the power $(F1)^{(-)}$ calculated in $\mathbf{C}^{\mathrm{op}}$) and $U^{\perp\perp} = \mathbf{C}^{\mathrm{op}}(-,F1)$. Then we calculate: $F^{\perp\perp} = (-)\cdot F1 \cong F$, similarly $U^{\perp\perp} = \mathbf{C}^{\mathrm{op}}(-,F1) = \mathbf{C}(F1,-) \cong U$, and also $F^{\underline{\mathsf{R}}}1 = \underline{\mathsf{R}}^1 \cong \underline{\mathsf{R}}$. □

Models of intuitionistic linear logic supply a natural collection of models for the enriched effect calculus. A *linear/nonlinear model* [2] consists of a cartesian-closed category $\mathbf{V}$ (the *intuitionistic category*), a symmetric monoidal closed category $\mathbf{C}$ (the *linear category*), and a symmetric monoidal adjunction $F \dashv G\colon \mathbf{C} \to \mathbf{V}$. The model has *additives* if $\mathbf{C}$ has finite products and coproducts. It is *classical* if $\mathbf{C}$ is $*$-autonomous [1]. In [4, Proposition 1], it is shown that every linear/nonlinear model with additives is a model of the enriched effect calculus. Given an object $\underline{\mathsf{R}}$ of $\mathbf{C}$, the dual model is thus a model of the enriched effect calculus. However, it is not (in general) a linear/nonlinear model (e.g., $\mathbf{C}^{\mathrm{op}}$ need not be symmetric monoidal closed). Thus models of the enriched effect calculus are closed under a natural construction which is not available for models of intuitionistic linear logic. However, models of classical linear logic are preserved by the dual model construction, when $\underline{\mathsf{R}}$ is chosen to be the dualizing object. In fact, unsurprisingly, such models are self-dual.

Proposition 3. *If $F \dashv G\colon \mathbf{C} \to \mathbf{V}$ is a model of classical linear logic with additives then, defining the pointed model $\mathcal{M} = (F \dashv G\colon \mathbf{C} \to \mathbf{V}, \perp)$, where $\perp$ is the dualizing object, it holds that the $\mathcal{M}$ is (pointed) isomorphic to $\mathcal{M}^{\perp}$.*

Proof (outline). The isomorphism is given by $(Id\colon \mathbf{V} \to \mathbf{V}, (-)^*\colon \mathbf{C} \to \mathbf{C}^{\mathrm{op}})$ from $\mathcal{M}$ to $\mathcal{M}^{\perp}$, where $(-)^*$ is the $*$-autonomous dualizing functor. $\square$

We next exhibit a more surprising example of self-duality, the syntactic model. As in Section 3, we assume a computation-type constant $\underline{\mathsf{R}}$. We then consider the syntactic model together with $\underline{\mathsf{R}}$ as the chosen object of $\mathbf{C}_{\mathrm{Syn}}$:

$$\mathcal{M}_{\mathrm{Syn},\underline{\mathsf{R}}} \;:=\; (F_{\mathrm{Syn}} \dashv U_{\mathrm{Syn}}\colon \mathbf{C}_{\mathrm{Syn}} \to \mathbf{V}_{\mathrm{Syn}}, \underline{\mathsf{R}}) \;.$$

By Theorem 4, there is a unique morphism (up to cni) from the model $F_{\mathrm{Syn}} \dashv U_{\mathrm{Syn}}\colon \mathbf{C}_{\mathrm{Syn}} \to \mathbf{V}_{\mathrm{Syn}}$ to the model $F^{\underline{\mathsf{R}}}_{\mathrm{Syn}} \dashv U^{\underline{\mathsf{R}}}_{\mathrm{Syn}}\colon \mathbf{C}^{\mathrm{op}}_{\mathrm{Syn}} \to \mathbf{V}_{\mathrm{Syn}}$ given by functors $S_{\underline{\mathsf{R}}}\colon \mathbf{V}_{\mathrm{Syn}} \to \mathbf{V}_{\mathrm{Syn}}$ and $T_{\underline{\mathsf{R}}}\colon \mathbf{C}_{\mathrm{Syn}} \to \mathbf{C}^{\mathrm{op}}_{\mathrm{Syn}}$ satisfying:

$$S_{\underline{\mathsf{R}}}(\alpha) \;\cong\; \alpha \qquad\qquad T_{\underline{\mathsf{R}}}(\underline{\alpha}) \;\cong\; \begin{cases} !1 & \text{if } \underline{\alpha} = \underline{\mathsf{R}} \\ \underline{\alpha} & \text{otherwise} \end{cases}$$

Obviously, this is a morphism of pointed models from $\mathcal{M}_{\mathrm{Syn},\underline{\mathsf{R}}}$ to $(\mathcal{M}_{\mathrm{Syn},\underline{\mathsf{R}}})^{\perp}$.

Theorem 6. *The morphism $(S_{\underline{\mathsf{R}}}, T_{\underline{\mathsf{R}}})$ is an equivalence of pointed models between $\mathcal{M}_{\mathrm{Syn},\underline{\mathsf{R}}}$ and $(\mathcal{M}_{\mathrm{Syn},\underline{\mathsf{R}}})^{\perp}$.*

Proof (outline). By Proposition 2, $(S_{\underline{\mathsf{R}}}, (T_{\underline{\mathsf{R}}})^{\mathrm{op}})$ is a morphism from $(\mathcal{M}_{\mathrm{Syn},\underline{\mathsf{R}}})^{\perp}$ to $(\mathcal{M}_{\mathrm{Syn},\underline{\mathsf{R}}})^{\perp\perp}$ and, by (the proof of) Theorem 5, (Id, Id) is a morphism from $(\mathcal{M}_{\mathrm{Syn},\underline{\mathsf{R}}})^{\perp\perp}$ to $\mathcal{M}_{\mathrm{Syn},\underline{\mathsf{R}}}$. The composite endomorphism $(S_{\underline{\mathsf{R}}}\, S_{\underline{\mathsf{R}}}, (T_{\underline{\mathsf{R}}})^{\mathrm{op}}\, T_{\underline{\mathsf{R}}})$ on $\mathcal{M}_{\mathrm{Syn},\underline{\mathsf{R}}}$ obviously maps value type constants α and computation type constants $\underline{\alpha}$, other than $\underline{\mathsf{R}}$, to themselves. In the case of $\underline{\mathsf{R}}$, we have

$$(T_{\underline{\mathsf{R}}})^{\mathrm{op}}\, T_{\underline{\mathsf{R}}}(\underline{\mathsf{R}}) \cong (T_{\underline{\mathsf{R}}})^{\mathrm{op}}(!1) \;\cong\; (T_{\underline{\mathsf{R}}})(!1) \;\cong\; 1 \to \underline{\mathsf{R}} \;\cong\; \underline{\mathsf{R}} \;,$$

where the penultimate isomorphism is because $T_{\underline{\mathsf{R}}}$ maps the left adjoint $!(-)$ in $\mathcal{M}_{\mathrm{Syn},\underline{\mathsf{R}}}$ to the left adjoint $(-) \to \underline{\mathsf{R}}$ in $(\mathcal{M}_{\mathrm{Syn},\underline{\mathsf{R}}})^{\perp}$. Thus the endomorphism $(S_{\underline{\mathsf{R}}}\, S_{\underline{\mathsf{R}}},\, (T_{\underline{\mathsf{R}}})^{\mathrm{op}}\, T_{\underline{\mathsf{R}}})$ on $\mathcal{M}_{\mathrm{Syn},\underline{\mathsf{R}}}$ preserves all type constants up to isomorphism. By Theorem 4, this endomorphism is isomorphic to the identity morphism. For the composite endomorphism $(S_{\underline{\mathsf{R}}}\, S_{\underline{\mathsf{R}}},\, T_{\underline{\mathsf{R}}}\, (T_{\underline{\mathsf{R}}})^{\mathrm{op}})$ on $(\mathcal{M}_{\mathrm{Syn},\underline{\mathsf{R}}})^{\perp}$, we have $T_{\underline{\mathsf{R}}}\, (T_{\underline{\mathsf{R}}})^{\mathrm{op}} = ((T_{\underline{\mathsf{R}}})^{\mathrm{op}}\, T_{\underline{\mathsf{R}}})^{\mathrm{op}}$, which is again isomorphic to the identity because $(T_{\underline{\mathsf{R}}})^{\mathrm{op}}\, T_{\underline{\mathsf{R}}}$ is (as shown above). Hence $(S_{\underline{\mathsf{R}}}, T_{\underline{\mathsf{R}}})\colon \mathcal{M}_{\mathrm{Syn},\underline{\mathsf{R}}} \to (\mathcal{M}_{\mathrm{Syn},\underline{\mathsf{R}}})^{\perp}$ and $(S_{\underline{\mathsf{R}}}, (T_{\underline{\mathsf{R}}})^{\mathrm{op}})\colon (\mathcal{M}_{\mathrm{Syn},\underline{\mathsf{R}}})^{\perp} \to \mathcal{M}_{\mathrm{Syn},\underline{\mathsf{R}}}$ together form an equivalence of models. $\square$

The self-duality of $\mathcal{M}_{\mathrm{Syn},\underline{\mathsf{R}}}$ exhibited above differs in two important respects from the self-duality for classical linear/nonlinear models of Proposition 3. First, $(S_{\underline{\mathsf{R}}}, T_{\underline{\mathsf{R}}})$ is not an isomorphism. Second, the functor $S_{\underline{\mathsf{R}}}$ is not (even isomorphic to) the identity on $\mathbf{V}_{\mathrm{Syn}}$. (Though $S_{\underline{\mathsf{R}}}S_{\underline{\mathsf{R}}}$ *is* isomorphic to the identity.)

It is now a straightforward matter to finally prove Theorem 1. The crucial observation is that the morphism $(S_{\underline{\mathsf{R}}}, T_{\underline{\mathsf{R}}})$ is the generic linearly-used CPS translation of the enriched effect calculus into itself. Indeed, the object actions of $S_{\underline{\mathsf{R}}}$ and $T_{\underline{\mathsf{R}}}$ are respectively the value-type and computation-type translations of Fig. 4, and the morphism actions are respectively $(\cdot)^{\mathcal{V}_{\underline{\mathsf{R}}}}$ and $(\cdot)^{\mathcal{C}_{\underline{\mathsf{R}}}}$ from Fig. 5.

Proof of Theorem 1 (outline). We have seen above that $S_{\underline{\mathsf{R}}}S_{\underline{\mathsf{R}}}$ is isomorphic to the identity on $\mathbf{V}_{\mathrm{Syn}}$, and $(T_{\underline{\mathsf{R}}})^{\mathrm{op}}\, T_{\underline{\mathsf{R}}}$ is isomorphic to the identity on $\mathbf{C}_{\mathrm{Syn}}$. We define the $j_{\mathsf{A}}\colon \mathsf{A}^{\mathcal{V}_{\underline{\mathsf{R}}}\mathcal{V}_{\underline{\mathsf{R}}}} \to \mathsf{A}$ to be components of the former natural isomorphism, and the $k_{\underline{\mathsf{A}}}\colon \underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}\mathcal{C}_{\underline{\mathsf{R}}}} \multimap \underline{\mathsf{A}}$ to be components of the latter. Statement 1 of the theorem is equivalent to the naturality of the j_{A}. The naturality condition which the $k_{\underline{\mathsf{A}}}$ are required to satisfy to be part of a cni, is the diagram below.

$$\begin{tikzcd}
(\underline{\mathsf{A}} \multimap \underline{\mathsf{B}})^{\mathcal{V}_{\underline{\mathsf{R}}}\mathcal{V}_{\underline{\mathsf{R}}}} \arrow[r, "j_{\underline{\mathsf{A}}\multimap\underline{\mathsf{B}}}"] \arrow[d, "="'] & \underline{\mathsf{A}} \multimap \underline{\mathsf{B}} \arrow[d, "k_{\underline{\mathsf{A}}} \multimap \underline{\mathsf{B}}"] \\
\underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}\mathcal{C}_{\underline{\mathsf{R}}}} \multimap \underline{\mathsf{B}}^{\mathcal{C}_{\underline{\mathsf{R}}}\mathcal{C}_{\underline{\mathsf{R}}}} \arrow[r, "\underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}\mathcal{C}_{\underline{\mathsf{R}}}} \multimap k_{\underline{\mathsf{B}}}"] & \underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}\mathcal{C}_{\underline{\mathsf{R}}}} \multimap \underline{\mathsf{B}}
\end{tikzcd}$$

Hence, if $\Gamma \,|\, z\colon \underline{\mathsf{A}} \vdash t\colon \underline{\mathsf{B}}$ then

$$\begin{aligned}
\underline{\lambda} z\!:\!\underline{\mathsf{A}}.\, t &= j_{\mathsf{A}\multimap\mathsf{B}}\, (\underline{\lambda} z\!:\!\underline{\mathsf{A}}.\, t)^{\mathcal{V}_{\underline{\mathsf{R}}}\mathcal{V}_{\underline{\mathsf{R}}}}\, [j^{-1}(\Gamma)] && \text{(statement 1)}\\
&= j_{\mathsf{A}\multimap\mathsf{B}}\, (\underline{\lambda} z\!:\!\underline{\mathsf{A}}^{\mathcal{C}_{\underline{\mathsf{R}}}\mathcal{C}_{\underline{\mathsf{R}}}}.\, (t^{\mathcal{C}_{\underline{\mathsf{R}}}\mathcal{C}_{\underline{\mathsf{R}}}}))\, [j^{-1}(\Gamma)] && \text{(def. } (-)^{\mathcal{V}_{\underline{\mathsf{R}}}})\\
&= (\underline{\lambda} z\!:\!\underline{\mathsf{A}}.\, k_{\underline{\mathsf{B}}}(t^{\mathcal{C}_{\underline{\mathsf{R}}}\mathcal{C}_{\underline{\mathsf{R}}}})\, [k_{\underline{\mathsf{A}}}^{-1}(z) \,/\, z])\, [j^{-1}(\Gamma)] && \text{(naturality } k)
\end{aligned}$$

proving statement 2 of the theorem. $\square$

We end the paper by mentioning a couple of issues that there is no space to cover in detail. We call a pointed model $\mathcal{M} = (F \dashv U\colon \mathbf{C} \to \mathbf{V},\, \underline{\mathsf{R}})$ *canonically pointed* if $\underline{\mathsf{R}}$ is isomorphic to $F1$. Because we have $F^{\underline{\mathsf{R}}}1 = \underline{\mathsf{R}}^1 \cong \underline{\mathsf{R}}$, we see that $\mathcal{M}^{\perp}$ is canonically pointed if and only if $\mathcal{M}$ is. We say that $F \dashv U\colon \mathbf{C} \to \mathbf{V}$ is *canonically self dual* if the canonically pointed $\mathcal{M} = (F \dashv U\colon \mathbf{C} \to \mathbf{V},\, F1)$ is self dual. By an argument similar to the proof of Theorem 6, the syntactic model can be shown to be canonically self dual. This has syntactic repercussions for EEC. If the computation-type constant $\underline{\mathsf{R}}$ is replaced uniformly by the computation type !1 in the canonical linearly-used CPS translation then one still obtains the involutivity property and its consequences.

Finally, we mention that the key to Theorem 5 is that every $\mathbf{V}$-enriched adjunction $F \dashv U \colon \mathbf{C} \to \mathbf{V}$ is isomorphic to a $\mathbf{V}$-adjunction of the standard form $(-)\cdot\underline{I} \dashv \mathbf{C}(\underline{I}, -)$, by setting $\underline{I} \cong F1$. This fact allows models of the enriched effect calculus to be described more simply as triples $(\mathbf{V}, \mathbf{C}, \underline{I})$, where the adjunction is replaced by simply requiring a specified object $\underline{I}$ of $\mathbf{C}$. This approach renders many properties of dual models obvious. What becomes less straightforward is to connect semantic and syntactic properties, since the semantic structure is less close to the syntactic primitives.

Acknowledgement. We thank the anonymous referees for helpful suggestions.

References

1. Barr, M.: $*$-autonomous categories. 752, vol. LNM (1979)
2. Benton, P.N.: A mixed linear and non-linear logic: Proofs, terms and models. In: Pacholski, L., Tiuryn, J. (eds.) CSL 1994. LNCS, vol. 933. Springer, Heidelberg (1995)
3. Berdine, J., O'Hearn, P.W., Reddy, U., Thielecke, H.: Linear continuation-passing. Higher Order and Symbolic Computation 15, 181–208 (2002)
4. Egger, J., Møgelberg, R.E., Simpson, A.: Enriching an effect calculus with linear types. In: Grädel, E., Kahle, R. (eds.) CSL 2009. LNCS, vol. 5771, pp. 240–254. Springer, Heidelberg (2009)
5. A.: Filinski. Controlling Effects. PhD thesis, School of Comp. Sci., CMU (1996)
6. Hasegawa, M.: Linearly used effects: Monadic and CPS transformations into the linear lambda calculus. In: Hu, Z., Rodríguez-Artalejo, M. (eds.) FLOPS 2002. LNCS, vol. 2441, pp. 167–182. Springer, Heidelberg (2002)
7. Hasegawa, M.: Semantics of linear continuation-passing in call-by-name. In: Kameyama, Y., Stuckey, P.J. (eds.) FLOPS 2004. LNCS, vol. 2998, pp. 229–243. Springer, Heidelberg (2004)
8. Kelly, G.M.: Basic Concepts of Enriched Category Theory. London Math. Society Lecture Note Series, vol. 64. Cambridge University Press, Cambridge (1982)
9. Lawvere, F.W.: Ordinal sums and equational doctrines. In: Seminar on Triples and Categorical Homology Theory (ETH, Zürich), pp. 141–155. Springer, Heidelberg (1969)
10. Levy, P.B.: Call-by-push-value. In: A functional/imperative synthesis. Semantic Structures in Computation. Springer, Heidelberg (2004)
11. Moggi, E.: Computational lambda-calculus and monads. In: Proc. 4th LICS, pp. 14–23 (1989)
12. Moggi, E.: Notions of computation and monads. Information and Computation 93, 55–92 (1991)
13. Parigot, M.: $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In: Voronkov, A. (ed.) LPAR 1992. LNCS, vol. 624, pp. 190–201. Springer, Heidelberg (1992)

Block Structure vs. Scope Extrusion: Between Innocence and Omniscience

Andrzej S. Murawski[⋆] and Nikos Tzevelekos[⋆⋆]

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

Abstract. We study the semantic meaning of block structure using game semantics and introduce the notion of block-innocent strategies, which turns out to characterise call-by-value computation with block-allocated storage through soundness, finitary definability and universality results. This puts us in a good position to conduct a comparative study of purely functional computation, computation with block storage and dynamic memory allocation respectively. For example, we show that dynamic variable allocation can be replaced with block-allocated variables exactly when the term involved (open or closed) is of base type and that block-allocated storage can be replaced with purely functional computation when types of order two are involved. To illustrate the restrictive nature of block structure further, we prove a decidability result for a finitary fragment of call-by-value Idealized Algol for which it is known that allowing for dynamic memory allocation leads to undecidability.

1 Introduction

Most programming languages manage memory by employing a stack for local variables and heap storage for dynamically allocated data that may live beyond their initial context. A prototypical example of the former mechanism is Reynolds's Idealized Algol [16], in which local variables can only be introduced inside blocks of ground type. Memory is then allocated on entry to the block and deallocated on exit. In contrast, languages such as ML permit variables to escape from their current context under the guise of pointers or references. In this case, after memory is allocated at the point of reference creation, the variable is allowed to persist indefinitely (in practice, garbage collection or explicit deallocation is used to put an end to its life).

In this paper we would like to compare the expressivity of the two paradigms. As a simple example of heap-based memory allocation we consider the language RML, introduced by Abramsky and McCusker in [2], which is a fragment of ML featuring integer-valued references. They also constructed a fully abstract game model of RML based on strategies (also referred to as *knowing* strategies) that allow the player to base his decisions on the full history of play. On the other hand, at around the same time Honda and Yoshida [6] showed that the purely functional core of RML, better known as call-by-value PCF [14], corresponds to *innocent* strategies [7], i.e. those that can only rely on a

⋆ Supported by an EPSRC Advanced Research Fellowship (EP/C539753/1).
⋆⋆ Supported by EPSRC (EP/F067607/1).

restricted view of the play when deciding on the next move. Since block-structured storage of Idealized Algol seems less expressive than dynamic memory allocation of ML and more expressive than PCF, it is natural to ask about its exact position in the range of strategies between innocence and omniscience. Our first result is an answer to this question. We introduce the family of *block-innocent* strategies, situated strictly between innocent and knowing strategies. As a vehicle for our study we use a call-by-value variant $\mathsf{IA}_{\mathsf{cbv}}$ of Idealized Algol and prove that each $\mathsf{IA}_{\mathsf{cbv}}$-term can be interpreted by a block-innocent strategy (soundness), each finitary block-innocent strategy corresponds to an $\mathsf{IA}_{\mathsf{cbv}}$-term (finitary definability) and each recursively presentable block-innocent strategy corresponds to an $\mathsf{IA}_{\mathsf{cbv}}$-term (universality). Block-innocence captures the particular kind of uniformity exhibited by strategies originating from block-structured programs, akin to innocence yet strictly weaker. In fact, we define block-innocence through innocence in a setting enriched with explicit store annotations added to standard moves. For instance, in the play shown below[1], if P follows a block-innocent strategy, he is free to use different moves as the fourth and sixth moves, but the tenth one and the twelfth one have to be the same.

$$\begin{tikzcd}[column sep=tiny]
q & q & q & 0 & q \arrow[lll, bend right] & 1 & a \arrow[lllll, bend right] & a \arrow[lllllll, bend right] & q & 0 & q \arrow[lll, bend right] & 0
\end{tikzcd}$$

Additionally, our framework detects "storage violations" resulting from an attempt to access a variable from outside of its block. For instance, no $\mathsf{IA}_{\mathsf{cbv}}$-term will ever produce the following play (the last move is the offending one).

$$\begin{tikzcd}[column sep=tiny]
q & q & q & 0 & q \arrow[lll, bend right] & 1 & a \arrow[lllll, bend right] & a \arrow[lllllll, bend right] & q & q \arrow[lll, bend right]
\end{tikzcd}$$

The notion of block-innocence provides us with a systematic methodology to address expressivity questions related to block structure such as "Does a given strategy originate from a stack-based memory discipline?" or "Can a given program using dynamic memory allocation be replaced with an equivalent program featuring stack-based storage?". To illustrate the approach we conduct a complete study of the relationship between the three classes of strategies according to type-thereotic order. We find that knowingness implies block-innocence when terms of base types (open or closed) are involved, that block-innocence implies innocence exactly for types of at most second order, and that knowingness implies innocence if the term is of base type and its free identifiers are of order 1.

As a further confirmation of the restrictive nature of the stack discipline of $\mathsf{IA}_{\mathsf{cbv}}$, we prove that program equivalence is decidable for a finitary variant of $\mathsf{IA}_{\mathsf{cbv}}$ which properly contains all second-order types as well as some third-order types (interestingly, our type discipline covers the available higher-order types in PASCAL). In contrast, the corresponding restriction of RML is known to be undecidable [10].

Related work. The stack discipline has always been regarded as part of the essence of Algol [16]. Accordingly, finding models embodying stack-oriented storage management has become an important goal of research into Algol-like languages. In this spirit, in the early 1980s, Reynolds [16] and Oles [11] devised a semantic model of Algol-like

[1] For the sake of clarity, we only include pointers pointing more than one move behind.

languages using a category of functors from a category of store shapes to the category of predomains. Perhaps surprisingly, in the 1990s Pitts and Stark [13,17] managed to adapt the techniques to languages with dynamic allocation. This would appear to create a common platform suitable for a comparative study such as ours. However, despite the valuable structural insights, the relative imprecision of the functor category semantics (failure of definability and full abstraction) makes it unlikely that the results obtained by us can be proved via this route. The semantics of local effects has also been investigated from the category-theoretic point of view in [15].

As for the game semantics literature, Ong's work [12] based on strategies-with-state is the work closest to ours. His paper defines a compositional framework that is proved sound for the third-order fragment of call-by-name Idealized Algol. Adapting the results to call-by-value and all types is far from immediate. For a start, to handle higher-order types, we note that the state of O-moves is no longer determined by its justifier and the preceding move. Instead, the right state has to be computed "globally" using the whole history of play. However, the obvious adaptation of so modified framework to call-by-value does not capture the block-structure of $\mathsf{IA}_{\mathsf{cbv}}$. Quite the opposite: it seems to be more compatible with RML than $\mathsf{IA}_{\mathsf{cbv}}$! Consequently, further changes are needed to characterize $\mathsf{IA}_{\mathsf{cbv}}$. Firstly, to restore definability, the explicit stores have to become lists instead of sets. Secondly, conditions controlling state changes must be tightened. In particular, P must be forbidden from introducing fresh variables at any step and, in a similar vein, must be forced to drop some variables from his moves in certain circumstances.

The paper cited above is part of a series that has eventually led to a complete classification of the decidable cases of call-by-name (finitary) Idealized Algol. Much less is known about the call-by-value case, we are only aware of two papers: one by Ghica [5] and the other by the first-named author [10]. Both rely on regular languages to capture the game semantics of fragments of $\mathsf{IA}_{\mathsf{cbv}}$ and RML respectively. Their other common feature is that the types considered are selected in such a way that no pointers need to be represented in the induced plays. Our results represent further progress with regard to $\mathsf{IA}_{\mathsf{cbv}}$. The type system of our language, named $\mathsf{IA}^{2+}_{\circlearrowleft}$, is designed in such a way that only pointers from O-moves need not be represented, but we must include an explicit representation of pointers from certain P-moves. In particular, in contrast to [5], we can account for all second-order types, as we allow all first-order types to occur in contexts. "Curried" types of the form $A \rightarrow B \rightarrow C$ are especially tricky to handle here, because they can only be dealt with correctly if pointers from P-moves are encoded explicitly (recall that in the call-by-value setting $A \rightarrow B \rightarrow C$ and $A \times B \rightarrow C$ are not isomorphic). Any further extension of the type system of $\mathsf{IA}^{2+}_{\circlearrowleft}$ leads either to context-free languages or to plays in which pointers from O-moves of unbounded length would have to be handled, which seemingly requires an infinite alphabet.

2 Syntax

To set a common ground for our investigations, we introduce a higher-order programming language $\mathcal{L}$ that features syntactic constructs for both block and dynamic

$$\frac{\Gamma, x : \mathsf{var} \vdash M : \beta}{\Gamma \vdash \mathsf{new}\, x\, \mathsf{in}\, M : \beta} \qquad \frac{}{\Gamma \vdash \mathsf{ref} : \mathsf{var}} \qquad \frac{}{\Gamma \vdash () : \mathsf{unit}} \qquad \frac{i \in \mathbb{Z}}{\Gamma \vdash i : \mathsf{int}} \qquad \frac{(x : \theta) \in \Gamma}{\Gamma \vdash x : \theta}$$

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

$$\frac{\Gamma \vdash M : \mathsf{var}}{\Gamma \vdash !M : \mathsf{int}} \qquad \frac{\Gamma \vdash M : \mathsf{var} \quad \Gamma \vdash N : \mathsf{int}}{\Gamma \vdash M := N : \mathsf{unit}} \qquad \frac{\Gamma \vdash M : \mathsf{unit} \to \mathsf{int} \quad \Gamma \vdash N : \mathsf{int} \to \mathsf{unit}}{\Gamma \vdash \mathsf{mkvar}(M, N) : \mathsf{var}}$$

$$\frac{\Gamma \vdash M : \theta \to \theta' \quad \Gamma \vdash N : \theta}{\Gamma \vdash MN : \theta'} \qquad \frac{\Gamma, x : \theta \vdash M : \theta'}{\Gamma \vdash \lambda x^{\theta}.M : \theta \to \theta'} \qquad \frac{\Gamma \vdash M : (\theta \to \theta') \to (\theta \to \theta')}{\Gamma \vdash \mathsf{Y}(M) : \theta \to \theta'}$$

memory allocation. Its types are generated by the grammar below, where β ranges over the ground types unit and int.

$$\theta ::= \quad \beta \quad | \quad \mathsf{var} \quad | \quad \theta \to \theta$$

The syntax of $\mathcal{L}$ is given in the Figure above. Note in particular the first two rules concerning variables. The order of a type is defined as follows: $\mathsf{ord}(\beta) = 0$, $\mathsf{ord}(\mathsf{var}) = 1$, $\mathsf{ord}(\theta_1 \to \theta_2) = \max(\mathsf{ord}(\theta_1) + 1, \mathsf{ord}(\theta_2))$. For any $i \geq 0$, terms that are typable using exclusively judgments of the form $x_1 : \theta_1, \cdots, x_n : \theta_n \vdash M : \theta$, where $\mathsf{ord}(\theta_j) < i$ $(1 \leq j \leq n)$ and $\mathsf{ord}(\theta) \leq i$, are said to form the ith-order fragment. To spell out the operational semantics of $\mathcal{L}$, we need to assume a countable set Loc of *locations*, which are added to the syntax as auxiliary constants of type var. We shall write α to range over them. The semantics then takes the form of judgments $s, M \Downarrow s', V$, where s, s' are finite partial functions from Loc to integers, M is a term and V is a value. Terms of the following shapes are values: $()$, integer constants, elements of Loc, λ-abstractions or terms of the form $\mathsf{mkvar}(\lambda x^{\mathsf{unit}}.M, \lambda y^{\mathsf{int}}.N)$. Here we only reproduce the two evaluation rules related to variable creation.

$$\frac{s \cup (\alpha \mapsto 0), M[\alpha/x] \Downarrow s', V}{s, \mathsf{new}\, x\, \mathsf{in}\, M \Downarrow s' \setminus \alpha, V}\ {\scriptstyle \alpha \notin \mathsf{dom}\, s} \qquad \frac{}{s, \mathsf{ref} \Downarrow s \cup (\alpha \mapsto 0), \alpha}\ {\scriptstyle \alpha \notin \mathsf{dom}\, s}$$

$s' \setminus \alpha$ is the restriction of s' to $\mathsf{dom}\, s' \setminus \{\alpha\}$. The former rule encapsulates the state within the newly created block, while the latter creates a reference to a new memory cell that can be passed around without restrictions on its scope.

Given a closed term $\vdash M : \mathsf{unit}$, we write $M \Downarrow$ if there exists s such that $\emptyset, M \Downarrow s, ()$. We shall call two programs equivalent if they behave identically in every context. This is captured by the following definition, parameterized by the kind of contexts that are considered, to allow for testing of terms with contexts originating from a designated subset of the language.

Definition 1. *Suppose $\mathcal{L}'$ is a subset of $\mathcal{L}$. We say that the terms-in-context $\Gamma \vdash M_1, M_2 : \theta$ are $\mathcal{L}'$-equivalent (written $\Gamma \vdash M_1 \cong_{\mathcal{L}'} M_2$) if, for any $\mathcal{L}'$-context $C[-]$ such that $\vdash C[M_1], C[M_2] : \mathsf{unit}$, $C[M_1] \Downarrow$ if and only if $C[M_2] \Downarrow$.*

We shall study three sublanguages of $\mathcal{L}$ called PCF^+, $\mathsf{IA}_{\mathsf{cbv}}$ and RML. The latter two have appeared in the literature as paradigmatic examples of programming languages with stack discipline and dynamic memory allocation respectively.

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

- PCF^+ is a purely functional language obtained from $\mathcal{L}$ by removing $\mathsf{new}\, x \,\mathsf{in}\, M$ and ref. It extends the language PCF [14] with primitives for variable access, but not for memory allocation.
- $\mathsf{IA}_{\mathsf{cbv}}$ is $\mathcal{L}$ without the ref constant. It can be viewed as a call-by-value variant of Idealized Algol [16]. Only block-allocated storage is available in $\mathsf{IA}_{\mathsf{cbv}}$.
- RML is $\mathcal{L}$ save the construct $\mathsf{new}\, x \,\mathsf{in}\, M$. It is exactly the language introduced in [2] as a prototypical language for ML-like integer references.

We shall often write $\mathsf{let}\, x = M \,\mathsf{in}\, N$ instead of $(\lambda x.N)M$. Note that, since $\mathsf{new}\, x \,\mathsf{in}\, M$ is equivalent to $\mathsf{let}\, x = \mathsf{ref} \,\mathsf{in}\, M$, RML and $\mathcal{L}$ merely differ on a syntactic level in that $\mathcal{L}$ contains "syntactic sugar" for blocks. In the opposite direction, our results will show that ref cannot in general be replaced with an equivalent term that uses $\mathsf{new}\, x \,\mathsf{in}\, M$. Indeed, our paper provides a general methodology for identifying and studying scenarios in which this expressivity gap occurs.

3 Game Semantics

Here we introduce the game models used throughout the paper, which are based on the Honda-Yoshida approach to modelling call-by-value computation [6].

Definition 2. *An* **arena** $A = (M_A, I_A, \vdash_A, \lambda_A)$ *is given by*
- *a set M_A of moves, and a subset $I_A \subseteq M_A$ of* initial *moves,*
- *a justification relation $\vdash_A \subseteq M_A \times (M_A \setminus I_A)$, and*
- *a labelling function $\lambda_A : M_A \to \{O, P\} \times \{Q, A\}$*

such that $\lambda_A(I_A) = \{PA\}$ and, whenever $m \vdash_A m'$, we have $(\pi_1\lambda_A)(m) \neq (\pi_2\lambda_A)(m')$ and $(\pi_2\lambda_A)(m') = A \implies (\pi_2\lambda_A)(m) = Q$.

The role of λ_A is to label moves as *Opponent* or *Proponent* moves and as *Questions* or *Answers*. We typically write them as $m, n, \ldots$, or $o, p, q, a, q_P, q_O, \ldots$ when we want to be specific about their kind. The simplest arena is $0 = (\emptyset, \emptyset, \emptyset, \emptyset)$. Other "flat" arenas are 1 and $\mathbb{Z}$, defined by $M_1 = I_1 = \{*\}$, $M_{\mathbb{Z}} = I_{\mathbb{Z}} = \mathbb{Z}$. The two standard constructions on arenas are presented in the figure above, where $\overline{I}_A$ stands for $M_A \setminus I_A$, the OP-complement of λ_A is written as $\bar{\lambda}_A$, and i_A, i_B range over initial moves in the respective arenas. Types of $\mathcal{L}$ can now be interpreted with arenas in the following way: $[\![\mathsf{unit}]\!] = 1$, $[\![\mathsf{int}]\!] = \mathbb{Z}$, $[\![\mathsf{var}]\!] = (1 \Rightarrow \mathbb{Z}) \otimes (\mathbb{Z} \Rightarrow 1)$ and $[\![\theta_1 \to \theta_2]\!] = [\![\theta_1]\!] \Rightarrow [\![\theta_2]\!]$. Although arenas model types, the actual games will be played in ***prearenas***, which are defined in the same way as arenas with the exception that initial moves must be O-questions. Given arenas A and B, we can construct the prearena $A \to B$ by setting:

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

For $\Gamma = \{x_1 : \theta_1, \cdots, x_n : \theta_n\}$, typing judgments $\Gamma \vdash \theta$ 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 $[\![\Gamma \vdash \theta]\!]$ or $[\![\theta_1, \cdots, \theta_n \vdash \theta]\!]$.

A ***justified sequence*** in a prearena A is a finite sequence s of moves of A satisfying the following condition: the first move must be initial, but all other moves m must be equipped with a pointer[2] to an earlier occurrence of a move m' such that $m' \vdash_A m$. A ***play*** in A is a justified sequence s satisfying the standard conditions of *Alternation*, *Well-Bracketing* and *Visibility* [7]. Visibility is based on the notions of *O-view* $\llcorner s \lrcorner$ and *P-view* $\ulcorner s \urcorner$ of a justified sequence s, given by: $\llcorner \epsilon \lrcorner = \epsilon,\ \llcorner s\, o \lrcorner = \llcorner s \lrcorner\, o,\ \llcorner s\, \overset{\frown}{o \cdots p} \lrcorner = \llcorner s \lrcorner\, o\, p$; $\ulcorner \epsilon \urcorner = \epsilon, \ulcorner s\, p \urcorner = \ulcorner s \urcorner\, p, \ulcorner s\, \overset{\frown}{p \cdots o} \urcorner = \ulcorner s \urcorner\, p\, o$. We write P_A to denote the set of plays in A.

Definition 3. *A* **(knowing) strategy** *σ on a prearena A, written $\sigma : A$, is a prefix-closed set of plays from A satisfying the first two conditions below. A strategy is* **innocent** *if, in addition, the third condition holds.*

O-CLOSURE *If even-length $s \in \sigma$ and $sm \in P_A$ then $sm \in \sigma$.*
DETERMINACY *If even-length $sm_1, sm_2 \in \sigma$ then $m_1 = m_2$.*
INNOCENCE *If $s_1 m, s_2 \in \sigma$ with odd-length s_1, s_2 and $\ulcorner s_1 \urcorner = \ulcorner s_2 \urcorner$ then $s_2 m \in \sigma$.*

Now we shall extend the framework to allow moves to be decorated with stores that contain *name*-integer pairs. The names should be viewed as semantic analogues of locations. When employing such moves-with-store, we are not interested in what exactly the names are, but 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, and all ensuing constructions and reasoning need to be compatible with it. This overhead can be dealt with robustly using the language of nominal set theory [4].

Let us fix a countably infinite set $\mathbb{A}$, the set of *names*, the elements of which we shall denote by α, β and variants. Consider the group PERM($\mathbb{A}$) of finite permutations of $\mathbb{A}$, denoted by π and variants. A ***strong nominal set*** [18] is a set equipped with a group action of PERM($\mathbb{A}$) such that each of its elements has *finite strong support*. That is to say, for any $x \in X$, there exists a finite set $\nu(x) \subseteq \mathbb{A}$, called ***the support of*** x, such that, for all permutations π, $(\forall \alpha \in \nu(x).\, \pi(\alpha) = \alpha) \iff \pi \cdot x = x$. Intuitively, $\nu(x)$ is the set of names "involved" in x. For example, the set $\mathbb{A}^{\#}$ of finite lists of distinct names with permutations acting elementwise is a strong nominal set. Name-variance in a strong nominal set X is represented by the relation: $x \sim x'$ if there exists π such that $x = \pi \cdot x'$.

We define a strong nominal set of ***stores***, the elements of which are finite sequences of name-integer pairs. Formally,

$$\Sigma, T ::= \quad \epsilon \quad | \quad (\alpha, i) :: \Sigma$$

where $i \in \mathbb{Z}$ and $\alpha \in \mathbb{A} \setminus \nu(\Sigma)$. We view stores as finite functions from names to integers, though their domains are lists rather than sets. Thus, we define the ***domain*** of a store to be the *list* of names obtained by applying the first projection to

[2] We then say that m' *justifies* m. If m is an answer, we might also say that m *answers* m'. If a question remains unanswered in s, it is *open*.

all of its elements. In particular, $\nu(\mathsf{dom}\,(\Sigma)) = \nu(\Sigma)$. If $\alpha \in \nu(\Sigma)$ then we write $\Sigma(\alpha)$ for the unique i such that (α, i) is an element of Σ. For stores Σ, T we write: $\Sigma \leq T$ for $\mathsf{dom}\,(\Sigma) \sqsubseteq \mathsf{dom}\,(T)$; $\Sigma \leq_p T$ for $\mathsf{dom}\,(\Sigma) \sqsubseteq_p \mathsf{dom}\,(T)$; $\Sigma \leq_s T$ for $\mathsf{dom}\,(\Sigma) \sqsubseteq_s \mathsf{dom}\,(T)$, where $\sqsubseteq, \sqsubseteq_p, \sqsubseteq_s$ denote the subsequence, prefix and suffix relations respectively. Note that $\Sigma \leq_{(p/s)} T \leq_{(p/s)} \Sigma$ implies $\mathsf{dom}\,(\Sigma) = \mathsf{dom}\,(T)$ but not $\Sigma = T$. Finally, let us write $\Sigma \setminus T$ for Σ restricted to $\nu(\Sigma) \setminus \nu(T)$.

An ***S-move*** (or *move-with-store*) in a prearena A is a pair consisting of a move and a store. We typically write S-moves as $m^\Sigma, n^T, o^\Sigma, p^T, q^\Sigma, a^T$. The first-projection function is viewed as *store erasure* and denoted by $\mathsf{erase}(\_)$. Note that moves contain no names and therefore, for any m^Σ, $\nu(m^\Sigma) = \nu(\Sigma) = \nu(\mathsf{dom}\,(\Sigma))$. A ***justified S-sequence*** in A is a sequence of S-moves equipped with justifiers, so that its erasure is a justified sequence. The notions of *O-view* and *P-view* are extended to S-sequences in the obvious manner. We say that a name α ***is closed*** in s if there are no open questions in s containing α.

Definition 4. *A justified S-sequence s in a prearena A is called an* **S-play***, also written $s \in SP_A$, if it satisfies the following conditions, for all $\alpha \in \mathbb{A}$.*

INIT *If $s = m^\Sigma \cdots$ then $\Sigma = \epsilon$.*
JUST-P *If $s = \cdots o^\Sigma \overset{\curvearrowleft}{\cdots} p^T \cdots$ then $\Sigma \leq_p T$. If $\lambda_A(p) = PA$ then* $\mathsf{dom}\,(\Sigma) = \mathsf{dom}\,(T)$.
JUST-O *If $s = \cdots p^\Sigma \overset{\curvearrowleft}{\cdots} o^T \cdots$ then* $\mathsf{dom}\,(\Sigma) = \mathsf{dom}\,(T)$.
PREV-PQ *If $s = \cdots o^\Sigma q_P^T \cdots$ then $\Sigma \setminus T \leq_s \Sigma$ and $\Sigma \setminus (\Sigma \setminus T) \leq_p T$ and*
(a). *if $\alpha \in \nu(T \setminus \Sigma)$ then $\alpha \notin \nu(s_{<q_P^T})$,*
(b). *if $\alpha \in \nu(\Sigma \setminus T)$ then α is closed in $s_{<q_P^T}$.*
VAL-O *If $s = \cdots p^\Sigma s' o^T \cdots$ and $\alpha \in (\nu(T) \cap \nu(\Sigma)) \setminus \nu(s')$ then $T(\alpha) = \Sigma(\alpha)$.*

For example, PREV-PQ stipulates that P-questions may drop some names from the store and append some others, but these changes may only take place *in blocks* at the tail of the store. Moreover, appended names must be fresh in the whole play, and a name can be dropped only if it has been closed.

Let us remark that, as stores have strong support, the set of S-plays SP_A is a strong nominal set. Further properties of S-plays include:

– If $s = \cdots m^\Sigma a_P^T \cdots$ then $\Sigma \setminus T \leq_s \Sigma$ and $\Sigma \setminus (\Sigma \setminus T) \leq_p T$ and
  (a) if $\alpha \in \nu(T)$ then $\alpha \in \nu(\Sigma)$,
  (b) if $\alpha \in \nu(\Sigma \setminus T)$ then α is closed in $s_{<a_P^T}$.
– If $s = s_1 o^\Sigma p^T s_2$ with $\alpha \in \nu(\Sigma) \setminus \nu(T)$ then $\alpha \notin \nu(s_2)$.

Definition 5. *An* **S-strategy** *σ on an arena A, written $\sigma : A$, is a prefix-closed set of S-plays from A satisfying the first three of the following conditions. An S-strategy is* **innocent** *if it also satisfies the last condition.*

NOMINAL CLOSURE *If $s' \sim s \in \sigma$ then $s' \in \sigma$.*
O-CLOSURE *If even-length $s \in \sigma$ and $sm^\Sigma \in SP_A$ then $sm^\Sigma \in \sigma$.*
DETERMINACY *If even-length $sm_1^{\Sigma_1}, sm_2^{\Sigma_2} \in \sigma$ then $sm_1^{\Sigma_1} \sim sm_2^{\Sigma_2}$.*
INNOCENCE *If $s_1 m^{\Sigma_1}, s_2 \in \sigma$ with s_1, s_2 odd-length and $\ulcorner s_1 \urcorner = \ulcorner s_2 \urcorner$ then there exists $s_2 m^{\Sigma_2} \in \sigma$ with $\ulcorner s_1 m^{\Sigma_1} \urcorner \sim \ulcorner s_2 m^{\Sigma_2} \urcorner$.*

Example 6. For any base type β, let us define the S-strategy $\mathsf{cell}_\beta : [\![\mathsf{var} \to \beta]\!] \to [\![\beta]\!]$ as the least innocent S-strategy containing the plays below. We use read and $\mathsf{write}(i)$ ($i \in \mathbb{Z}$) to refer to the question-moves of $[\![\mathsf{var}]\!]$, and i ($i \in \mathbb{Z}$) and ok for the non-initial answers.

$$\begin{tikzcd}[column sep=small]
q_0 & q_1^{(\alpha,0)} \arrow[l, bend right] & a_1^{(\alpha,i)} \arrow[l, bend right] & a_0 \arrow[lll, bend right]
\end{tikzcd}
\quad
\begin{tikzcd}[column sep=small]
q_0 & q_1^{(\alpha,0)} \arrow[l, bend right] & \mathsf{read}^{(\alpha,i)} \arrow[l, bend right] & i^{(\alpha,i)} \arrow[l, bend right]
\end{tikzcd}
\quad
\begin{tikzcd}[column sep=small]
q_0 & q_1^{(\alpha,0)} \arrow[l, bend right] & \mathsf{write}(j)^{(\alpha,i)} \arrow[l, bend right] & \mathsf{ok}^{(\alpha,j)} \arrow[l, bend right]
\end{tikzcd}$$

Example 7. Had we used sets instead of lists for representing stores, the following "S-strategy", which represents incorrect overlap of scopes (α and β are in scope of one another, but at the same time have different scopes), would be innocent.

$$\begin{tikzcd}[column sep=small]
q_0 & q_1^{(\alpha,0),(\beta,0)} \arrow[l, bend right] & 0_1^{(\alpha,0),(\beta,0)} \arrow[l, bend right] & q_1^{(\alpha,0)} \arrow[lll, bend right]
\end{tikzcd}
\qquad
\begin{tikzcd}[column sep=small]
q_0 & q_1^{(\alpha,0),(\beta,0)} \arrow[l, bend right] & 1_1^{(\alpha,0),(\beta,0)} \arrow[l, bend right] & q_1^{(\beta,0)} \arrow[lll, bend right]
\end{tikzcd}$$

Arenas and S-strategies form a category, which we call $\mathcal{S}$, and so do innocent S-strategies. $\mathcal{S}$ turns out to exhibit the same kind of categorical structure as that discussed in [6], which can be employed to model call-by-value higher-order computation with recursion. Thus, the functional part of $\mathsf{IA}_{\mathsf{cbv}}$ can be interpreted in $\mathcal{S}$ according to the standard recipe. Assignment, dereferencing and mkvar can in turn be modelled using the innocent strategies without stores from [2]. Finally, the denotation of $\mathsf{new}\, x\, \mathsf{in}\, M$ is obtained by composing the denotation of $\lambda x^{\mathsf{var}}.M$ with the innocent S-strategy cell_β. Let us write $[\![\cdots]\!]_{\mathrm{S}}$ for the resultant semantic map.

Proposition 8 (Soundness). *For any* $\mathsf{IA}_{\mathsf{cbv}}$*-term* $\Gamma \vdash M : \theta$, $[\![\Gamma \vdash M : \theta]\!]_{\mathrm{S}}$ *is an innocent S-strategy.*

Innocent S-strategies can be *decomposed* in a similar way to the innocent strategies of [6]. There is one important exception, though, which occurs when the second-move introduces a non-empty store (our rules of play imply that the move must be a question). Let α be the first variable from the non-empty store. In order to decompose the strategy, consider a P-view s in which α occurs in the second move q_α. It turns out that $s = q q_\alpha s_\alpha s'$, where (the store of) a move m^Σ from s contains α if, and only if, it is q_α or in s_α. In addition, no justification pointers connect s' to $q_\alpha s_\alpha$. This separation can be applied to decompose the view-function of an innocent S-strategy. The s_α parts, put together as a single S-strategy, can subsequently be dealt with in the style of factorization arguments, which remove α from moves at the cost of an additional var-component. Finally, to relate s_α's to the suitable s' one can use numerical codes for $q_\alpha s_\alpha$. These ideas lie at the heart of the following result. By a finitary innocent S-strategy we mean an innocent strategy whose view-function quotiented by name-variance is finite.

Proposition 9 (Finitary Definability and Universality).
- *Any finitary innocent S-strategy is* $\mathsf{IA}_{\mathsf{cbv}}$*-definable.*
- *Any recursively presentable innocent S-strategy is* $\mathsf{IA}_{\mathsf{cbv}}$*-definable.*

It is worth noting that the universality result for innocent S-strategies implies an analogous result for innocent strategies and PCF. Thanks to call-by-value, the result is actually sharper than the universality results of [1,7], which had to be proved "up to observational equivalence". This was due to the fact that partial recursive functions could not always be represented in the canonical way (i.e. by terms for which the corresponding strategy contained plays of the form $q\,q\,n\,f(n)$). This is no longer the case under the call-by-value regime, where each partially recursive function f can be coded by a term whose denotation will be the strategy based on plays of the shape $n\, f(n)$.

With the soundness and definability results in place, we could now proceed in the familiar way to define a fully abstract model of $\mathsf{IA}_{\mathsf{cbv}}$ via the intrinsic quotient. However, this would be somewhat counterproductive. It turns out that RML is a conservative extension of $\mathsf{IA}_{\mathsf{cbv}}$ (Corollary 14), so the (simpler) fully abstract model of RML from [2], based on knowing strategies, is already fully abstract for $\mathsf{IA}_{\mathsf{cbv}}$. In fact, our model can be related to knowing strategies more precisely. Observe that by erasing storage annotations in an innocent S-strategy σ we obtain a knowing strategy, which we call $\mathsf{erase}(\sigma)$ (determinism follows from the fact that stores in O-moves are uniquely determined and from block-innocence). Let us write $[\![\cdots]\!]$ for the knowing strategy semantics (cast in [6]).

Lemma 10. *For any* $\mathsf{IA}_{\mathsf{cbv}}$*-term* $\Gamma \vdash M : \theta$, $[\![\Gamma \vdash M : \theta]\!] = \mathsf{erase}([\![\Gamma \vdash M : \theta]\!]_{\mathrm{S}})$.

This means the intrinsic quotient we would construct in the setting with stores can be represented more explicitly via the induced *complete* plays (without stores)[3]. Even though innocent S-strategies have not led us to a direct account of full abstraction for $\mathsf{IA}_{\mathsf{cbv}}$, we have obtained important insights into the structure of knowing strategies representing $\mathsf{IA}_{\mathsf{cbv}}$-terms: they are erasures of innocent S-strategies. Knowing strategies with this property will be referred to as ***block-innocent***. The knowledge that strategies determined by $\mathsf{IA}_{\mathsf{cbv}}$ are block-innocent will be crucial in establishing a series of results in the following sections.

Example 11. Let us revisit the two plays from the Introduction. The first one indeed comes from an innocent S-strategy (we reveal the stores below). For the second one to become innocent (in the setting with stores), a store with variable α, say, would need to be introduced in the second move. Then α must also occur in the seventh move by JUST-O, but it must not occur in the eighth move by JUST-P (the PA clause). Hence, it will not be present in the ninth move by JUST-O. Consequently, the last move (justified by the seventh move) is bound to break either PREV-PQ(a) (if it contains α) or JUST-P (if it does not).

$$\begin{tikzcd}[column sep=tiny]
q & q^{(\alpha,0)} \arrow[l] & q^{(\alpha,0)} & 0^{(\alpha,1)} & q^{(\alpha,1)} \arrow[lll, bend right] & 1^{(\alpha,1)} & a^{(\alpha,0)} \arrow[lllll, bend right] & a \arrow[lllllll, bend right] & q & 0 & q \arrow[lll, bend right] & 0
\end{tikzcd}$$

4 From Omniscience to Innocence

In Section 2 we introduced the three languages: PCF^{+}, $\mathsf{IA}_{\mathsf{cbv}}$ and RML. By the soundness and universality results of the previous section (as well as the soundness results from [6,2]) the languages correspond respectively to innocent, block-innocent and knowing strategies. Let A be an arena. We write $\mathcal{I}_A$, $\mathcal{B}_A$ and $\mathcal{K}_A$ for the corresponding classes of (store-free) strategies in A. Obviously, $\mathcal{I}_A \subseteq \mathcal{B}_A \subseteq \mathcal{K}_A$. Next we shall study type-theoretic conditions under which one kind of strategy collapses to another. Thanks to

[3] A play is complete if it does not contain unanswered questions. That such plays capture program equivalence in RML follows from the argument in [3], readily adaptable to RML. By Corollary 14, the same characterization will apply to $\mathsf{IA}_{\mathsf{cbv}}$.

the universality results, this corresponds to the existence of an equivalent program in a weaker language.

Lemma 12. *Let $A = [\![\theta_1, \cdots, \theta_n \vdash \theta \to \theta']\!]$. Then $\mathcal{B}_A \subsetneq \mathcal{K}_A$.*

Proof. Observe that there exist moves q_0, a_0, q_1, a_1 such that $q_0 \vdash_A a_0 \vdash_A q_1 \vdash_A a_1$ and consider $\sigma = \{\epsilon, q_0, q_0a_0, q_0a_0q_1, q_0a_0q_1a_1\}$, i.e. σ has no response at $q_0a_0q_1a_1q_1$. Then $\sigma \in \mathcal{K}_A \setminus \mathcal{B}_A$. It is worth remarking that a strategy of the above kind denotes the RML-term $\vdash \mathsf{let}\, v = \mathsf{ref}\ \mathsf{in}\ \lambda x^{\mathsf{unit}}.\mathsf{if}\ !v\ \mathsf{then}\ \Omega\ \mathsf{else}\ v := !v + 1 : \mathsf{unit} \to \mathsf{unit}$. □

Lemma 12 confirms that, in general, block structure restricts expressivity. However, the next result shows this not to be the case for open terms of base type.

Lemma 13. *Let $A = [\![\theta_1, \cdots, \theta_n \vdash \beta]\!]$. Then $\mathcal{B}_A = \mathcal{K}_A$.*

Proof. Observe that any knowing strategy for A becomes block-innocent if in the second move P introduces a store with one variable that keeps track of the history of play (this is reminiscent of the factorization arguments in game semantics). The variable should be removed from the store by P only when he plays an answer to the initial question, in which case the play becomes complete and cannot be extended further. □

By universality, we can conclude that each RML-term of base type is equivalent to an $\mathsf{IA_{cbv}}$-term. Since contexts used for testing equivalence are exactly of this kind, we obtain the following corollaries. The first one amounts to saying that RML is a conservative extension of $\mathsf{IA_{cbv}}$. The second one states that block-structured contexts suffice to distinguish terms that might use scope extrusion.

Corollary 14. *For any $\mathsf{IA_{cbv}}$-terms $\Gamma \vdash M_1, M_2$ and RML-terms $\Gamma \vdash N_1, N_2$*

– $\Gamma \vdash M_1 \cong_{\mathsf{RML}} M_2$ *if, and only if,* $\Gamma \vdash M_1 \cong_{\mathsf{IA_{cbv}}} M_2$.
– $\Gamma \vdash N_1 \cong_{\mathsf{RML}} N_2$ *if, and only if,* $\Gamma \vdash N_1 \cong_{\mathsf{IA_{cbv}}} N_2$.

Now we investigate the boundary between block structure and lack of state.

Lemma 15. *Let A be an arena such that each question enables an answer*[4]*. The following conditions are equivalent.*

1. $\mathcal{B}_A \subseteq \mathcal{I}_A$.
2. *No O-question is enabled by a P-question:* $m \vdash_A q_O$ *implies* $\lambda_A(m) = PA$.
3. *Store content of O-questions is trivial:* $sq_O^{\Sigma} \in SP_A$ *implies* $\mathsf{dom}\,(\Sigma) = \epsilon$.

We can now determine at which types block-innocence implies innocence.

Lemma 16. $[\![\theta_1, \cdots, \theta_n \vdash \theta]\!]$ *satisfies condition 2 of Lemma 15 iff* $\mathsf{ord}(\theta_i) \leq 1$ *($i = 1, \cdots, n$) and* $\mathsf{ord}(\theta) \leq 2$.

Consequently, second-order $\mathsf{IA_{cbv}}$-terms always have purely functional equivalents. Finally, we can pinpoint the types at which strategies are bound to be innocent: it suffices to combine the previous findings.

[4] All denotable arenas enjoy this property.

Lemma 17. *Let* $A = [\![\theta_1, \cdots, \theta_n \vdash \theta]\!]$. *Then* $\mathcal{K}_A = \mathcal{I}_A$ *iff* $\mathsf{ord}(\theta_i) \leq 1$ *(*$i = 1, \cdots, n$*) and* $\mathsf{ord}(\theta) = 0$.

In the next section we demonstrate that the gap in expressivity between $\mathcal{K}_A$ and $\mathcal{B}_A$ also bears practical consequences. The undecidable equivalence problem for second-order finitary RML becomes decidable in second-order finitary $\mathsf{IA}_{\mathsf{cbv}}$ (as well as at some third-order types).

5 Decidability of a Finitary Fragment of $\mathsf{IA}_{\mathsf{cbv}}$

To prove program equivalence decidable we restrict the base datatype of integers to the finite segment $\{0, \cdots, N\}$ ($N > 0$) and replace recursive definitions ($\mathsf{Y}(M)$) with looping ($\mathsf{while}\, M\, \mathsf{do}\, N$). Let us call the resultant language $\mathsf{IA}_{\circlearrowleft}$. Our decidability result will hold for a subset $\mathsf{IA}_{\circlearrowleft}^{2+}$ of $\mathsf{IA}_{\circlearrowleft}$, in which type order is restricted. $\mathsf{IA}_{\circlearrowleft}^{2+}$ will reside inside the third-order fragment of $\mathsf{IA}_{\circlearrowleft}$ and contain its second-order fragment. Note that the second-order fragment of similarly restricted RML is known be undecidable (even without loops) [10].

The decidability of program equivalence in $\mathsf{IA}_{\circlearrowleft}^{2+}$ will be shown by translating terms to regular languages representing the corresponding *knowing* strategies. We stress that we are *not* going to work with the induced S-plays. Nevertheless, the translation will crucially rely on insights gleaned from the semantics with explicit stores. More precisely, we will be interested in capturing the induced *complete* (store-free) plays. It is worth mentioning that, unlike in the (single-threaded) call-by-name setting, complete plays need not be maximal.

To represent plays as words, one needs to consider carefully how to represent pointers, should that be necessary. For example, this can be done by decorating moves with integers that encode the distance from the target in some way. Only pointers from questions require attention, since those from answers are uniquely reconstructible through the well-bracketing condition. Next we analyse two typing scenarios that look hopeless from the point of view of encoding pointers, since the distance from the pointer can grow arbitrarily. In the first case, thanks to block-innocence, we will be able to overcome the difficulties. The other case must remain a challenge for future work (or an undecidability result). On the basis of our discussion we shall subsequently introduce the type system of $\mathsf{IA}_{\circlearrowleft}^{2+}$.

Consider the arena $[\![\theta \vdash \theta_1 \to \ldots \to \theta_k \to \beta]\!]$. Due to the presence of the k arrows on the right-hand side we obtain chains of enablers $q_0 \vdash a_0 \vdash \cdots \vdash q_k \vdash a_k$, where q_0 is initial and each q_i ($i = 1, \cdots, k$) is initial in $[\![\theta_i]\!]$. We shall call the moves ***spinal***. Consider $[\![\, \vdash \lambda x^{\mathsf{unit}}.\lambda y^{\mathsf{unit}}.() : \mathsf{unit} \to \mathsf{unit} \to \mathsf{unit}]\!]$ (i.e. $k = 2$), which contains plays of the form $q_0 a_0 (q_1 a_1)^j$ for any $j \geq 0$. Pointers are still uniquely determined in these plays, but everything changes once O plays q_2 next. Then the target might be any of the j occurrences of q_1. The strategy in question actually offers responses in all such cases, so it would seem that all of these plays need to be represented (thus necessitating the use of an infinite alphabet). Fortunately, thanks to block-innocence, we can restrict ourselves to the case $j = 1$ and make the problem disappear. To see why, observe that none of the moves q_i, a_i will ever carry a non-empty store in an S-play, by

Definition 4. Thus, because the strategy is block-innocent, its behaviour is already represented faithfully by the single play $q_0 a_0 q_1 a_1 q_2 a_2$. In fact, this is one of the cases when block-innocence implies innocence, but in general this will not be true for denotations of $\mathsf{IA}^{2+}_{\circlearrowleft}$-terms. Hence, we generalize the observation as follows. Since the move q_1 never carries a non-trivial store, it follows that no additional information about the strategy is hidden in plays containing two occurrences of q_1. This is because a block-innocent strategy has to behave uniformly after each q_1 and in general will depend only on what happened between q_0 and a_0, and not on what happened after a previous copy of q_1 was played (there can be no communication between the "threads" started with q_1 because q_1 cannot carry a non-trivial store). Now that it is known that O need only play one occurrence of q_1, we can apply a similar reasoning to q_2, and so on. This yields the following lemma. Note that, due to Visibility, insisting on the presence of a unique copy of $q_1, \cdots, q_k$ in a play amounts to asking that each q_i be preceded by a_{i-1}.

Lemma 18. *Call a play* **spinal** *if each spinal question q_i ($0 < i \leq k$) occurring in it is the immediate successor of a_{i-1}. Let P_A^{sp} be the set of spinal plays of A. Let $\sigma, \tau : A$ be block-innocent strategies. Then $\sigma \cap P_A^{sp} = \tau \cap P_A^{sp}$ implies $\sigma = \tau$.*

Hence, for the purpose of checking program equivalence, it suffices to compare the induced sets of *spinal* complete plays.

Now that we have dealt with one challenge, let us introduce another one, which cannot be overcome so easily. Consider the arena $[\![(\theta_1 \to \theta_2 \to \theta_3) \to \theta_4 \vdash \theta]\!]$ and the enabling sequence $q_0 \vdash q_1 \vdash q_2 \vdash a_2 \vdash q_3$ it contains. Now consider the plays $q_0 q_1 (q_2 a_2)^j q_3$, where $j \geq 0$. Again, to represent the pointer from q_3 to one of the j occurrences of a_2, one would need an unbounded number of indices. This time it is not sufficient to restrict j to 1, because the behaviour need not be uniform after each q_2 (this is because in the setting with stores a non-empty store can be introduced as soon as in the second move q_1). To see that the concern is real, consider the term $f : (\mathsf{unit} \to \mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit} \vdash \mathsf{new}\, x \,\mathsf{in}\, f(\lambda y^{\mathsf{unit}}.\cdots \lambda z^{\mathsf{unit}}.\cdots) : \mathsf{unit}$, where $(\cdots)$ contain some code inspecting and changing the value of x.

This leads us to introduce $\mathsf{IA}^{2+}_{\circlearrowleft}$ via a type system that will not generate the configuration just discussed. Another restriction is to omit third-order types in the context, as they lead beyond the realm of regular languages (cf. $f : ((\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit} \vdash f(\lambda g^{\mathsf{unit} \to \mathsf{unit}}.g())$). Since var leads to identical problems as $\mathsf{unit} \to \mathsf{unit}$, we restrict its use accordingly.

Definition 19. $\mathsf{IA}^{2+}_{\circlearrowleft}$ *consists of $\mathsf{IA}_{\circlearrowleft}$-terms whose typing derivations rely solely on typing judgments of the shape $x_1 : ctype_1, \cdots, x_n : ctype_n \vdash M : ttype$, where ctype and ttype are defined by the grammar below.*

$$\begin{array}{lllllllll} ctype ::= & \beta & \mid & \mathsf{var} & \mid & \beta \to ctype & \mid & \mathsf{var} \to ctype & \mid & (\beta \to \beta) \to ctype \\ ttype ::= & \beta & \mid & \mathsf{var} & \mid & ctype \to ttype \end{array}$$

A lot of pointers from questions become uniquely determined in strategies representing $\mathsf{IA}^{2+}_{\circlearrowleft}$ terms, namely, all pointers from any O-questions and all pointers from P-questions to O-questions.

Lemma 20. *Let $A = [\![ctype_1, \cdots, ctype_n \vdash ttype]\!]$ and s_1, s_2 be spinal plays of A that are equal after all pointers from O-questions and all pointers from P-questions to O-questions have been erased. Then $s_1 = s_2$.*

Thus, the only pointers that need to be accounted for are those from P-questions to O-answers. Here is the simplest scenario illustrating that they can be ambiguous. Consider the terms

$$f : \mathsf{unit} \to \mathsf{unit} \to \mathsf{unit} \vdash \mathsf{let}\, g_1 = f()\,\mathsf{in}\,(\mathsf{let}\, g_2 = f()\,\mathsf{in}\, g_i()) : \mathsf{unit}$$

where $i = 1, 2$. They lead to the following plays, respectively for $i = 1$ and $i = 2$, which are equal up to pointers from P-questions to O-answers.

$$\begin{tikzcd} q_0 & q_1 \arrow[l, bend right] & a_1 \arrow[l, bend right] & q_1 \arrow[lll, bend right] & a_1 \arrow[l, bend right] & q_2 \arrow[lll, bend right] \end{tikzcd} \qquad \begin{tikzcd} q_0 & q_1 \arrow[l, bend right] & a_1 \arrow[l, bend right] & q_1 \arrow[lll, bend right] & a_1 \arrow[l, bend right] & q_2 \arrow[l, bend right] \end{tikzcd}$$

We are going to represent such pointers with numerical indices encoding the target of the pointer inside the current P-view. More precisely, let us enumerate (starting from 0) all question-enabling O-answers in the P-view. Then pointers from P-questions to O-answers can be encoded by decorating the P-question with the index of the O-answer. The plays above will be encoded as $q_0q_1a_1q_1a_1q_2^0$ and $q_0q_1a_1q_1a_1q_2^1$ respectively (other pointers are uniquely recoverable by Lemma 20 and will not be represented explicitly). So that we need not study the behaviour of the representation scheme for pointers under general composition (after which the indices might need to be recalculated), we restrict our translation to terms in a canonical shape, to be defined next. Any $\mathsf{IA}_{\circlearrowleft}$-term can be converted effectively to such a form and the conversion preserves denotation.

The canonical forms are defined by the following grammar. We use types as superscripts, whenever we want to highlight the type of an identifier (u, v, x, y, z range over identifier names). Note that the only identifiers in canonical form are those of base type, represented by x^β below.

$$\begin{aligned} \mathbb{C} ::= {} & () \mid i \mid x^\beta \mid x^\beta \oplus y^\beta \mid \mathsf{if}\, x^\beta\, \mathsf{then}\, \mathbb{C}\, \mathsf{else}\, \mathbb{C} \mid x^{\mathsf{var}} := y^{\mathsf{int}} \mid !x^{\mathsf{var}} \mid \lambda x^\theta.\mathbb{C} \mid \\ & \mathsf{mkvar}(\lambda x^{\mathsf{unit}}.\mathbb{C}, \lambda y^{\mathsf{int}}.\mathbb{C}) \mid \mathsf{new}\, x^{\mathsf{var}}\, \mathsf{in}\, \mathbb{C} \mid \mathsf{while}\, \mathbb{C}\, \mathsf{do}\, \mathbb{C} \mid \mathsf{let}\, x^\beta = \mathbb{C}\, \mathsf{in}\, \mathbb{C} \mid \\ & \mathsf{let}\, x = zy^\beta\, \mathsf{in}\, \mathbb{C} \mid \mathsf{let}\, x = z\, \mathsf{mkvar}(\lambda u^{\mathsf{unit}}.\mathbb{C}, \lambda v^{\mathsf{int}}.\mathbb{C})\, \mathsf{in}\, \mathbb{C} \mid \mathsf{let}\, x = z(\lambda x^\theta.\mathbb{C})\, \mathsf{in}\, \mathbb{C} \end{aligned}$$

Lemma 21. *Let $\Gamma \vdash M : \theta$ be an $\mathsf{IA}_{\circlearrowleft}$-term. There is an $\mathsf{IA}_{\circlearrowleft}$-term $\Gamma \vdash N : \theta$ in canonical form, effectively constructible from M, such that $[\![\Gamma \vdash M]\!] = [\![\Gamma \vdash N]\!]$.*

Proof. N can be obtained via a series of η-expansions, β-reductions and commuting conversions involving let and if. $\square$

A useful feature of the canonical form is that the problems with pointers can be related to the syntactic shape: they concern references to let-bound identifiers x^θ such that θ is *not* a base type (i.e. $\theta = \mathsf{var}$ or θ is a function type). The representation scheme for pointers corresponds then to enumerating such let bindings along branches of the syntactic tree of the canonical form (using 0 for topmost bindings). Below we state our representability theorem for $\mathsf{IA}^{2+}_{\circlearrowleft}$-terms. The definition of $\mathcal{A}_M$ is actually too generous, as we shall only need indices to decorate P-questions enabled by O-answers (in concrete examples the indices will be superscripts).

Proposition 22. *Suppose $\Gamma \vdash M : \theta$ is an $\mathsf{IA}^{2+}_{\circlearrowleft}$-term. Let $\mathcal{A}_M = M_A + (M_A \times \mathbb{N})$, where $A = [\![\Gamma \vdash \theta]\!]$. Let $\mathcal{C}_{\Gamma \vdash M}$ be the set of non-empty spinal complete plays from $[\![\Gamma \vdash M : \theta]\!]$. Then $\mathcal{C}_{\Gamma \vdash M}$ can be represented as a regular language over a* finite *subset of $\mathcal{A}_M$.*

Proof. For brevity, we shall write $\mathcal{C}_M$ instead of $\mathcal{C}_{\Gamma\vdash M}$ whenever it is clear what Γ should be. $\mathcal{C}_M$ can be decomposed as $\sum_{i\in I_A}(i\ \mathcal{C}_M^i)$. Obviously $\mathcal{C}_M$ is regular if, and only if, so is any of $\mathcal{C}_M^i$ $(i \in I_A)$. Hence, it suffices to show that $\mathcal{C}_M^i$ is regular for any relevant i. The proof proceeds by induction on the structure of canonical forms. The most difficult cases are those involving let. Note that whenever a canonical form of an $\mathsf{IA}_{\circlearrowleft}^{2+}$-term is of the shape $\mathsf{let}\, x = z(\lambda x^\theta.\mathbb{C})\,\mathsf{in}\,\mathbb{C}$, z's type must be of the form $(\beta_1 \to \beta_2) \to (\theta_1 \to \theta_2)$ (and θ is a base type). We handle this case below. Consider the terms:

$$\Gamma, z : (\beta_1 \to \beta_2) \to (\theta_1 \to \theta_2), y : \beta_1 \vdash M : \beta_2,$$
$$\Gamma, z : (\beta_1 \to \beta_2) \to (\theta_1 \to \theta_2), x : \theta_1 \to \theta_2 \vdash N : \theta'.$$

Assuming that M and N satisfy the Proposition, we show that so does $N' \equiv \mathsf{let}\, x = z(\lambda y^{\beta_1}.M)\,\mathsf{in}\, N$. We shall refer to moves contributed by $x : \theta$ with m_x. If we want to range solely over O- or P-moves from the component, we use o_x and p_x respectively. Moreover, we use $m_{z,x}, o_{z,x}, p_{z,x}$ to refer to copies of m_x, o_x, p_x in the $z : \theta' \to \theta$ component. The most common operation performed using this notation will be the relabelling of m_x to $m_{z,x}$. If θ is a function type, then there is a unique P-question q_x enabled by the initial move $\star_x$. Whenever we have a separate substitution rule for q_x, the rule for m_x or p_x will not apply to q_x. In most cases we will want to substitute $q_{z,x}^0$ ($q_{z,x}$ decorated with index 0 represent a topmost binding) for q_x. In addition, $i + 1/i$ is used to increment all numerical indices by 1. Then we have

$$\mathcal{C}_{N'}^{(i_\Gamma,\star_z)} = q_z\,\mathcal{C}'\,\star_{z,x}\,\mathcal{C}_N^{(i_\Gamma,\star_z,\star_x)}[i+1/i,\ q_{z,x}^0\mathcal{C}'/q_x,\ p_{z,x}\mathcal{C}'/p_x,\ o_{z,x}/o_x]$$

where $\mathcal{C}' = (\sum_{i\in I_{[\![\beta_1]\!]}} i_z\,\mathcal{C}_M^{(i_\Gamma,\star_z,i_y)}[j_z/j])^*$ and j ranges over $I_{[\![\beta_2]\!]}$. □

Theorem 23. *Program equivalence of* $\mathsf{IA}_{\circlearrowleft}^{2+}$*-terms is decidable.*

We remark that adding dynamic memory allocation in the form of ref to $\mathsf{IA}_{\circlearrowleft}^{2+}$, or its second-order sublanguage, results in undecidability [10]. Hence, at second order, block structure is "strictly weaker" than scope extrusion.

6 Summary

In this paper we have introduced the notion of block-innocence that has been linked with call-by-value Idealized Algol in a sequence of results. Thanks to the faithfulness of block-innocence, we could investigate the interplay between type theory, functional computation and stateful computation with block structure and dynamic allocation respectively. We have also shown a new decidability result for a carefully designed fragment of $\mathsf{IA}_{\mathsf{cbv}}$. Its extension to product types poses no particular difficulty. In fact, it suffices to follow the way we have tackled the var type, which is itself a product type. The result thus extends those from [5] and is a step forward towards a full classification of decidable fragments of $\mathsf{IA}_{\mathsf{cbv}}$: the language $\mathsf{IA}_{\circlearrowleft}^{2+}$ we considered features all second-order types and some third-order types, while finitary $\mathsf{IA}_{\mathsf{cbv}}$ is known to be undecidable at order 5 [9]. Interestingly, $\mathsf{IA}_{\circlearrowleft}^{2+}$ features restrictions that are compatible with the use of higher-order types in PASCAL [8], in which procedure parameters cannot be procedures with procedure parameters. An interesting topic for future work is a category-theoretic characterization of block-innocence.

References

1. Abramsky, S., Jagadeesan, R., Malacaria, P.: Full abstraction for PCF. Information and Computation 163, 409–470 (2000)
2. Abramsky, S., McCusker, G.: Call-by-value games. In: Nielsen, M. (ed.) CSL 1997. LNCS, vol. 1414, pp. 1–17. Springer, Heidelberg (1998)
3. Abramsky, S., McCusker, G.: Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In: O'Hearn, P.W., Tennent, R.D. (eds.) Algol-like languages, pp. 297–329. Birkhaüser, Basel (1997)
4. Gabbay, M.J., Pitts, A.M.: A new approach to abstract syntax with variable binding. Formal Aspects of Computing 13, 341–363 (2002)
5. Ghica, D.R.: Regular-language semantics for a call-by-value programming language. In: Proceedings of MFPS. Electronic Notes in Computer Science, vol. 45. Elsevier, Amsterdam (2001)
6. Honda, K., Yoshida, N.: Game-theoretic analysis of call-by-value computation. Theoretical Computer Science 221(1-2), 393–456 (1999)
7. Hyland, J.M.E., Ong, C.-H.L.: On Full Abstraction for PCF: I. Models, observables and the full abstraction problem, II. Dialogue games and innocent strategies, III. A fully abstract and universal game model. Information and Computation 163(2), 285–408 (2000)
8. Mitchell, J.C.: Concepts in programming languages. Cambridge University Press, Cambridge (2002)
9. Murawski, A.S.: About the undecidability of program equivalence in finitary languages with state. ACM Transactions on Computational Logic 6(4), 701–726 (2005)
10. Murawski, A.S.: Functions with local state: regularity and undecidability. Theoretical Computer Science 338(1/3), 315–349 (2005)
11. Oles, F.: Type algebras, functor categories and block structure. In: Nivat, M., Reynolds, J.C. (eds.) Algebraic Methods in Semantics, pp. 543–573. Cambridge University Press, Cambridge (1985)
12. Ong, C.-H.L.: Observational equivalence of 3rd-order Idealized Algol is decidable. In: Proceedings of IEEE Symposium on Logic in Computer Science, pp. 245–256. Computer Society Press (2002)
13. Pitts, A.M., Stark, I.: On the observable properties of higher order functions that dynamically create local names, or: What's new? In: Borzyszkowski, A.M., Sokolowski, S. (eds.) MFCS 1993. LNCS, vol. 711, pp. 122–141. Springer, Heidelberg (1993)
14. Plotkin, G.D.: LCF considered as a programming language. Theoretical Computer Science 5, 223–255 (1977)
15. Power, J.: Semantics for local computational effects. Electr. Notes Theor. Comput. Sci. 158, 355–371 (2006)
16. Reynolds, J.C.: The essence of Algol. In: de Bakker, J.W., van Vliet, J.C. (eds.) Algorithmic Languages, pp. 345–372. North Holland, Amsterdam (1981)
17. Stark, I.D.B.: Names and Higher-Order Functions. PhD thesis, University of Cambridge Computing Laboratory, Technical Report No. 363 (1995)
18. Tzevelekos, N.: Full abstraction for nominal general references. Logical Methods in Computer Science 5(3) (2009)

Completeness for Algebraic Theories of Local State

Sam Staton

Computer Laboratory, University of Cambridge

Abstract. Every algebraic theory gives rise to a monad, and monads allow a meta-language which is a basic programming language with side-effects. Equations in the algebraic theory give rise to equations between programs in the meta-language. An interesting question is this: to what extent can we put equational reasoning for programs into the algebraic theory for the monad?

In this paper I focus on local state, where programs can allocate, update and read the store. Plotkin and Power (FoSSaCS'02) have proposed an algebraic theory of local state, and they conjectured that the theory is complete, in the sense that every consistent equation is already derivable. The central contribution of this paper is to confirm this conjecture. To establish the completeness theorem, it is necessary to reformulate the informal theory of Plotkin and Power as an enriched algebraic theory in the sense of Kelly and Power (JPAA, 89:163–179). The new presentation can be read as 14 program assertions about three effects.

The completeness theorem for local state is dependent on certain conditions on the type of storable values. When the set of storable values is finite, there is a subtle additional axiom regarding quotient types.

1 Introduction

In this paper we are interested in reasoning about local state, about programs such as

```
let val a = ref(3) in a:=4; !a end;        (1)
```

As Moggi suggested [16], one way to give a denotational semantics to a side-effecting program of type $\tau_1 \to \tau_2$ is to give a morphism $[\![\tau_1]\!] \to T([\![\tau_2]\!])$, in a category equipped with a monad T.

Many monads arise as free models of algebraic theories. Formally, we may say that an equation for a monad T is a pair of morphisms $B \to T(A)$. Thus an equation can be thought of as a pair of denotations of programs, i.e. an assertion that two programs are the same. A system of equations for a monad typically gives rise to a quotient monad in which the equations are satisfied. In summary: by specifying equations between denotations of programs, we can construct a new denotational semantics which is sound for these equations.

In assessing the power of this technique, it is important to ask whether a monad is complete in the following sense: every equation $(e_1, e_2)\colon B \to T(A)$ is either already true (i.e. $e_1 = e_2$), or it is inconsistent (i.e. only satisfied in trivial models). This property is sometimes called Hilbert-Post completeness. It means that our monad is 'as good as possible', given the base category. There are no further equations between programs that can be accommodated in our model.

For local state, the category of sets is insufficient, and so we move to the category of nominal sets (first considered by Gabbay and Pitts [9] as a model of variable binding). This category has an object $\mathbb{A}$ of 'atoms', which we will think of as locations. This object captures many of the important properties of locations. It is infinite, although there is no injection $\mathbb{N} \rightarrowtail \mathbb{A}$ from the natural numbers; informally, the locations are local and cannot be globally enumerated.

The theory of local state is enriched in the category of nominal sets. Let $\mathbb{V}$ be some (nominal) set of values. The operations of the theory induce generic effects, which include $\mathsf{upd}\colon \mathbb{A} \times \mathbb{V} \to T(1)$ (update a location with a value), $\mathsf{lk}\colon \mathbb{A} \to T(\mathbb{V})$ (lookup the value in a location and return it), $\mathsf{ref}\colon \mathbb{V} \to T(\mathbb{A})$ (return a new cell containing a given value). With the let construction of Moggi's meta-language (and some infix notation), we can write programs such as (1).

The central contribution of this paper is Theorem 5, where we show that the enriched algebraic theory of local state is complete. This solves a conjecture of Plotkin and Power [20].

For Theorem 5 to hold, the set of storable values must be infinite. When it is finite, the theory of local state is not complete. If there are only two values, then there is an interesting additional equation involving quotient types.

Aside: limitations of a naive semantics. The denotational semantics in nominal sets works well at first order, but it is well known that such a naive semantics is limited at higher-order. At higher order, a more careful treatment of functions is needed. Consider the following programs, of type $1 \to T(\mathbb{A} \to \{\mathsf{tt}, \mathsf{ff}\})$ (see e.g. [19]).

$$\mathsf{let}\ \mathrm{a} \Leftarrow \mathsf{ref}(\mathrm{v})\ \mathsf{in}\ (\lambda \mathrm{b}.\ \mathsf{ff}) \qquad \mathsf{let}\ \mathrm{a} \Leftarrow \mathsf{ref}(\mathrm{v})\ \mathsf{in}\ (\lambda \mathrm{b}.\ [\mathrm{a} \overset{?}{=} \mathrm{b}]) \tag{2}$$

(Here, $[(-) \overset{?}{=} (-)]\colon \mathbb{A} \to \{\mathsf{tt}, \mathsf{ff}\}$ is the equality test function.) There is a compelling operational argument for considering these expressions to be equivalent: they both allocate a new location (a), but they never reveal it, so it should never be received as an argument. However, in the category of nominal sets, there is an isomorphism $i\colon [\mathbb{A} \to \{\mathsf{tt}, \mathsf{ff}\}] \cong P(\mathbb{A})$ between the nominal set of functions $[\mathbb{A} \to \{\mathsf{tt}, \mathsf{ff}\}]$ and the nominal set $P(\mathbb{A})$ of finite and cofinite sets of atoms, such that $i(\lambda \mathrm{b}.\ [a = \mathrm{b}]) = \{a\}$ and $i(\lambda \mathrm{b}.\ \mathsf{ff}) = \emptyset$. There is also a cardinality function $\mathit{card}\colon P(\mathbb{A}) \to \mathbb{N} \uplus \{\omega\}$. The following argument is typical for showing that equations between programs cannot be made into equations in the algebraic theory. If we equate the programs (2) in our theory of local state, we will be able to conclude that

$$\begin{aligned} 0 &= \mathsf{let}\ \mathrm{f} \Leftarrow \big(\mathsf{let}\ \mathrm{a} \Leftarrow \mathsf{ref}(\mathrm{v})\ \mathsf{in}\ (\lambda \mathrm{b}.\ \mathsf{ff})\big)\ \mathsf{in}\ \mathit{card}(i(\mathrm{f})) \\ &= \mathsf{let}\ \mathrm{f} \Leftarrow \big(\mathsf{let}\ \mathrm{a} \Leftarrow \mathsf{ref}(\mathrm{v})\ \mathsf{in}\ (\lambda \mathrm{b}.\ [\mathrm{a} \overset{?}{=} \mathrm{b}])\big)\ \mathsf{in}\ \mathit{card}(i(\mathrm{f})) \\ &= 1 \end{aligned}$$

The problem here is that the composite $(\mathit{card} \cdot i) : [\mathbb{A} \to \{\mathsf{tt}, \mathsf{ff}\}] \to (\mathbb{N} \uplus \{\omega\})$ will not be definable in any reasonable programming language. Authors have used logical relations [3,4,15,18,22,26], game semantics [1,14,17], and bisimulation [12,24] to make denotational semantics that can support some higher-order properties.

Structure. After recalling background material, I give a new presentation of a theory of local state in Section 4. In Section 5 I prove that the theory of local state is complete when the set of storable values is infinite, and we discuss the situation when the set of values is finite. I conclude in Section 6 by outlining the innovations that were helpful in moving from the theory proposed by Plotkin and Power [20] to the theory in Sec. 4, so that the completeness result could be stated and proved.

2 Presentations of Enriched Algebraic Theories and Strong Monads

We now recall some aspects of the presentation of enriched algebraic theories from the exposition by Kelly and Power [13], simplified and adapted to the needs of this work. In particular, we focus on the case of cartesian structure, rather than arbitrary monoidal structure.

Let $\mathcal{C}$ be a cartesian closed category. A *signature* in $\mathcal{C}$ is a set Op, thought of as a set of operators, and an assignment to each operator $\mathit{op} \in \mathit{Op}$ of two objects of $\mathcal{C}$, called the arity and coarity of op. When $\mathit{op} \in \mathit{Op}$ has arity A and coarity B, we write $(\mathit{op}\colon B \to A) \in \mathit{Op}$.

An *algebra* for a signature Op is an object X of $\mathcal{C}$ together with, for every $(\mathit{op}\colon B \to A) \in \mathit{Op}$, a morphism $\mathit{op}_X \colon B \times X^A \to X$. Homomorphism of algebras is defined in the evident way.

We will only consider signatures Op for which the category of Op-algebras is monadic over $\mathcal{C}$. A sufficient condition for this is that $\mathcal{C}$ is a Grothendieck topos.

Every monad T_{Op} arising from a signature Op on $\mathcal{C}$ has a *strength*. For every pair of objects X, Y, there is a morphism $\mathit{str}_{X,Y} \colon T_{\mathit{Op}}(X) \times Y \to T_{\mathit{Op}}(X \times Y)$ making certain diagrams commute (see e.g. [16, def. 32]).

For any strong monad T on $\mathcal{C}$, every morphism $f\colon B \to T(A)$ induces interpretations in T-algebras. The interpretation of f in a T-algebra $(X, x\colon T(X) \to X)$ is the following composite:

$$f_{(X,x)} \;=\; B \times X^A \xrightarrow{f\times\mathsf{id}} T(A) \times X^A \xrightarrow{\mathit{str}} T(A \times X^A) \xrightarrow{T(\mathit{eval})} T(X) \xrightarrow{x} X.$$

Informally, $f_{(X,x)}$ takes an element of B and a valuation of A in X, and returns an element of X.

An *equation* for a monad T is a pair of morphisms $\lambda, \rho\colon B \to T(A)$ with common domain and codomain. The object B is to be thought of as the context of the equation, while A is to be thought of as the type of the variables. A T-algebra (X, x) is said to *satisfy* an equation $\lambda, \rho\colon B \to T(A)$ if we have two equal morphisms: $\lambda_{(X,x)} = \rho_{(X,x)}\colon B \times X^A \to X$.

A *theory* in $\mathcal{C}$ is a pair (Op, Eq) of a signature Op and a set of equations Eq for T_{Op}. An *algebra* for a theory (Op, Eq) is an Op-algebra that satisfies all the equations in Eq. We will only consider theories for which the category of (Op, Eq)-algebras is monadic over $\mathcal{C}$ (e.g. $\mathcal{C}$ is a Grothendieck topos). The resulting monad $T_{(Op,Eq)}$ is again strong.

2.1 Simple Meta-language for a Strong Monad

We will use a variant of Moggi's simple meta-language [16] for reasoning about generalized elements of a strong monad. Let T be a strong monad on a category $\mathcal{C}$ with products.

The types of the meta-language are the objects of $\mathcal{C}$. The terms of the meta-language are built from variables (roman type) and the following grammar:

$$\mathrm{t} ::= \mathsf{let}\ \mathrm{y} : Y \Leftarrow \mathrm{t}\ \mathsf{in}\ \mathrm{t} \mid f(\mathrm{x}_1, \ldots, \mathrm{x}_n) \qquad (\text{for } f \colon X_1 \times \cdots \times X_n \to T(Y)).$$

When $f = \eta \cdot g$, we will elide η, the unit of T.

A typing context is an assignment of variables to types. The typing rules include structural rules such as

$$\frac{}{\mathrm{x} : X \vdash \mathrm{x} : X} \qquad \text{and} \qquad \frac{\Gamma \vdash \mathrm{t} : Z}{\Gamma, \mathrm{y} : Y \vdash \mathrm{t} : Z}$$

For every morphism $f \colon X_1 \times \cdots \times X_n \to T(Y)$ in $\mathcal{C}$, we have a well-typed term $\mathrm{x}_1 : X_1, \ldots, \mathrm{x}_n : X_n \vdash f(\mathrm{x}_1, \ldots, \mathrm{x}_n) \colon Y$. The let construction is typed by the following rule:

$$\frac{\Gamma \vdash \mathrm{t} : Y \quad \Gamma, \mathrm{y} : Y \vdash \mathrm{u} : Z}{\Gamma \vdash \mathsf{let}\ \mathrm{y} : Y \Leftarrow \mathrm{t}\ \mathsf{in}\ \mathrm{u} : Z}$$

We use the common syntactic sugar, pattern matching in let, writing $(\mathrm{t}; \mathrm{u})$ for $\mathsf{let}\ \_ : Y \Leftarrow \mathrm{t}\ \mathsf{in}\ \mathrm{u}$, etc.

For a typing context $\Gamma = (\mathrm{x}_1 : X_1, \ldots, \mathrm{x}_n : X_n)$, we let $[\![\Gamma]\!] = X_1 \times \cdots \times X_n$. Every typed term-in-context $(\Gamma \vdash \mathrm{t} : X)$ has a semantics in the category $\mathcal{C}$, $[\![\Gamma \vdash \mathrm{t} : X]\!] : [\![\Gamma]\!] \to T(X)$, given by induction on the structure of typing derivations. The interesting case is the let construction: $\Gamma \vdash \mathsf{let}\ \mathrm{y} : Y \Leftarrow \mathrm{t}\ \mathsf{in}\ \mathrm{u} : Z$ is interpreted as the following composite:

$$[\![\Gamma]\!] \xrightarrow{(\mathsf{id}, [\![\mathrm{t}]\!])} [\![\Gamma]\!] \times T(Y) \xrightarrow{str} T([\![\Gamma]\!] \times Y) \xrightarrow{T([\![\mathrm{u}]\!])} T(T(Z)) \xrightarrow{\mu} T(Z) \qquad .$$

For a monad T_{Op} arising from a signature Op, every operation $(op : B \to A) \in Op$ induces a morphism $B \to T_{Op}(A)$. These morphisms can be thought of as "generic effects" in our meta-language.

When two terms are typed in the same context, e.g. $\Gamma \vdash \mathrm{t} : X$ and $\Gamma \vdash \mathrm{u} : X$, then we will write $\Gamma \vdash \mathrm{t} = \mathrm{u} : X$ to indicate that the corresponding morphisms $[\![\mathrm{t}]\!], [\![\mathrm{u}]\!] \colon [\![\Gamma]\!] \to T(X)$ are equal. There are various sound rules for this notion of equality. For instance,

$$\frac{\Gamma \vdash \mathrm{t}_1 : X_1 \qquad \Gamma \vdash \mathrm{t}_2 : X_2 \qquad \Gamma, \mathrm{x}_1 : X_1, \mathrm{x}_2 : X_2 \vdash \mathrm{t}_3 : X_3}{\Gamma \vdash \big(\mathsf{let}\ \mathrm{x}_2 \Leftarrow (\mathsf{let}\ \mathrm{x}_1 \Leftarrow\ \mathsf{in}\ \mathrm{t}_1 \mathrm{t}_2)\ \mathsf{in}\ \mathrm{t}_3\big) = \big(\mathsf{let}\ \mathrm{x}_1 \Leftarrow\ \mathsf{in}\ \mathrm{t}_1 (\mathsf{let}\ \mathrm{x}_2 \Leftarrow \mathrm{t}_2\ \mathsf{in}\ \mathrm{t}_3)\big) : X_3}$$

Another important rule is the following substitution law:

$$\frac{\Gamma, \mathrm{x} : X \vdash \mathrm{t}_1 = \mathrm{t}_2 : Y \quad \Gamma \vdash \mathrm{u} : X}{\Gamma \vdash [{}^{\mathrm{u}}/_{\mathrm{x}}]\mathrm{t}_1 = [{}^{\mathrm{u}}/_{\mathrm{x}}]\mathrm{t}_2 \ : Y}$$

3 Rudiments of Nominal Sets

We now recall the category of nominal sets. As a category of continuous group actions, it was considered as a base category for local state by Stark [22]. The present formulation is due to Gabbay and Pitts [9].

We begin by fixing an infinite set $\mathbb{A}$ of atoms. In this paper, we think of these atoms as locations in the store. Let $\mathsf{Sym}(\mathbb{A})$ be the group of permutations on $\mathbb{A}$, ranged over by π. Recall that a $\mathsf{Sym}(\mathbb{A})$-set is a set X together with a function $\mathsf{Sym}(\mathbb{A}) \times X \to X$, such that $(\pi' \cdot \pi) \bullet x = \pi' \bullet (\pi \bullet x)$ and $\mathsf{id} \bullet x = x$.

A finite set of atoms, $A \subseteq_{\mathrm{f}} \mathbb{A}$, is said to support $x \in X$ if whenever $\pi|_A = \mathsf{id}$, $\pi \bullet x = x$. A $\mathsf{Sym}(\mathbb{A})$-set is a *nominal set* if every element has a finite support. In this case, every element $x \in X$ has a smallest supporting set, $\mathit{supp}(x)$.

Nominal sets form a category, **Nom**. A morphism $f \colon X \to Y$ is an equivariant function, i.e. for all $\pi \in \mathsf{Sym}(\mathbb{A})$ and $x \in X$, we have $f(\pi \bullet x) = \pi \bullet (f(x))$. The category of nominal sets has lots of structure.

- The set $\mathbb{A}$ of atoms is a nominal set, with action $\pi \bullet a = \pi(a)$.
- Any set X can be made into a nominal set with discrete action: for all $x \in X$, let $\pi \bullet x = x$. For example, the terminal nominal set has one element.
- The product of two nominal sets can be made into a nominal set.
- The set of all (not necessarily equivariant) functions $X \to Y$ between two nominal sets has a $\mathsf{Sym}(\mathbb{A})$-set structure given by $(\pi \bullet f)(x) = \pi \bullet (f(\pi^{-1} \bullet x))$. With this structure, not all functions have finite support. We write $[X \to_{\mathrm{fs}} Y]$ or Y^X for the set of finitely supported functions $X \to Y$. This is the cartesian closed structure of the category of nominal sets.
- If X is a nominal set and R is an equivalence relation on X that is equivariant (i.e. $x \ R \ x' \implies (\pi \bullet x) \ R \ (\pi \bullet x')$) then the quotient $X/_R$ has a natural nominal set structure.
- Given two nominal sets X, Y, we can form the disjoint product:

  $$X \otimes Y = \{(x, y) \mid x \in X,\ y \in Y,\ \mathit{supp}(x) \cap \mathit{supp}(y) = \emptyset\} \quad .$$

  In particular $\mathbb{A} \otimes \mathbb{A}$ is the set of pairs of distinct atoms. We will write $\mathbb{A}^{\otimes n}$ for the n-fold disjoint product of $\mathbb{A}$. The nominal sets $\{\mathbb{A}^{\otimes n} \mid n \in \mathbb{N}\}$ form a generator of **Nom**: if two equivariant functions $f, g \colon X \to Y$ are different then there is $n \in \mathbb{N}$ and $h \colon \mathbb{A}^{\otimes n} \to X$ such that $f \cdot h \neq g \cdot h$.

4 A Theory of Local State

A new presentation of the theory of local state is given in Figure 1. It is an algebraic theory enriched in the category of nominal sets. We build it from a theory of global state, a theory of block, and four additional equations specifying how these theories interact. I have specified the equations for the theory using the syntax for the meta-language for T_{Op}. For example, the notation

$$\text{GS2.}\quad \mathrm{a} : \mathbb{A} \vdash \mathsf{let}\ \mathrm{v} \Leftarrow {!\mathrm{a}}\ \mathsf{in}\ \mathsf{let}\ \mathrm{w} \Leftarrow {!\mathrm{a}}\ \mathsf{in}\ (\mathrm{v},\mathrm{w}) \ \approx\ \mathsf{let}\ \mathrm{v} \Leftarrow {!\mathrm{a}}\ \mathsf{in}\ (\mathrm{v},\mathrm{v}) \ : \mathbb{V} \times \mathbb{V}$$

describes two morphisms $\mathbb{A} \to T_{Op}(\mathbb{V} \times \mathbb{V})$.

The theory is parametrized in a set $\mathbb{V}$ of values. We consider $\mathbb{V}$ as a discrete nominal set.

Note that, by equation $\mathrm{B3}_1$, and basic properties of nominal sets, we have

$$\mathrm{v},\mathrm{w} : \mathbb{V} \vdash \mathsf{let}\ \mathrm{a} \Leftarrow \mathsf{ref}(\mathrm{v})\ \mathsf{in}\ \mathsf{let}\ \mathrm{b} \Leftarrow \mathsf{ref}(\mathrm{w})\ \mathsf{in}\ [\mathrm{a} \overset{?}{=} \mathrm{b}] \ = \ \mathsf{ff} \ : \{\mathsf{tt},\mathsf{ff}\} \quad .$$

The theory of **global state** has two operations, $\mathsf{lk}\colon \mathbb{A} \to \mathbb{V}$ and $\mathsf{upd}\colon \mathbb{A} \times \mathbb{V} \to 1$. We use infix notation, respectively $!a$ ("*look-up location* a") and $a := v$ ("*update location* a *to* v"). There are 7 equations:

GS1. $\mathrm{a} : \mathbb{A} \vdash \mathsf{let}\ \mathrm{v} \Leftarrow {!\mathrm{a}}\ \mathsf{in}\ \mathrm{a} := \mathrm{v} \ \approx\ () \ : 1$

GS2. $\mathrm{a} : \mathbb{A} \vdash \mathsf{let}\ \mathrm{v} \Leftarrow {!\mathrm{a}}\ \mathsf{in}\ \mathsf{let}\ \mathrm{w} \Leftarrow {!\mathrm{a}}\ \mathsf{in}\ (\mathrm{v},\mathrm{w}) \ \approx\ \mathsf{let}\ \mathrm{v} \Leftarrow {!\mathrm{a}}\ \mathsf{in}\ (\mathrm{v},\mathrm{v}) \ : \mathbb{V} \times \mathbb{V}$

GS3. $\mathrm{a} : \mathbb{A}, \mathrm{v},\mathrm{w} : \mathbb{V} \vdash \mathrm{a} := \mathrm{v};\ \mathrm{a} := \mathrm{w} \ \approx\ \mathrm{a} := \mathrm{w} \ : 1$

GS4. $\mathrm{a} : \mathbb{A}, \mathrm{v} : \mathbb{V} \vdash \mathrm{a} := \mathrm{v};\ \mathsf{let}\ \mathrm{w} \Leftarrow {!\mathrm{a}}\ \mathsf{in}\ \mathrm{w} \ \approx\ \mathrm{a} := \mathrm{v};\ \mathrm{v} \ : \mathbb{V}$

GS5. $\mathrm{a},\mathrm{b} : \mathbb{A} \vdash \mathsf{let}\ \mathrm{v} \Leftarrow {!\mathrm{a}}\ \mathsf{in}\ \mathsf{let}\ \mathrm{w} \Leftarrow {!\mathrm{b}}\ \mathsf{in}\ (\mathrm{v},\mathrm{w})$
$\qquad\approx\ \mathsf{let}\ \mathrm{w} \Leftarrow {!\mathrm{b}}\ \mathsf{in}\ \mathsf{let}\ \mathrm{v} \Leftarrow {!\mathrm{a}}\ \mathsf{in}\ (\mathrm{v},\mathrm{w}) \ : \mathbb{V} \times \mathbb{V}$

GS6. $(\mathrm{a},\mathrm{b}) : \mathbb{A} \otimes \mathbb{A}, \mathrm{v},\mathrm{w} : \mathbb{V} \vdash \mathrm{a} := \mathrm{v};\ \mathrm{b} := \mathrm{w} \ \approx\ \mathrm{b} := \mathrm{w};\ \mathrm{a} := \mathrm{v} \ : 1$

GS7. $(\mathrm{a},\mathrm{b}) : \mathbb{A} \otimes \mathbb{A}, \mathrm{v} : \mathbb{V} \vdash \mathrm{a} := \mathrm{v};\ {!\mathrm{b}} \ \approx\ \mathsf{let}\ \mathrm{w} \Leftarrow {!\mathrm{b}}\ \mathsf{in}\ \mathrm{a} := \mathrm{v};\ \mathrm{w} \ : \mathbb{V}$

The theory of **block** has an operation, $\mathsf{ref}^n\colon \mathbb{A}^{\otimes n} \times \mathbb{V} \to \mathbb{A}^{\otimes(n+1)}$, for every natural number $n \in \mathbb{N}$. Infix, we write, $\mathsf{ref}^n(\vec{a}; v)$; the intuition is "*allocate a new location, different from* $\vec{a}$, *initialized with* v". We use a shorthand: $\mathsf{ref}(v) = \mathsf{ref}^0(\vec{a}; v)$.

There are two equations and one equation schema. For each $n \in \mathbb{N}$, we write p_n for the injection $\mathbb{A}^{\otimes(n+1)} \rightarrowtail \mathbb{A}^{\otimes n} \times \mathbb{A}$.

B1. $\mathrm{v} : \mathbb{V} \vdash \mathsf{let}\ \mathrm{a} \Leftarrow \mathsf{ref}(\mathrm{v})\ \mathsf{in}\ () \ \approx\ () \ : 1$

B2. $\mathrm{v},\mathrm{w} : \mathbb{V} \vdash \mathsf{let}\ \mathrm{a} \Leftarrow \mathsf{ref}(\mathrm{v})\ \mathsf{in}\ \mathsf{let}\ \mathrm{b} \Leftarrow \mathsf{ref}(\mathrm{w})\ \mathsf{in}\ (\mathrm{a},\mathrm{b})$
$\qquad\approx\ \mathsf{let}\ \mathrm{b} \Leftarrow \mathsf{ref}(\mathrm{w})\ \mathsf{in}\ \mathsf{let}\ \mathrm{a} \Leftarrow \mathsf{ref}(\mathrm{v})\ \mathsf{in}\ (\mathrm{a},\mathrm{b}) \ : \mathbb{A} \times \mathbb{A}$

$\mathrm{B3}_n$. $\mathrm{v}\colon \mathbb{V}, \vec{\mathrm{a}}\colon \mathbb{A}^{\otimes n} \vdash \mathsf{let}\ \vec{\mathrm{b}} \Leftarrow \mathsf{ref}^n(\vec{\mathrm{a}}; \mathrm{v})\ \mathsf{in}\ p_n(\vec{\mathrm{b}}) \ \approx\ \mathsf{let}\ \mathrm{b} \Leftarrow \mathsf{ref}(\mathrm{v})\ \mathsf{in}\ (\vec{\mathrm{a}},\mathrm{b}) \ : \mathbb{A}^{\otimes n} \times \mathbb{A}$

The theory of **local state** combines the theory of global state with the theory of block, with 4 additional equations:

LS1. $\mathrm{v},\mathrm{w} : \mathbb{V} \vdash \mathsf{let}\ \mathrm{a} \Leftarrow \mathsf{ref}(\mathrm{v})\ \mathsf{in}\ \mathrm{a} := \mathrm{w};\ \mathrm{a} \ \approx\ \mathsf{let}\ \mathrm{a} \Leftarrow \mathsf{ref}(\mathrm{w})\ \mathsf{in}\ \mathrm{a} \ : \mathbb{A}$

LS2. $\mathrm{v} : \mathbb{V} \vdash \mathsf{let}\ \mathrm{a} \Leftarrow \mathsf{ref}(\mathrm{v})\ \mathsf{in}\ \mathsf{let}\ \mathrm{w} \Leftarrow {!\mathrm{a}}\ \mathsf{in}\ (\mathrm{w},\mathrm{a})$
$\qquad\approx\ \mathsf{let}\ \mathrm{a} \Leftarrow \mathsf{ref}(\mathrm{v})\ \mathsf{in}\ (\mathrm{v},\mathrm{a}) \ : \mathbb{V} \times \mathbb{A}$

LS3. $\mathrm{a} : \mathbb{A}, \mathrm{v},\mathrm{w} : \mathbb{V} \vdash \mathsf{let}\ \mathrm{b} \Leftarrow \mathsf{ref}(\mathrm{v})\ \mathsf{in}\ \mathrm{a} := \mathrm{w};\ \mathrm{b} \ \approx\ \mathrm{a} := \mathrm{w};\ \mathsf{let}\ \mathrm{b} \Leftarrow \mathsf{ref}(\mathrm{v})\ \mathsf{in}\ \mathrm{b} \ : \mathbb{A}$

LS4. $\mathrm{a} : \mathbb{A}, \mathrm{v} : \mathbb{V} \vdash \mathsf{let}\ \mathrm{b} \Leftarrow \mathsf{ref}(\mathrm{v})\ \mathsf{in}\ \mathsf{let}\ \mathrm{w} \Leftarrow {!\mathrm{a}}\ \mathsf{in}\ (\mathrm{w},\mathrm{b})$
$\qquad\approx\ \mathsf{let}\ \mathrm{w} \Leftarrow {!\mathrm{a}}\ \mathsf{in}\ \mathsf{let}\ \mathrm{b} \Leftarrow \mathsf{ref}(\mathrm{v})\ \mathsf{in}\ (\mathrm{w},\mathrm{b}) \ : \mathbb{A}$

Fig. 1. The theory of local state, enriched in the category of nominal sets. The theory is parametrized on a set $\mathbb{V}$ of values.

(We could alternatively reason about the theory of local state in a 'nominal equational logic' (e.g. [5,6,8]) but we would then have to restrict to a finite set of values and we would have no guarantee of the strength of the resulting monad.)

4.1 Algebras for Local State

We now construct algebras for the theory of local state. We begin with algebras for global state; we then consider algebras for block; and finally we combine these ideas to arrive at algebras for local state.

To begin, we consider the nominal set $\mathbb{S} = [\mathbb{A} \to_{\mathrm{fs}} \mathbb{V}]$. These functions are to be thought of as stores. Notice that a function $S \colon \mathbb{A} \to \mathbb{V}$ has support $A \subseteq_{\mathrm{f}} \mathbb{A}$ if and only if there is $v \in \mathbb{V}$ such that for all $a \in (\mathbb{A} \setminus A)$ we have $s(a) = v$. We can think of a store as being initialized to some value, and then subjected to a finite modification.

For any nominal set X, a *computation in* X is a finitely supported function $\chi \colon \mathbb{S} \to_{\mathrm{fs}} (\mathbb{S} \times X)$. We write χ_1 and χ_2 for the left and right projections, respectively. The computations form a model of the theory of global state:

– We define an equivariant function $\mathsf{upd}_X \colon \mathbb{A} \times \mathbb{V} \times (\mathbb{S} \times X)^{\mathbb{S}} \to (\mathbb{S} \times X)^{\mathbb{S}}$ as follows: let $(\mathsf{upd}_X(a, v, \chi))(S) = ((\chi_1(S))[{}^{v}\!/\!_{a}], \chi_2(S))$, where $(\chi_1(S))[{}^{v}\!/\!_{a}]$ is the store which behaves like $\chi_1(S)$, except that location a maps to v.
– We define an equivariant function $\mathsf{lk}_X : \mathbb{A} \times \left((\mathbb{S} \times X)^{\mathbb{S}}\right)^{\mathbb{V}} \to (\mathbb{S} \times X)^{\mathbb{S}}$ as follows: let $(\mathsf{lk}_X(a, \bar{\chi}))(S) = (\bar{\chi}(S(a)))(S)$. Here, $\bar{\chi}$ is a finitely supported function $\mathbb{V} \to_{\mathrm{fs}} (\mathbb{S} \times X)^{\mathbb{S}}$.

We are primarily interested in the free model of global state. The structure $(\mathbb{S} \times X)^{\mathbb{S}}$ is not the free model of global state on X because it typically contains too much. For instance, there is a computation in $(\mathbb{S} \times \mathbb{N})^{\mathbb{S}}$ that counts the number of different values in memory. Assuming that the values are numbers, there is a computation in $(\mathbb{S} \times 1)^{\mathbb{S}}$ that adds 7 to every memory cell.

To cut down our model, we say a finite set $A \subseteq_{\mathrm{f}} \mathbb{A}$ *storage-supports* χ if whenever two stores $S, S' \in \mathbb{S}$ agree on A (i.e. $S|_A = S'|_A$) then we have

1. $(\chi_1(S))|_A = (\chi_1(S'))|_A$; and
2. $\chi_2(S) = \chi_2(S')$; and
3. $(\chi_1(S))|_{\mathbb{A}\setminus A} = S|_{\mathbb{A}\setminus A}$ and $(\chi_1(S'))|_{\mathbb{A}\setminus A} = S'|_{\mathbb{A}\setminus A}$.

Storage-supporting is an equivariant property, and so we can define a nominal set $T_{\mathrm{GS}}(X)$ as follows:

$$T_{\mathrm{GS}}(X) = \{\chi : \mathbb{S} \to_{\mathrm{fs}} (\mathbb{S} \times X) \mid \mathit{supp}(\chi) \text{ storage-supports } \chi\} \ . \tag{3}$$

It is straightforward to check that the above model of global state in $(\mathbb{S} \times X)^{\mathbb{S}}$ restricts to a model in $T_{\mathrm{GS}}(X)$.

Proposition 1. *For every nominal set X, the nominal set $T_{\mathrm{GS}}(X)$ is the free algebra of the theory of global state over X.*

Next, we consider the free block algebra $T_{\mathrm{BK}}(X)$ on a nominal set:

$$T_{\mathrm{BK}}(X) \;=\; \{(s,x) \mid x \in X, s\colon \mathit{supp}(x) \rightharpoonup \mathbb{V}\}/_{\doteqdot} \quad . \tag{4}$$

Here, we are writing $s\colon \mathit{supp}(x) \rightharpoonup \mathbb{V}$ to indicate that s might be partially defined, and $\doteqdot$ is the equivalence relation generated as follows: if $\pi \in \mathsf{Sym}(\mathbb{A})$ is permutation such that $\pi|_{\mathit{supp}(x)\setminus \mathrm{dom}(s)} = \mathsf{id}$, then $(s,x) \doteqdot (s\cdot\pi^{-1}, \pi \bullet x)$. The $\mathsf{Sym}(\mathbb{A})$-set structure is given by $\pi \bullet [s,x]_{\doteqdot} = [s\cdot\pi^{-1}, \pi\bullet x]_{\doteqdot}$.

– Define an equivariant function $\mathsf{ref}^n_X\colon \mathbb{A}^{\otimes n} \times \mathbb{V} \times (T_{\mathrm{BK}}(X))^{\mathbb{A}^{\otimes(n+1)}} \to T_{\mathrm{BK}}(X)$ as follows: let

$$\mathsf{ref}^n_X(\vec{a},v,f) = \begin{cases} [s+(b\mapsto v),x]_{\doteqdot} & \text{where there is } b\in\mathbb{A} \text{ such that } [s,x]_{\doteqdot} = f(\vec{a},b) \\ & \quad\text{and } b \notin \mathit{supp}(f,\vec{a}),\ b\in \mathit{supp}(x),\ b\notin \mathrm{dom}(s). \\ [s,x]_{\doteqdot} & \text{where there is } b\in\mathbb{A} \text{ such that } [s,x]_{\doteqdot} = f(\vec{a},b) \\ & \quad\text{and } b\notin \mathit{supp}(x),\ b\notin \mathit{supp}(f(\vec{a},b)). \end{cases}$$

It is important to note that the definition of $\mathsf{ref}^n_X(\vec{a},v,f)$ is independent of the particular choice of b.

An intuition for an equivalence class $[s,x]_{\doteqdot}$ in $T_{\mathrm{BK}}(X)$ is that s is a local store that assigns values to some of the locations involved in x. Notice that $\mathit{supp}([s,x]_{\doteqdot}) = \mathit{supp}(x) \setminus \mathrm{dom}(s)$: those locations that are assigned values are local, so that it doesn't matter if they are renamed.

Now, we can consider free algebras for the full theory of local state:

Proposition 2. *The free algebra over X for the theory of local state has carrier $T_{\mathrm{GS}}(T_{\mathrm{BK}}(X))$.*

Recall that the composition of two monads is not a monad unless one can give a distributive law (e.g. [2]). Equations LS1–4 can be understood as defining a distributive law $\delta\colon T_{\mathrm{BK}}\cdot T_{\mathrm{GS}} \to T_{\mathrm{GS}}\cdot T_{\mathrm{BK}}$ (see also [11, §4], [21]): let

$$\big(\delta_X[s,\chi]_{\doteqdot}\big)(S) \;=\; \Big(S'|_{(\mathbb{A}\setminus\mathrm{dom}(s))} \cup S|_{\mathrm{dom}(s)}\ ,\ \big[S'|_{(\mathrm{dom}(s)\cap\mathit{supp}(x))}\ ,\ x\big]_{\doteqdot}\Big)$$
$$\text{where } (S',x) = \chi(S|_{(\mathbb{A}\setminus\mathrm{dom}(s))} \cup s).$$

5 Completeness

We now show that the theory of local state is complete, in the following sense. To make some preliminary definitions, we return to the situation of Section 2.

Definition 3. *A theory (Op,Eq) in a cartesian closed category is* complete *if every additional equation $B \rightrightarrows T_{(Op,Eq)}(A)$ is either satisfied in all algebras, or satisfied only in subterminal algebras.*

(Recall that a subterminal object is a subobject of the terminal object. In the category of nominal sets, the only proper subterminal object is the empty set.) Some authors call this property "Hilbert-Post completeness", after Hilbert and Post proved this property of the propositional calculus.

A useful technique for showing that an equation is only satisfied in subterminal algebras is to derive the equation $\vdash \mathsf{tt} = \mathsf{ff}\ : \{\mathsf{tt},\mathsf{ff}\}$ from it.

Lemma 4. *Let T be a strong monad on a distributive category. The two injections $1 \to T(1+1)$ are equal if and only if all T-algebras are subterminal.*

5.1 Completeness when the Storable Values Are Infinite

Theorem 5. *If the set $\mathbb{V}$ of values is infinite then the theory of local state is complete.*

This subsection is devoted to the proof of this theorem.

Consider an equation $\mathit{\Gamma} \vdash \lambda \approx \rho : X$. Suppose that it is not satisfied in the free algebra, $T_{\mathrm{LS}}(X)$, so that $\lambda \neq \rho \colon [\![\mathit{\Gamma}]\!] \to T_{\mathrm{LS}}(X)$. We proceed roughly as follows: by considering the ways that λ and ρ could be different, we construct a context $\mathcal{E}$ so that $\mathsf{tt} = \mathcal{E}[\lambda]$ and $\mathsf{ff} = \mathcal{E}[\rho]$. We then use Lemma 4 to conclude that the extra equation $(\lambda \approx \rho)$ is only satisfied in the subterminal models.

It is helpful to make use of Proposition 2 and to prove the theorem in the following two steps.

Step 1. We will first prove the following result. Let $(\mathit{Op}, \mathit{Eq})$ and $(\mathit{Op}', \mathit{Eq}')$ be theories in **Nom**, such that $T_{(\mathit{Op},\mathit{Eq})} = T_{\mathrm{GS}} \cdot T_{(\mathit{Op}',\mathit{Eq}')}$. We will assume that the theory $(\mathit{Op}', \mathit{Eq}')$ is complete with respect to $(\mathit{Op}, \mathit{Eq})$-algebras, that is, that every equation of the form $\mathit{\Gamma} \rightrightarrows T_{(\mathit{Op}',\mathit{Eq}')}(X)$ is either satisfied in all $(\mathit{Op}', \mathit{Eq}')$-algebras, or satisfied only in subterminal $(\mathit{Op}, \mathit{Eq})$-algebras. From this assumption we will conclude that the theory $(\mathit{Op}, \mathit{Eq})$ is complete.

Step 2. To conclude Theorem 5, we will assume that the nominal set $\mathbb{V}$ of values is infinite, and prove that the theory of block is complete with respect to local state algebras.

Under the hypothesis of Step 1, we consider an equation $\mathit{\Gamma} \vdash \lambda \approx \rho : X$ with $\lambda \neq \rho \colon [\![\mathit{\Gamma}]\!] \to T_{\mathrm{GS}}(T_{(\mathit{Op}',\mathit{Eq}')}(X))$. Since the nominal sets $\{\mathbb{A}^{\otimes n} \mid n \in \mathbb{N}\}$ form a generator, we have $n \in \mathbb{N}$ and an equivariant function $\gamma : \mathbb{A}^{\otimes n} \to [\![\mathit{\Gamma}]\!]$ such that $\lambda \cdot \gamma \neq \rho \cdot \gamma$. Pick an enumeration of distinct atoms $\{b_1, \dots, b_n\}$. From the characterization of T_{GS} (see (3)), we know that there must be a store $S_0 \in \mathbb{S}$ with support $\{b_1, \dots, b_n\}$ such that $\lambda(\gamma(b_1, \dots, b_n))(S_0) \neq \rho(\gamma(b_1, \dots, b_n))(S_0)$. Either

$$\begin{array}{rrcl} & (\pi_1(\lambda \cdot \gamma(b_1, \dots, b_n)))(S_0) & \neq & (\pi_1(\rho \cdot \gamma(b_1, \dots, b_n)))(S_0) \\ \text{or} & (\pi_2(\lambda \cdot \gamma(b_1, \dots, b_n)))(S_0) & \neq & (\pi_2(\rho \cdot \gamma(b_1, \dots, b_n)))(S_0) \quad . \end{array} \tag{5}$$

In the first case, we have two different stores, both supported by $\{b_1, \dots, b_n\}$. There must therefore be $i \leq n$ such that we have two different values:

$$((\pi_1(\lambda \cdot \gamma(b_1, \dots, b_n)))(S_0))(b_i) \neq ((\pi_1(\rho \cdot \gamma(b_1, \dots, b_n)))(S_0))(b_i) \text{ in } \mathbb{V}.$$

We now rewrite this observation in the monadic metalanguage. We define a term $\vec{\mathrm{a}} : \mathbb{A}^{\otimes n} \vdash \mathbf{S_0}(\vec{\mathrm{a}}) : 1$ by $\vec{\mathrm{a}} : \mathbb{A}^{\otimes n} \vdash \mathrm{a}_1 := S_0(b_1);\ \dots \mathrm{a}_n := S_0(b_n)$. We have the following equations.

$$\vec{\mathrm{a}} : \mathbb{A}^{\otimes n} \vdash \mathbf{S_0}(\vec{\mathrm{a}});\ (\lambda \cdot \gamma)(\vec{\mathrm{a}});\ !\mathrm{a}_i \approx \mathbf{S_0}(\vec{\mathrm{a}});\ (\lambda \cdot \gamma)(\vec{\mathrm{a}});\ (\pi_1(\lambda(\gamma(\vec{b}))(S_0))(b_i))$$

$$\vec{\mathrm{a}} : \mathbb{A}^{\otimes n} \vdash \mathbf{S_0}(\vec{\mathrm{a}});\ (\rho \cdot \gamma)(\vec{\mathrm{a}});\ !\mathrm{a}_i \approx \mathbf{S_0}(\vec{\mathrm{a}});\ (\rho \cdot \gamma)(\vec{\mathrm{a}});\ (\pi_1(\rho(\gamma(\vec{b}))(S_0))(b_i))$$

We will use the context $\mathcal{S}nap_{\vec{\mathrm{a}}}[-]$:

$$\begin{aligned}&\mathcal{S}nap_{\vec{\mathrm{a}}}[-] = \\ &\quad \mathsf{let}\ \mathrm{v}_1 \Leftarrow !\mathrm{a}_1\ \mathsf{in}\ \ldots \mathsf{let}\ \mathrm{v}_n \Leftarrow !\mathrm{a}_n\ \mathsf{in}\ \mathsf{let}\ \mathrm{r} \Leftarrow [-]\ \mathsf{in}\ \mathrm{a}_1 := \mathrm{v}_1;\ \ldots \mathrm{a}_n := \mathrm{v}_n;\ \mathrm{r}.\end{aligned}$$

This simple context has the property that for any nominal set X and for any term $\vec{\mathrm{a}}\colon \mathbb{A}^{\otimes n} \vdash t(\vec{\mathrm{a}})\colon 1$,

$$\vec{\mathrm{a}} : \mathbb{A}^{\otimes n} \vdash \mathcal{S}nap_{\vec{\mathrm{a}}}[t(\vec{\mathrm{a}});\ \mathsf{ff}] \approx \mathsf{ff} \qquad \text{and} \qquad \vec{\mathrm{a}} : \mathbb{A}^{\otimes n} \vdash \mathcal{S}nap_{\vec{\mathrm{a}}}[t(\vec{\mathrm{a}});\ \mathsf{tt}] \approx \mathsf{tt}\ : \{\mathsf{tt}, \mathsf{ff}\} \quad .$$

So, in this situation we can conclude the following sequence of equations:

$$\begin{aligned}\vec{\mathrm{a}}\colon \mathbb{A}^{\otimes n} \vdash \mathsf{tt} &\approx \mathcal{S}nap_{\vec{\mathrm{a}}}[\mathbf{S_0}(\vec{\mathrm{a}});\ (\lambda \cdot \gamma)(\vec{\mathrm{a}});\ \mathsf{tt}] \\ &\approx \mathcal{S}nap_{\vec{\mathrm{a}}}[\mathbf{S_0}(\vec{\mathrm{a}});\ (\lambda \cdot \gamma)(\vec{\mathrm{a}});\ [!\mathrm{a}_i \overset{?}{=} \pi_1(\lambda(\gamma(\vec{b}))(S_0))(b_i)]] \\ &\approx \mathcal{S}nap_{\vec{\mathrm{a}}}[\mathbf{S_0}(\vec{\mathrm{a}});\ (\rho \cdot \gamma)(\vec{\mathrm{a}});\ [!\mathrm{a}_i \overset{?}{=} \pi_1(\lambda(\gamma(\vec{b}))(S_0))(b_i)]] \\ &\approx \mathcal{S}nap_{\vec{\mathrm{a}}}[\mathbf{S_0}(\vec{\mathrm{a}});\ (\rho \cdot \gamma)(\vec{\mathrm{a}});\ \mathsf{ff}] \\ &\approx \mathsf{ff} \quad .\end{aligned}$$

At this point, we note that the following rule is valid for the metalanguage in nominal sets, because the projection function $[\![\Gamma]\!] \times \mathbb{A}^{\otimes n} \to [\![\Gamma]\!]$ is always epi:

$$\frac{\Gamma \vdash \mathrm{t} : X \quad \Gamma \vdash \mathrm{u} : X \quad \Gamma, \vec{\mathrm{a}} : \mathbb{A}^{\otimes n} \vdash \mathrm{t} = \mathrm{u} : X}{\Gamma \vdash \mathrm{t} = \mathrm{u} : X} \tag{6}$$

We have derived $\vdash \mathsf{tt} = \mathsf{ff}$, and so the subterminal algebras are the only algebras satisfying $\Gamma \vdash \lambda \approx \rho$. This concludes the case where $\pi_1(\lambda(\gamma(\vec{b})))(S_0) \neq \pi_1(\rho(\gamma(\vec{b})))(S_0)$.

For the other case in (5), where $\pi_2(\lambda(\gamma(\vec{b})))(S_0) \neq \pi_2(\rho(\gamma(\vec{b})))(S_0)$, we proceed as follows. We consider the equation

$$\vec{\mathrm{a}}\colon \mathbb{A}^{\otimes n} \vdash \mathcal{S}nap_{\vec{\mathrm{a}}}[\mathbf{S_0}(\vec{\mathrm{a}});\ (\lambda \cdot \gamma)(\vec{\mathrm{a}})] \approx \mathcal{S}nap_{\vec{\mathrm{a}}}[\mathbf{S_0}(\vec{\mathrm{a}});\ (\rho \cdot \gamma)(\vec{\mathrm{a}})]\ : X \quad .$$

The function described by the left hand side of this equation always returns $\pi_2(\lambda(\gamma(\vec{b})))(S_0)$, leaving the global store unchanged; and the function described by the right hand side of this equation returns $\pi_2(\rho(\gamma(\vec{b})))(S_0)$, leaving the global store unchanged. We thus have two unequal functions $\mathbb{A}^{\otimes n} \to T_{(Op',Eq')}(X)$: an equation that is not satisfied in the free (Op', Eq')-algebra $T_{(Op',Eq')}(X)$. By the assumption for Step 1, the only (Op, Eq)-algebras that satisfy this equation are subterminal. This concludes Step 1.

We now tackle Step 2: we will show that the theory of block is complete with respect to T_{LS}-algebras. We consider a pair of distinct equivariant functions $\lambda, \rho\colon \Gamma \to T_{\mathrm{BK}}(X)$ and show that this equation is only satisfied in subterminal T_{LS}-algebras. As above, we have $n \in \mathbb{N}$ and $\gamma\colon \mathbb{A}^{\otimes n} \to \Gamma$ such that $\lambda \cdot \gamma \neq \rho \cdot \gamma$.

We begin by setting up some notation. For any nominal set Y, a natural number m, and a permutation group $G < \mathsf{Sym}(m)$, we write $Y^m/_G$ for the nominal set of n-tuples $(y_1,\ldots,y_m)$ up to the equivalence relation generated by $(y_1,\ldots,y_m) \sim_G (y_{\pi(1)},\ldots,y_{\pi(m)})$ for $\pi \in G$.

The nominal set X admits the following analysis (as does every nominal set). There is an ordinary set Orb and for each $o \in Orb$, a natural number m_o and a permutation group $G_o < \mathsf{Sym}(m_o)$, together with an isomorphism

$$i\colon X \cong \coprod_{o\in Orb} \mathbb{A}^{\otimes m_o}/_{G_o} \quad .$$

We write $orb\colon X \to Orb$ for the evident projection function. This is a version of the orbit-stabilizer theorem: Orb is the set of orbits of X; and the finite groups G_o generate the stabilizers. This characterization forms the correspondence between nominal sets and named sets with symmetry [7,10].

We now pick an enumeration of distinct atoms $\{b_1,\ldots,b_n\}$, and we pick representatives (s,x) and (s',x') of the $\doteq$-equivalence classes $\lambda(\gamma(\vec{b}))$ and $\rho(\gamma(\vec{b}))$ respectively. Without loss of generality, by the definition of $\doteq$, we assume that $\mathrm{dom}(s)\cap supp(x') = \emptyset = \mathrm{dom}(s')\cap supp(x)$.

We proceed differently depending on whether x and x' are in the same orbit, whether $orb(x) = orb(x')$. If $orb(x) \neq orb(x')$, then we have the following sequence of equations in any algebra satisfying $(\lambda \approx \rho)$.

$$\begin{aligned}\vec{\mathrm{a}}\colon \mathbb{A}^{\otimes n} \vdash \mathsf{tt} &\approx \mathsf{let}\ \mathrm{r} \Leftarrow (\lambda\cdot\gamma)(\vec{\mathrm{a}})\ \mathsf{in}\ [orb(\mathrm{r}) \stackrel{?}{=} orb(x)] \\ &\approx \mathsf{let}\ \mathrm{r} \Leftarrow (\rho\cdot\gamma)(\vec{\mathrm{a}})\ \mathsf{in}\ [orb(\mathrm{r}) \stackrel{?}{=} orb(x)] \ \approx \mathsf{ff}\ \ : \{\mathsf{tt},\mathsf{ff}\} \quad .\end{aligned}$$

If x and x' are in the same orbit then we proceed as follows. It is at this point that we make use of the fact that the set $\mathbb{V}$ of values is infinite: pick n distinct values $v_1,\ldots,v_n$ that lie outside the ranges of s and s'. We define a partial function $s'' : \mathbb{A} \rightharpoonup \mathbb{V}$ as follows:

$$\begin{array}{ll} \textit{For } i \le n\textit{:} & s''(b_i) = v_i \\ \textit{For } b \in \mathrm{dom}(s)\textit{:} & s''(b) = s(b) \\ \textit{For } b' \in \mathrm{dom}(s')\textit{:} & s''(b') = s'(b') \end{array} \quad .$$

We write $\bar{o}$ for $orb(x)$, and let $\mathsf{in}_{\bar{o}}[b_1 \ldots b_{m_{\bar{o}}}]_{G_{\bar{o}}} = i(x)$ and $\mathsf{in}_{\bar{o}}[b'_1 \ldots b'_{m_{\bar{o}}}]_{G_{\bar{o}}} = i(x')$. Notice that $\mathrm{dom}(s'') = supp(x)\cup supp(x')$, and so the function s'' is defined at b_i and b'_i for every $i \le m_{\bar{o}}$. Crucially, the tuple-quotients

$$[s''(b_1),\ldots,s''(b_{m_{\bar{o}}})]_{G_{\bar{o}}} \text{ and } [s''(b'_1),\ldots,s''(b'_{m_{\bar{o}}})]_{G_{\bar{o}}} \text{ in } \mathbb{V}^{m_{\bar{o}}}/_{G_{\bar{o}}} \qquad (7)$$

are different, because $[s,x]_{\doteq} \neq [s',x']_{\doteq}$. In what follows, we abbreviate (7) by writing $[\vec{w}]_{G_{\bar{o}}}$ for the left hand tuple and $[\vec{w}']_{G_{\bar{o}}}$ for the right hand tuple.

We now translate these observations into the monadic metalanguage. We consider a derived effect

$$\mathsf{lk}\colon \coprod_{o\in Orb} (\mathbb{A}^{m_o}/_{G_o}) \longrightarrow T_{\mathrm{LS}}\Big(\coprod_{o\in Orb} (\mathbb{V}^{m_o}/_{G_o})\Big) \quad .$$

Informally, $\mathsf{lk}(\mathsf{in}_o[a_1,\ldots,a_{m_o}]_{G_o})$ returns the result of simultaneously looking up the values in locations $a_1,\ldots,a_{m_o}$. Formally, it is the unique equivariant function making the following diagram commute:

$$\begin{tikzcd}
\coprod_{o\in Orb}(\sim_{G_o}) \arrow[r] \arrow[d, shift left] \arrow[d, shift right] & \coprod_{o\in Orb}(\sim_{G_o}) \arrow[d, shift left] \arrow[d, shift right] \\
\coprod_{o\in Orb}(\mathbb{A}^{m_o}) \arrow[r, "(\star)"] \arrow[d] & T_{\mathrm{LS}}\big(\coprod_{o\in Orb}\mathbb{V}^{m_o}\big) \arrow[d] \\
\coprod_{o\in Orb}(\mathbb{A}^{m_o}/_{G_o}) \arrow[r, dotted, "\mathsf{lk}"'] & T_{\mathrm{LS}}\big(\coprod_{o\in Orb}(\mathbb{V}^{m_o}/_{G_o})\big)
\end{tikzcd} \tag{8}$$

The arrow labelled $(\star)$ corresponds to the *Orb*-fold coproduct of terms

$$\vec{\mathrm{a}}\colon \mathbb{A}^{m_o} \vdash \mathsf{let}\ \mathrm{v}_1 \Leftarrow !\mathrm{a}_1\ \mathsf{in}\ \ldots \mathsf{let}\ \mathrm{v}_{m_o} \Leftarrow !\mathrm{a}_{m_o}\ \mathsf{in}\ (\mathrm{v}_1,\ldots,\mathrm{v}_{m_o})\colon \mathbb{V}^{m_o}$$

and the upper diagrams commute by axiom GS5. Informally, it does not matter which order the locations are read, and so the G_o-equivalence classes are respected.

In any T_{LS}-algebra satisfying $(\lambda \approx \rho)$, we have

$$\begin{aligned}
\vec{\mathrm{a}} : \mathbb{A}^{\otimes n} \vdash\ \ \mathsf{tt} &\approx \mathsf{let}\ \mathrm{r} \Leftarrow (\lambda\cdot\gamma)(\vec{\mathrm{a}})\ \mathsf{in}\ \mathrm{a}_1 := v_1;\ \ldots \mathrm{a}_n := v_n;\ [\mathsf{lk}(i(\mathrm{r})) \overset{?}{=} \mathsf{in}_{\bar{o}}[\vec{w}]_{G_{\bar{o}}}] \\
&\approx \mathsf{let}\ \mathrm{r} \Leftarrow (\rho\cdot\gamma)(\vec{\mathrm{a}})\ \mathsf{in}\ \mathrm{a}_1 := v_1;\ \ldots \mathrm{a}_n := v_n;\ [\mathsf{lk}(i(\mathrm{r})) \overset{?}{=} \mathsf{in}_{\bar{o}}[\vec{w}]_{G_{\bar{o}}}] \\
&\approx \mathsf{ff}\ \ : \{\mathsf{tt},\mathsf{ff}\} \quad .
\end{aligned}$$

Using (6), we complete Step 2 and finish our proof of Theorem 5.

5.2 An Additional Axiom when the Set of Values Is Finite

The proof of Theorem 5 relies on the hypothesis that the set of values is infinite. This is the situation considered by Plotkin and Power in [20].

There are many applications where the set of storable values is finite. If there is only one value, so we have a model of the ν-calculus [19], the theory is complete, and the proof can be adapted: the theory of block remains complete with respect to the theory of local state. If $\mathbb{V}$ is empty, then all algebras are terminal. If $\mathbb{V}$ is finite and has more than one element, then the theorem fails. We will give a counter-example in the case where $\mathbb{V} = \{\mathsf{tt},\mathsf{ff}\}$. Consider the quotient nominal set $\mathbb{A}^3/_{\mathrm{C}_3}$, with $(a,b,c) \sim_{\mathrm{C}_3} (b,c,a)$. Then the following computations are distinguished in $T_{\mathrm{LS}}(\mathbb{A}^3/_{\mathrm{C}_3})$, but they are equated in some non-trivial T_{LS}-algebras.

$$\begin{aligned}
&\mathrm{c} : \mathbb{A} \vdash \mathsf{let}\ \mathrm{a} \Leftarrow \mathsf{ref}(\mathsf{tt})\ \mathsf{in}\ \mathsf{let}\ \mathrm{b} \Leftarrow \mathsf{ref}(\mathsf{ff})\ \mathsf{in}\ [\mathrm{a},\mathrm{b},\mathrm{c}]_{\mathrm{C}_3} :\ \mathbb{A}^3/_{\mathrm{C}_3} \\
&\mathrm{c} : \mathbb{A} \vdash \mathsf{let}\ \mathrm{a} \Leftarrow \mathsf{ref}(\mathsf{ff})\ \mathsf{in}\ \mathsf{let}\ \mathrm{b} \Leftarrow \mathsf{ref}(\mathsf{tt})\ \mathsf{in}\ [\mathrm{a},\mathrm{b},\mathrm{c}]_{\mathrm{C}_3} :\ \mathbb{A}^3/_{\mathrm{C}_3}
\end{aligned} \tag{L5}$$

Notice that if we add an additional constant $\star$ to $\mathbb{V}$, then these terms *are* distinguished in all algebras, using the context $\mathcal{E}[-] = \big(\mathrm{c} := \star; \mathsf{let}\ \mathrm{r} \Leftarrow [-]\ \mathsf{in}\ \mathsf{lk}_{\mathrm{C}_3}(\mathrm{r})\big)$. Here, the effect $\mathsf{lk}_{\mathrm{C}_3} : \mathbb{A}^3/_{\mathrm{C}_3} \to T_{\mathrm{LS}}(\mathbb{V}^3/_{\mathrm{C}_3})$ is to be defined as in (8), in the previous subsection.

The theory of local state is thus not complete, in the sense of Defn. 3, when the set of values is finite. This notion of completeness is arguably too strong, because the nominal set $(\mathbb{A}^3/_{C_3})$ would never arise as the denotation of an intensional type.

Recall from the work of Tzevelekos [25] that a nominal set X is said to be *strong* if, for all $x \in X$ and $a \in supp(x)$, if $\pi \bullet x = x$ then $\pi(a) = a$. Equivalently, a nominal set is strong if it is isomorphic to a coproduct of nominal sets of the form $\mathbb{A}^{\otimes n}$. Arguably, the denotations of first-order intensional types are always strong nominal sets.

Theorem 6. *Let X be a strong nominal set. Every equation that is of the form $\Gamma \rightrightarrows T_{\mathrm{LS}}(X)$ is either satisfied in all T_{LS}-algebras, or satisfied only in subterminal T_{LS}-algebras.*

This result is proved in much the same way as Theorem 5. The crucial lemma is a refined form of Step 2 in that proof.

> *Every equation of the form $\Gamma \rightrightarrows T_{\mathrm{BK}}(X)$, with X a strong nominal set, is either satisfied in all T_{BK}-algebras, or satisfied only in subterminal T_{LS}-algebras.*

6 Comparison with the Theory of Plotkin and Power

Plotkin and Power [20] propose a theory for local state, which is the starting point for the present paper. We conclude this paper by making precise the relationship between their theory and ours. Throughout this section, we suppose that $\mathbb{V}$ is a countably infinite set.

Let $\mathbf{I}$ be the category of natural numbers and injections between them, and consider the category $[\mathbf{I}, \mathbf{Set}]$ of covariant presheaves $\mathbf{I} \to \mathbf{Set}$ and natural transformations between them. We can define a presheaf $\mathbb{A}$ of locations by $\mathbb{A}(n) = n$. There is an endofunctor δ on $[\mathbf{I}, \mathbf{Set}]$, given by $(\delta X)(n) = X(n+1)$.

Definition 7. *A Plotkin-Power algebra for local state is a presheaf $X \colon \mathbf{I} \to \mathbf{Set}$ together with three natural transformations*

$$l \colon X^{\mathbb{A}} \to X^{\mathbb{V}} \qquad u \colon X \to X^{\mathbb{A} \times \mathbb{V}} \qquad b \colon \delta X \to X^{\mathbb{V}}$$

subject to 13 commuting diagrams [20, Sec. 4].

Theorem 8. *The category of Plotkin-Power algebras for local state is equivalent to the category of algebras for the monad T_{LS} on* **Nom** *(as in Section 4).*

The theory of Plotkin-Power algebras appears very similar to the theory in Figure 1, but there are two subtle points, outlined in the following proof sketch.

Every block-algebra preserves pullbacks. The first discrepancy between Fig. 1 and Defn. 7 is that the carrier of an algebra in Defn. 7 is a presheaf in $[\mathbf{I}, \mathbf{Set}]$, rather than a nominal set. It is well known that the category **Nom** of nominal sets is equivalent to the category of pullback-preserving-functors $\mathbf{I} \to \mathbf{Set}$, but not every presheaf $\mathbf{I} \to \mathbf{Set}$ preserves pullbacks.

The proof of completeness (Sec. 5) is significantly simplified by working in the category of nominal sets. Equality is decidable, and we have the principle (6).

Define a *block algebra* to be a presheaf X together with a natural transformation $\delta X \to X^{\mathbb{V}}$, such that the relevant diagrams from [20,21, def. 5.1] commute.

Proposition 9. *The carrier of every block algebra preserves pullbacks.*

One way to prove Prop. 9 is to consider a small category $\mathbf{B}_{\mathbb{V}}$ whose objects are natural numbers, and where a morphism $f\colon n \to m$ is a function $f\colon n \to (m \uplus \mathbb{V})$ that is injective on $f^{-1}(m)$. Composition is in the style of a Kleisli category. The important observation is that the category of block algebras is equivalent to the category of functors $[\mathbf{B}_{\mathbb{V}}, \mathbf{Set}]$. Indeed, the inclusion $\mathbf{I} \to \mathbf{B}_{\mathbb{V}}$ induces a monadic forgetful functor $[\mathbf{B}_{\mathbb{V}}, \mathbf{Set}] \to [\mathbf{I}, \mathbf{Set}]$. We will say that a morphism $f\colon n \to m$ in $\mathbf{B}_{\mathbb{V}}$ is *total* if $\mathrm{im}(f) \subseteq m$. Pullbacks of total morphisms in $\mathbf{B}_{\mathbb{V}}$ are absolute, in the sense that they are preserved by every functor.

An alternative presentation of block. The presentation of Defn. 7 is not an enriched algebraic theory as in Section 2, because it involves the operation $b\colon \delta X \to X^{\mathbb{V}}$. Plotkin and Power are able to give a strength for the resulting monad by hand, but they leave open the problem of finding an algebraic presentation and hence they do not have all the effects in their meta-language.

The following result plays a crucial role in the proof of Theorem 8. The first datum corresponds to the Plotkin-Power block operator, while the second corresponds to our family $\{\mathsf{ref}^n\}$ of operators, with equations $\mathrm{B3}_n$ (Fig. 1). We write $\mathbb{A}^{\otimes n}$ for the representable functor $\mathbf{I}(n, -)\colon \mathbf{I} \to \mathbf{Set}$.

Proposition 10. *Let X, Z be presheaves in $[\mathbf{I}, \mathbf{Set}]$. The following data are equivalent.*

1. *A natural transformation $\delta X \to X^Z$*
2. *A family of natural transformations $\{\beta^n : X^{\mathbb{A}^{\otimes(n+1)}} \to X^{(\mathbb{A}^{\otimes n} \times Z)}\}_{n \in \mathbb{N}}$ making the following diagrams commute:*

$$\begin{tikzcd}
X^{(\mathbb{A}^{\otimes n} \times \mathbb{A})} \arrow[r, "X^{pn}"] \arrow[dr, bend right, "(\beta^0)^{\mathbb{A}^{\otimes n}}"'] & X^{\mathbb{A}^{\otimes(n+1)}} \arrow[d, "(\beta^n)"] \\
 & X^{(\mathbb{A}^{\otimes n} \times Z)}
\end{tikzcd} \qquad \textit{for every } n \in \mathbb{N}.$$

I present an alternative solution in [23], but using a more generous enrichment for which the completeness theorem (Thm. 5) does *not* hold.

Acknowledgements. It has been helpful to talk with Marcelo Fiore, Chung-Kil Hur, Paul Levy, Tadeusz Litak, Rasmus Møgelberg, Gordon Plotkin, John Power, and Nikos Tzevelekos. The reviewers gave very helpful feedback.

References

1. 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: LICS 2004 (2004)
2. Barr, M., Wells, C.: Toposes, Triples and Theories. Springer, Heidelberg (1984)
3. 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)
4. Birkedal, L., Stø, K.: Realizability semantics of parametric polymorphism, references, and recursive types. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 456–470. Springer, Heidelberg (2009)
5. Clouston, R.A., Pitts, A.M.: Nominal equational logic. In: Articles dedicated to Gordon Plotkin. ENTCS, vol. 172, pp. 223–257 (2007)
6. Fiore, M.P., Hur, C.-K.: Term equational systems and logics. In: Proc. of MFPS XXIV. Electron. Notes Theor. Comput. Sci, vol. 218, pp. 171–192 (2008)
7. Fiore, M.P., Staton, S.: Comparing operational models of name-passing process calculi. Inform. and Comput. 204(4), 435–678 (2006)
8. Gabbay, M., Mathijssen, A.: A formal calculus for informal equality with binding. In: Leivant, D., de Queiroz, R. (eds.) WoLLIC 2007. LNCS, vol. 4576, pp. 162–176. Springer, Heidelberg (2007)
9. Gabbay, M.J., Pitts, A.M.: A new approach to abstract syntax with variable binding. Formal Aspects of Computing 13, 341–363 (2001)
10. Gadducci, F., Miculan, M., Montanari, U.: About permutation algebras (pre)sheaves and named sets. Higher-Order Symb. Comput. 19(2-3), 283–304 (2006)
11. Hyland, M., Plotkin, G.D., Power, J.: Combining effects: Sum and tensor. Theoret. Comput. Sci. 357, 70–99 (2006)
12. Jeffrey, A., Rathke, J.: Towards a theory of bisimulation for local names. In: Proc. of LICS 1999, pp. 56–66 (1999)
13. Kelly, G.M., Power, A.J.: Adjunctions whose counits are coequalizers, and presentations of finitary enriched monads. J. Pure Appl. Algebra 89(1-2), 163–179 (1993)
14. Laird, J.: A game semantics of names and pointers. Ann. Pure Appl. Logic 151(2-3), 151–169 (2008)
15. Møgelberg, R.E.: A nominal relational model for local variables. Draft (2009)
16. Moggi, E.: Notions of computation and monads. Inform. and Comput. 93(1), 55–92
17. Murawski, A.S., Tzevelekos, N.: Full abstraction for Reduced ML. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 32–47. Springer, Heidelberg (2009)
18. O'Hearn, P.W., Tennant, R.D.: Parametricity and local variables. J. ACM 42(3), 658–709 (1995)
19. Pitts, A., Stark, I.: Observable properties of higher order functions that dynamically create local names, or: What's new? In: Borzyszkowski, A.M., Sokolowski, S. (eds.) MFCS 1993. LNCS, vol. 711. Springer, Heidelberg (1993)
20. Plotkin, G.D., Power, J.: Notions of computation determine monads. In: Nielsen, M., Engberg, U. (eds.) FOSSACS 2002. LNCS, vol. 2303, pp. 342–356. Springer, Heidelberg (2002), http://homepages.inf.ed.ac.uk/gdp/publications/
21. Power, J.: Semantics for local computational effects. In: Proc. of MFPS XXII (2006)
22. Stark, I.: Categorical models for local names. LISP and Symbolic Computation 9(1), 77–107 (1996)

23. Staton, S.: Two cotensors in one: Presentations of algebraic theories for local state and fresh names. In: Proc. of MFPS XXV (2009)
24. Sumii, E.: A complete characterization of observational equivalence in polymorphic λ-calculus with general references. In: Grädel, E., Kahle, R. (eds.) CSL 2009. LNCS, vol. 5771, pp. 455–469. Springer, Heidelberg (2009)
25. Tzevelekos, N.: Full abstraction for nominal general references. Logical Methods in Computer Science 5(3) (2009)
26. Zhang, Y., Nowak, D.: Logical relations for dynamic name creation. In: Baaz, M., Makowsky, J.A. (eds.) CSL 2003. LNCS, vol. 2803, pp. 575–588. Springer, Heidelberg (2003)

Fair Adversaries and Randomization in Two-Player Games

Eugene Asarin[1], Raphaël Chane-Yack-Fa[2,*], and Daniele Varacca[3]

[1] LIAFA - CNRS & Univ. Paris Diderot, France
[2] Univ. de Sherbrooke - Département d'informatique - Quebec, Canada
[3] PPS - CNRS & Univ. Paris Diderot, France

Abstract. Two-player games are used to model open systems. One player models the system, trying to respect some specification, while the other player models the environment. In classical model checking, the objective is to verify that the system can respect its specification, whatever the environment does.

In this article, we consider a more realistic scenario when the environment is supposed to be fair. We define a notion of fair player in two-player games. Our solution is inspired by Banach-Mazur games, and leads to a definition of a novel class of 3-player games called ABM-games. For ω-regular specifications on finite arenas, we explore the properties of ABM-games and devise an algorithm for solving them. As the main result, we show that winning in an ABM-game (i.e. winning against a fair player) is equivalent to winning with probability one against the randomized adversary.

Keywords: Games, Markov decision processes, fairness.

1 Introduction

Two-player games are used to model open systems. One player (sometimes called Adam) models the system, trying to achieve some goal, while the other player (sometimes called Eve) models the environment. In classical model checking, one wants to verify that the system can achieve the goal in *any* kind of environment. Therefore, one can assume a very evil Eve, able to exploit the smallest weakness of Adam. In many situations, the environment is not that evil. It can make choices against the system, but sometimes it can play in its favour. There are different ways to model such an environment. One possible way is to suppose that the environment makes random choices. This leads to the notion of $1\,{}^1\!/_2$-player games, or Markov decision processes (MDP). In such models, one wishes to verify whether the system can reach its goal with probability 1.

Another point of view is to suppose that the environment is *fair*. Fairness assumptions are well known in the context of closed systems. In most cases [2,14], fairness assumptions are of the form "if an action is allowed infinitely often

* During this research the second author was with LIAFA and PPS.

during a run, then it is done infinitely often". But what does it mean for a *player* to be fair? A general definition of fairness for closed systems has been proposed in [15]. It is based on a different notion of game: the Banach-Mazur game. A property is defined to be a fairness property if the good player has a winning strategy in the game. This definition has a topological characterisation in terms of comeager sets. In [14], a comparison between the general notion of fairness and Markov chains is performed. It is shown that for ω-regular specifications, a finite Markov chain satisfies the specification with probability 1, if and only if the specification is a fairness property in the sense of [15].

In this work, we propose a similar definition of fair player. The idea is to split Eve into two "sub-players", Banach (good) and Mazur (evil), playing the Banach-Mazur game between themselves. Eve is thus not always playing against Adam, but sometimes she shares his goals. However Adam does not know when this is the case, he only knows that Eve is split. To show that our notion of 2-player game with fair Eve is correct, we compare it with Markov Decision Processes. Similarly to the result of [14], we show that for games with ω-regular conditions, Adam can reach his goal with probability 1 in an MDP if and only if he can win the game against fair Eve.

The characterisation in terms of Banach-Mazur games gives a different point of view of qualitative probabilistic model checking. In [13], this point of view is used to simplify and modify the classic algorithm by Courcoubetis and Yannakakis [7]. Also it helps providing a notion of counterexample for probabilistic model checking [12]. We hope that our proposal can be the starting point for similar results in the model checking of MDPs.

Structure of the paper: In Section 2, we introduce the known notions of two-player game, Markov decision process, Banach-Mazur game, etc. We present the theorem that links Banach-Mazur games on graphs with Markov chains. In Section 3, we introduce our new game, the ABM game. We show that for parity winning condition, if player Adam wins, then he has a memoryless winning strategy, and for ω-regular condition, he has a finite memory strategy. This is the main technical result of the paper, and we present the complete proof. This proof is indeed constructive and it generates an algorithm to decide whether Adam has a winning strategy, and to produce the winning strategy when it exists. We also show that, contrary to two-player games, the ABM game is not determined, by showing a game where no player has a winning strategy. Finally, in Section 4, we show that, for parity and ω-regular goals, Adam wins the ABM game, if and only if he almost surely wins in the corresponding Markov decision process. This result uses the existence of memoryless and finite memory strategies.

2 Infinite Games on Finite Graphs

2.1 Preliminaries

A *(directed) graph* is a pair $G = (V, T)$ where V is a set of *vertices* (also called states) and T a set of *edges* (or transitions) such that $T \subseteq V \times V$. In this article, we assume that the graphs are finite, i.e. V is always a finite set. We also consider only graphs without dead-ends, i.e. for all $v \in V$ there exists a vertex $w \in V$ such that $(v, w) \in T$. We assume the reader is familiar with the notion of *strongly connected component*. A *bottom strongly connected component* U is a strongly connected component such that for all $v \in U$, $(v, v') \in T \implies v' \in U$. A *path* of a graph is an infinite sequence $x = (v_0, v_1, \dots)$ such that $(v_k, v_{k+1}) \in T$ for each $k \in \mathbb{N}$. A *path fragment* is a finite prefix $\alpha = (v_0, v_1, \dots, v_n)$ of some path.

An *initialized graph* is a pair (G, v_0), where $v_0 \in V$. An *arena* is a 4-tuple $\mathscr{G} = (G, V_A, V_E, v_0)$ where (G, v_0) is an initialized graph and $\{V_A, V_E\}$ is a partition of V . A vertex of V_A is typically represented by a square and a vertex of V_E by a circle. In case that one of the two sets V_A, V_E is empty, we identify the arena with the underlying initialized graph. A *winning condition* Ω on $\mathscr{G}$ is a subset of the set V^ω of infinite sequences on V. For $x \in V^\omega$, we denote by $\inf(x)$ the set of elements of V which appear infinitely often in x.

2.2 Two-Player Games

We define first classical 2-player games on arenas [9]. In this kind of games, each player plays successively on the arena and the game never stops.

Definition 1. *A* 2-player game *on an arena $\mathscr{G}$ is a pair $\mathbb{G}^2 = (\mathscr{G}, \Omega)$ where Ω a winning condition on $\mathscr{G}$. A* play *on $\mathscr{G}$ is a path on G starting at v_0.*

We call the two players Adam and Eve. Intuitively, the players play the game by moving on vertices a token initially placed in v_0. In each vertex v_i, if $v_i \in V_A$ then Adam moves the token to some vertex v_{i+1} so that $(v_i, v_{i+1}) \in T$. If $v_i \in V_E$ then Eve chooses a successor. Adam wins a play x if $x \in \Omega$, otherwise Eve wins.

Definition 2. *A* strategy *for Adam is a mapping $\phi : V^* \to V$ which, for each path fragment α ending in $v_i \in V_A$, returns a successor vertex $\phi(\alpha) = v_{i+1}$ such that $(v_i, v_{i+1}) \in T$. Adam* follows a strategy *ϕ during a play $x = (v_0, v_1, \dots)$ if for all $i \in \mathbb{N}$ such that $v_i \in V_A$, $\phi(v_0, \dots, v_i) = v_{i+1}$. We say that ϕ is a* winning strategy *for Adam in $\mathbb{G}^2$, if Adam wins each play beginning in v_0 following the strategy ϕ. We say also that Adam wins the game if he has a winning strategy.*

The definition is analogous for Eve. To choose the strategy, a player may just need a memory of bounded size, or no memory at all.

Definition 3. *A* finite memory *strategy for Adam is a mapping $\phi : V \times M \to V$ where M is a finite set, together with an update function $up : V \times M \to M$, and an initial state memory $m_0 \in M$. Adam follows a finite memory strategy ϕ during a play $x = (v_0, v_1, \dots)$ if there is a sequence of memory states $(m_0, m_1, \dots)$ such that for all $i \in \mathbb{N}$, $up(v_i, m_i) = m_{i+1}$ and whenever $v_i \in V_A$, then $\phi(v_i, m_i) = v_{i+1}$. A* memoryless *(or* positional*) strategy for Adam is a mapping $\phi : V \to V$, which for all vertices $v_i \in V_A$ gives a successor vertex $\phi(v_i) = v_{i+1}$.*

Positional strategies can be seen as finite memory strategies where M is a singleton.

2.3 Winning Conditions

We can consider several classes of winning conditions on arenas [9], such as reachability conditions, Büchi conditions, Muller conditions or parity conditions.

Definition 4. *A* parity game *on an arena $\mathcal{G}$ is a 2-player game together with a colouring function $c : V \to \mathbb{N}$, such that the winning condition Ω is the set of plays x such that the number* $\min(c(\inf(x)))$ *is even.*

A classical theorem is the following:

Theorem 5 ([8,16]). *If Adam has a winning strategy on a parity game, then he has a positional winning strategy.*

We can define a more general class of winning conditions, using parity automata. The definition of parity automata is standard, see for instance [9]. A winning condition Ω is said to be ω-regular if it is accepted by some parity automaton. One can transform a game with ω-regular winning condition into a parity game by making the synchronised product of the game with the automaton. In this way, one can prove the following well known fact:

Theorem 6. *If Adam has a winning strategy for an ω-regular winning condition, then Adam has a finite memory winning strategy.*

The memory needed by Adam is essentially the automaton recognizing the winning condition.

2.4 Probabilistic Models

In certain cases, we want to model unpredictable events in systems. Therefore, we want to be able to take transitions in a probabilistic way.

Definition 7. *A* $1\,{}^1\!/_2$-player game *is a triple $\mathbb{G}^{1{}^1\!/_2} = (\mathcal{G}, p, \Omega)$ where $\mathcal{G} = (G, V_A, V_E, v_0)$ is an arena, Ω a winning condition like in 2-player games, while p is a probabilistic transition function defined as $p : V_E \times V \to [0,1]$ such that $p(v_e, v) = 0$ iff $(v_e, v) \notin T$ and for all $v_e \in V_E$, $\sum_{v \in V} p(v_e, v) = 1$. If the graph G is bipartite then the game is also called Markov Decision Process (MDP). If $V_A = \emptyset$, then the game is a* ${}^1\!/_2$-player *game $\mathbb{G}^{{}^1\!/_2}$ and it is also known as Markov Chain (MC).*

In $1\,{}^1\!/_2$-player games, Eve is the ${}^1\!/_2$ player because she does not really make any choices. So we will not talk about strategies for Eve. However, strategies for Adam are defined in the same way as in 2-player games. They can be finite memory or memoryless. In this article, we will not need randomized strategies that can be found in the literature [6].

Given a strategy for the full player Adam, one generates a (possibly infinite state) Markov chain by taking the "execution tree" of the graph G and by pruning out all the choices Adam does not take. If the strategy is finite memory the corresponding Markov chain has finitely many states. One can calculate with standard techniques the probability of measurable sets of infinite paths. See for instance [6] for more details. Here we are interested in the following definition:

Definition 8. *We say that Adam has an* almost sure winning strategy *if in the Markov chain generated by the strategy, the winning condition has probability 1. Adam wins almost surely if he has an almost sure winning strategy.*

Note that ω-regular winning conditions are always measurable.

2.5 Banach-Mazur Games

A different kind of 2-player game is a game where players do not play only one transition but a sequence of transitions successively, and the alternation is not decided by the arena, but by the players themselves. This game is called a Banach-Mazur game or a path game. It is played on a graph.

Definition 9. *A* Banach-Mazur game $\mathbb{G}^{BM}$ *is given by an initialized graph* (G, v_0)*, and a winning condition* Ω*.*

The two players are B (Banach) and M (Mazur). M begins in v_0 and chooses a path fragment $(v_0^0, v_1^0, \dots, v_{n_0}^0)$ of size n_0 (also chosen). Then player B does the same from the vertex $v_{n_0}^0$. The game goes infinitely alternating player B and player M turns, so we get an infinite sequence x. Banach wins if $x \in \Omega$, otherwise Mazur wins.

Definition 10. *A play* z *of a Banach-Mazur game* $\mathbb{G}^{BM}$ *is an infinite sequence of path fragments* $z = (\beta_0\beta_1 \dots)$ *where* $\beta_k = (v_0^k, \dots, v_{n_k}^k)$*. The* flattening *of a play* z *is the corresponding infinite path. Banach wins a play* z*, if its flattening belongs to* Ω*, otherwise Mazur wins. A strategy for Banach is a mapping* $\psi : (V^*)^* \to V^+$ *which for each finite sequence* $\gamma = (\beta_0 \dots \beta_i)$ *of path fragments, gives a feasible path fragment* $\psi(\gamma) = \beta_{i+1} = (v_0^{i+1}, \dots, v_{n_{(i+1)}}^{i+1})$*. Banach follows a strategy* ψ *during a play* $x = (\beta_0\beta_1 \dots)$ *if for all* $i \in \mathbb{N}$ *such that* i *is even,* $\psi(\beta_0 \dots \beta_i) = \beta_{i+1}$*. The strategy* ψ *for Banach is winning in* $\mathbb{G}^{BM}$ *from* v_0*, if Banach wins each play starting in* v_0 *following the strategy* ψ*. The strategy for Mazur is defined analogously.*

The Banach-Mazur game was proposed by Mazur (see [11], problem 43) as a way of characterising the topological notion of co-meagerness of subsets of the unit interval. Mazur conjectured that the second player has a winning strategy if and only if the winning condition is co-meager in the standard topology. Banach proved this (and he won, as prize, a bottle of wine). Banach-Mazur games were adapted later on graphs (see [4]). Völzer, Varacca and Kindler [15] used them to give a definition of fairness in Kripke structures (equivalent to closed systems). They argued that this game generalizes the known notions of fairness in systems, and they proposed the following definition:

Definition 11. $\Omega \subseteq V^\omega$ *is a fairness property on* (G, v_0) *if* B *has a winning strategy in the Banach-Mazur game on* (G, v_0) *with winning condition* Ω*.*

We will adapt the game in order to express fairness in open systems, i.e. classical 2-player games.

An important result is the link between the Banach-Mazur games and Markov chains, shown by the following theorem.

Theorem 12 ([14]). *If (G, v_0) is an initialized graph, Ω an ω-regular winning condition and p any probabilistic transition function on V, then Ω has probability 1 in the Markov chain generated on (G, v_0) by p iff Banach wins the Banach-Mazur game on (G, v_0, Ω).*

3 ABM Games

3.1 Definitions

After having recalled the known notions of 2-player games and Banach-Mazur games, we now propose a new kind of game that somehow combines those two. In this new game, Adam plays as usual, but Eve is split in two. The two halves are called Banach and Mazur. Intuitively Banach helps Adam (he is good or "Bon" in French), while the real adversary is Mazur (evil, or "Mauvais" in French).

Definition 13. *An* ABM game *is given by an arena $\mathscr{G} = (G, V_A, V_E, v_0)$, and a winning condition Ω.*

The game $\mathbb{G}^{ABM} = (\mathscr{G}, \Omega)^{ABM}$ is played by players A, E_B and E_M. At the beginning of the game, if $v_0 \in V_A$ then A chooses a transition $(v_0, v_1) \in T$. If $v_0 \in V_E$ then E_M chooses the transition. The game goes on in the same way as in 2-player games. States that belong to V_A are controlled by player A and those in V_E by player E_M. After a while, player E_M has to let the control of V_E states to player E_B. Then, it is this one's turn to play against player A before passing the lead to E_M again and so on. The following definition formalizes the rules of play for Banach and Mazur:

Definition 14. *A* move tree *λ of player E_B (or E_M) from a state $v_k \in V_E$, is a finite, prefix closed set of path fragments starting at v_k, and verifying the following conditions:*

- *for each path fragment $\alpha \in V^*$ and each vertex $v \in V_E$, if $\alpha v \in \lambda$ then there is at most one vertex w such that $(v, w) \in T$ and $\alpha v w \in \lambda$;*
- *for each path fragment $\alpha \in V^*$ and each vertex $v \in V_A$, $\alpha v w \in \lambda$ if and only if $(v, w) \in T$.*

In a state of V_A, player A chooses a transition as in the classical 2-player game. But in a state $v_i \in V_E$ the token is moved according to the move tree λ given by E_M or E_B. In fact, if E_M (or E_B) gets the lead in v_k then $(v_k, \ldots, v_i)$ is a prefix of a branch of λ. If v_i has a successor, then E_M (E_B resp.) plays v_{i+1}, which is the unique successor of v_i in the tree. Else, E_M (E_B resp.) passes the lead.

Intuitively, in open systems, player A represents the system while players E_B and E_M the environment. When we want to verify a specification, we assume that the system always makes the right choice but the environment is not necessarily always against. Sometimes the environment helps A to satisfy the property, sometimes the environment plays against. In this way, ABM games allow us to model a *fair* environment.

Like in Banach-Mazur games, we can see an infinite play x as an infinite sequence of tree branches $x = (\beta_0\beta_1\dots)$ where $\beta_k = (v_0^k, \dots, v_{n_k}^k)$ is a path fragment where E_B is leading if k is odd and E_M is leading if k is even. Players A and E_B win the play x if the flattening of $x \in \Omega$, else E_M wins. In the following sections we will sometimes identify a play with its flattening.

Definition 15. *A strategy ϕ for player A is defined as in 2-player games. A strategy for player E_B or E_M is a mapping $\psi : V^* \to \mathscr{T}$ where $\mathscr{T}$ is the set of move trees. Player A follows a strategy ϕ during a play $x = (\beta_0\beta_1\dots)$ with flattening $x = (v_0v_1\dots)$ if for all $i \in \mathbb{N}$ such that $v_i \in V_A$, $\phi(v_0, \dots, v_i) = v_{i+1}$. E_B (E_M resp.) follows a strategy ψ during a play $x = (\beta_0\beta_1\dots)$ if for all $i \in \mathbb{N}$ such that i is even (odd resp.), $\phi(\beta_0\dots\beta_i) = \lambda_i$ and β_{i+1} is a branch of λ_i.*

A pair of strategies (ϕ, ψ) (for A and E_B respectively) is said to be winning if A and E_B win the play following their respective strategies. We say that A has a winning strategy ϕ if there exists a winning pair of strategies (ϕ, ψ) for A and E_B. Player A wins the game if he has a winning strategy.

If E_M plays the move tree on Fig. 1 at the beginning of the game, he will reach q_3 then q_4 before passing the initiative or will pass his turn if player A chooses the transition to q_1.

Remark. In the definition of ABM games, it is important to notice that a strategy for player A depends only on the sequence of the previously visited vertices. Therefore, A never knows whether he is playing against Banach or Mazur at any given time. That is why there are three players and not only two. This is important for the main result (Theorem 32) to hold.

In the example of Fig. 2, consider the following winning condition: $\Box\Diamond q_3 \wedge \Box\neg q_4$ (infinitely often q_3 but never q_4). Consider the ABM game with this winning condition. If we suppose that player A always knows who is leading in states of V_E, we can construct a winning strategy for players A and E_B.

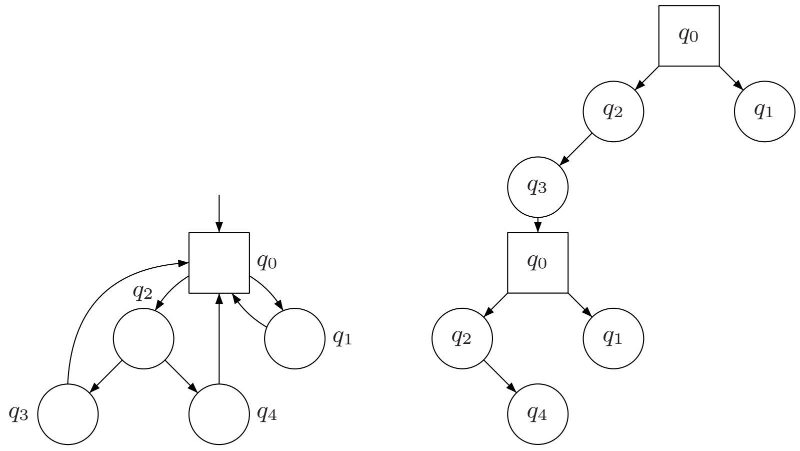


Fig. 1. Example of an arena and a move tree in this arena

While player E_M is leading, player A takes the transition going to q_1. Then he takes transition to q_2 when E_B gets the lead. Player E_B always takes transition to q_3. Thus, state q_3 is visited infinitely often but state q_4 never. However, if Eve plays randomly (with any Markovian distribution), it is not difficult to see that for any strategy of A, the winning condition has probability 0. This contradicts Theorem 32.

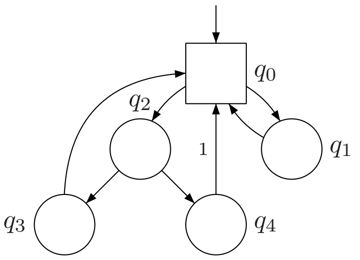


Fig. 2. A 1 1/2-player game

This is also the reason why we cannot code our game in terms of classic 2-player games. A comparison with games with imperfect information (see e.g. [5,3]) remains to be explored.

3.2 Traps and Attractors

Before presenting the main theorems, we need to study some properties of ABM arenas. The following notions were often used in proofs in classical 2-player games [9,16].

Definition 16. *Let $\mathscr{G} = ((V,T), V_A, V_E, v_0)$ be an arena. A trap for Eve (or E-trap) in $\mathscr{G}$ is a subset of vertices $U \subseteq V$ such that:*

- *for all $v \in U \cap V_E$, $(v,v') \in T \implies v' \in U$,*
- *for all $v \in U \cap V_A$, there exists $v' \in U$ such that $(v,v') \in T$.*

Trap for Adam can be defined in the same way.

The idea of the E-trap is to consider the set of vertices in which Adam can keep the token no matter what does Eve. The following easy proposition was expressed in [16] for 2-player games. It depends only on Adam, and thus it holds also for ABM games.

Proposition 17. *In an ABM game, player A has a memoryless strategy to keep the play in an E-trap.*

Definition 18. *Let $\mathscr{G} = ((V,T), V_A, V_E, v_0)$ be an arena. The attractor of $U \subseteq V$ for Eve (or E-attractor) written $Attr_E(U)$, is the limit of the sequence defined as:*

- *$Attr_E^0(U) = U$,*
- *$\forall i \geq 0, Attr_E^{i+1}(U) = Attr_E^i(U) \cup \{v \in V_E \mid \exists v' \in Attr_E^i(U)$ such that $(v,v') \in T\} \cup \{v \in V_A \mid (v,v') \in T \implies v' \in Attr_E^i(U)\}$.*

Attractor for Adam can be defined in the same way.

In 2-player games, an E-attractor of a set U induces a strategy for Eve to reach U. In ABM games, vertices of V_E are alternately controlled by E_B and E_M and thus there is no strategy for any of these 2 players to reach U. However, we have the following property on the complement anyway.

Proposition 19 ([16]). *The complement of an E-attractor in an arena is an E-trap.*

3.3 Positional Strategies

In the rest of this section, we will show that in the case of ABM games with parity winning conditions, winning strategies for Adam can be memoryless. A classical method to prove such a result is to compute the set of winning positions and show that from these states there exists a positional winning strategy. This technique was used for the 2-player games [16]. In doing so, the set of states is partitioned into winning and losing states. Later, we will observe that ABM games are not determined. However, we still can compute winning and non-winning positions. In the following, G denotes a graph, $\mathscr{G} = (G, V_A, V_E, v_0)$ an arena and $c : V \to \mathbb{N}$ a colouring mapping defining a parity winning condition as in 2-player games.

Theorem 20. *If player A wins the parity game $(\mathscr{G}, c)^{ABM}$ then A has a memoryless winning strategy.*

To start proving this result, let $\mathscr{C}_G^M = \{C_0^M, \dots, C_i^M \dots, C_n^M\}$ be the set of bottom strongly connected components of graph G that have the following property: for all $i < n$, for all $U \subseteq C_i^M$, if U is an E-trap and U is strongly connected then $min(c(U))$ is odd. We write $\mathscr{C}_G^B = \{C_0^B, \dots, C_i^B \dots, C_m^B\}$ the set of bottom strongly connected components of G that do not have that property.

This construction draws its inspiration from the proof [6] of the existence of memoryless strategies in MDP with parity winning condition. In fact, the sets $\mathscr{C}_G^M$ and $\mathscr{C}_G^B$ are inspired from the concept of *controllably win recurrent vertices* [7,6].

Lemma 21. *For all strategies ϕ for A and ψ for E_B, there exists a play x following ϕ and ψ such that* $\inf(x)$ *is an E-trap.*

Proof. Fix a pair of strategies (ϕ, ψ) for A and E_B. We will show that E_M can play in order to make x an E-trap. Therefore, we suppose that E_M knows players A and E_B's strategies and chooses his moves according to these. Anytime during the play where E_M has the initiative, we will say that a vertex $q \in V_E$ is *explored* if E_M has already taken all outgoing edges from q since the beginning of his turn. We will say that q is *exhausted* if there is no play that follows ϕ and reaches q. Player E_M will play in this way: when he gets the lead, if there exists a vertex $q \in V_E$ not *explored* and not *exhausted* then E_M tries to reach it. This is possible because q is not *exhausted*. There exists a play following ϕ that allows this. So in this visited state q, he chooses a transition that has not yet been taken since he is leading. E_M repeats this process until all vertices of V_E are marked *explored* or *exhausted*, and then passes the initiative. We notice that an *exhausted* vertex will stay forever *exhausted*. Formally, a move tree λ of E_M is a tree that owns at most one branch λ_0 such that each node $n \in V_E$ of the branch is not a leaf. Indeed, that branch represents the revealed strategy of player A. To agree with the definition of a move tree, E_M passes the lead if A does not follow his strategy ϕ. The branch λ_0 is finite. Suppose that λ_0 is infinite, then there always exists a vertex that is not *explored* and not *exhausted*. That contradicts the finiteness of the graph. Each vertex is eventually either *explored* or *exhausted*. So the tree λ is finite and it is a proper move tree.

Suppose now that for a play x obtained in that way, $\inf(x)$ is not an E-trap. Then there exist vertices $s \in V_E$ and $t \in V$ such that $(s,t) \in T$, $s \in \inf(x)$ and $t \notin \inf(x)$. $s \in \inf(x)$ then s will never be *exhausted* and each time E_M gets the initiative, he will be able to visit its successor t. So we visit t infinitely often. But $t \notin \inf(x)$, contradiction. We can conclude that $\inf(x)$ is an E-trap. □

Lemma 22. *Player A has no winning strategy in the parity game on $\mathscr{C}_G^M$.*

Proof. Let $C_i^M \in \mathscr{C}_G^M$. Assume that A and E_B have a winning pair of strategies (ϕ, ψ) on C_i^M. Then for each play x following the strategies (ϕ, ψ), $U = \inf(x)$ is a strongly connected set such that $min(c(U))$ is even. Thus, by definition of $\mathscr{C}_G^M$, U is not an E-trap. That is a contradiction regarding to previous lemma. As a consequence, A and E_B do not have winning strategies on C_i^M. So A has no winning strategy on $\mathscr{C}_G^M$. □

Lemma 23. *Player A has a memoryless winning strategy in the parity game on $\mathscr{C}_G^B$.*

Proof. Let $C_i^B \in \mathscr{C}_G^B$. By definition of $\mathscr{C}_G^B$, we know that there exists an E-trap: $U \subseteq C_i^B$ such that $min(c(U))$ is even. We write m a vertex, which has minimal colour in U. The strategy ϕ of player A is the following: for all $u \in U$ such that $u \in V_A$, A chooses a successor $v \in U$ such that for all others successors w of u, the distance between v and m is shorter than the one between w and m. And for all $u \notin U$ such that $u \in V_A$, A will choose similarly the successor with shortest distance to reach the set U. The strategy ψ for player E_B consists in using a similar strategy of shortest distance when it is his turn and passing the lead to player E_M each time the vertex m is reached. (In the same way that in Lemma 21, we do not consider the case where player A does not follow his strategy. So the move tree is finite.) Remark that distances to the vertex m and to the set U are well-defined for each vertex of the component C_i^B because of its strong connection. Let x be a play following strategies ϕ and ψ. Then $\inf(x) \subseteq U$ because the strategies allow to reach the set U and to stay in it forever. Thus the minimal vertex infinitely often reached is $min(c(\inf(x))) = m$. We can conclude that the pair of strategies (ϕ, ψ) is winning. Moreover, ϕ is clearly memoryless. So A has a memoryless winning strategy on $\mathscr{C}_G^B$. □

Now we will show that we can partition the set of vertices V of the graph into a winning region for A written W and a non-winning region for A written L. We construct L by induction:

- $G_0 = G$,
- $L_0 = Attr_E(\mathscr{C}_{G_0}^M)$,
- $G_{i+1} = G_i$ restricted to $V \setminus L_i$,
- $L_{i+1} = L_i \cup Attr_E(\mathscr{C}_{G_{i+1}}^M)$.

G is a finite graph so the sequences $(G_i)_i$ and $(L_i)_i$ converge. We write G_W the limit of the sequence $(G_i)_i$, L the limit of $(L_i)_i$ and $W = V \setminus L$.

Lemma 24. *For all $i \in \mathbb{N}$, player A has no winning strategy on L_i if E_M is leading the states of V_E.*

Proof. Player A has no winning strategy on $\mathscr{C}_G^M$ so he does not have one on L_0 either. Indeed, player E_M can play the strategy induced by the E-attractor to reach $\mathscr{C}_G^M$ where A has no winning strategy. Let $n \in \mathbb{N}$, suppose that A has no winning strategy on L_n if E_M is leading. According to Lemma 22, we know that A has no winning strategy on $\mathscr{C}_{G_{n+1}}^M$ if the play is restricted to the graph G_{n+1}. The only way for A to win would be to always reach L_n when player E_B has the initiative in L_n. By induction hypothesis, if E_M leads in L_n then A has no winning strategy. But for each play where we can reach L_n with E_B, there exists a play where we can reach L_N with E_M. Player E_M only needs to simulate the moves of E_B until he arrives at L_N. So A has no winning strategy on $\mathscr{C}_{G_{n+1}}^M$. As shown previously, player A does not have a strategy in its E-attractor either. Thus A has no winning strategy in $L_{n+1} = L_n \cup Attr_E(\mathscr{C}_{G_{n+1}}^M)$ if E_M has the initiative. □

Lemma 25. *W is an E-trap.*

Proof. For all $i \in \mathbb{N}$, $V \setminus L_i$ is an E-trap because it is the complement of an E-attractor. So W is an E-trap. □

Lemma 26. *Player A has a memoryless winning strategy on W.*

Proof. When the token is in $W \setminus \mathscr{C}_{G_W}^B$, strategies for A and E_B consist in strategies of shortest paths to the set $\mathscr{C}_{G_W}^B$ in W similar to previously described strategies. According to Lemma 25, W is an E-trap, so A has a strategy to prevent E_M from reaching L. The shortest distance strategy also allows to stay in W. Furthermore, we can always reach $\mathscr{C}_{G_W}^B$. Indeed, the only bottom strongly connected components reachable from a state of W belong to $\mathscr{C}_{G_W}^B$ by construction of L and W. The strategy of distance is then a winning and memoryless strategy. When $\mathscr{C}_{G_W}^B$ is reached, we can use the strategy described in Lemma 23, which is also winning and memoryless. □

Proof of Theorem 20. At the beginning of the game, E_M leads, so A has no winning strategy on L according to Lemma 24. Lemma 26 says that A has a memoryless winning strategy on $W = V \setminus L$. In consequence, if A has a winning strategy in the initial state then he has a memoryless winning strategy. □

We observe that the proof of Theorem 20 is constructive. It provides explicitly the winning region of Adam. Also, for each state of this region, the winning strategy is explicitly given by Lemma 23 and Lemma 26.

3.4 ABM Games Are Not Determined

Lemma 22 says that players A and E_B have no winning strategy in the parity game on $\mathscr{C}_G^M$. In general, player E_M has no winning strategy either on $\mathscr{C}_G^M$. We can infer that the game is not determined.

Consider the parity game represented on Fig. 3. Lemma 22 says that players A and E_B do not have any strategies to win the game. However, we can observe here that player E_M does not have a strategy either. If he wants to win, E_M has either to reach infinitely often state q_4 or reach infinitely often q_1 and a finite number times q_3. But E_M does not know player A's strategy. If A were to take the transition going to q_2 a finite number of times, then E_M could just pass to E_B without making any moves. But since he does not know actually if player A intends to take this transition infinitely often or not, there are two cases.

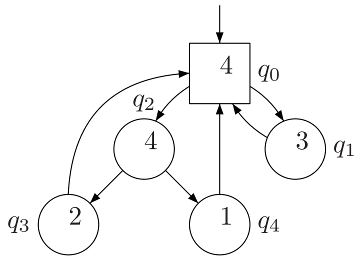


Fig. 3. Example of parity game

- If player E_M supposes that player A will reach q_2 infinitely often, then his strategy has to wait for this move and reach state q_4 before passing the lead to player E_B. However, if eventually A never reaches q_2, then E_M would never pass his turn. As a consequence, this is not a strategy because the move tree chosen by E_M must be finite.
- If E_M supposes that A will never reach q_2 again at a certain point, then he will pass his turn in q_1. But we can imagine a scenario where each time E_M passes the initiative, A take the transition to q_2 and let E_B reach q_3.

In any case, we cannot define any winning strategy for player E_M in this game.

3.5 Finite Memory

We conclude the section by extending Theorem 20 to ω-regular winning conditions, similarly to Theorem 6.

Theorem 27. *Let $\mathscr{G}$ be an arena and $\Omega \subseteq V^\omega$ an ω-regular condition. If player A has a winning strategy in the game $(\mathscr{G}, \Omega)^{ABM}$ then he has a finite-memory winning strategy.*

The proof technique is similar to the one used for Theorem 6. One makes the product with a deterministic parity automaton recognising the winning condition. The automaton is essentially the memory needed by Adam.

4 Fairness as Randomization

In this section, we intend to demonstrate a theorem similar to Theorem 12 in the context of open systems. That is we want to build a connection between ABM games and $1\,{}^1\!/\!_2$-player games. We first do this for the parity case, and then extend to all ω-regular conditions.

We start by noting that the existence of memoryless strategies for ABM games is mirrored in MDP.

Theorem 28 ([7,6]). *Let $\mathscr{G} = (G, V_A, V_E, v_0)$ be an arena, $p : V_E \times V \to [0, 1]$ a probabilistic transition function on V_E such that $p(v_e, v) = 0$ iff $(v_e, v) \notin T$ and for all $v_e \in V_E$, $\sum_{v \in V} p(v_e, v) = 1$, and $c : V \to \mathbb{N}$ a colouring mapping. If Adam wins almost surely the parity $1\,{}^1\!/\!_2$-player game $(\mathscr{G}, p, c)$ then Adam has a memoryless almost sure winning strategy.*

4.1 The Parity Case

Theorem 29. *Let $\mathscr{G} = (G, V_A, V_E, v_0)$ be an arena, c a colouring, p a probabilistic transition function on V_E and $\Omega \subseteq V^\omega$ the parity winning condition defined by c. If A has a winning strategy in the game $(\mathscr{G}, \Omega)^{ABM}$ then Adam has an almost-sure winning strategy in the $1\,{}^1\!/\!_2$-player game $(\mathscr{G}, p, \Omega)$.*

Proof. Assume that player A has a winning strategy in the game $(\mathscr{G}, \Omega)^{ABM}$. Then according to Theorem 20, A has a memoryless winning strategy ϕ. For each state of V_A, we keep only the outgoing edge provided by ϕ. Let $\mathscr{K} = ((V, T^K), v_0)$ be the initialized graph where $T^K = T \setminus \{(v_a, v) \in T \mid v_a \in V_A \text{ and } \phi(v_a) \neq v\}$. We can easily see that, if player A wins in $(\mathscr{G}, \Omega')^{ABM}$ then Banach wins in $(\mathscr{K}, \Omega')^{BM}$. Indeed, each play that is winning for A on $\mathscr{G}$ can be simulated on $\mathscr{K}$. It does not matter if Banach or Mazur has the initiative in a state of V_A because there is only one outgoing edge.

Let p' be the probabilistic transition function on V such that for all $v_e \in V_E$, $v \in V$, $p'(v_e, v) = p(v_e, v)$ and for all $v_a \in V_A$, $(v_a, v) \in T \implies p'(v_a, v) = 1$. This defines a Markov chain on $((V, T^K), v_0)$. Thanks to Theorem 12, we know that the winning condition Ω has probability 1 in this Markov chain. But this shows that ϕ is an almost sure winning strategy for Adam in the $1\,{}^1\!/\!_2$-player game $(\mathscr{G}, p, \Omega)$. □

Theorem 30. *Let $\mathscr{G} = (G, V_A, V_E, v_0)$ be an arena, c a colouring, p a probabilistic transition function on V_E and Ω the parity winning condition defined by c. If Adam has an almost sure winning strategy in the $1\,{}^1\!/\!_2$-player game $(\mathscr{G}, p, \Omega)$, then A has a winning strategy in the game $(\mathscr{G}, \Omega)^{ABM}$.*

Proof. Suppose that Adam has an almost sure winning strategy in the game $(\mathscr{G}, p, \Omega)$. Then according to Theorem 28, Adam has a memoryless almost sure winning strategy. Let $\mathscr{K} = ((V, T^K), v_0)$ be the initialized graph where $T^K = T \setminus \{(v_a, v) \in T \mid v_a \in V_A \text{ and } \phi(v_a) \neq v\}$. Let p' be the probabilistic transition function on V such that for all $v_e \in V_E$, $v \in V$, $p'(v_e, v) = p(v_e, v)$ and for all $v_a \in V_A$, $(v_a, v) \in T \implies p'(v_a, v) = 1$. This generates a Markov chain. As Adam wins almost surely in $(\mathscr{G}, p, \Omega)$ then Ω has probability 1 in the Markov chain. By Theorem 12, Banach has a winning strategy in the game $(\mathscr{K}, \Omega)^{BM}$. This means that ϕ is a winning strategy for Adam in the game $(\mathscr{G}, \Omega)^{ABM}$. The winning strategy of player E_B is to simulate Banach winning strategy in $(\mathscr{K}, \Omega)^{BM}$. Thus Adam and Banach have also a winning strategy in the game $(\mathscr{G}, \Omega)^{ABM}$. □

The following theorem results from Theorem 30 and Theorem 29.

Theorem 31. *Let $\mathscr{G} = (G, V_A, V_E, v_0)$ be an arena, c a colouring, p a probabilistic transition function on V_E and Ω the parity winning condition defined by c. Adam has an almost-sure winning strategy in the $1\,^1\!/_2$-player game $(\mathscr{G}, p, \Omega)$ if and only if A has a winning strategy in the game $(\mathscr{G}, \Omega)^{ABM}$.*

4.2 ω-Regular Conditions

The key fact in order to exploit Theorem 12 is that the graph obtained after applying the memoryless strategy of Adam is finite, as Theorem 12 applies only to finite graphs. We notice thus that Theorem 20 on the existence of memoryless strategies is essential in the proof of Theorem 32. In the case of ω-regular winning conditions, the strategy of Adam is finite memory. As in the memoryless case, the key observation is that the graph one gets by applying the strategy is finite (though larger than the original graph). Thus, it is still possible to apply Theorem 12. We omit the straightforward details of the proof.

Theorem 32. *Let $\mathscr{G} = (G, V_A, V_E, v_0)$ be an arena, p a probabilistic transition function on V_E and $\Omega \subseteq V^\omega$ an ω-regular condition. Adam has an almost-sure winning strategy in the $1\,^1\!/_2$-player game $(\mathscr{G}, p, \Omega)$ if and only if A has a winning strategy in the game $(\mathscr{G}, \Omega)^{ABM}$.*

We can notice that if $V_A = \emptyset$ then we have the special case of Theorem 12.

Thus, we showed that playing against a fair player is equivalent to playing against a probabilistic player in the case of ω-regular properties.

5 Related and Future Work

The Banach-Mazur game is one possible definition of fairness in closed systems. An equivalent topological definition can be given in terms of co-meagerness. In [1], the topological definition is used to prove the equivalence between probabilistic and fair semantics of timed automata. Interestingly, this equivalence holds only for one-clock automata, but it breaks down once we allow more than one clock. Another equivalent definition is in terms of α-fairness [10]. Of the three definitions, this is the one that most resembles the intuitive notion of fairness "if something is often possible, it will be often performed". It would be interesting to define fair strategies for Eve in terms of α-fairness. We also expect that this game-theoretic point of view can be applied to improve existing algorithms, or to find new ones, in the qualitative model checking of MDPs.

In this paper, we have applied a definition of fairness to one of the players of 2-player games. In general, we could study what happens to other players and other games. For instance, a $1\,^1\!/_2$-player game where Adam plays fairly should be equivalent to a Markov chain. Finally, we would like to explore the connections between ABM games and games with imperfect information ([5,3]).

References

1. Baier, C., Bertrand, N., Bouyer, P., Brihaye, T., Größer, M.: Almost-sure model checking of infinite paths in one-clock timed automata. In: LICS 2008, pp. 217–226. IEEE Computer Society, Los Alamitos (2008)
2. Baier, C., Kwiatkowska, M.: Model checking for a probabilistic branching time logic with fairness. Distributed Computing 11(3), 125–155 (1998)
3. Bertrand, N., Genest, B., Gimbert, H.: Qualitative determinacy and decidability of stochastic games with signals. In: LICS 2009, pp. 319–328. IEEE Computer Society, Los Alamitos (2009)
4. Berwanger, D., Grädel, E., Kreutzer, S.: Once upon a time in the West. Determinacy, complexity and definability of path games. In: Vardi, M.Y., Voronkov, A. (eds.) LPAR 2003. LNCS, vol. 2850, pp. 226–240. Springer, Heidelberg (2003)
5. Chatterjee, K., Doyen, L., Henzinger, T.A., Raskin, J.-F.: Algorithms for omega-regular games with imperfect information. Logical Methods in Computer Science 3(3) (2007)
6. Chatterjee, K., Jurdziński, M., Henzinger, T.A.: Quantitative stochastic parity games. In: SODA 2004, pp. 114–123. ACM/SIAM (2004)
7. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. Journal of the ACM 42(4), 857–907 (1995)
8. Emerson, E.A., Jutla, C.: Tree automata, mu-calculus and determinacy (extended abstract). In: FOCS 1991, pp. 368–377. IEEE Computer Society, Los Alamitos (1991)
9. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games. LNCS, vol. 2500. Springer, Heidelberg (2002)
10. Lichtenstein, O., Pnueli, A., Zuck, L.D.: The glory of the past. In: Parikh, R. (ed.) Logic of Programs 1985. LNCS, vol. 193, pp. 196–218. Springer, Heidelberg (1985)
11. Mauldin, D. (ed.): The Scottish Book. Birkhäuser, Basel (1981)
12. Schmalz, M., Varacca, D., Völzer, H.: Counterexamples in probabilistic LTL model checking for Markov chains. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009 - Concurrency Theory. LNCS, vol. 5710, pp. 587–602. Springer, Heidelberg (2009)
13. Schmalz, M., Völzer, H., Varacca, D.: Model checking almost all paths can be less expensive than checking all paths. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 532–543. Springer, Heidelberg (2007)
14. Varacca, D., Völzer, H.: Temporal logics and model checking for fairly correct systems. In: LICS 2006, pp. 389–398. IEEE Computer Society, Los Alamitos (2006)
15. Völzer, H., Varacca, D., Kindler, E.: Defining fairness. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 458–472. Springer, Heidelberg (2005)
16. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theor. Computer Science 200, 135–183 (1998)

Retaining the Probabilities in Probabilistic Testing Theory

Sonja Georgievska and Suzana Andova

Department of Mathematics and Computer Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
s.georgievska@tue.nl, s.andova@tue.nl

Abstract. This paper considers the probabilistic may/must testing theory for processes having external, internal, and probabilistic choices. We observe that the underlying testing equivalence is too strong and distinguishes between processes that are observationally equivalent. The problem arises from the observation that the classical compose-and-schedule approach yields unrealistic overestimation of the probabilities, a phenomenon that has been recently well studied from the point of view of compositionality (de Alfaro/Henzinger/Jhala 2001, Cheung/Lynch/Segala/Vaandrager 2006), in the context of randomized protocols (Chatzikokolakis/Palamidessi 2007), and in probabilistic model checking (Giro/D'Argenio/Ferrer Fioriti 2009). To that end, we propose a new testing theory, aiming at preserving the probability information in a parallel context. The resulting testing equivalence is insensitive to the exact moment the internal and the probabilistic choices occur. We also give an alternative characterization of the testing preorder as a probabilistic ready-trace preorder.

1 Introduction

The testing theory for concurrent processes [7, 13] is based on the criterion that two processes are equivalent iff they cannot be distinguished when interacting with their environment, which is an arbitrary process itself. The aim is to obtain the coarsest congruence for parallel composition by the definition of the relation. It turned out that, for a broad class of processes, the testing equivalence [7, 13] and the failures equivalence [1] coincide [20]. Both theories, [1] and [7, 13] gave prominence to the notion of "unobservability" of the exact moment in which a process makes an internal choice.

Originally, only qualitative properties were in the focus of theories for reactive processes. In many applications with unreliable components, however, it is more useful to prove that a system will deadlock less than 0.1% of the time, rather than never. This is one of the reasons why probabilistic choice, as a refinement of internal choice, was introduced in process semantics, in addition to action choice and standard internal choice. To be able to reason about the probabilistic behavior of systems, *schedulers* were adopted, which resolve the nondeterminism and yield fully probabilistic systems.

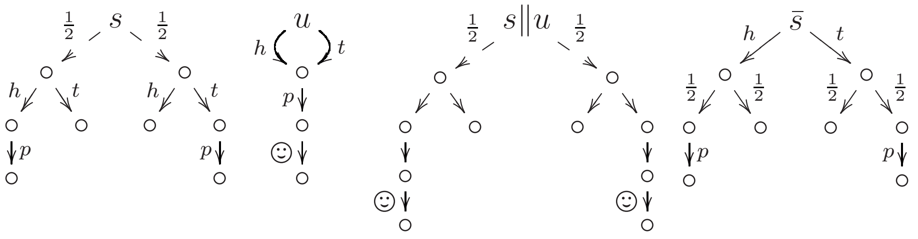


Fig. 1. The coin-flipping machine and the guessing user

Originally, schedulers were monolithic: processes were composed and schedulers were applied to the composed system. With this reasoning, an extension of the may/must testing theory for the case with probabilistic choice was defined [24]. However, it was soon observed that the compose-and-schedule approach could yield unrealistic overestimation of the probabilistic behaviour of the composed system [17, 19, 22]. We explain this phenomenon via the examples that follow.

The gambling machine. Consider a system consisting of a machine and a user, that communicate via a menu of two buttons "head" and "tail" positioned at the machine. The machine first makes a fair choice (i.e. flips a coin) whether to give a prize if "head" is chosen or if "tail" is chosen. Then the user is allowed to choose "head" or "tail" by pressing the appropriate button. If the user chooses the right outcome, a prize is awarded. Note that by no means the user could have detected the machine's choice beforehand. The machine can be modeled by process graph s in Fig. 1. Thus, in half of the machine runs, it offers a prize after the "head" button has been pressed (out of the two-button menu "head" and "tail"), while in the other half of the runs it offers a prize after the "tail" button has been pressed (out of the two-button menu "head" and "tail"). The user can be modeled by process graph u in Fig. 1. Sometimes she would press "head" and sometimes "tail"; however, she is "happy" (denoted by action ☺) iff, after pressing a button, a prize follows. Note that the user makes her choice based on the options the machine offers; in this example the menu consists of two options.

Let the user and the machine interact, by synchronizing on their common actions, except on action ☺. In terms of testing theory [7], process s is tested with test u. By means of probability theory, it can be calculated that the probability that the user has guessed the output of flipping is $\frac{1}{2}$. That is, the probability of a ☺ action being reported is $\frac{1}{2}$. However, most of the existing approaches for probabilistic testing, in particular probabilistic may/must testing [8, 9, 15, 21, 23, 24], do not give this answer. As we mentioned above, in order to compute the probability of ☺ being reported, the probabilistic may/must testing uses schedulers. These schedulers have access to the nodes of the graph of the synchronized process.

Consider the synchronization $s \parallel u$ represented by the process graph in Fig. 1, where actions are hidden after they have synchronized (τ-labels are omitted in the figure). Each scheduler resolves the choice in the nondeterministic nodes of $s \parallel u$ and yields a fully probabilistic process graph. For $s \parallel u$ in Fig. 1 there are four possible schedulers, which yield the following set of probabilities with which s passes the test u: $\{0, \frac{1}{2}, 1\}$. We can see that, because the power of the schedulers is unrestricted, unrealistic upper and lower bounds for the probabilities are obtained. Observe that this happens due to the effect of "cloning" the synchronization nondeterminism. The choice between synchronizing on h or t has been "cloned" in both futures after the probabilistic choice in $s \parallel u$. In this way, when resolving the nondeterminism in $s \parallel u$, a scheduler assumes that the user has unrealistic power to *see* the result of the coin-flipping *before* guessing.

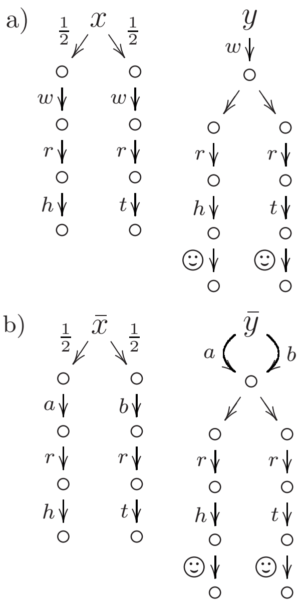


Fig. 2. The coin tosser and result-guesser game: a) fair play; b) unfair play

Consider now process graph $\bar{s}$ in Fig. 1. To the user $\bar{s}$ may as well represent the behaviour of the coin-flipping machine – the user cannot see whether the machine flips the coin *before* or *after* making the "head or tail" offers. According to the user, the machine acts as specified as long as she is able to guess the result in half of the cases. In fact, both schedulers defined by the probabilistic may/must testing, when applied to $\bar{s} \parallel u$ yield exactly probability $\frac{1}{2}$ of reporting a ☺ action. Consequently, none of the approaches in [8, 9, 15, 21, 23, 24] equate processes s and $\bar{s}$: when tested with u, they produce different bounds for the probabilities of reporting ☺[1].

The coin tosser – result guesser game. Consider the following game. Player x tosses a fair coin and hides the outcome. Player y guesses the outcome of the tossing and writes it down. While he is writing the result down, player x waits (i.e. he may write down something meaningless). Then, they both agree to reveal their outcomes, i.e. x to uncover the coin and y to show what he has written. Players x and y are modeled in Fig. 2a. Obviously, the probability that the second player guesses correctly, i.e. reports ☺, equals $\frac{1}{2}$. However, the schedulers applied to the resulting graph for the synchronization of both players give the set of probabilities $\{0, \frac{1}{2}, 1\}$, thus, suggesting that there is a strategy such that player y can always guess the result correctly. On the other hand, if process $x' = w.r.(h \oplus_{\frac{1}{2}} t)$ is synchronized with y, both schedulers applied to the resulting process graph suggest that the probability of reporting a ☺ action is exactly $\frac{1}{2}$. Hence, in probabilistic may/must testing theory processes x and x' cannot

[1] If we ignore the probabilities, processes s and $\bar{s}$ are testing-equivalent by [7].

be equated, and therefore prefix does not distribute over internal probabilistic choice. In other words, the internal (probabilistic) choice is "observable". [2]

It is important to note that, if the players are as in Fig. 2b, i.e. the coin tosser reveals the outcome by offering different actions for synchronization (a or b), then the guesser can surely guess the result: he would make his guess depending on the action on which both players previously synchronized (a or b).

1.1 Contributions

The testing preorder relations defined in [23, 24] were shown to be branching time (simulations) in [9, 15, 18]. Thus, distribution of prefix over internal/probabilistic choice is not allowed. This implies that if we are about to combine external, classical internal and internal probabilistic choice, for verification purposes we can only rely on equivalences that inspect the internal structure of the process graphs. Nevertheless, the most discouraging fact is that in a parallel composition the probability information might be lost, as the previous examples show. This questions the reasons for adding probabilities in the model initially.

The main contribution of this paper is testing semantics in the style of [7] for processes exhibiting external, internal and probabilistic choices that does not yield over-estimated or under-estimated probabilities with which a process passes a test. To achieve this, we propose a novel method for labeling the internal transitions that arise in the parallel composition of the process and the test. The usage of the labels solves the problem with the "cloned nondeterminism" and allows us to compute the probabilities with which a process passes a test. Moreover, unlike the schedulers-based approaches [23, 24], our approach makes the resolution of the nondeterminism state-independent, which, we believe, is more appropriate for testing-based semantics.

The technical contributions of the paper are the following. In our model the internal transitions with the same node of origin have different labels which serve as identifiers for the transitions in a parallel context (Sec. 3). Having this, we first define resolution of the internal nondeterminism in a process (Sec. 4.1). Then, when testing a process with a test, given a resolution of the internal nondeterminism of the process, we define how the synchronization nondeterminism and the internal nondeterminism of the test are resolved (Sec. 4.2). Our solution avoids overestimation and underestimation of the probabilities with which the process passes the test. Based on the testing semantics, we define a preorder relation between processes from which it follows that one process implements another iff the former contains "less" nondeterminism (Sec. 4.2). Finally, we characterize the testing preorder as a probabilistic ready-trace preorder relation (Sec. 5). The latter reveals that, indeed, our testing equivalence is insensitive to the exact moment of occurrence of internal/probabilistic choices.

Due to lack of space and to avoid technical complications while presenting the ideas and the results, in this paper we restrict to finitely-branching, divergence-free, though recursive processes and to finite tests.

[2] Variants of this problem were initially pointed in [17, 19, 22] and treated in [3, 5, 6, 11].

2 Preliminaries

In this paper we use the Bayesian definition of probability because it is more suitable than the measure-theoretic one for our definition of ready-trace preorder.

Bayesian probability For a set A, 2^A denotes its power-set. The following definitions are taken from [16].

We consider a sample space, Ω, consisting of points called *elementary events.* Selection of a particular $a \in \Omega$ is referred to as an "a has occurred". An *event* $A \subseteq \Omega$ is a set of elementary events. $A, B, C, \ldots$ range over events. An event A *has occurred* iff a has occurred for some $a \in A$. Let $A_1, A_2, \ldots$ be a sequence of events and C be an event. The members of the sequence are *exclusive given* C, if whenever C has occurred no two of them can occur together, that is, if $A_i \cap A_j \cap C = \emptyset$ whenever $i \neq j$. C is called a *conditioning* event. If the conditioning event is Ω, then "given Ω" is omitted.

For certain pairs of events A and B, a real number $P(A|B)$ is defined and called the *probability* of A given B. These numbers satisfy the following axioms:

A1: $0 \leq P(A|B) \leq 1$ and $P(A|A) = 1$.
A2: If the events in $\{A_i\}_{i=1}^{\infty}$ are exclusive given B, then $P(\cup_{i=1}^{\infty} A_i \mid B) = \sum_{i=1}^{\infty} P(A_i|B)$.
A3: $P(C|A \cap B) \cdot P(A|B) = P(A \cap C|B)$.

For $P(A|\Omega)$ we simply write $P(A)$.

3 Model

We define our model of process graphs and operations on them. Presuppose a finite set of actions $\mathcal{A}$ and a countable set of labels $\mathcal{L}$ such that $\mathcal{A} \cap \mathcal{L} = \emptyset$. A process graph has three types of transitions (edges), representing action, internal, and probabilistic transitions; a state (node) can perform only one type of transitions. The action transitions are decorated with actions from $\mathcal{A}$, while the internal transitions with labels from $\mathcal{L}$. No two action transitions with the same state of origin are labeled the same and no two internal transitions with the same state of origin are labeled the same. Moreover, if two states have transitions each with the same label, then the sets of all labels of their outgoing (internal) transitions coincide. All states are reachable from a designated initial state via zero or more transitions.

Definition 1 (Process graph). *A* process graph *(or simply process) is a tuple* $\mathcal{P} = (S_{\mathcal{A}}, S_p, S_\tau, \rightarrow, \twoheadrightarrow, \dashrightarrow, r)$, *where*

- $S_{\mathcal{A}}, S_p, S_\tau$ *are finite sets of* action, probabilistic, and nondeterministic states *respectively,*
- $r \in S_{\mathcal{A}} \cup S_p \cup S_\tau$ *is the* initial *state,*
- $\rightarrow \subseteq S_{\mathcal{A}} \times \mathcal{A} \times (S_{\mathcal{A}} \cup S_p \cup S_\tau)$ *is an* action transition relation *such that* $(s, a, t) \in \rightarrow$ *and* $(s, a, t') \in \rightarrow$ *implies* $t = t'$,

- $\twoheadrightarrow \subseteq S_\tau \times \mathcal{L} \times (S_\mathcal{A} \cup S_p \cup S_\tau)$ *is an* internal transition relation *such that*
  - $(s,l,t) \in \twoheadrightarrow$ *and* $(s,l,t') \in \twoheadrightarrow$ *implies* $t = t'$, *and*
  - *if* $(s,l,t) \in \twoheadrightarrow$ *and* $(s',l,t') \in \twoheadrightarrow$ *for some* $l \in \mathcal{L}$ *and states* t,t', *then, for every* $k \in \mathcal{L}$, $(s,k,q) \in \twoheadrightarrow$ *for some* q *iff* $(s',k,q') \in \twoheadrightarrow$ *for some* q',
- $\dashrightarrow \subseteq S_p \times (0,1] \times (S_\mathcal{A} \cup S_p \cup S_\tau)$ *is a* probabilistic transition relation *such that* $(s,\pi,t) \in \dashrightarrow$ *and* $(s,\rho,t) \in \dashrightarrow$ *implies* $\pi = \rho$, *and for all* $s \in S$, $\sum_{(s,\pi,t)\in\dashrightarrow} \pi = 1$, *and*
- *for every state* $s \in S_\mathcal{A} \cup S_p \cup S_\tau$, *there exists a sequence* $\{(s_i,\alpha_i,s_{i+1})\}_{i=1}^n$ *such that* $s_1 = r, s_{n+1} = s$, *and for every* $i \in \{1,\ldots n\}$, (s_i,α_i,s_{i+1}) *belongs to one of the transition relations.*

The set of all process graphs is denoted by $\mathcal{G}$. A process graph is usually named by its initial state. We write $s \xrightarrow{a} t$ rather than $(s,a,t) \in \rightarrow$, $s \overset{l}{\twoheadrightarrow} t$ rather than $(s,l,t) \in \twoheadrightarrow$, and $s \overset{\pi}{\dashrightarrow} t$ rather than $(s,\pi,t) \in \dashrightarrow$ (or $s \dashrightarrow t$ if the value of π is irrelevant in the context). We write $s \xrightarrow{a}$ to denote that there exists an action transition $s \xrightarrow{a} s'$ for some $s' \in S$, where by S we denote $S_\mathcal{A} \cup S_p \cup S_\tau$. Given a process s and an action $a \in \mathcal{A}$, by s_a we denote the process (if exists) for which $s \xrightarrow{a} s_a$.

For a finite index set I, by $s \xrightarrow{a_i} \{s_i\}_{i\in I}$ (resp. $s \overset{\pi_i}{\dashrightarrow} \{s_i\}_{i\in I}$, $s \overset{\tau_i}{\twoheadrightarrow} \{s_i\}_{i\in I}$) we denote that there exist transitions $\{s \xrightarrow{a_i} s_i\}_{i\in I}$ (resp. $\{s \overset{\pi_i}{\dashrightarrow} s_i\}_{i\in I}$, $\{s \overset{\tau_i}{\twoheadrightarrow} s_i\}_{i\in I}$) and s has no other outgoing transitions.

We define the parametric operators $\oplus$, $\sqcap$, $\Box$ [3] on process graphs.

Definition 2. *Let* I *be a finite index set and* $\{s_i\}_{i\in I}$ *be a set of process graphs. For* $\{\pi_i\}_{i\in I} \subset (0,1]$ *such that* $\sum_{i\in I} \pi_i = 1$, $\{\tau_i\}_{i\in I} \subset \mathcal{L}$ *and* $\{a_i\}_{i\in I} \subseteq \mathcal{A}$, *the parametric operators* $\oplus$, $\sqcap$, $\Box$ *on process graphs are defined as follows:* $\oplus_{i\in I}\pi_i s_i$ *(resp.* $\Box_{i\in I}\ a_i s_i$, $\sqcap_{i\in I}\tau_i s_i$ *) is constructed by creating a new state* s, *a set of disjoint copies* $\{s_i'\}_{i\in I}$ *of the process graphs* $\{s_i\}_{i\in I}$, *and transitions* $\{s \overset{\pi_i}{\dashrightarrow} s_i'\}_{i\in I}$ *(resp.* $\{s \xrightarrow{a_i} s_i'\}_{i\in I}$, $\{s \overset{\tau_i}{\twoheadrightarrow} s_i'\}_{i\in I}$ *). The constant* δ *denotes a process that has only one state,* $\delta \in S_\mathcal{A}$, *and no transitions.*

Given a process graph $\mathcal{P} = (S_\mathcal{A}, S_p, S_\tau, \rightarrow, \twoheadrightarrow, \dashrightarrow, r)$, let $I\colon S_\mathcal{A} \mapsto 2^\mathcal{A}$ be a function such that, for all $a \in \mathcal{A}, s \in S_\mathcal{A}$, it holds $a \in I(s)$ iff $s \xrightarrow{a}$. $I(s)$ is called the *menu* of s. Intuitively, for $s \in S_\mathcal{A}$, $I(s)$ is the set of actions that process s can perform initially.

A *finite* process graph is a process graph in which only finite paths exist. A *divergence-free* process graph is a process graph without infinite sequences of probabilistic or internal transitions. To simplify the presentation, throughout the paper we assume that processes are divergence-free.

4 Testing Preorder

In this section we define a testing preorder relation in the style of [7] for the model introduced in Sec. 3. We first define the notion of unfolding a process and

[3] These operators are not to be confused with the CSP operators; however, we choose them so that they have the same intuitive meaning.

the notion of resolving internal nondeterminism in a process. They are needed in Sec. 4.2, where we define the testing preorder.

4.1 Resolving Internal Nondeterminism

The internal nondeterminism in a process is resolved by appropriately assigning probability distributions to the internal choices, i.e. assigning probabilities to the labels of the internal transitions. We first define a function that unfolds a process to a certain depth and relabels the internal transitions. Then we define how the probabilities are assigned.

Let $\mathbb{E}$ denote the set of equations of type $\sum_{i\in I} x_i = 1$ such that I is a finite index set and $\{x_i\}_{i\in I} \subset \mathcal{L}$. For a given set of non-negative numbers I, by $\max I$ we denote the maximum of the numbers in I, if $I \neq \emptyset$, or 0, if $I = \emptyset$. We slightly abuse the notation $\oplus$ and by $\oplus_{i\in I}\pi_i(s_i, \mathcal{C}_i)$ we denote the pair $(\oplus_{i\in I}\pi_i s_i, \cup_{i\in I}\mathcal{C}_i)$, for a given finite index set I, a set of process graphs $\{s_i\}_{i\in I}$, and a set $\{\mathcal{C}_i\}_{i\in I}$ such that $\mathcal{C}_i \subset \mathbb{E}$ for $i \in I$. Similarly for $\Box$ and $\sqcap$.

We next define the unfolding recursive function. The unfolding "depth" equals the maximal length of a sequence of observable actions that the unfolded process could perform.

Definition 3. *The partial function* $\mathrm{U}\colon \mathcal{G} \times 2^{\mathbb{E}} \times \mathbb{N} \mapsto \mathcal{G} \times 2^{\mathbb{E}}$, *called* unfolding, *is defined as* [4]

$$\mathrm{U}(s, \mathcal{C}, m) =$$

$$\begin{cases} \oplus_{i\in I}\, \pi_i \mathrm{U}(s_i, \mathcal{C}, m), & \text{if } m > 0, s \overset{\pi_i}{\dashrightarrow} \{s_i\}_{i\in I} \\ \sqcap_{i\in I} \tau_i^{n+1} \mathrm{U}(s_i, \mathcal{C} \cup \{\sum_{i\in I} \tau_i^{n+1} = 1\}, m), & \text{if } m > 0, s \overset{\tau_i}{\twoheadrightarrow} \{s_i\}_{i\in I}, \\ & n = \max\{k \mid \sum_{i\in I} \tau_i^k = 1 \in \mathcal{C}\} \\ \Box_{i\in I}\, a_i \mathrm{U}(s_i, \mathcal{C}, m-1), & \text{if } m > 0, s \xrightarrow{a_i} \{s_i\}_{i\in I} \\ (\delta, \mathcal{C}) & \text{otherwise} \end{cases}$$

where the labels $\{\tau_i^{n+1}\}$ *are fresh labels.*

By unfolding of a process graph a finite tree is obtained. The labels of the internal transitions in the obtained tree are fresh with respect to the labels in the original process graph. Every time a state with outgoing internal transitions labeled with the set $\{\tau_i\}_{i\in I}$ is to be unfolded, a fresh label set $\{\tau_i^{n+1}\}_{i\in I}$ is used for labeling this particular internal choice in the tree. This label set, in fact, contains implicit information , n, about the number of times a state in the original graph with the same label set $\{\tau_i\}_{i\in I}$ has been unfolded up to that moment. The set of equations obtained by unfolding are used later for assigning probabilities to the labels.

Intuitively, if in the original process graph one internal choice happens in the future of another, although they have the same labels, they are different from

[4] In the definition of the function, the index n in τ_i^n is not to be confused with nth power of τ_i; we would denote the latter with $(\tau_i)^n$.

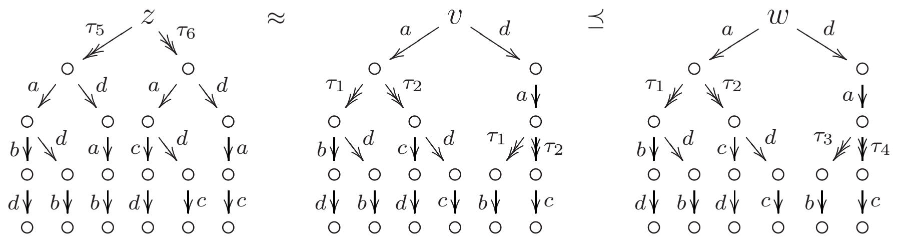


Fig. 3. Relations between several processes; processes z and v are obtained by interleaving $\tau_5(ab) \sqcap \tau_6(ac)$, resp. $a(\tau_1 b \sqcap \tau_2 c)$, with action d

each other. Therefore, they are given different labels in the unfolded tree. On the contrary, if choices with the same set of labels are placed "in parallel", as those in process v in Fig. 3, then they represent the same choice. Namely, since v in Fig. 3 is the parallel composition of process $q = a(\tau_1 b \sqcap \tau_2 c)$ and action d, and process q does not synchronize on action d, the internal choice of q cannot be influenced by action d. Consequently, both internal choices that appear in the graph of process v must be resolved in the same manner.

Given a tree that results from unfolding a process, we define how probabilities are assigned to the labels of the internal transitions.

Definition 4. *Let s be a process graph, $m \geq 0$, $\mathrm{U}(s, \emptyset, m) = (\bar{s}, \mathcal{C})$, and $\mathcal{L}_{/\mathcal{C}}$ be the set of all variables that appear in $\mathcal{C}$. Let $\lambda \colon \mathcal{L}_{/\mathcal{C}} \mapsto [0,1]$ be a function assigning values to the variables in $\mathcal{L}_{/\mathcal{C}}$ respecting the constraints $\mathcal{C}$. We call λ a* resolution *of $\mathcal{C}$. Given λ, let s^m be the process graph obtained when every transition $s_i \overset{\tau_i^k}{\twoheadrightarrow} s_i'$ that belongs to the process graph $\bar{s}$ is replaced by $s_i \overset{\lambda(\tau_i^k)}{\dashrightarrow} s_i'$, if $\lambda(\tau_i^k) \neq 0$, or erased if $\lambda(\tau_i^k) = 0$. We call s^m a m-*resolution *of process s.*

Intuitively, s^m is the process that results when the internal choices in $\bar{s}$ have been replaced by probabilistic choices with labels given by λ.

4.2 Definition of the Testing Preorder

A *test* T, as usual, is a finite process graph given as a tree, such that, for a symbol $\omega \notin \mathcal{A}$, there may exist transitions $s \xrightarrow{\omega}$ for some states s of T, denoting success. Additionally, we assume that no two labels of the internal transitions of the test are the same (we justify this assumption below). By $\mathcal{T}$ we denote the set of all tests. Given a test T, by $\mathsf{length}(T)$ we denote the maximal length of a sequence of observable actions which T can perform before performing ω.

In the previous subsection we defined how the internal nondeterminism of a process is resolved. However, two other types of nondeterminism arise when a process is being tested: first, the nondeterminism with respect to the action on which the process and the test synchronize, if there are more than one candidate-actions for synchronization at the moment (*synchronization* nondeterminism), and, second, the internal nondeterminism of the test.

Regarding the synchronization nondeterminism, note that the resolution must not depend on the internal transitions of the processes; otherwise the problem with overestimated probabilities, as discussed in the example on the gambling machine in Sec. 1, would remain. Second, we must not *underestimate* the power of the entity that resolves the synchronization nondeterminism, which can be even more hazardous [5]. For example, consider again the machine s, the user u, and their synchronization $s \| u$ in Fig. 1, described in Sec. 1. [6] Obviously, in order to keep the probability of $\frac{1}{2}$ with which s passes test u, we need to "remember" that both internal choices in $s \| u$ are resolved in the same way. Therefore, if the process and the test do not have a history of synchronization, the way the synchronization nondeterminism is resolved should depend on the set of candidate-actions for synchronization and on nothing else. On the other hand, as the process and the test proceed interacting, the entity that resolves the synchronization nondeterminism may actually "remember" its history of resolution. Therefore, we also take into account this history in the resolution of the current synchronization nondeterminism – otherwise, we would risk underestimation of the probabilities with which the process passes the test.

Concerning the test, we assume that all labels of internal transitions that it contains are distinct. That is, we restrict to the subset of tests that are most powerful and have "full control" over their nondeterminism. In fact, if two processes are not distinguished by this subset of tests, then they should not be distinguished by the rest of the (less powerful) tests either. Now, to avoid underestimation of the probabilities, we must not exclude the possibility that the test itself is the entity that resolves the synchronization nondeterminism (as in the gambling machine). Therefore, the test can also take into account the history of resolving the synchronization nondeterminism, when making decisions on resolving its internal nondeterminism. For example, consider the test $a(\tau_1 d\omega \sqcap \tau_2 c\omega) \; \Box \; bc\omega$ and the process $\frac{1}{2}(ad \; \Box \; b) \oplus \frac{1}{2}(ac)$. When testing the process, the test can choose transition τ_1 if action a was performed out of the options a and b, and the transition τ_2 if a was the only option on the menu. In this way, the chances to report success increase.

We now formalize the above discussions. By $\mathbb{P}$ we denote the set of all polynomials with variable names in the labels set $\mathcal{L}$. For a given finite index set I, a set of polynomials $\{\phi_i\}_{i \in I} \subset \mathbb{P}$, a set $\{\mathcal{C}_i\}_{i \in I}$ with $\mathcal{C}_i \subset \mathbb{E}$ for $i \in I$, and a set $\{\alpha_i\}_{i \in I}$ of scalars or variables in $\mathcal{L}$, by $\sum_{i \in I} \alpha_i(\phi_i, \mathcal{C}_i)$ we denote the pair $(\sum_{i \in I} \alpha_i \phi_i, \cup_{i \in I} \mathcal{C}_i)$.

Let $\mathcal{G}_{\not\twoheadrightarrow} \subset \mathcal{G}$ be the set of all process graphs that contain no internal transitions. We next define recursively the function for computing the result of testing a process in $\mathcal{G}_{\not\twoheadrightarrow} \subset \mathcal{G}$ with a test.

[5] With overestimated probabilities we might fail to verify a correct protocol [3, 12], but with underestimated probabilities we might claim a protocol to be correct, even if it is not.

[6] Here actually the user itself resolves the synchronization nondeterminism, but in general this is not the case.

Definition 5. *The partial function* $\mathsf{R}\colon \mathcal{G}_{\not\twoheadrightarrow} \times \mathcal{T} \times \mathcal{L} \times 2^{\mathbb{E}} \mapsto \mathbb{P} \times 2^{\mathbb{E}}$ *is defined as*

$$\mathsf{R}(s,T,l,\mathcal{C}) =$$

$$\begin{cases} (1,\mathcal{C}), & \text{if } T \xrightarrow{\omega} \\ \sum_{i\in I} \pi_i \cdot \mathsf{R}(s_i,T,l,\mathcal{C}), & \text{if } s \overset{\pi_i}{\dashrightarrow} \{s_i\}_{i\in I}, T \overset{\omega}{\not\rightarrow} \\ \sum_{i\in I} \pi_i \cdot \mathsf{R}(s,T_i,l,\mathcal{C}), & \text{if } T \overset{\pi_i}{\dashrightarrow} \{T_i\}_{i\in I}, s \not\dashrightarrow \\ \sum_{i\in I} \tau_i^l \cdot \mathsf{R}(s,T_i,l,\mathcal{C} \cup \{\sum_{i\in I} \tau_i^l = 1\}), & \text{if } T \overset{\tau_i}{\twoheadrightarrow} \{T_i\}_{i\in I}, s \not\dashrightarrow \\ \sum_{a\in K} \tau_{(a,K)}^l \mathsf{R}(s_a,T_a,\tau_{(a,K)}^l,\mathcal{C} \cup \{\sum_{a\in K} \tau_{(a,K)}^l{=}1\}), & \text{for } K = I(s) \cap I(T) \neq \emptyset \\ (0,\mathcal{C}) & \text{otherwise.} \end{cases}$$

where the labels $\{\tau_i^l\}$ *and* $\{\tau_{(a,K)}^l\}$ *are fresh labels.*

Definition 6. *Let* ε *be a special label that cannot appear in any process or test,* $s \in \mathcal{G}_{\not\twoheadrightarrow}$ *and* T *be a test. Let* $\mathsf{R}(s,T,\varepsilon,\emptyset) = (\phi,\mathcal{C})$ *and* $\mathcal{L}_{/\phi}$ *be the set of all variables that appear in* ϕ. *Let* $\lambda\colon \mathcal{L}_{/\phi} \mapsto [0,1]$ *be a function assigning values to the variables in* $\mathcal{L}_{/\phi}$ *respecting the constraints* $\mathcal{C}$. *We call* λ *a* resolution *of* $s|T$. *The value of the polynomial* ϕ *for the values of its variables given by* λ *is denoted by* $\Pr(s,T,\lambda)$.

Intuitively, for a process s without internal transitions, $\Pr(s,T,\lambda)$ is the probability with which s passes test T, given that the synchronization nondeterminism and the internal nondeterminism of the test are resolved by λ.

We now define the testing preorder relation. Intuitively, a process s *implements* a process t iff for every test T it holds that, for every resolution of the internal nondeterminism in s, there exists a resolution of the internal nondeterminism in t, such that the options for resolving the synchronization nondeterminism and the internal nondeterminism in the test are the same for both processes when tested with T, and, moreover, for every possible resolution of the synchronization nondeterminism and the internal nondeterminism in the test, the probability with which s passes T is equal to the probability with which t passes T.

Definition 7. *Let* s *and* t *be two processes.* s implements t, *denoted by* $s \preceq_{\mathcal{T}} t$, *iff for every test* T *with* $\mathsf{length}(T) = m$ *and for every* m*-resolution* s^m *of* s, *there exists an* m*-resolution* t^m *of* t, *such that* $\mathsf{R}(s^m,T,\varepsilon,\emptyset) = \mathsf{R}(t^m,T,\varepsilon,\emptyset)$, *and, given an arbitrary resolution* λ *of* $s^m|T$, *it holds that* $\Pr(s^m,T,\lambda) = \Pr(t^m,T,\lambda)$.

In general, process s implements process t iff s contains "less" internal nondeterminism than process t.

Definition 8. *Processes* s *and* t *are* testing-equivalent, *denoted by* $s \approx_{\mathcal{T}} t$, *iff* $s \preceq_{\mathcal{T}} t$ *and* $t \preceq_{\mathcal{T}} s$.

Examples. Consider processes s and $\bar{s}$ and test u in Fig. 1. Neither of them has internal nondeterminism. When test u is applied to s and $\bar{s}$, the set of options for resolving the synchronization nondeterminism is the same for both processes. Moreover, for every resolution of the synchronization nondeterminism, the probability with which s and $\bar{s}$ pass test u is $\frac{1}{2}$. Consider now process x and test y in

Fig. 2. Process x has no internal nondeterminism and there is no synchronization nondeterminism. No matter how y resolves its internal nondeterminism, the probability with which x passes y is $\frac{1}{2}$.

Remark 1. Note that Definitions 3 and 5 can be merged into one definition for the process graph resulting from testing a process with a test. The process would have only probabilistic, internal and ω-transitions and the labels of the internal transitions would be created as the resulting tree is being created, according to Def. 3 and Def. 5. We separated the definitions for clarity and because Def. 3 is also needed in Sec. 5 and it simplifies the proof of Theorem 1.

5 Characterizing the Testing Preorder as a Probabilistic Ready-Trace Preorder

In this section we characterize the testing preorder with a probabilistic ready-trace preorder.

Definition 9 (Ready trace). *A ready trace of length n is a sequence $\mathcal{O} = (M_1, a_1, M_2, a_2, \ldots, M_{n-1}, a_{n-1}, M_n)$ where $M_i \in 2^{\mathcal{A}}$ for all $i \in \{1, 2, \ldots, n\}$ and $a_i \in M_i$ for all $i \in \{1, 2, \ldots, n-1\}$.*

We assume that the observer has the ability to observe the actions that the process performs, together with the menus out of which actions are chosen. Intuitively, a ready trace $\mathcal{O} = (M_1, a_1, M_2, a_2, \ldots, M_{n-1}, a_{n-1}, M_n)$ can be observed if the initial menu is M_1, then action $a_1 \in M_1$ is performed, then the next menu is M_2, then action $a_2 \in M_2$ is performed and so on, until the observing ends at a point when the menu is M_n.

Next, given a process $s \in \mathcal{G}_{\not\twoheadrightarrow}$, i.e. without internal transitions, we define a process $s_{(M,a)}$. Intuitively, $s_{(M,a)}$ is the process that s becomes, assuming that menu M was offered to s and action a was performed. For example, for process s in Fig. 1, $s_{(\{h,t\},h)} = \frac{1}{2}p \oplus \frac{1}{2}\delta$.

Suppose $s \overset{\pi_1}{\dashrightarrow} s_1 \overset{\pi_2}{\dashrightarrow} s_2 \ldots \overset{\pi_n}{\dashrightarrow} s_n$ and $s_n \in S_{\mathcal{A}}$. Denote the product $\pi_1\pi_2 \cdots \pi_n$ by π. We write $s \overset{\pi}{\Rightarrow} s_n$ rather than $s \overset{\pi_1}{\dashrightarrow} s_1 \overset{\pi_2}{\dashrightarrow} s_2 \ldots \overset{\pi_n}{\dashrightarrow} s_n$.

Definition 10. *Let $s \in \mathcal{G}_{\not\twoheadrightarrow}$. Let $M \subseteq \mathcal{A}$, $a \in M$ be such that $I(s) = M$ if $s \in S_{\mathcal{A}}$, or otherwise there exists a transition $s \Rightarrow s'$ such that $I(s') = M$. The process graph $s_{(M,a)}$ is obtained from s in the following way:*

– *if $s \xrightarrow{a} s_a$ then $s_{(M,a)} = s_a$;*
– *if $s \dashrightarrow$ then $s_{(M,a)} \equiv \oplus_{i\in I} \frac{\pi_i}{\pi} s'_i$, where $\{s'_i\}_{i\in I}$ are all process graphs s.t. for $i \in I$ there exists a sequence of transitions $s \overset{\pi_i}{\Rightarrow} s_i \xrightarrow{a} s'_i$ s.t. $I(s_i) = M$, and $\pi = \sum_{s \overset{\pi_i}{\Rightarrow} s_i, I(s_i)=M} \pi_i$.*

Definition 11. *Let $(M_1, a_1, M_2, a_2, \ldots, M_{n-1}, a_{n-1}, M_n)$ be a ready trace of length n and $s \in \mathcal{G}_{\not\twoheadrightarrow}$. Functions $P^1_s(M)$ and $P^n_s(M_n | M_1, a_1, \ldots M_{n-1}, a_{n-1})$ (for $n > 1$) are defined in the following way:*

$$P_s^1(M) = \begin{cases} \sum_{s \overset{\pi}{\dashrightarrow} s'} \pi \cdot P_{s'}^1(M) & \textit{if } s \in S_p, \\ 1 & \textit{if } s \in S_{\mathcal{A}},\ I(s) = M, \\ 0 & \textit{otherwise.} \end{cases}$$

$$P_s^2(M_2|M_1, a_1) = \begin{cases} P^1_{s_{(M_1,a_1)}}(M_2) & \textit{if } P_s^1(M_1) > 0, \\ \textit{undefined} & \textit{otherwise.} \end{cases}$$

$$P_s^n(M_n|M_1, a_1, \ldots, a_{n-1}) = \begin{cases} P^{n-1}_{s_{(M_1,a_1)}}(M_n|M_2, a_2, \ldots, a_{n-1}) & \textit{if } P_s^1(M_1) > 0, \\ \textit{undefined} & \textit{otherwise.} \end{cases}$$

Let the sample space consist of all possible menus and let $s \in \mathcal{G}_{\not\twoheadrightarrow}$. Function $P_s^1(M)$ can be interpreted as the probability that menu M is observed when process s starts executing. Let the sample space consist of all ready traces of length n. Function $P_s^n(M_n|M_1, a_1, \ldots M_{n-1}, a_{n-1})$ can be interpreted as the probability of the event $\{(M_1, a_1, \ldots, M_{n-1}, a_{n-1}, M_n)\}$, given the event $\{(M_1, a_1, \ldots M_{n-1}, a_{n-1}, X) : X \in 2^{\mathcal{A}}\}$, when observing ready traces of process s. These probabilities are well defined, i.e. they satisfy axioms A1-A3 of Sec. 2.

Definition 12. *Let s and t be two processes. We say s* implements *t* w.r.t. ready traces *(notation $s \preceq_{\mathcal{O}} t$) iff for every $n \geq 0$ and every n-resolution $\bar{s}$ of s, there exists a n-resolution $\bar{t}$ of t such that for all $k \leq n$ and for all ready traces $(M_1, a_1, \ldots M_k)$,*

- *$P_{\bar{s}}^1(M_1) = P_{\bar{t}}^1(M_1)$ and*
- *if $k > 1$, then $P_{\bar{s}}^k(M_k|M_1, a_1, \ldots M_{k-1}, a_{k-1})$ is defined if and only if $P_{\bar{t}}^k(M_k|M_1, a_1, \ldots M_{k-1}, a_{k-1})$ is defined, and, in case they are both defined, they are equal.*

Informally, a process s implements a process t iff, for every n-resolution $\bar{s}$ of the nondeterminism in s, there is an n-resolution $\bar{t}$ of the nondeterminism in t such that for every ready trace $(M_1, a_1, M_2, a_2, \ldots M_k)$ of length $k \leq n$, the probability to observe M_k, under the condition that the sequence $(M_1, a_1, M_2, a_2, \ldots M_{k-1}, a_{k-1})$ was previously observed, is defined at the same time for both $\bar{s}$ and $\bar{t}$, and, moreover, in case both probabilities are defined, they coincide.

Definition 13. *Let s and t be two processes. s and t are ready-trace-equivalent, denoted by $s \approx_{\mathcal{O}} t$, iff $s \preceq_{\mathcal{O}} t$ and $t \preceq_{\mathcal{O}} s$.*

Examples. Processes s and $\bar{s}$ in Fig.1 are ready-trace equivalent. Processes z and v in Fig. 3 are ready-trace equivalent and they both implement process w in the same figure. Processes z and v can be actually seen as an interleaving of processes $\tau_5 ab \sqcap \tau_6 ac$, resp. $a(\tau_1 b \sqcap \tau_2 c)$, none of which can recognize action d, with action d, while process w has "full control" over its nondeterminism. Fig. 4 also represents relations between several processes.

Theorem 1. *Let s and t be two processes. $s \preceq_{\mathcal{O}} t$ if and only if $s \preceq_{\mathcal{T}} t$.*

Proof. See [10].

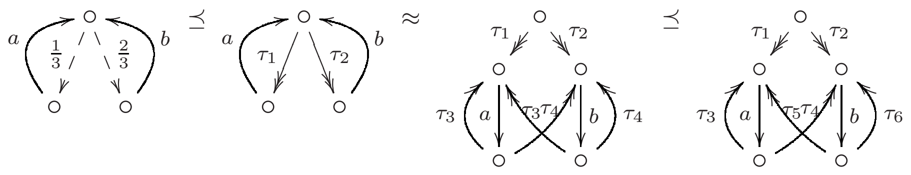


Fig. 4. Relations between several recursive processes

6 Related Work

Testing preorders in the style of [7] for systems exhibiting external, internal and internal probabilistic choices were first introduced in [24], where centralized (monolithic) schedulers, as discussed in Sec. 1, were used. These preorders were later extensively studied and characterized as simulations [8, 9, 15, 21, 23]. Motivated to allow distribution of prefix over internal (probabilistic) choice, as in [7, 13, 14], proposals for testing semantics were given in [19] and [2]. In these approaches all probabilistic choices in a process are resolved at the beginning of the execution. Thus, internal choices are "pushed" downwards and replicated, having as a side-effect loss of the probability information and loss of idempotence of internal choice.

The phenomenon of overestimated probabilities that results from system analysis using centralized schedulers was first observed in [17] and also pointed at in [19, 22]; however, it started being treated relatively recently.

The first paper to introduce distributed schedulers for concurrent systems to overcome the problem with the super-powerful centralized schedulers, discussed in Sec. 1, is [6], for a setting and parallel composition that is rather different than ours. The motivation is to allow compositionality for the trace distributions inclusion (see e.g. [22]). Reference [5] also treats the compositionality problem for trace distributions inclusion for reactive-generative probabilistic systems, by exploiting distributed schedulers.

The first paper to introduce the finer reasoning of testing semantics [7] under restricted schedulers is [3]. It introduces an *explicit* scheduler, which communicates with the process via labels: two nodes in the process are indistinguishable to a scheduler if they have the same labels. For example, the probabilities with which process s (Fig. 1) passes test u, and whether processes s and $\bar{s}$ can be equated, depend on the labeling system.

Unlike the previous approaches for solving the problem with overestimated probabilities in concurrent processes, the "schedulers" that we define do not use information about the states of the processes. We believe that this is essential if the equivalence relation is based on the notion of observability or testing. On the other hand, taking into consideration all possible resolutions of internal nondeterminism seems to be unavoidable when defining preorder relations between probabilistic-nondeterministic systems. For example, even in statistical *black-box* testing (see [4]), one needs to consider all possible schedulers in order to come up with a reasonable notion of trace distribution inclusion.

7 Conclusion

We defined a testing preorder relation in the style of [7] for processes exhibiting internal, external, and probabilistic internal nondeterminism. The goal of the testing semantics was to preserve the probability information in a parallel context and to produce realistic estimates of the probabilities with which a process passes a given test, based on the information that both entities exchange. To our knowledge, this is the first testing semantics of its kind that tackles this problem. We also showed that the testing preorder relation can be characterized with a probabilistic ready-trace preorder, which is easier to grasp from an observer's point of view. From this characterization it easily follows that the equivalence relation is insensitive to the exact moment of occurrence of internal and probabilistic choices.

In the future we plan to extend our work to divergent and infinitely-branching processes, to consider recursive tests and to study the preorder relation defined here in a process algebraic setting. It would be also interesting to apply the testing preorder for verification purposes.

Acknoledgements. We are grateful to the anonymous reviewers for FoSSaCS'10, whose detailed comments improved the final version of this paper.

References

1. Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A theory of communicating sequential processes. Journal of ACM 31(3), 560–599 (1984)
2. Cazorla, D., Cuartero, F., Valero, V., Pelayo, F.L., Pardo, J.J.: Algebraic theory of probabilistic and nondeterministic processes. Journal of Logic and Algebraic Programming 55(1-2), 57–103 (2003)
3. 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)
4. Cheug, L., Stoelinga, M.I.A., Vaandrager, F.W.: A testing scenario for probabilistic processes. Journal of ACM 54(6), 29:1–29:45 (2007)
5. Cheung, L., Lynch, N., Segala, R., Vaandrager, F.: Switched PIOA: Parallel composition via distributed scheduling. Theoret. Comp. Science 365(1-2), 83–108 (2006)
6. de Alfaro, L., Henzinger, T., Jhala, R.: Compositional methods for probabilistic systems. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 351–365. Springer, Heidelberg (2001)
7. De Nicola, R., Hennessy, M.C.B.: Testing equivalences for processes. Theoret. Comp. Science 34, 83–133 (1984)
8. Deng, Y., van Glabbeek, R., Hennessy, M., Morgan, C.: Testing finitary probabilistic processes (extended abstract). In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009 - Concurrency Theory. LNCS, vol. 5710, pp. 274–288. Springer, Heidelberg (2009)
9. Deng, Y., van Glabbeek, R.J., Hennessy, M., Morgan, C.: Characterising testing preorders for finite probabilistic processes. Logical Methods in Computer Science 4(4:4), 1–33 (2008)

10. Georgievska, S., Andova, S.: Retaining the probabilities in probabilistic testing theory. Technical Report (to appear, 2010), http://www.win.tue.nl/~sgeorgie/general_testing.pdf
11. Giro, S., D'Argenio, P.: On the expressive power of schedulers in distributed probabilistic systems. In: Proc. QAPL 2009. ENTCS (2009) (to appear)
12. Giro, S., D'Argenio, P., Ferrer Fioriti, L.M.: Partial order reduction for probabilistic systems: A revision for distributed schedulers. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009 - Concurrency Theory. LNCS, vol. 5710, pp. 338–353. Springer, Heidelberg (2009)
13. Hennessy, M.: Algebraic Theory of Processes. MIT Press, Cambridge (1988)
14. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall, Englewood Cliffs (1985)
15. Jonsson, B., Wang, Y.: Testing preorders for probabilistic processes can be characterized by simulations. Theoret. Comp. Science 282(1), 33–51 (2002)
16. Lindley, D.V.: Introduction to Probability and Statistics from a Bayesian Viewpoint. Cambridge University Press, Cambridge (1980)
17. Lowe, G.: Representing nondeterministic and probabilistic behaviour in reactive processes. Technical Report PRG-TR-11-93, Oxford Univ. Comp. Labs (1993)
18. Lynch, N., Segala, R., Vaandrager, F.: Observing branching structure through probabilistic contexts. SIAM Journal on Computing 37(4), 977–1013 (2007)
19. Morgan, C., McIver, A., Seidel, K., Sanders, J.W.: Refinement-oriented probability for CSP. Formal Aspects of Computing 8(6), 617–647 (1996)
20. De Nicola, R.: Extensional equivalences for transition systems. Acta Informatica 24(2), 211–237 (1987)
21. Palmeri, M.C., De Nicola, R., Massink, M.: Basic observables for probabilistic testing. In: Proc. QEST 2007, pp. 189–200. IEEE Computer Society, Los Alamitos (2007)
22. Segala, R.: Modeling and verification of randomized distributed real-time systems. PhD thesis. MIT (1995)
23. Segala, R.: Testing probabilistic automata. In: Sassone, V., Montanari, U. (eds.) CONCUR 1996. LNCS, vol. 1119, pp. 299–314. Springer, Heidelberg (1996)
24. Wang, Y., Larsen, K.G.: Testing probabilistic and nondeterministic processes. In: Proceedings of the IFIP TC6/WG6.1 Twelth International Symposium on Protocol Specification, Testing and Verification XII, pp. 47–61 (1992)

Forward Analysis of Depth-Bounded Processes

Thomas Wies, Damien Zufferey, and Thomas A. Henzinger

IST Austria (Institute of Science and Technology Austria)

Abstract. Depth-bounded processes form the most expressive known fragment of the π-calculus for which interesting verification problems are still decidable. In this paper we develop an adequate domain of limits for the well-structured transition systems that are induced by depth-bounded processes. An immediate consequence of our result is that there exists a forward algorithm that decides the covering problem for this class. Unlike backward algorithms, the forward algorithm terminates even if the depth of the process is not known a priori. More importantly, our result suggests a whole spectrum of forward algorithms that enable the effective verification of a large class of mobile systems.

1 Introduction

We are interested in the verification of π-calculus processes [21, 22], i.e., message passing systems that admit unbounded creation of processes and name mobility. We can think of a configuration of such a system as a graph [14, 20]. The vertices of the graph are the processes labelled by their current local state. Edges between processes indicate whether the respective processes share a channel, i.e., whether they are able to communicate with each other.

The most expressive known fragment of the π-calculus for which interesting verification problems are still decidable is the class of depth-bounded processes [18]. Intuitively, in a depth-bounded process there is a bound on the length of all simple paths in all reachable configuration graphs (the graphs may contain cycles). A typical example of a depth-bounded process is a server-client architecture where a server answers requests of clients and where each client only knows the name of the server but not the names of other clients. Both the number of simultaneously active clients as well as the number of pending requests for the server can be unbounded.

In this paper we are concerned with the covering problem for depth-bounded processes. Intuitively, the covering problem asks whether a system can reach a configuration that contains some process that is in a local error state. A decision procedure for the covering problem therefore enables the automated verification of an interesting class of safety properties. Meyer showed in [18] that depth-bounded processes admit well-structured transition systems (WSTS) [1, 9, 12]. This implies that the covering problem for depth-bounded processes of known depth can be decided using a standard backward algorithm for WSTSs. The question whether the covering problem is decidable for the entire class of depth-bounded processes was open.

We present the first forward algorithm for this problem. Unlike backward algorithms, our algorithm terminates even if the bound of the system is not known a priori. We thus show that the covering problem is decidable for the entire class. Our algorithm is an instance of the expand, enlarge, and check algorithm schema for WSTSs that exhibit a so-called *adequate domain of limits* (ADL) [10, 13]. An adequate domain of limits for the well-quasi-ordering of a WSTS provides an effective representation of all downward-closed sets of configurations, i.e., ADLs are the key for ensuring termination of forward analyses of WSTSs. Our main technical contribution is the development of an adequate domain of limits for depth-bounded processes. For this purpose we show that downward-closed sets of configurations in depth-bounded processes are characterized by finite unions of regular languages of unranked trees.

Besides our theoretical interest in forward analysis of π-calculus processes there are also practical considerations that make forward algorithms more appealing than their backward counterparts. A backward analysis needs to consider all possible unifications between names that may enable processes to synchronize. A forward analysis instead knows which names are equal and which are not. In practice, the search space of a forward analysis is therefore often significantly smaller than the search space of a backward analysis. We give an example that demonstrates this phenomenon in Section 3. While the forward algorithm that we consider in this paper is mainly of theoretical interest, our adequate domain of limits suggests a whole spectrum of forward algorithms that enable the effective verification of a large class of mobile systems. This spectrum ranges from acceleration-based algorithms in the style of Karp-Miller [8, 11, 15] to approximation algorithms based on abstract interpretation [6].

Further related work. Depth-bounded processes are semantically defined in terms of reachable configurations. While checking depth-boundedness is in general undecidable, many fragments of the π-calculus that are defined syntactically [2, 7] or in terms of type systems [4, 25, 26] are subsumed by depth-bounded processes. Our result carries over to these fragments. Further related work can be found in the context of graph rewriting systems. Bauer and Wilhelm [3] developed an overapproximating shape analysis for graph rewriting systems whose reachable configurations have a star-like shape. Such systems are bounded in the length of the acyclic paths. Our result naturally generalizes to such systems and promises complete algorithms for their verification.

2 Preliminaries

We first fix the syntax and semantics of our version of the π-calculus and briefly recall depth-bounded processes, well-quasi-orderings, better-quasi-orderings, and well-structured transition systems.

2.1 The π-Calculus and Depth-Bounded Processes

We consider systems of recursive equations in the polyadic π-calculus that have a specific normal form inspired by Amadio and Meyssonnier [2].

Assume a countable infinite set of names with typical elements x, y and a countable infinite set of process identifiers with typical elements A, B. We assume that each name and identifier has an associated *arity* in $\mathbb{N}$. We denote by $\boldsymbol{x}$ a (possibly empty) vector over names and denote by $[\boldsymbol{x}/\boldsymbol{y}]$ a substitution on names.

Process terms P are recursively composed of the unit process 0, parameterized process identifiers $A(\boldsymbol{x})$, and the standard operations of parallel composition $P_1 \mid P_2$, external choice $\pi_1.P_1 + \pi_2.P_2$, and name restriction $(\nu x)P$. Hereby, a prefix π is either an *input prefix* of the form $x(\boldsymbol{y})$ or an *output prefix* of the form $\overline{x}(\boldsymbol{y})$. All parameter vectors occuring in process terms must respect the arities of names and identifiers. We call the terms of the form $A(\boldsymbol{x})$ *threads*. We write Π in order to denote indexed parallel composition and Σ for indexed external choice. We further write $(\nu \boldsymbol{x})$ for $(\nu x_1) \dots (\nu x_n)$ where $\boldsymbol{x} = x_1, \dots, x_n$. An occurrence of a name x in a process term P is called *free* if it is not below a (νx) or an input prefix $y(x)$. We denote by $\mathsf{fn}(P)$ the set of all free occurring names in P. We say that P is *closed* if $\mathsf{fn}(P) = \emptyset$. We denote by $P \equiv Q$ the usual structural congruence relation on process terms, i.e., P is syntactically equal to Q up to renaming and reordering of restricted names, scope extrusion, elimination of units, and associativity and commutativity of parallel composition and external choice.

A *configuration* is a closed process term of the following form

$$(\nu \boldsymbol{x})(\prod_{i \in I} A_i(\boldsymbol{x}_i))$$

A *process* $\mathcal{P}$ is a pair $(I, \mathcal{E})$ where I is an *initial* configuration and $\mathcal{E}$ is a finite set of parametric equations $A(\boldsymbol{x}) = P$ such that (1) every process identifier in P and I is defined by exactly one equation in $\mathcal{E}$ and (2) $\mathsf{fn}(P) \subseteq \{\boldsymbol{x}\}$. We assume that all equations in $\mathcal{E}$ have the following normal form:

$$A(\boldsymbol{x}) = \sum_{i \in I} \pi_i.(\nu \boldsymbol{x}_i)(\prod_{j \in J_i} A_j(\boldsymbol{x}_j))$$

Operational semantics. Given a process $\mathcal{P} = (I, \mathcal{E})$, we define a transition relation $\rightarrow_{\mathcal{E}}$ on configurations that captures the usual π-calculus reduction rules as follows. Let P and Q be configurations then we have $P \rightarrow_{\mathcal{E}} Q$ if and only if the following conditions hold:

1. $P \equiv (\nu \boldsymbol{u})(A(\boldsymbol{v}) \mid B(\boldsymbol{w}) \mid P')$,
2. the defining equation of A in $\mathcal{E}$ is of the form $A(\boldsymbol{x}) = x(\boldsymbol{x}').(\nu \boldsymbol{x}'')(M) + M'$,
3. the defining equation of B in $\mathcal{E}$ is of the form $B(\boldsymbol{y}) = \overline{y}(\boldsymbol{y}').(\nu \boldsymbol{y}'')(N) + N'$,
4. $\sigma = [\boldsymbol{v}/\boldsymbol{x}, \boldsymbol{w}/\boldsymbol{x'}, \boldsymbol{z}_A/\boldsymbol{x''}, \boldsymbol{w}/\boldsymbol{y}, \boldsymbol{z}_B/\boldsymbol{y''}]$ where $\boldsymbol{z} = \boldsymbol{z}_A, \boldsymbol{z}_B$ are fresh names,
5. $\sigma(x) = \sigma(y)$,
6. $Q \equiv (\nu \boldsymbol{u}, \boldsymbol{z})(\sigma(M) \mid \sigma(N) \mid P')$.

We denote by $\rightarrow_{\mathcal{E}}^*$ the reflexive transitive closure of the relation $\rightarrow_{\mathcal{E}}$. We say that a configuration P is *reachable* in process $\mathcal{P}$ if and only if $I \rightarrow_{\mathcal{E}}^* P$. Finally, we denote by $\mathit{Reach}(\mathcal{P})$ the set of all reachable configurations of process $\mathcal{P}$.

Depth-Bounded Processes. We now recall the definition of the class of depth-bounded processes [18]. The *nesting of restrictions* nest_ν of a process term is measured recursively as follows $\mathit{nest}_\nu(0) = \mathit{nest}_\nu(A(\boldsymbol{x})) = \mathit{nest}_\nu(P_1 + P_2) = 0$, $\mathit{nest}_\nu((\nu x)P) = 1 + \mathit{nest}_\nu(P)$, and $\mathit{nest}_\nu(P_1 \mid P_2) = \max\{\mathit{nest}_\nu(P_1), \mathit{nest}_\nu(P_2)\}$. The *depth* of a process term P is the minimal nesting of restrictions of process terms in the congruence class of P:

$$\mathit{depth}(P) = \min\{\, \mathit{nest}_\nu(Q) \mid Q \equiv P \,\}\,.$$

Definition 1 (Depth-Boundedness). *A set of configurations $\mathcal{C}$ is called* depth-bounded *if there is $k_D \in \mathbb{N}$ such that $depth(P) \leq k_D$ for all $P \in \mathcal{C}$. A process $\mathcal{P}$ is called* depth-bounded *if its set of reachable configurations $Reach(\mathcal{P})$ is depth-bounded.*

Example 2. The following equations describe a simple client-server system where a server can spawn clients and answer their requests. The server is given by process identifier $\mathit{Server}(x, y)$. The channel x is used for communication with clients. The channel y is used to trigger creation of new clients.

$$\begin{aligned}
\mathit{Server}(x, y) &= (x(z).\mathit{Answer}(z) \mid \mathit{Server}(x, y)) \\
&\quad + (y().(\nu u)(\mathit{Client}(u, x) \mid \mathit{Answer}(u) \mid \mathit{New}(y) \mid \mathit{Server}(x, y))) \\
\mathit{Client}(u, x) &= u().(\mathit{Client}(u, x) \mid \mathit{Request}(x, u)) \\
\mathit{Answer}(u) &= \overline{u}().0 \qquad \mathit{New}(y) = \overline{y}().0 \qquad \mathit{Request}(x, u) = \overline{x}(u).0
\end{aligned}$$

If the initial configuration is given by $(\nu x, y)(\mathit{New}(y) \mid \mathit{Server}(x, y))$ then the depth of all reachable configurations is bounded by 2.

2.2 WQOs, BQOs, and WSTSs

We briefly recall the relevant theory of well-quasi-orderings, better-quasi-orderings [23], and well-structured transition systems [1, 9, 12].

Well-quasi-ordering. A pair $(X, \leq)$ of a set X and a binary relation $\leq$ on X is called *well-quasi-ordered set* (wqo) if and only if (1) $\leq$ is a quasi-ordering (i.e., reflexive and transitive) and (2) any infinite sequence $x_0, x_1, x_2, \ldots$ of elements from X contains an increasing pair $x_i \leq x_j$ with $i < j$. A nonempty set $Y \subseteq X$ is called *directed* if for any $x, y \in Y$ there exists $z \in Y$ with $x, y \leq z$. A set $Y \subseteq X$ is called *upward-closed* if for any pair x, y such that $x \in Y$ and $y \geq x$ implies $y \in Y$. Similarly, Y is called *downward-closed* if for any pair x, y such that $x \in Y$ and $y \leq x$ implies $y \in Y$. The upward-closure of $Y \subseteq X$ is defined as $\uparrow Y = \{x \mid \exists y \in Y. x \geq y\}$. Correspondingly, we denote by $\downarrow Y$ the downward-closure of Y. We extend the ordering $\leq$ to an ordering $\leq$ on subsets of X as expected: for $Y_1, Y_2 \subseteq X$, we have $Y_1 \leq Y_2$ iff for all $y_1 \in Y_1$ there exists $y_2 \in Y_2$ if $y_1 \leq y_2$. For $Y \subseteq X$ we call $Y' \subseteq X$ *large* in Y iff $Y \leq Y'$. Conversely, we call Y' *small* in Y if $Y' \leq Y$. A subset $Y \subseteq X$ of X is called *irreducible* if for any $Y_1, Y_2 \subseteq X$, $Y \leq Y_1 \cup Y_2$ implies $Y \leq Y_1$ or $Y \leq Y_2$.

Better-quasi-orderings. Let $\leq$ be a quasi-ordering on a set X then define the quasi-ordering $\leq_1$ on subsets of X as follows: for $Y_1, Y_2 \subseteq X$, we have $Y_1 \leq_1 Y_2$ iff there exists an injection $\phi : Y_1 \rightarrow Y_2$ such that for all $y_1 \in Y_1$, $y_1 \leq \phi(y_1)$. We are interested in wqo sets $(X, \leq)$ whose powerset is again a wqo with respect to $\leq_1$. For this purpose we consider Nash-William's better-quasi-orderings [23]. Better-quasi-ordered (bqo) sets are particular well-behaved wqo sets. Unlike general well-quasi-orderings, bqo sets are closed under the powerset construction. The formal definition of better-quasi-orderings is rather technical and not required for understanding this paper. We therefore refer to [23] for the actual definition. We only state the properties of bqo sets that we will use in our proofs.

Proposition 3. *Let $(X, \leq)$ be a bqo then*

1. *$(X, \leq)$ is a wqo,*
2. *$(2^X, \leq_1)$ is a bqo,*
3. *every $Y \subseteq X$ is a bqo with respect to the restriction of $\leq$ to Y.*

Properties 1 and 2 are proved in [23]. Property 3 immediately follows from the definition of bqo sets.

Well-structured transition system. A *well-structured transition systems* (WSTS) is a transition system $T = (S, s_0, \rightarrow, \leq)$ where S is a set of configurations, $s_0 \in S$ an initial configuration, $\rightarrow \subseteq S \times S$ a transition relation, and $\leq \subseteq S \times S$ a relation satisfying the following two conditions: (well-quasi-ordering) $\leq$ is a well-quasi-ordering on S; and (compatibility) $\leq$ is upward compatible with respect to $\rightarrow$, i.e., for all s_1, s_2, t_1 such that $s_1 \leq t_1$ and $s_1 \rightarrow s_2$, there exists t_2 such that $t_1 \rightarrow^* t_2$ and $s_2 \leq t_2$.

Definition 4 (Covering Problem). *Given a WSTS $(S, s_0, \rightarrow, \leq)$ and a configuration $t \in S$, the* covering problem *asks whether there exists a configuration $t' \in S$ such that $s_0 \rightarrow^* t'$ and $t \leq t'$.*

3 The Covering Problem for Depth-Bounded Processes

We define the following natural quasi-ordering $\leq$ on configurations of processes: let $P \equiv (\nu \boldsymbol{x})P'$ and Q be configurations then $P \leq Q$ if and only if $Q \equiv (\nu \boldsymbol{x})(P' \mid R)$ for some process term R. Meyer [18] proved that depth-bounded sets of configurations are well-quasi-ordered by $\leq$. Thus, a depth-bounded process $\mathcal{P} = (I, \mathcal{E})$ induces a well-structured transition systems $(\mathit{Reach}(\mathcal{P}), I, \rightarrow_E, \leq)$. We are interested in the covering problem for these WSTSs.

Forward vs. backward algorithms. The standard algorithm for deciding the covering problem for a WSTS is a backward algorithm that works as follows. Starting from the configuration t that is to be covered one computes the set of backward-reachable configurations of the upward closure of t and then checks whether this set contains the initial configuration. The well-quasi-ordering ensures that the backward analysis terminates.

In the WSTS $(\mathit{Reach}(\mathcal{P}), I, \rightarrow_E, \leq)$ that is induced by a depth-bounded process $\mathcal{P}$ we implicitly restrict the transition relation $\rightarrow_E$ to the forward-reachable configurations $\mathit{Reach}(\mathcal{P})$. The predecessor configurations with respect to this restricted transition relation are not effectively computable, i.e., the backward algorithm is not applicable to this WSTS. On the other hand, predecessor configurations for the unrestricted transition relation are effectively computable, but the induced set of backward-reachable configurations is in general not depth-bounded (and thus not well-quasi-ordered by $\leq$). A backward algorithm can only be effectively applied to the WSTS $(\mathcal{C}(k), I, \rightarrow_E, \leq)$. Here $\mathcal{C}(k)$ is the set of all configurations of depth k and k is the maximal depth of configurations in $\mathit{Reach}(\mathcal{P})$, i.e., $\mathit{Reach}(\mathcal{P}) \subseteq \mathcal{C}(k)$. Since k must be known in advance, Meyer's result only implies that the covering problem is decidable for depth-bounded processes of known depth. We will show that there exists a forward algorithm that overcomes this limitation.

Besides the theoretical deficiency of backward algorithms there is also a practical reason why forward algorithms are more attractive. We explain this with an example.

Example 5. Consider the parameterized process $\mathcal{P}(n)$ for $n \in \mathbb{N}$ that is given by the initial configuration $I(n)$:

$$(\nu x, z, y, y_1, \dots, y_n)(\mathit{Buffer}_n(x, z, y_1, \dots, y_n) \mid \mathit{Env}(z, x, y))$$

and the equations $\mathcal{E}(n)$:

$$\begin{aligned}
\mathit{Buffer}_0(x, z) &= x(y).\mathit{Buffer}_1(x, z, y) \\
\mathit{Buffer}_i(x, z, y_1, \dots, y_i) &= x(y).\mathit{Buffer}_{i+1}(x, z, y_1, \dots, y_i, y) \\
&\quad + \overline{z}(y_1).\mathit{Buffer}_{i-1}(x, z, y_2, \dots, y_i) \qquad \text{for } 0 < i < n \\
\mathit{Buffer}_n(x, z, y_1, \dots, y_n) &= \overline{z}(y_1).\mathit{Buffer}_{n-1}(x, z, y_2, \dots, y_n) \\
\mathit{Env}(z, x, y) &= \overline{x}(y).(\nu u)(\mathit{Env}(z, x, u)) \; + \; z(u).\mathit{Env}(z, x, u)
\end{aligned}$$

The process $\mathcal{P}(n)$ models a finite FIFO buffer that stores data sent by the environment in a queue of maximal length n. The queue is modeled using the parameter lists of the process identifiers Buffer_i.

Suppose we want to check that the configuration $P \equiv (\nu x, z)(\mathit{Buffer}_0(x, z))$ is coverable in $\mathcal{P}(n)$. The number of representatives for the set of configurations that are backward-reachable from the upward-closure of P grows exponential in n. The reason is that in one of the continuations of the choices that define $\mathit{Buffer}_i(x, z, y_1, \dots, y_i)$ the parameter y_1 does not occur. A backward algorithm that computes the predecessors for the execution of this choice has no knowledge about the name that the parameter y_1 denotes. It has to guess whether it is a fresh name or whether it is equal to one of the other names appearing in the continuation. On the other hand, a forward algorithm always knows which name the parameter y_1 denotes. Therefore, the set of representatives for the configurations that are forward reachable from $I(n)$ grows only linear in n. It is this phenomenon that makes forward algorithms more appealing for the analysis of π-calculus processes.

4 An Adequate Domain of Limits

Most forward algorithms for solving the covering problem of WSTSs compute the *cover*, i.e., the downward-closure of the forward-reachable configurations and then check whether this set contains the configuration to be covered. In order to effectively compute the cover, one needs to find a *completion* of the wqo set that contains all the limits of downward-closed sets. The canonical example is the completion for the well-quasi-ordering on markings of Petri nets. It is given by vectors over the set $\mathbb{N}_\omega$ of natural numbers extended with the limit ordinal ω. This completion is the basis for the Karp-Miller algorithm [15] that computes the covering tree of a Petri net. The notion of an *adequate domain of limits* [10, 13] formalizes the completions of wqo sets.

An *adequate domain of limits* (ADL) [13] for a well-quasi-ordered set $(X, \leq)$ is a tuple $(Y, \sqsubseteq, \gamma)$ where Y is a set disjoint from X; (L1) the map $\gamma : Y \cup X \to 2^X$ is such that $\gamma(z)$ is downward-closed for all $z \in X \cup Y$, and $\gamma(x) = \downarrow \{x\}$ for all $x \in X$; (L2) there is a limit point $\top \in Y$ such that $\gamma(\top) = X$; (L3) $z \sqsubseteq z'$ if and only if $\gamma(z) \subseteq \gamma(z')$; and (L4) for any downward-closed set D of X, there is a finite subset $E \subseteq Y \cup X$ such that $\gamma(E) = D$, where γ is extended to sets as expected: $\gamma(E) = \bigcup_{z \in E} \gamma(z)$. A *weak adequate domain of limits* (WADL) [10] for $(X, \leq)$ is a tuple $(Y, \sqsubseteq, \gamma)$ satisfying (L1),(L3), and (L4). Note that any weak adequate domain of limits can be extended to an adequate domain of limits.

4.1 Limit Configurations

We now describe a weak adequate domain of limits for depth-bounded configurations. In order to finitely represent the limits of infinite downward-closed sets we need to be able to express that certain subterms in a configuration can be replicated arbitrarily often. A natural solution to this problem is to extend configurations with the replication operator ! that is used as a recursion primitive in alternative definitions of the π-calculus [21, 22]. Instead of using replication to express recursion, we use it to effectively represent infinite sets of configurations.

A *limit configuration* E is constructed recursively from process identifiers $A(\boldsymbol{x})$, parallel composition $E_1 \mid E_2$, name restriction $(\nu x)E$ and replication $!E$. We extend the congruence relation $\equiv$ from configurations to limit configurations by adding the axiom $!E \equiv (E \mid !E)$. We carry over the definitions of the transition relations of processes and the quasi-ordering $\leq$ from configurations to limit configurations by replacing the congruence relation in the definitions with the extended congruence relation. We then define the denotation $\gamma(E)$ of a limit configuration E as its downward-closure restricted to non-limit configurations:

$$\gamma(E) = \{ P \mid P \text{ configuration and } P \leq E \}$$

The quasi-ordering $\sqsubseteq$ on limit configurations that is required for the adequate domain of limits is defined by condition (L3).

Example 6. Consider again the client-server process presented in Example 2. The following limit configuration denotes the cover of this process:

$$\begin{aligned}(\nu x)((\nu y)(&\mathit{New}(y) \mid \mathit{Server}(x, y))\\ &\mid !(\nu z)(\mathit{Client}(z, x) \mid \mathit{Answer}(z))\\ &\mid !(\nu z)(\mathit{Client}(z, x) \mid \mathit{Request}(x, z)))\end{aligned}$$

We now state the main technical result of this paper. Given a finite set of process identifiers PI, we denote by $\mathcal{C}(PI, k)$ the set of all configurations over PI that have depth at most k. We further denote by $\mathcal{L}(PI, k)$ the set of all limit configurations over PI whose elements denote sets of k-bounded configurations such that $\mathcal{L}(PI, k)$ itself does not contain the configurations in $\mathcal{C}(PI, k)$.

Theorem 7. *Let $k \in \mathbb{N}$ and let PI be a finite set of process identifiers. Then $(\mathcal{L}(PI, k), \sqsubseteq, \gamma)$ is a weak adequate domain of limits for the well-quasi-ordered set $(\mathcal{C}(PI, k), \leq)$.*

In the remainder of this section we prove Theorem 7.

4.2 Tree Encoding of Depth-Bounded Configurations

We first relate depth-bounded configurations with graphs of bounded tree-depth, which in turn can be encoded into trees of bounded height. The construction is similar to the one used in [18]. However, we prove that the tree encodings of depth-bounded configurations are not just well-quasi-ordered, but in fact better-quasi-ordered.

Communication topology. We use standard notation for (undirected) graphs. A *labelled graph* over a finite set of labels L is a tuple (G, l_v, l_e) where G is a graph, $l_v : V(G) \to L$ is a *vertex labelling function*, and $l_e : V(E) \to L$ is an *edge labelling function*.

Let $\mathcal{P} = (I, \mathcal{E})$ be a process. Let further n be the maximal arity of all vectors of names occurring in I and $\mathcal{E}$, and let $\mathcal{A}$ be the set of all process identifiers occurring in $I, \mathcal{E}$. Define the set of labels $L \overset{\text{def}}{=} 2^{\{0,\dots,n\}} \cup \mathcal{A} \cup \{\bullet\}$ where $\bullet$ is distinct from all process identifiers. Let P be a configuration of process $\mathcal{P}$ of the form

$$(\nu \boldsymbol{x})(\mathit{\Pi}_{j \in J} A_j(\boldsymbol{x}_j))$$

where $\boldsymbol{x} = x_1, \dots, x_m$, and the index sets $\{1..m\}$, and J are disjoint. The function ct maps P to a labelled graph over L as follows: the graph consists of vertices corresponding to threads and names occurring in the configuration. Each thread vertex is labelled by the process identifier of the corresponding thread in the configuration. There are edges between thread vertices and name vertices indicating that one of the names in the parameter vector of the thread is the name associated with that name vertex. Formally, $\mathsf{ct}(P)$ is a graph $((V, E), l_v, l_e)$ where

- V is a union of disjoint sets of vertices $\{v_j\}_{j \in J}$ and $\{v_1, \dots, v_m\}$,
- $E = \{\, \{v_j, v_i\} \mid j \in J \ \wedge\ 1 \le i \le m \ \wedge\ x_{j_r} = x_i \text{ for some } 1 \le r \le n \,\}$,
- $l_v(v_k) = \begin{cases} A_k & \text{if } k \in J \\ \bullet & \text{otherwise,} \end{cases}$
- $l_e(\{v_j, v_i\}) = \{\, r \mid j \in J \ \wedge\ 1 \le i \le m \ \wedge\ x_{j_r} = x_i \,\}$.

We call $\mathsf{ct}(P)$ the *communication topology* of configuration P.

Tree-depth. We relate depth-bounded sets of configurations to sets of graphs of bounded tree-depth [24]. A *path* π in a graph G is a sequence $v_1, \dots, v_n$ of vertices in $V(G)$ that are consecutively connected by edges in $E(G)$. We say that π *connects* vertices v_1 and v_n. We call π *simple path* if for all $1 \le i < j \le n$, $v_i \ne v_j$. A *tree* T is a graph such that every pair of distinct vertices in T is connected by exactly one path and this path is simple. A *rooted tree* is a tree with a dedicated root vertex. A *rooted forest* is a disjoint union of rooted trees. The *height* of a vertex v in a rooted forest F, denoted $\mathit{height}(F, v)$, is the number of vertices on the path from the root (of the tree to which v belongs) to v. The *height* of F is the maximal height of the vertices in F. Let v, w be vertices of F and let T be the tree in F to which w belongs. The vertex v is an *ancestor* of vertex w in F, denoted $v \preceq w$, if v belongs to the path connecting w and the root of T. The *closure* $\mathit{clos}(F)$ of a rooted forest F is the graph consisting of the vertices of F and the edge set $\{\, \{v, w\} \mid v \preceq w, v \ne w \,\}$. The *tree-depth* $\mathsf{td}(G)$ of a graph G is the minimum height of all rooted forests F such that $G \subseteq \mathit{clos}(F)$. The tree-depth of a labelled graph is the tree-depth of the enclosed graph. Finally, we say that a set of graphs $\mathcal{G}$ has *bounded tree-depth* if there exists $k \in \mathbb{N}$ such that all graphs $G \in \mathcal{G}$ have tree-depth at most k.

Proposition 8. *A set of configurations $\mathcal{C}$ is depth-bounded iff its communication topologies $\mathsf{ct}(\mathcal{C})$ have bounded tree-depth.*

The proof of Proposition 8 uses Meyer's characterization of sets of depth-bounded configurations in terms of sets of graphs that are bounded in the length of the simple paths [18, Theorem 1]. One can easily show that a set of graphs is bounded in the length of the simple paths if and only if it has bounded tree-depth.

We now relate the ordering on configurations $P \leq Q$ with an ordering on the underlying communication topologies. Given two labelled graphs G_1 and G_2, we say G_1 is (isomorphic to) a *subgraph* of G_2, written $G_1 \hookrightarrow G_2$, iff there exists an injective label-preserving homomorphism from G_1 to G_2.

Lemma 9. *Let P and Q be configurations. Then $P \leq Q$ iff $\mathsf{ct}(P) \hookrightarrow \mathsf{ct}(Q)$.*

Tree encoding. A labelled rooted tree over a finite set of labels L is a pair (T, l) where T is a rooted tree and $l : V(T) \to L$ a vertex labelling function. We extend the relation $\hookrightarrow$ to rooted labelled trees, as expected, and we say that a tree T_1 is a *subtree* of tree T_2 whenever $T_1 \hookrightarrow T_2$ holds. In the following, we fix a finite set of labels L. Let L_k be the set of all isomorphism classes of labelled graphs G over labels $L \cup (L \times \{1..k\})$ such that G has at most k vertices. Clearly, since L is finite, L_k is finite.

Given a labelled graph G over labels L that has tree-depth at most k, we can construct a labelled rooted tree (T, l) over the set of labels L_k from G as follows. First, let F be a rooted forest of minimal height whose closure contains the graph induced by G. The rooted tree T is constructed from the forest F by extending F with a fresh root vertex r that has edges to all the roots of the trees in F. The labelling function l is defined as follows. Let $v \in V(T)$ be a vertex in T. If $v = r$ then $l(r)$ is the empty graph. Otherwise v is a vertex in F (and thus in G). Let P be the subgraph of G that is induced by the vertices on the path from v to the root (of the tree in F to which v belongs). Now construct a graph P_h from P by adding to the label of each vertex of P its height in F. Then $l(v)$ is the isomorphism class of P_h. Since G has tree-depth at most k, $P_h \in L_k$. Thus, l is well-defined. Let Trees_k be the function mapping a labelled graph G of tree-depth at most k to the set of all labelled rooted trees over L_k that can be constructed from G as described above. We denote by $\mathrm{rng}(\mathit{Trees}_k)$ the set of labelled trees $\bigcup \{\, \mathit{Trees}_k(G) \mid G \text{ labelled graph over } L \text{ with } \mathsf{td}(G) \leq k \,\}$.

Lemma 10. *Let $k \in \mathbb{N}$ and T_1, T_2 be trees in $\mathrm{rng}(\mathit{Trees}_k)$. If T_1 is a subtree of T_2 then $G_1 \hookrightarrow G_2$ for all G_1, G_2 with $T_1 \in \mathit{Trees}_k(G_1)$ and $T_2 \in \mathit{Trees}_k(G_2)$.*

Let T be a rooted tree and $x, y \in V(T)$ two vertices. The infimum of x and y, denoted $x \inf y$, is the vertex $z \in V(T)$ with the greatest height such that $z \preceq x$ and $z \preceq y$. Given rooted trees T_1 and T_2, a function φ is an *inf-preserving embedding* from T_1 into T_2 iff (1) $\varphi : V(T_1) \to V(T_2)$ is injective, and (2) for all $x, y \in V(T_1)$, $\varphi(x \inf y) = \varphi(x) \inf \varphi(y)$. An embedding between two rooted labelled trees over the same set of labels is *label-preserving* iff it maps vertices to vertices with the same label.

Clearly, if a tree is a subtree of another tree then there exists an inf and label preserving embedding between these trees. For trees that result from the tree encoding of configurations the converse holds, too. Vertices of different height in such trees have always different labels. Thus, an inf and label-preserving embedding between such trees also preserves antecedence of vertices.

Lemma 11. *Let $k \in \mathbb{N}$ and T_1, T_2 be trees in $\mathrm{rng}(\mathit{Trees}_k)$. Then the following two properties are equivalent:*

1. *there exists an inf and label-preserving embedding from T_1 to T_2;*
2. *T_1 is a subtree of T_2.*

Laver [16] proved a variation of Kruskal's tree theorem for trees labelled by a bqo set, namely that countable rooted trees labelled by a bqo set are a bqo under inf-preserving embedding. Similar to Friedman's special case of Kruskal's tree theorem, we get the special case of Laver's theorem that rooted trees labelled by a finite set of labels are better-quasi-ordered by inf and label-preserving embedding. Thus, together with Lemma 10 we get the following proposition.

Proposition 12. *For any $k \in \mathbb{N}$, $(\mathrm{rng}(\mathit{Trees}_k), \hookrightarrow)$ is a bqo set.*

4.3 Limit Configurations as Ideal Completions

Finkel and Goubault-Larrecq [10] characterize the minimal candidates for the WADLs of a wqo set X in terms of its ideal completion. This means that the set of all downward-closed directed subsets of X forms a WADL for X. We use this observation to prove that limit configurations form WADLs for depth-bounded configurations.

Proposition 13. *The directed downward-closed sets of depth-bounded configurations are exactly the denotations of limit configurations.*

By [10, Proposition 3.3] the above proposition implies Theorem 7. In our proof of Proposition 13 we characterize the tree encodings of downward-closed sets of configurations in terms of the languages of *hedge automata* [5, Chapter 8].

Hedge automata. A *(nondeterministic) finite hedge automaton* $\mathcal{A}$ over a finite alphabet Σ is a tuple (Q, Σ, Q_f, Δ) where Q is a finite set of states, $Q_f \subseteq Q$ is a set of final states, and Δ is a finite set of transition rules of the following form:

$$a(R) \to q$$

where $a \in \Sigma$, $q \in Q$, and $R \subseteq Q^*$ is a regular language over Q. These languages R occuring in the transition rules are called *horizontal languages*.

A *run* of $\mathcal{A}$ on a rooted labelled tree T with vertex label function $l : V(T) \to \Sigma$ is a vertex label function $r : V(T) \to Q$ such that for each vertex $v \in V(T)$ with $a = l(v)$ and $q = r(v)$ there is a transition rule $a(R) \to q$ with $r(v_1) \dots r(v_n) \in R$ where $v_1, \dots, v_n$ are the immediate successors of v in T. In particular, to apply a rule to a leaf, the empty word ϵ has to be in the horizontal language of the rule R.

A rooted labelled tree T is *accepted* by $\mathcal{A}$ if there is a run r of $\mathcal{A}$ on T such that r labels the root of T by a final state. The *language* $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of all rooted labelled trees over Σ that are accepted by $\mathcal{A}$.

Finite partitions of well-quasi-ordered sets. In order to characterize the horizontal languages of the constructed hedge automaton we will define equivalence classes on the vertices of the individual levels of the tree encodings. For this purpose, the following definition will be useful. Let $(X, \leq)$ be a well-quasi-ordered set. We call a partition $\mathcal{P} \subseteq 2^X$ of X an *infinite chain partition* if and only if (1) $\mathcal{P}$ is finite and (2) for all $Y \in \mathcal{P}$, either Y is a singleton or Y contains an infinite chain C such that $Y \leq C$.

Proposition 14. *If $(X, \leq)$ is a countable well-quasi-ordered set then there exists an infinite chain partition of X.*

In order to prove Proposition 13, we first prove that every directed downward-closed set of depth-bounded configurations is the denotation of a limit configuration.

Lemma 15. *Let D be a downward-closed set of configurations then there exists a limit configuration E such that $D = \gamma(E)$.*

For proving the lemma, let $D = (P_i)_{i \in \mathbb{N}}$ be a downward-closed directed family of configurations and let k be the maximal tree-depth of the graphs in $\mathsf{ct}(D)$. Choose some $Q_0 \in D$ whose communication topology has tree-depth k. Using Q_0 construct an ascending chain $D' = (Q_i)_{i \in \mathbb{N}}$ as follows: for each $i \in \mathbb{N}$ choose $Q_i \in D$ such that $P_i \leq Q_i$ and $Q_{i-1} \leq Q_i$. Such Q_i exists for each $i \in \mathbb{N}$ since D is directed and, by induction, $Q_{i-1} \in D$. Then by construction (1) $D = \downarrow D'$ and (2) all elements in D' have tree-depth k. Let $(G_i)_{i \in \mathbb{N}}$ be the family of labelled graphs $G_i = \mathsf{ct}(Q_i)$. Now for each $i \in \mathbb{N}$ choose a tree $T_i \in \mathit{Trees}_k(G_i)$ such that the family $\mathcal{T} = (T_i)_{i \in \mathbb{N}}$ is an ascending chain with respect to the subtree relation. Such a family exists because the G_i are ordered by subgraph isomorphism and all G_i have the same tree-depth. Without loss of generality we assume that the vertex sets of all trees T_i are pairwise disjoint.

Let $V = \bigcup_{i \in \mathbb{N}} V(T_i)$, $E = \bigcup_{i \in \mathbb{N}} E(T_i)$, and let l be the union of all the vertex labelling functions of the labelled trees T_i. The height of the vertices in the trees T_i range from 1 to $k+1$. For a node $x \in V$ of height $h > 1$ we denote by $\mathit{parent}(v) \in V$ the parent of v in the tree T_i to which v belongs. Similarly, for a node $v \in V$ we denote by $\mathit{Children}(v)$ the set of all vertices that are direct successors of v in the tree to which v belongs. We extend the functions parent to sets of vertices, as expected. Furthermore, let $T(v)$ be the subtree rooted in v of the tree T_i with $v \in V(T_i)$. For all $1 \leq h \leq k+1$, let V_h be the set of all vertices in V that have height h. For all h we extend the relation $\hookrightarrow$ from labelled rooted trees to vertices in V_h as expected: for all $v, w \in V_h$, $v \hookrightarrow w$ iff $T(v) \hookrightarrow T(w)$. From Proposition 12 and Property 3 of Proposition 3 follows that for all h, $(V_h, \hookrightarrow)$ is a bqo.

We will now construct a finite hedge automaton $\mathcal{A}$ from the family of trees $\mathcal{T}$ whose language is both small and large in $\mathcal{T}$. For this purpose we define an equivalence relation on each V_h that partitions V_h into finitely many equivalence classes. These equivalence classes serve as the states of the automaton.

For each $i \in \mathbb{N}$ fix some injective label-preserving homomorphism $\phi_i : V(T_i) \to V(T_{i+1})$ and denote by $\phi_{[i,j]}$ the composition $\phi_{j-1} \circ \cdots \circ \phi_i$ if $j > i$ and the identify function id if $j = i$. Then define an equivalence relation $\sim$ on V as follows: for all $v_i \in V(T_i)$ and $v_j \in V(T_j)$

$$v_i \sim v_j \quad \text{iff} \quad \begin{array}{l} i \leq j \text{ and } \phi_{[i,j]}(v_i) = v_j \text{ or} \\ i \geq j \text{ and } \phi_{[j,i]}(v_j) = v_i \end{array}$$

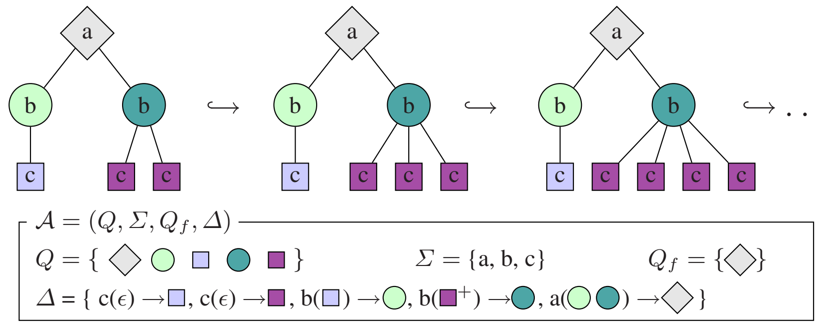

a
b
b
c
c
c
$\hookrightarrow$
$\hookrightarrow$
$\hookrightarrow \ldots$
$\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$
$Q = \{$ ◇ ● ■ ● ■ $\}$
$\Sigma = \{$a, b, c$\}$
$Q_f = \{$◇$\}$
$\Delta = \{$ c(ϵ) → ■, c(ϵ) → ■, b(■) → ●, b(■$^+$) → ●, a(● ●) → ◇ $\}$

Fig. 1. A chain of labelled trees with the equivalence classes under the relations $\simeq_h$ and the constructed hedge automaton

Now, recursively define an equivalence relation $\simeq_h$ on V_h for each $1 \leq h \leq k+1$ as follows: for $h = 1$ we simply have $v \simeq_1 w$ for all $v, w \in V_1$. In order to define $\simeq_h$ for $h > 1$ we need some intermediate definitions. Given an equivalence class U in the quotient of V_{h-1} wrt. $\simeq_{h-1}$, let $\mathit{Children}(U)$ be the set of equivalence classes $\tilde{v}$ in the quotient $V_h/_\sim$ such that some $v \in \tilde{v}$ has a parent in U. Since $(V_h, \hookrightarrow)$ is a bqo, and $\mathit{Children}(U) \subseteq 2^{V_h}$, it follows from Proposition 3 that $(\mathit{Children}(U), \hookrightarrow_1)$ is also a bqo and thus a wqo. Furthermore, $\mathit{Children}(U)$ is countable. Thus, by Proposition 14 there exists an infinite chain partition of $\mathit{Children}(U)$. For each U, choose one such infinite chain partition $\mathcal{P}(U)$ of $\mathit{Children}(U)$. Then for $v, w \in V_h$ we define: $v \simeq_h w$ iff there exists $U \in V_{h-1}/_{\simeq_{h-1}}$ such that (1) $\mathit{parent}(v), \mathit{parent}(w) \in U$ and (2) there is $P \in \mathcal{P}(U)$ such that $v, w \in \bigcup P$.

We can easily prove by induction on h that each $\simeq_h$ is indeed an equivalence relation on V_h and that each $\simeq_h$ partitions V_h into finitely many equivalence classes. Furthermore, using the definition of infinite chain partitions one can easily prove the following properties.

Lemma 16. *Let $U \in V_h/_{\simeq_h}$ then*

1. *all $v \in U$ have the same label,*
2. *U is directed with respect to $\hookrightarrow$,*
3. *if $h = 1$ then U contains exactly the root vertices of all the trees T_i,*
4. *if $h > 1$ then $\mathit{parent}(U) \subseteq U'$ for some $U' \in V_{h-1}/_{\simeq_{h-1}}$ and*
   (a) either all vertices in U' have at most one child in U or
   (b) every $v \in U$ is contained in a proper infinite chain $C \subseteq U$ and for every finite subset $V \subseteq U$ there exists $v' \in U'$ such that $V \hookrightarrow_1 \mathit{Children}(v') \cap U$.

Now let $\simeq$ be the union of all the relations $\simeq_h$. Then $\simeq$ is an equivalence relation on V that partitions V into finitely many equivalence classes. For an equivalence class $U \in V/_\simeq$, let $C(U)$ be the set of all equivalence classes that contain children of vertices in U. Furthermore, let $l(U)$ be the unique label of all vertices in U, and let $m(U)$ denote 1 if every parent of a vertex $v \in U$ has at most one child in U and, otherwise, let $m(U)$ denote the symbol $+$. Now define the hedge automaton $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$ where:

- $Q = V/_\simeq$,
- $\Sigma = L_k$,

- $Q_f = V_1/_{\simeq}$,
- Δ consists of transition rules of the following form for each $U \in V/_{\simeq}$
  - $l(U)(U_1^{m(U_1)} \cdots U_n^{m(U_n)}) \to U$ if $C(U) = \{U_1, \dots, U_n\}$
  - $l(U)(\epsilon) \to U$ if $C(U) = \emptyset$.

Figure 1 depicts a chain of trees and the constructed automaton $\mathcal{A}$. The equivalence classes in the quotient $V/\simeq$ are highlighted in the trees.

Using Lemma 16 we can now prove that the language accepted by $\mathcal{A}$ is both small and large in $\mathcal{T}$.

Lemma 17. *$\mathcal{L}(\mathcal{A})$ is small in $\mathcal{T}$, i.e., $\forall T \in \mathcal{L}(\mathcal{A})\, \exists i \in \mathbb{N} :\ T \hookrightarrow T_i$.*

Lemma 18. *$\mathcal{L}(\mathcal{A})$ is large in $\mathcal{T}$, i.e., $\forall i \in \mathbb{N}\, \exists T \in \mathcal{L}(\mathcal{A}) :\ T_i \hookrightarrow T$.*

Note that by construction of $\mathcal{A}$ the tree encoding operation can be reversed on the trees in $\mathcal{L}(\mathcal{A})$. Let $D_{\mathcal{A}}$ be the corresponding set of configurations. From Lemmas 17,18,10, and 9 follows that $D = \downarrow D' = \downarrow D_{\mathcal{A}}$. From $\mathcal{A}$ we can now easily construct a limit configuration E whose denotation is the downward closure of $D_{\mathcal{A}}$. It follows that $D = \downarrow D_{\mathcal{A}} = \gamma(E)$ which proves Lemma 15.

Lemma 19. *For any limit configuration E, $\gamma(E)$ is a downward-closed directed set.*

Clearly $\gamma(E)$ is downward-closed. For proving that $\gamma(E)$ is directed, we can again construct a hedge automaton $\mathcal{A}$ from E, such that the tree encoding operation can be reversed on all trees accepted by $\mathcal{A}$ and the downward-closure of the resulting configurations $D_{\mathcal{A}}$ coincides with $\gamma(E)$. Using a simple pumping argument for the language $\mathcal{L}(\mathcal{A})$ we can show that for every two trees $T_1, T_2 \in \mathcal{L}(\mathcal{A})$ there exists a tree $T \in \mathcal{L}(\mathcal{A})$ such that $T_1 \hookrightarrow T$ and $T_2 \hookrightarrow T$. It follows that $D_{\mathcal{A}}$ is directed and thus $\gamma(E)$.

5 Forward Analysis of Depth-Bounded Processes

The expand, enlarge, and check (EEC) algorithm of Geeraerts et al. [13] is a forward algorithm that decides the covering problem for *effective* WSTSs with appropriate adequate domain of limits.

A WSTS $(X, x_0, \to, \leq)$ and an adequate domain of limits $(Y, \sqsubseteq, \gamma)$ for the wqo $(X, \leq)$ are *effective* if the following conditions are satisfied: (E1) X and Y are recursively enumerable; (E2) for any $x_1, x_2 \in X$, one can decide whether $x_1 \to x_2$; (E3) for any $z \in X \cup Y$ and for any finite subset $Z \subseteq X \cup Y$, one can decide whether $Post(\gamma(z)) \subseteq \gamma(Z)$; and (E4) for any finite subsets $Z_1, Z_2 \subseteq X \cup Y$, one can decide whether $\gamma(Z_1) \subseteq \gamma(Z_2)$.

We argue that the WSTS induced by a depth-bounded process together with its WADL of limit configurations are effective. The conditions (E1) and (E2) are clearly satisfied. Also given a limit configuration z we can compute a finite set of limit configurations denoting $Post(\gamma(z))$. Note further that Proposition 13 implies that for any finite subsets $Z_1, Z_2 \subseteq \mathcal{L}(PI, k)$, $\gamma(Z_1) \subseteq \gamma(Z_2)$ holds if and only if for all $z_1 \in Z_1$ there exists $z_2 \in Z_2$ such that $\gamma(z_1) \subseteq \gamma(z_2)$. The inclusion problem $\gamma(z_1) \subseteq \gamma(z_2)$ can be reduced to the language inclusion problem for deterministic hedge automata, which is decidable. For this purpose, one computes deterministic hedge automata from the finitely many tree encodings of the configurations of z_1 and z_2 and then checks whether the language of some automaton of z_1 is included in the language of some automaton of z_2. Thus conditions (E3) and (E4) are also satisfied.

Finally, let us explain why the EEC algorithm terminates on depth-bounded systems even if the bound of the system is not known a priori. The idea of the algorithm is to simultaneously enumerate two infinite increasing chains. The first chain $X_0 \subseteq X_1 \ldots$ is a sequence of finite subsets of X that contains all reachable configurations of the analyzed system. The second chain $Y_0 \subseteq Y_1 \subseteq \ldots$ is a sequence of finite subsets of Y that contains all limits Y. In each iteration i the algorithm computes an under and an over-approximation of the analyzed system for the current pair (X_i, Y_i) of elements in the chain. These approximations are such that the under-approximation is guaranteed to detect that t can be covered if X_i contains a path to a covering state. The over-approximation is guaranteed to detect that t cannot be covered if Y_i can express $\downarrow Post^*(\downarrow s_0)$ and this set does not cover t. The conditions on the chains ensure that one of the two conditions eventually holds for some $i \in \mathbb{N}$.

For deciding the covering problem of depth-bounded systems we can now simply enumerate the sets $\mathcal{C}(PI) = \bigcup_{k \in \mathbb{N}} \mathcal{C}(PI, k)$ and $\mathcal{L}(PI) = \bigcup_{k \in \mathbb{N}} \mathcal{L}(PI, k)$. Then in each iteration of the EEC algorithm the pair (X_i, Y_i) is contained in some limit domain $\mathcal{C}(PI, k), \mathcal{L}(PI, k))$ and the conditions on the chains for termination of the EEC algorithm are still satisfied.

Theorem 20. *The covering problem for depth-bounded processes is decidable.*

Complexity. Depth-bounded processes subsume Petri nets [19]. This implies an exponential space lower bound on the complexity of the covering problem for depth-bounded processes [17]. The exact complexity is open.

6 Conclusion

At the dawn of cloud computing and processors that put an enormous number of cores at disposal of the programmer, message passing concurrency is gaining more and more importance. Many typical use cases of message passing such as client-server and consumer-producer communication with an unbounded number of clients/producers, and master-worker load balancing, can be modeled by depth-bounded processes. The covering problem plays a central role in the automated verification of the correctness of message passing systems. We prepared the ground for a spectrum of forward algorithms that solve the covering problem for depth-bounded processes. The exploitation of our result for the development of practical forward algorithms for this class of systems is our primary goal for future research.

References

1. Abdulla, P.A., Cerans, K., Jonsson, B., Tsay, Y.-K.: General decidability theorems for infinite-state systems. In: LICS, pp. 313–321 (1996)
2. Amadio, R.M., Meyssonnier, C.: On decidability of the control reachability problem in the asynchronous pi-calculus. Nord. J. Comput. 9(1), 70–101 (2002)

3. Bauer, J., Wilhelm, R.: Static analysis of dynamic communication systems by partner abstraction. In: SAS, pp. 249–264 (2007)
4. 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)
5. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Löding, C., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications (2008), http://tata.gforge.inria.fr/ (release November 18, 2008)
6. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL, pp. 238–252 (1977)
7. Dam, M.: Model checking mobile processes. In: Best, E. (ed.) CONCUR 1993. LNCS, vol. 715, pp. 22–36. Springer, Heidelberg (1993)
8. Finkel, A.: A Generalization of the Procedure of Karp and Miller to Well Structured Transition Systems. In: Ottmann, T. (ed.) ICALP 1987. LNCS, vol. 267, pp. 499–508. Springer, Heidelberg (1987)
9. Finkel, A.: Reduction and covering of infinite reachability trees. Inf. Comput. 89(2), 144–179 (1990)
10. Finkel, A., Goubault-Larrecq, J.: Forward Analysis for WSTS, Part I: Completions. In: STACS. Dagstuhl Sem. Proc., vol. 09001, pp. 433–444 (2009)
11. Finkel, A., Goubault-Larrecq, J.: Forward Analysis for WSTS, Part II: Complete WSTS. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 188–199. Springer, Heidelberg (2009)
12. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theor. Comput. Sci. 256(1-2), 63–92 (2001)
13. Geeraerts, G., Raskin, J.-F., Van Begin, L.: Expand, Enlarge and Check: New algorithms for the coverability problem of WSTS. J. Comput. Syst. Sci. 72(1), 180–203 (2006)
14. Janssens, D., Lens, M., Rozenberg, G.: Computation graphs for actor grammars. J. Comput. Syst. Sci. 46(1), 60–90 (1993)
15. Karp, R.M., Miller, R.E.: Parallel program schemata. J. Comput. Syst. Sci. 3(2), 147–195 (1969)
16. Laver, R.: On Fraïssé's Order Type Conjecture. Ann. of Math. 93(1), 89–111 (1971)
17. Lipton, R.J.: The reachability problem requires exponential space. Technical Report 62, Yale University (1976)
18. Meyer, R.: On boundedness in depth in the pi-calculus. In: IFIP TCS. IFIP, vol. 273, pp. 477–489. Springer, Heidelberg (2008)
19. Meyer, R., Gorrieri, R.: On the relationship between pi-calculus and finite place/transition petri nets. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 463–480. Springer, Heidelberg (2009)
20. Milner, R.: Flowgraphs and flow algebras. J. ACM 26(4), 794–818 (1979)
21. Milner, R., Parrow, J., Walker, D.: A Calculus of Mobile Processes, I. Inf. Comput. 100(1), 1–40 (1992)
22. Milner, R., Parrow, J., Walker, D.: A Calculus of Mobile Processes, II. Inf. Comput. 100(1), 41–77 (1992)
23. Nash-Williams, C.S.J.A.: On better-quasi-ordering transfinite sequences. Proc. Camb. Phil. Soc. 64, 273–290 (1968)
24. Nešetřil, J., de Mendez, P.O.: Tree-depth, subgraph coloring and homomorphism bounds. Eur. J. Comb. 27(6), 1022–1041 (2006)
25. Ostrovský, K.: On Modelling and Analysing Concurrent Systems. PhD thesis, Chalmers University of Technology and Goteborg University (2005)
26. Sangiorgi, D.: pi-calculus, internal mobility, and agent-passing calculi. Theor. Comput. Sci. 167(1&2), 235–274 (1996)

Incremental Pattern-Based Coinduction for Process Algebra and Its Isabelle Formalization

Andrei Popescu and Elsa L. Gunter

University of Illinois at Urbana-Champaign

Abstract. We present a coinductive proof system for bisimilarity in transition systems specifiable in the *de Simone* SOS format. Our coinduction is *incremental*, in that it allows building incrementally an a priori unknown bisimulation, and *pattern-based*, in that it works on equalities of process patterns (i.e., universally quantified equations of process terms containing process variables), thus taking advantage of equational reasoning in a "circular" manner, inside coinductive proof loops. The proof system has been formalized and proved sound in Isabelle/HOL.

1 Introduction

Bisimilarity is arguably the most natural equivalence on interactive processes. Assuming process transitions are labeled by (observable) actions a, processes P and P' are bisimilar iff: **(I)** whenever P can a-transit to a process Q, P' can also a-transit to some process Q' such that P' and Q' are again bisimilar; **(II)** and vice versa; **(III)** and so on, *indefinitely* (as in an infinite game).

The above informal description of the bisimilarity relation can of course be made rigorous by defining bisimilarity to be the largest *bisimulation*, i.e., the largest relation θ for which (I) and (II) hold (with "bisimilar" replaced by "in θ"). But the largest-fixpoint description loses (at least superficially) the game-theoretic flavor of the intuitive description, so we stick to the latter for a while. How would one go about proving that P and Q are bisimilar? Well, if one were allowed an infinite proof, one could try to show that each transition of P is matched by a transition of Q so that the continuations P' and Q' are (claimed to be) bisimilar (and vice versa), and then prove the bisimilarity claims about all pairs of continuations P' and Q', and so on. This way, one would build an infinite tree whose nodes contain bisimilarity claims about pairs of processes. Now assume that, while expanding the tree, one encounters a repetition of a previous claim (that appeared on an ancestor node). A reasonable "optimization" of the infinite proof would then be to stop and "seal" that node, because the bisimilarity argument for its ancestor can be repeated ad litteram. In other words, one may take the (yet unresolved!) goal of the ancestor as a hypothesis, which discharges the repetitive goal – this is the upside of trying to build an infinite proof: non-well-foundedness (i.e., circularity) works in our advantage. Assume now one finds such repetitions on all paths when building the tree. Then our bisimilarity proof is done! In terms of the fixpoint definition, we have proved that the pair (P, Q) of processes located at the root are bisimilar by *coinduction*, i.e., by exhibiting a bisimulation that contains (P, Q). In terms of proof engineering however, the needed bisimulation did not appear out of nowhere, but was built incrementally from the goal, essentially by an exploration that discovered a regular pattern for an infinite proof tree. In fact, *coinductive proofs are intuitively all about discovering regular patterns.*

This paper provides formal support for this intuition. Here is an illustration of our approach, for a mini process calculus. Fix a set of actions **act** with a given silent action $\tau \in \mathbf{act}$ and a map on $\mathbf{act} \setminus \{\tau\}$, $a \mapsto \overline{a}$, such that $\overline{\overline{a}} = a$ for all $a \in Act$. The processes P are generated by the grammar: $P ::= 0 \mid a.P \mid P|Q \mid !P$. Thus, we have idle process, action prefix, parallel composition, and replication. "!" binds more strongly than "|". The behavior of processes is specified by the following labeled transition system:

$$\frac{\cdot}{a.P \overset{a}{\rightsquigarrow} P}(\mathsf{PREF}) \quad \frac{P_0 \overset{a}{\rightsquigarrow} Q_0}{P_0|P_1 \overset{a}{\rightsquigarrow} Q_0|P_1}(\mathsf{PARL}) \quad \frac{P_1 \overset{a}{\rightsquigarrow} Q_1}{P_0|P_1 \overset{a}{\rightsquigarrow} P_0|Q_1}(\mathsf{PARR})$$

$$\frac{P_0 \overset{a}{\rightsquigarrow} Q_0 \quad P_1 \overset{\overline{a}}{\rightsquigarrow} Q_1}{P_0|P_1 \overset{\tau}{\rightsquigarrow} Q_0|Q_1}(\mathsf{PARS}) \quad \frac{P \overset{a}{\rightsquigarrow} Q}{!P \overset{a}{\rightsquigarrow} !P|Q}(\mathsf{REPL}) \quad \frac{P \overset{a}{\rightsquigarrow} Q_0 \quad P \overset{\overline{a}}{\rightsquigarrow} Q_1}{!P \overset{\tau}{\rightsquigarrow} !P|(Q_0|Q_1)}(\mathsf{REPLS})$$

We may wish to prove in this context that parallel composition is associative and commutative and that replication absorbs self-parallel composition, i.e., that $(P_0|P_1)|P_2 = P_0|(P_1|P_2)$, $P_0|P_1 = P_1|P_0$, and $P|!P = !P$ for all processes P_0, P_1, P_2, P, where we write "=" for *strong bisimilarity*. In fact, assume we already proved the first two facts and are left with proving the third, $P|!P = !P$. For this, we first check to see if the equations we already know so far (associativity and commutativity of |) imply this new one by pure equational reasoning – no, they don't. This means we cannot discharge the goal right away, and therefore we need to perform *unfoldings* of the two terms in the goal. We unfold $P|!P$ and $!P$ until we reach hypotheses involving only process meta-variables. The upper side of Figure 1 contains all possible *derived rules* (i.e., compositions of primitive rules in the system, all the way down to non-decomposable hypotheses) that can be matched by $P|!P$ in order to infer a transition from $P|!P$. And, similarly, the lower side for the term $!P$ – in this latter case, the matched derived

$$\frac{P \overset{a}{\rightsquigarrow} Q}{P|!P \overset{a}{\rightsquigarrow} Q|!P}(\mathsf{PARL})\ (1) \qquad \frac{\dfrac{P \overset{a}{\rightsquigarrow} Q}{!P \overset{a}{\rightsquigarrow} !P|Q}(\mathsf{REPL})}{P|!P \overset{a}{\rightsquigarrow} P|(!P|Q)}(\mathsf{PARR})\ (2)$$

$$\frac{\dfrac{P \overset{a}{\rightsquigarrow} Q_0 \quad P \overset{\overline{a}}{\rightsquigarrow} Q_1}{!P \overset{\tau}{\rightsquigarrow} !P|(Q_0|Q_1)}(\mathsf{REPLS})}{P|!P \overset{\tau}{\rightsquigarrow} P|(!P|(Q_0|Q_1))}(\mathsf{PARR})\ (3) \qquad \frac{P \overset{a}{\rightsquigarrow} Q_0 \quad \dfrac{P \overset{\overline{a}}{\rightsquigarrow} Q_1}{!P \overset{\overline{a}}{\rightsquigarrow} !P|Q_1}(\mathsf{REPL})}{P|!P \overset{\tau}{\rightsquigarrow} Q_0|(!P|Q_1)}(\mathsf{PARS})\ (4)$$

$$\frac{P \overset{a}{\rightsquigarrow} Q}{!P \overset{a}{\rightsquigarrow} !P|Q}(\mathsf{REPL})\ (5) \qquad \frac{P \overset{a}{\rightsquigarrow} Q_0 \quad P \overset{\overline{a}}{\rightsquigarrow} Q_1}{!P \overset{\tau}{\rightsquigarrow} !P|(Q_0|Q_1)}(\mathsf{REPLS})\ (6)$$

Fig. 1. The matching derived rules for $P|!P$ and $!P$

rules coincide with the matched primitive rules. To see how the derived rules are obtained, the figure shows whole *derivation trees*, but we only care about the leaves and the roots of these trees.

Next, we try to pair these derived rules (upper versus lower), by the accordance of their hypotheses and their transition labels. The only valid pairing possibilities are: (1) with (5), (2) with (5), (3) with (6), and (4) with (6). The targets of the conclusions of the rules in these pairs yield four new goals: **(i)** $Q|!P = !P|Q$; **(ii)** $P|(!P|Q) = !P|Q$; **(iii)** $P|(!P|(Q_0|Q_1)) = !P|(Q_0|Q_1)$; **(iv)** $Q_0|(!P|Q_1) = !P|(Q_0|Q_1)$. The original goal, $P|!P = !P$, is replaced by the above four goals, *and is also henceforth taken as a hypothesis*. Notice that our goals are generic, i.e., universally quantified over the occurring process meta-variables, P, Q, Q_0, Q_1. Now, *equational reasoning* (by standard equational rules, *including substitution*) with hypothesis $P|!P = !P$ together with the already known lemmas $(P_0|P_1)|P_2 = P_0|(P_1|P_2)$ and $P_0|P_1 = P_1|P_0$ is easily seen to discharge each of the remaining four goals, and the proof is done.

Why is this proof valid, i.e., why does it represent a proof of the fact that, for all process terms P, $P|!P$ and $!P$ are bisimilar? The rigorous justification for this is the topic of this paper. But the short answer has to do with our previous discussion on discovering patterns: the above is really a proof by coinduction (on universally quantified equalities of terms up to equational closure), which *builds incrementally* the relation representing the coinductive argument. Notice the appearance of *circular reasoning*: a goal that cannot be locally discharged is expanded according to the SOS definition transition relation and *becomes a hypothesis*. In this particular example, the proof is finished after only one expansion, but the process of expanding the goals with taking them as hypotheses may in principle continue, obtaining arbitrarily large proof trees.

We show that deductions such as the above are sound for a wide class of process algebras – those specifiable by SOS rules in the de Simone format [11]. Our results have been given a formalization in Isabelle/HOL [3], which was desirable for two reasons: first, the very technical constructions (especially in Sec. 4) and arguments (in both Secs. 3 and 4) were asking for a means to be absolutely sure of their correctness; second, the formalization has the potential of leading to an implementation of a coinductive tool. Here is the structure of this paper. The rest of this section establishes some notation. Sec. 2 discusses our representation of the de Simone format. Secs. 3 and 4 contain our original theoretic contribution: incremental proof systems for bisimilarity – Sec. 3 for standard bisimilarity, Sec. 4 for universally quantified bisimilarity equations. Sec. 5 discusses related and future work. More details on our Isabelle scripts and on various other technical topics, as well as more examples, can be found in the technical report [28], to which we occasionally refer from this paper. [28] is an identical copy of this paper, except that it has an appendix with more details. The Isabelle scripts can be found at http://hdl.handle.net/2142/14857 in both html-browsable and pdf formats (App. C in [28] has details about the scripts.)

Conventions and notations. By “Isabelle”, here we mean “Isabelle/HOL”. We present our work in the usual mathematical language, but partly employ the *Isabelle dialect* of this language in order to allow the interested reader to easily relate this paper with our Isabelle formal proofs. (We believe that this choice does not decrease readability, since the Isabelle notation is very close to standard mathematical notation and also occasionally allows for clear and concise formulations, as, e.g., with datatypes and records. A priori familiarity with the Isabelle dialect is *not* required from the reader.)

Isabelle distinguishes between a *type* and a *set*, but the set-theoretical-oriented reader is free to ignore this distinction; as a matter of syntax though, membership to a type is denoted by “::” and membership to a set by “$\in$”. **nat** is the type of naturals. Given types α and β, $\alpha \times \beta$ is their product type, $\alpha \Rightarrow \beta$ the type of functions between α and β, α $\underline{\textbf{list}}$ the type of lists of items in α, and α $\underline{\textbf{set}}$ the type of sets of items in α.[1] fst and snd are the two projections from $\alpha \times \beta$. $[]$ is the empty list and $[a_0, \ldots, a_{n-1}]$ the list consisting of the n indicated items; given a list L and $i < \mathsf{length}\, L$, $L!i$ is the $(i+1)$-th element in L (thus, the first element is $L!0$). The operator $\mathsf{set} : \alpha\ \underline{\textbf{list}} \Rightarrow \alpha\ \underline{\textbf{set}}$ gives the set of the items appearing in a list. A list L is said to be *nonrepetitive* if $L!i \neq L!j$ for all $i, j < \mathsf{length}\ L$ with $i \neq j$. For readability, we consistently use: sans serif fonts for constants, such as length and set; boldface for types, such as **nat**; underlined boldface for type constructors, such as $\underline{\textbf{list}}$ and $\underline{\textbf{set}}$.

2 Syntax and Operational Semantics of Processes

Process variables, terms and substitution. We fix the following types: **param**, of *parameters*, ranged over by p; **opsym**, of *operation symbols* (*opsyms* for short), ranged over by f, g; **var**, of *(process) variables*, ranged over by X, Y, Z – this latter type is assumed to be infinite. The type **term**, of *(process) terms*, ranged over by P, Q, R, T, S, U, V, is defined as an Isabelle datatype (i.e., initial algebra): Datatype **term** = Var **var** | Op **opsym** (**param** $\underline{\textbf{list}}$) (**term** $\underline{\textbf{list}}$).

Thus, a term can have any opsym at the top, applied to any list of parameters and any list of terms (of any length), without being subject to further well-formedness conditions. Hence an opsym f does *not* have an a priori associated numeric rank (m, n) (indicating that f takes m parameters and n terms). Rather, we allow in **term** the whole pool of all possible terms under all possible rankings of the operation symbols. This looseness w.r.t. terms is admittedly a formalization shortcut (fitting nicely the Isabelle simply-typed framework), but is completely unproblematic for the concepts and results of this paper: while an SOS specification of a transition system will of course employ only certain (possibly overloaded) ranks for the opsyms, the unused ranks will be harmless, since they will not affect transitions or bisimilarity.

[1] Note the use of postfix notation for type constructors – this is not standard mathematically, but is intuitive, as it matches natural language: while an element of **int** is an integer, an element of **int** $\underline{\textbf{list}}$ is an integer list.

σ and τ will range over **var** $\Rightarrow$ **term**. We consider the operators:
- vars :: **term** $\Rightarrow$ **var** **<u>set</u>**, giving the set of variables occurring in a term.
- $\_[\_]$ :: **term** $\times$ (**var** $\Rightarrow$ **term**) $\Rightarrow$ **term**, such that $T[\sigma]$ is the term obtained from T by substituting all its variables X by $\sigma\, X$.

Next we represent the meta-SOS notion of a *transition-system specification* [15,26]. Given any type α, the type α **<u>ftrans</u>**, of formal α-transitions, consists of pairs, written $k \rightsquigarrow l$, with $k, l :: \alpha$, where k is called the *source* and l the *target*. We fix a type **act**, of *actions*, ranged over by a, b.

Rules syntax. The type **rule**, of *(SOS-)rules*, ranged over by rl, is defined to be the following record type (i.e., a product with named projections): Record **rule** =

hyps :: (**var** **<u>ftrans</u>**) **<u>list</u>** (read "hypotheses")
cnc :: **term** **<u>ftrans</u>** (read "conclusion")
side :: (**nat** $\Rightarrow$ **act**) $\Rightarrow$ **act** $\Rightarrow$ **bool** (read "side-condition")

The hypotheses and the conclusions of our rules are therefore formal transitions between variables, and between terms, respectively. I.e., for any rule rl:
- $\mathsf{hyps}\ rl$ has the form $[XX_0 \rightsquigarrow Y_0, \ldots, XX_{n-1} \rightsquigarrow Y_{n-1}]$, with XX_j, Y_j variables;
- $\mathsf{cnc}\ rl$ has the form $S \rightsquigarrow T$, with S and T terms.

One can visualize rl as

$$\frac{XX_0 \rightsquigarrow Y_0, \ \ldots\ , XX_{n-1} \rightsquigarrow Y_{n-1}}{S \rightsquigarrow T}\ [\lambda\, as, b.\ \mathsf{side}\ rl\ as\ b]$$

where as :: **nat** $\Rightarrow$ **act** and b :: **act**. Actually, we think of rl as follows:

$$\frac{XX_0 \overset{as\,0}{\rightsquigarrow} Y_0\ ,\ \ldots\ , XX_{n-1} \overset{as\,(n-1)}{\rightsquigarrow} Y_{n-1}}{S \overset{b}{\rightsquigarrow} T}\ [\mathsf{side}\ rl\ as\ b]$$

Note however that the side condition $\mathsf{side}\ rl$ is (for now) allowed to take into consideration the whole function as, and not only its first n values $as\ 0, \ldots, as\ (n-1)$, as one would expect – this is corrected below by "saneness".

Given a rule rl with $\mathsf{hyps}\ rl$ and $\mathsf{cnc}\ rl$ as above, we write: $\mathsf{theXXs}\ rl$, for the variable list $[XX_0, \ldots, XX_{n-1}]$; $\mathsf{theYs}\ rl$, for the variable list $[Y_0, \ldots, Y_{n-1}]$; $\mathsf{theS}\ rl$, for the term S; $\mathsf{theT}\ rl$, for the term T.

A rule rl is said to be *sane* if the following hold:

-**(1)** $\mathsf{theYs}\ rl$ is nonrepetitive;
-**(2)** $\mathsf{set}(\mathsf{theXXs}\ rl) \subseteq \mathsf{vars}(\mathsf{theS}\ rl)$;
-**(3)** $\mathsf{vars}(\mathsf{theS}\ rl) \cap \mathsf{set}(\mathsf{theYs}\ rl) = \emptyset$;
-**(4)** $\mathsf{vars}(\mathsf{theT}\ rl) \subseteq \mathsf{vars}(\mathsf{theS}\ rl) \cup \mathsf{set}(\mathsf{theYs}\ rl)$;
-**(5)** $\forall as, as'.\ (\forall i < \mathsf{length}(\mathsf{theYs}\ rl).\ as\ i = as'\ i) \longrightarrow \mathsf{side}\ rl\ as = \mathsf{side}\ rl\ as'$.

A rule rl is said to be *amenable* if $\mathsf{theS}\ rl$ has the form $\mathsf{Op}\ f\ ps\ [\mathsf{Var}\ X_0, \ldots, \mathsf{Var}\ X_{m-1}]$, where f is an opsym, ps a list of parameters, and $[X_0, \ldots, X_{m-1}]$ a *nonrepetitive* list of variables. Given an amenable rule rl as above, we write $\mathsf{thef}\ rl$ for f, $\mathsf{theps}\ rl$ for ps, and $\mathsf{theXs}\ rl$ for $[X_0, \ldots, X_{m-1}]$.

Saneness expresses a natural property for well-behaved SOS rules: Think of a term S as a generic composite process, built from its unspecified components (its variables) by means of opsyms. Then a sane rule is one that describes the behavior of the composite S in terms of the behavior of (some of) its components: condition (2) says that indeed the hypotheses refer to the components, (1) and (3) that the hypotheses only assume that some components transit "somewhere" (without any further information), (4) that the resulted continuation of the composite depends only on the components and their continuations, and (5) that the side-condition may only depend on the action labels of the hypotheses and of the conclusion. In addition, amenability asks that the composite process S be obtained by a primitive operation f applied to unspecified components. The conjunction of saneness and amenability is precisely the de Simone format requirement [11], hence we call a rule *de Simone* if it is sane and amenable.

Running example. We show what the example in the introduction becomes under our representation. Assume that **act** is an unspecified type with constants $^{-} :: \textbf{act} \Rightarrow \textbf{act}$ and $\tau :: \textbf{act}$ such that $\overline{\overline{a}} = a$ for all $a \neq \tau$. Define the relation $\mathsf{sync} :: \textbf{act} \Rightarrow \textbf{act} \Rightarrow \textbf{act} \Rightarrow \textbf{bool}$ by $\mathsf{sync}\ a\ b\ c = (a \neq \tau \wedge b \neq \tau \wedge \overline{a} = b \wedge c = \tau)$. We take **opsym** to be a three-element datatype $\mathsf{Pref} \mid \mathsf{Par} \mid \mathsf{Repl}$ and **param** to be **act**. For readability, in our running example (including throughout the future continuations of this example), for all $X :: \textbf{var}$, $S, T :: \textbf{term}$ and $a :: \textbf{act}$, we use the following abbreviations: X for $\mathsf{Var}\ X$; $a.S$ for $\mathsf{Op\ Pref}\ [a]\ [S]$; $S \mid T$ for $\mathsf{Op\ Par}\ []\ [T, S]$; $!\,S$ for $\mathsf{Op\ Repl}\ []\ [S]$.

Rls consists of the rules $\{\mathsf{PREF}_a.\ a :: \textbf{act}\} \cup \{\mathsf{PARL}, \mathsf{PARR}, \mathsf{PARS}, \mathsf{REPL}, \mathsf{REPLS}\}$ listed below, where X, Y, X_0, X_1, Y_0, Y_1 are fixed *distinct* variables.

$$\frac{\cdot}{a.X \overset{b}{\rightsquigarrow} X}\,(\mathsf{PREF}_a)\,[a = b] \qquad \frac{X_0 \overset{as\,0}{\rightsquigarrow} Y_0}{X_0 \mid X_1 \overset{b}{\rightsquigarrow} Y_0 \mid X_1}\,(\mathsf{PARL})\,[as\,0 = b]$$

$$\frac{X_0 \overset{as\,0}{\rightsquigarrow} Y_0}{X_1 \mid X_0 \overset{b}{\rightsquigarrow} X_1 \mid Y_0}\,(\mathsf{PARR})\,[as\,0 = b] \qquad \frac{X_0 \overset{as\,0}{\rightsquigarrow} Y_0 \qquad X_1 \overset{as\,1}{\rightsquigarrow} Y_1}{X_0 \mid X_1 \overset{b}{\rightsquigarrow} Y_0 \mid Y_1}\,(\mathsf{PARS})\,[\mathsf{sync}\,(as\,0)\,(as\,1)\,b]$$

$$\frac{X \overset{as\,0}{\rightsquigarrow} Y}{!X \overset{b}{\rightsquigarrow} !X \mid Y}\,(\mathsf{REPL})\,[as\,0 = b] \qquad \frac{X \overset{as\,0}{\rightsquigarrow} Y_0 \qquad X \overset{as\,1}{\rightsquigarrow} Y_1}{!X \overset{b}{\rightsquigarrow} !X \mid (Y_0 \mid Y_1)}\,(\mathsf{REPLS})\,[\mathsf{sync}\,(as\,0)\,(as\,1)\,b]$$

For listing the rules, we employed the previously discussed visual representation. E.g., the formal description of PARS is $(\!|\ \mathsf{hyps} = [X_0 \rightsquigarrow Y_0,\ X_1 \rightsquigarrow Y_1];\ \mathsf{cnc} = (X_0 \mid X_1 \rightsquigarrow Y_0 \mid Y_1);\ \mathsf{side} = (\lambda\ as, b.\ \mathsf{sync}\,(as\,0)\,(as\,1)\,b)\ |\!)$. All the rules in this example are easily seen to be de Simone.

Rules semantics. We fix *Rls*, a set of *de Simone* rules. The one-step transition relation induced by *Rls* on terms is a (curried) ternary relation $\mathsf{step} :: \textbf{term} \Rightarrow \textbf{act} \Rightarrow \textbf{term} \Rightarrow \textbf{bool}$, where we write $P \overset{a}{\rightsquigarrow} Q$ instead of $\mathsf{step}\ P\ a\ Q$, defined inductively by the following clause:

- if $rl \in Rls$, $\sigma((\mathsf{theXXs}\ rl)!j) \overset{as\,j}{\rightsquigarrow} \sigma((\mathsf{theYs}\ rl)!j)$ for all $j < \mathsf{length}(\mathsf{theYs}\ rl)$, and $\mathsf{side}\ rl\ as\ b$ holds, then $(\mathsf{theS}\ rl)[\sigma] \overset{b}{\rightsquigarrow} (\mathsf{theT}\ rl)[\sigma]$

(where $\sigma :: \textbf{var} \Rightarrow \textbf{term}$, $as :: \textbf{nat} \Rightarrow \textbf{act}$, and $b :: \textbf{act}$).

The above definition is the expected one: each (generic) rule in *Rls* yields, for each substitution of the variables in the rule and for each choice of the actions fulfilling the side-condition, an inference of the instance of the rule's conclusion from the instances of the rule's hypotheses.

Bisimilarity. We write **rel** for (**term** $\times$ **term**) <u>**set**</u>, the type of relations between terms, ranged over by θ, η, ξ. The (monotonic) *retract* operator $\mathsf{Retr} :: \mathbf{rel} \Rightarrow \mathbf{rel}$, named so because it maps each θ to a relation retracted (w.r.t. transitions) back from θ, is defined by: $\mathsf{Retr}\ \theta = \{(P, Q).\ (\forall a, P'.\ P \overset{a}{\rightsquigarrow} P' \longrightarrow (\exists Q'.\ (P', Q') \in \theta \wedge Q \overset{a}{\rightsquigarrow} Q'\ )) \wedge (\forall a, Q'.\ Q \overset{a}{\rightsquigarrow} Q' \longrightarrow (\exists P'.\ (P', Q') \in \theta \wedge P \overset{a}{\rightsquigarrow} P'\ ))\}$. The *bisimilarity* relation, $\mathsf{bis} :: \mathbf{rel}$, is the *greatest fixed point* of Retr.

Notice that we defined bisimilarity for *open* terms (i.e., terms possibly containing variables), while often in the literature both transition and bisimilarity are defined for *closed* terms only (with step and Retr defined by the same conditions as above, but acting on closed terms and relations on closed terms, respectively). However, for the de Simone format of our rules (as well as for more general formats, e.g., well-founded pure tyft [15]), transition does *not* bring any variables (in the sense that, if $P \overset{a}{\rightsquigarrow} P'$, then the free variables of P are among those of P') implying that two closed terms are bisimilar according to our definition iff they are bisimilar according to the aforementioned "closed" version.

Because of the particular format of the rules, bis is a congruence on terms. This is in fact true for rule formats more expressive than the one considered here [6,15,32]. However, we shall need to exploit a stronger property specific to the de Simone format, namely: whenever θ is a congruence, it follows that $\theta \cap (\mathsf{Retr}\ \theta)$ is also a congruence. Let, for any relation θ, $\mathsf{congCl}\ \theta$ be its congruence closure. From the above, we infer a powerful "up to" coinduction rule (that is, up to bisimilarity and up to arbitrary contexts), due to de Simone [11] and Sangiorgi [34], improving on traditional coinduction:

Theorem 1. *For all* $\theta :: \mathbf{rel}$*, if* $\theta \subseteq \mathsf{Retr}(\mathsf{congCl}(\theta \cup \mathsf{bis}))$*, then* $\theta \subseteq \mathsf{bis}$*.*

3 The Raw Coinductive Proof System

We now present the core of our original theoretical contribution: defining an incrementally-coinductive proof system for bisimilarity and proving it sound. We define the *raw deduction* relation $\Vdash\ :: \mathbf{rel} \Rightarrow \mathbf{rel} \Rightarrow \mathbf{bool}$ (with infix notation) inductively by the clauses:

$$\frac{\cdot}{\theta \Vdash \theta'}(\mathsf{Ax})\,[\theta' \subseteq \mathsf{congCl}(\theta \cup \mathsf{bis})] \qquad \frac{\forall \theta' \in \Theta.\ \theta \Vdash \theta'}{\theta \Vdash \bigcup \Theta}(\mathsf{Split})\,[\Theta \neq \emptyset] \qquad \frac{\theta' \cup \theta \Vdash \theta''}{\theta \Vdash \theta'}(\mathsf{Coind})\,[\theta' \subseteq \mathsf{Retr}\ \theta'']$$

$\theta \Vdash \theta'$ is eventually intended to mean: "θ implies θ' modulo bisimilarity and congruence closure". Here is the intuitive reading of the rules (thinking of them as being applied backwards for expanding or discharging goals). (Ax) allows to deduce θ' from θ right away. (Split) allows for splitting the goal according to a chosen partition of its conclusion. (Coind) is the interesting rule, and is the actual engine of the proof system. To get an intuitive grasp of this rule, let us first assume that $\theta = \emptyset$ (i.e., that θ is empty). Then the goal is to show θ' included in

$\mathsf{congCl}\ \mathsf{bis}$, i.e., in bis. For this, it would suffice that $\theta' \subseteq \mathsf{Retr}(\theta')$; alternatively, we may "defer" the goal by coming up with an "interpolant" θ'' such that $\theta' \subseteq \mathsf{Retr}(\theta'')$ and θ' implies θ'' modulo bisimilarity and congruence. (As we shall see in the next section, working symbolically with open terms provides natural interpolant candidates.) In case $\theta \neq \emptyset$, θ should be thought of *temporally* as the collection of auxiliary facts gathered from previous coinductive expansions.

Note that, for the aforementioned intention of the proof system, (Coind) is not sound *by itself*: regarded as applied backwards to a goal, it moves the conclusion θ' to the hypotheses, creating a circularity. In other words, of course it is not true that the conjunction of $\theta'' \subseteq \mathsf{congCl}(\theta' \cup \theta \cup \mathsf{bis})$ and $\theta' \subseteq \mathsf{Retr}\ \theta''$ implies $\theta' \subseteq \mathsf{congCl}(\theta \cup \mathsf{bis})$ for all $\theta, \theta', \theta''$. Yet, *the proof system as a whole* is sound in the following sense:

Theorem 2. *If* $\emptyset \Vdash \theta$, *then* $\theta \subseteq \mathsf{bis}$.

In the remainder of this section, we outline the proof of this theorem.

(I) In order to gain more control on the proof system, we *objectify* it in a standard fashion, by considering proofs (i.e., proof trees) explicitly, at the object level (as opposed to merely implicitly as they appear in the inductive definition of $\Vdash$). For this, we pick a sufficiently large type **index**, ranged over by i, and define the type **prf**, of *proof trees*, ranged over by Pf, with constructors mirroring the clauses in the definition of $\Vdash$:

DATATYPE **prf** = Ax **rel rel** | Split (**index** $\Rightarrow$ **prf**) **rel rel** | Coind **prf rel rel**

We let Pfs range over **index** $\Rightarrow$ **prf**. The pair of relations that a proof tree Pf "proves", which is (θ, θ') when Pf has the one of the forms $\mathsf{Ax}\ \theta\ \theta'$, $\mathsf{Split}\ Pfs\ \theta\ \theta'$, or $\mathsf{Coind}\ Pf\ \theta\ \theta'$, is denoted by $\mathsf{proves}\ Pf$. The conclusion-hypothesis dependencies and the side-conditions of the clauses defining $\Vdash$ are captured by the predicate $\mathsf{correct} :: \mathbf{prf} \Rightarrow \mathbf{bool}$, defined recursively as expected:

- $\mathsf{correct}\,(\mathsf{Ax}\ \theta\ \theta') = (\theta' \subseteq \mathsf{congCl}(\theta \cup \mathsf{bis}))$;
- $\mathsf{correct}\,(\mathsf{Split}\ Pfs\ \theta\ \theta') =$
$((\forall i.\ \mathsf{correct}(Pfs\ i) \wedge \mathsf{fst}(\mathsf{proves}(Pfs\ i)) = \theta) \wedge \bigcup i.\ \mathsf{snd}(\mathsf{proves}(Pfs\ i)) = \theta')$;
- $\mathsf{correct}\,(\mathsf{Coind}\ Pf\ \theta\ \theta') =$
$(\mathsf{correct}\,Pf \wedge \mathsf{fst}(\mathsf{proves}\,Pf) = \theta' \cup \theta \wedge \theta' \subseteq \mathsf{Retr}(\mathsf{snd}(\mathsf{proves}\,Pf)))$.

It is immediate that $\theta \Vdash \theta'$ holds iff $\exists Pf.\ \mathsf{correct}(Pf) \wedge \mathsf{proves}(Pf) = (\theta, \theta')$.

(II) Thus, it suffices to show that $\theta \subseteq \mathsf{bis}$ whenever there exists a correct proof tree Pf such that $\mathsf{proves}(Pf) = (\theta, \theta')$. For showing the latter, we introduce a couple of auxiliary concepts. Given Pf, a *label* in Pf is a pair (θ, θ') "appearing" in Pf – formally, we define $\mathsf{labels} :: \mathbf{prf} \Rightarrow (\mathbf{rel} \times \mathbf{rel})\ \underline{\mathbf{set}}$ by:

- $\mathsf{labels}\,(\mathsf{Ax}\ \theta\ \theta') = \{(\theta, \theta')\}$;
- $\mathsf{labels}\,(\mathsf{Split}\ Pfs\ \theta\ \theta') = \{(\theta, \theta')\} \cup \bigcup i.\ \mathsf{labels}(Pfs\ i)$;
- $\mathsf{labels}\,(\mathsf{Coind}\ Pf\ \theta\ \theta') = \{(\theta, \theta')\} \cup \mathsf{labels}\,Pf$.

We let $\mathsf{Left}\,Pf$ denote the union of the lefthand sides of all labels in Pf, and $\mathsf{Right}\,Pf$ the union of the righthand sides of all labels in Pf.

Lemma 1. *If Pf is correct, then* $\mathsf{Right}\ Pf \subseteq \mathsf{congCl}((\mathsf{Left}\ Pf) \cup \mathsf{bis})$.

Lemma 2. *If Pf is correct and* $\mathsf{fst}(\mathsf{proves}\ Pf) \subseteq \mathsf{Retr}(\mathsf{Right}\ Pf)$, *then* $\mathsf{Left}\ Pf \subseteq \mathsf{Retr}(\mathsf{Right}\ Pf)$.

Lemma 1 follows by an easy induction on proof trees. By contrast, Lemma 2 requires some elaboration – before getting into that, let us show how the two lemmas imply our desired fact. Assume that Pf is correct and $\mathsf{proves}\ Pf = (\emptyset, \theta)$. Then the hypotheses of both lemmas are satisfied by Pf, and therefore (since also Retr is monotonic) $\mathsf{Left}\ Pf \subseteq \mathsf{Retr}(\mathsf{Right}\ Pf) \subseteq \mathsf{Retr}(\mathsf{congCl}((\mathsf{Left}\ Pf) \cup \mathsf{bis}))$, implying, by Theorem 1, $\mathsf{Left}\ Pf \subseteq \mathsf{bis}$. With Lemma 1, we obtain $\mathsf{Right}\ Pf \subseteq \mathsf{congCl}(\mathsf{bis})$, which means (given that bis is a congruence) $\mathsf{Right}\ Pf \subseteq \mathsf{bis}$. And since $\theta \subseteq \mathsf{Right}\ Pf$, we obtain $\theta \subseteq \mathsf{bis}$, as desired.

It remains to prove Lemma 2. This lemma states a property of proof trees that depends on a hypothesis concerning their roots (i.e., the pair (θ, θ') that they "prove"). The task of finding a strengthening of that hypothesis so that a direct proof by structural induction goes through seems rather difficult, if not impossible. We instead take the roundabout route of identifying an invariant satisfied on backwards paths in the proof trees whose roots satisfy our hypothesis. First, we define the notion of a *path* (independently of proof trees): a list $[(\theta_0, \theta'_0), \ldots, (\theta_{m-1}, \theta'_{m-1})]$ is called a *path* if the following is true for all $n < m-1$: either $\theta_{n+1} = \theta_n$, or $\theta_{n+1} \subseteq \mathsf{Retr}(\theta'_{n+1}) \cup \theta_n$. Then one can verify the following:
-(a) Fix ξ :: **rel**. If $[(\theta_0, \theta'_0), \ldots, (\theta_{m-1}, \theta'_{m-1})]$ is a path, $\theta_0 \subseteq \mathsf{Retr}\ \xi$ and $\forall n < m.\, \theta'_n \subseteq \xi$, then $\forall n < m.\, \theta_n \subseteq \mathsf{Retr}\ \xi$. (By easy induction on n.)
-(b) If Pf is correct, $\mathsf{proves}(Pf) = (\theta, \theta')$, and (η, η') is a label in Pf, then there exists a path $[(\theta_0, \theta'_0), \ldots, (\theta_{m-1}, \theta'_{m-1})]$ consisting of labels in Pf (i.e., such that (θ_n, θ'_n) are labels in Pf for all $n < m$) and connecting (θ, θ') with (η, η') (i.e., such that $(\theta_0, \theta'_0) = (\theta, \theta')$ and $(\theta_{m-1}, \theta'_{m-1}) = (\eta, \eta')$). (By induction on Pf.)

With these preparations, we can prove Lemma 2: Assume $\mathsf{proves}(Pf) = (\theta, \theta')$ and $\theta \subseteq \mathsf{Retr}(\mathsf{Right}\ Pf)$. Fix a label (η, η') in Pf. According to (b), there exists a path connecting (θ, θ') with (η, η') and going through labels in Pf only. Then the hypotheses of (a) are satisfied by the aforementioned path and $\xi = \mathsf{Right}\ Pf$, and therefore all the lefthand sides of the pairs in this path are included in $\mathsf{Retr}(\mathsf{Right}\ Pf)$. In particular, $\eta \subseteq \mathsf{Retr}(\mathsf{Right}\ Pf)$. Since the choice of the label (η, η') was arbitrary, it follows that $\mathsf{Left}\ Pf \subseteq \mathsf{Retr}(\mathsf{Right}\ Pf)$, as desired.

Remarks. (1) The soundness of $\Vdash$ was established not locally (rule-wise), as is customary in soundness results, but globally, by analyzing entire proof trees. What the potential backwards applications of the clause (Coind) do is to *improve the candidate relation for the coinductive argument.* In the end, as shown by the proof of Theorem 2, the (successful) relation is synthesized by putting together the righthand sides of all labels in the proof tree.

(2) The proof system represented by $\Vdash$ is not a typical syntactic system, but contains semantic intrusions – in effect, the system is complete already by its axiom (Ax), which allows for an instantaneous "oracle proof" that the considered relation is included in bisimilarity. But of course, the realistic employment of this system will appeal to such instantaneous proofs only through the available (already proved) lemmas. (Thus, the purpose of including bis in the side-condition of (Ax) was not to ensure completeness (in such a trivial manner), but to allow the usage of previously known facts about bisimilarity.) A more syntactic and syntax-driven system for terms (also featuring oracles though, for the same reason as this one) will be presented in the next section.

4 Deduction of Universally Quantified Bisimilarity Equations

Next we introduce a deduction system for term equalities, where, as before, we interpret equality as bisimilarity, but now we interpret the occurring variables as being *universally quantified over the domain of terms.*

Universal bisimilarity, $\mathsf{ubis} :: \mathbf{rel}$, is defined as follows: $(U, U') \in \mathsf{ubis}$ iff $(U[\tau], U'[\tau]) \in \mathsf{bis}$ for all substitutions $\tau :: \mathbf{var} \Rightarrow \mathbf{term}$. Thus, e.g., given distinct variables X and Y and an opsym f, $(\mathsf{Op}\ f\ []\ [\mathsf{Var}\ X, \mathsf{Var}\ Y], \mathsf{Op}\ f\ []\ [\mathsf{Var}\ Y, \mathsf{Var}\ X])$ $\in \mathsf{ubis}$ is equivalent to $\forall U, V :: \mathbf{term}.\ (\mathsf{Op}\ f\ []\ [U, V], \mathsf{Op}\ f\ []\ [V, U]) \in \mathsf{bis}$.

Matched derived rules. Derived rules appear by composing primitive rules (i.e., the de Simone rules in *Rls*) within maximal composition chains. I.e., they come from considering, in the SOS system, derivation trees that are completely backwards-saturated (in that their leaves involve only variables as sources and targets) and then forgetting the intermediate steps in these trees. A derived rule may not be amenable (hence not de Simone), but will always be sane. We shall let *drl* denote derived rules, keeping the symbol *rl* for primitive rules.

We are interested in constructing all derived rules that are matched by a given term U in such a way that U becomes the source of the conclusion of the derived rule; in doing so, we also care about avoiding any overlap between the freshly generated variables (required to build the rules) and the variables of another given term V (that we later wish to prove universally bisimilar with U). We thus introduce the operator $\mathsf{mdr} :: \mathbf{term} \Rightarrow \mathbf{term} \Rightarrow \mathbf{rule}\ \underline{\mathbf{set}}$, read "matched derived rules", such that, given $U, V :: \mathbf{term}$, $\mathsf{mdr}\ V\ U$ is the set of all the derived rules with U as the source of their conclusion and with "the Ys" fresh for V. We write $\mathsf{mdr}_V\, U$ instead of $\mathsf{mdr}\ V\ U$.

The definition of mdr is both intuitive and standard (and was already sketched in the pioneering paper [11]), but its formalities are very technical, due to the need to avoid name overlapping and compose side-conditions. Here, we count on its understanding by examples and by its abstract properties, but App. A in [28] gives the general definition. (In [6,4], where what we call "matched derived rules" are called "ruloids", mdr is not even defined, but rather the existence of such an operator satisfying suitable properties (essentially the same with what we call below soundness and completeness of the matched derived rules) is proved.)

Running example (continued). We again assume that all the variables X, Y etc. that we refer to below are *fixed distinct variables.*
- $\mathsf{mdr}_{!X}(X \mid !X)$, the set of derived rules matched by $X \mid !X$ and with "the Ys" avoiding the variables of $!X$, consists of $\{\mathsf{DRL}_1, \mathsf{DRL}_2, \mathsf{DRL}_3, \mathsf{DRL}_4\}$ (see below);
- $\mathsf{mdr}_{X \mid !X}(!X)$, the set of derived rules matched by $!X$ and with "the Ys" avoiding the variables of $X \mid !X$, consists of $\{\mathsf{DRL}_5, \mathsf{DRL}_6\}$ (given below).

$$\frac{X \overset{as\,0}{\rightsquigarrow} Y}{X \mid !X \overset{b}{\rightsquigarrow} Y \mid !X}\,(\mathsf{DRL}_1)\,[as\ 0 = b] \qquad \frac{X \overset{as\,0}{\rightsquigarrow} Y}{X \mid !X \overset{b}{\rightsquigarrow} X \mid (!X \mid Y)}\,(\mathsf{DRL}_2)\,[\exists c.\ as\ 0 = c \wedge c = b]$$

$$\frac{X \overset{as\,0}{\rightsquigarrow} Y_0 \qquad X \overset{as\,1}{\rightsquigarrow} Y_1}{X \mid !X \overset{b}{\rightsquigarrow} X \mid (!X \mid (Y_0 \mid Y_1))}\,(\mathsf{DRL}_3)\left[\exists c.\ \begin{array}{l}\mathsf{sync}\,(as\,0)\,(as\,1)\,c\\ \wedge\ c = b\end{array}\right] \qquad \frac{X \overset{as\,0}{\rightsquigarrow} Y_0 \qquad X \overset{as\,1}{\rightsquigarrow} Y_1}{X \mid !X \overset{b}{\rightsquigarrow} Y_0 \mid (!X \mid Y_1)}\,(\mathsf{DRL}_4)\left[\exists c.\ \begin{array}{l}as\ 1 = c\ \wedge\\ \mathsf{sync}\,(as\,0)\,c\,b\end{array}\right]$$

$$\frac{X \overset{as\,0}{\rightsquigarrow} Y}{!X \overset{b}{\rightsquigarrow} !X \mid Y}\,(\mathsf{DRL}_5)\,[as\ 0 = b] \qquad \frac{X \overset{as\,0}{\rightsquigarrow} Y_0 \qquad X \overset{as\,1}{\rightsquigarrow} Y_1}{!X \overset{b}{\rightsquigarrow} !X \mid (Y_0 \mid Y_1)}\,(\mathsf{DRL}_6)\,[\mathsf{sync}\,(as\,0)\,(as\,1)\,b]$$

Remarks. (1) Because the term $!X$ has only depth 1, the matched derived rules $\mathsf{DRL}_5, \mathsf{DRL}_6$ are essentially the primitive rules $\mathsf{REPL}, \mathsf{REPLS}$. Moreover, DRL_1 was obtained by a single (backwards) application of the rule PARL.

(2) Each of $\mathsf{DRL}_2, \mathsf{DRL}_3, \mathsf{DRL}_4$ arises from the composition of two primitive rules. For example, DRL_3 is obtained by applying PARR, and then applying REPLS to the resulted hypothesis:

$$\frac{\dfrac{X \overset{as\,0}{\rightsquigarrow} Y_0 \qquad X \overset{as\,1}{\rightsquigarrow} Y_1}{!X \overset{c}{\rightsquigarrow} !X|(Y_0 \mid Y_1)}\,(\mathsf{REPLS})\,[\mathsf{sync}\,(as\,0)\,(as\,1)\,c]}{X \mid !X \overset{b}{\rightsquigarrow} X \mid (!X \mid (Y_0 \mid Y_1))}\,(\mathsf{PARR})\,[c = b]$$

The side-condition of DRL_3 is obtained by composing (essentially as relations) the two side-conditions, of PARR and REPLS, yielding existential quantification over c. Of course, the side-conditions of $\mathsf{DRL}_2, \mathsf{DRL}_3, \mathsf{DRL}_4$ can be readily simplified to the equivalent forms $as\ 0 = b$, $\mathsf{sync}\,(as\,0)\,(as\,1)\,b$ and again $\mathsf{sync}\,(as\,0)\,(as\,1)\,b$, but eliminating the existential quantifiers may not be possible in general – recall that side-conditions are *arbitrary* predicates.

The only property we care about concerning elements drl of $\mathsf{mdr}_V\,U$ w.r.t. V is that $\mathsf{theYs}(drl)$ are all distinct from the variables of V. On the other hand, concerning the relationship between $\mathsf{mdr}_V\,U$ and U, we have the crucial facts of *soundness* and *completeness* w.r.t. transition:
- For all $drl \in \mathsf{mdr}_V\,U$, drl is *sound*, i.e.: for all τ :: **var** $\Rightarrow$ **term**, as :: **nat** $\Rightarrow$ **act**, and b :: **act**, if $\tau((\mathsf{theXXs}\ drl)!j) \overset{as\ j}{\rightsquigarrow} \tau((\mathsf{theYs}\ drl)!j)$ for all $j < \mathsf{length}(\mathsf{theYs}\ drl)$ and $\mathsf{side}\ drl\ as\ b$ holds, then $(\mathsf{theS}\ drl)[\tau] \overset{b}{\rightsquigarrow} (\mathsf{theT}\ drl)[\tau]$.
- $\mathsf{mdr}_V\,U$ is *complete* for inference of transitions with sources that match U, i.e.: for all τ :: **var** $\Rightarrow$ **term**, b :: **act** and Q :: **term** such that $U[\tau] \overset{b}{\rightsquigarrow} Q$, there exist $drl \in \mathsf{mdr}_V\,U$, τ' :: **var** $\Rightarrow$ **term** and as :: **nat** $\Rightarrow$ **act** such that:
— τ' coincides with τ on $\mathsf{vars}\ U$ (hence $U[\tau] = U[\tau']$);
— $\tau'((\mathsf{theXXs}\ drl)!j) \overset{as\ j}{\rightsquigarrow} \tau'((\mathsf{theYs}\ drl)!j)$ for all $j < \mathsf{length}(\mathsf{theYs}\ drl)$;
— $\mathsf{side}\ drl\ as\ b$ holds;
— $(\mathsf{theT}\ drl)[\tau'] = Q$ (and also, remember that $\mathsf{theS}\ drl = U$).

Deduction of universal bisimulation. An *equation* will be simply a pair of terms, written $U \cong V$, and we write **equation** for the type of equations. (Note that **rel** is the same as **equation** <u>**set**</u>.) Our goals will consist of pairs (set of equations) – equation, where all equations shall be thought of as being *universally quantified*. We shall mostly use S, T, U, V for terms thought of as *patterns*, and P, Q, R for terms thought of as *instances*.

Given U, U' :: **term**, $G :: \mathsf{mdr}_{U'}\, U \Rightarrow \mathsf{mdr}_U\, U'$, and $g :: \prod_{drl \in \mathsf{mdr}_{U'}\, U} \{0, \ldots, \mathsf{length}(\mathsf{theXXs}(G\ \ drl)) - 1\} \Rightarrow \{0, \ldots, \mathsf{length}(\mathsf{theXXs}\ \ drl) - 1\}$, we define the predicate $\mathsf{simul}\ U\ U'\ G\ g$, read "$U$ is *(one-step-)simulated* by U' via G and g", to mean that, for all $drl \in \mathsf{mdr}_U\, U'$, the following holds: Assume drl has the form

$$\frac{XX_0 \overset{as\ 0}{\rightsquigarrow} Y_0\ ,\ \ldots\ , XX_{n-1} \overset{as\ (n-1)}{\rightsquigarrow} Y_{n-1}}{S \overset{b}{\rightsquigarrow} T}\ [\mathsf{side}\ drl\ as\ b] \qquad (*)$$

and $drl' = G\ drl$ has the form

$$\frac{XX'_0 \overset{as\ 0}{\rightsquigarrow} Y'_0\ ,\ \ldots\ , XX'_{n'-1} \overset{as\ (n'-1)}{\rightsquigarrow} Y'_{n'-1}}{S' \overset{b}{\rightsquigarrow} T'}\ [\mathsf{side}\ drl'\ as\ b] \qquad (**)$$

(and therefore $g\ drl :: \{0, \ldots, n' - 1\} \Rightarrow \{0, \ldots, n - 1\}$) Then:
- (1) $XX_{g\ drl\ j} = XX'_j$ (i.e., syntactically equal, as variables) for all $j < n'$.
- (2) $\forall as :: \mathbf{nat} \Rightarrow \mathbf{act}, b :: \mathbf{act}.\ \mathsf{side}\ drl\ as\ b \longrightarrow \mathsf{side}\ (G\ drl)\ (as \circ (g\ drl))\ b$.

Given the rules drl, of the form $(*)$, and drl', of the form $(**)$, and given $h :: \{0, \ldots, n' - 1\} \Rightarrow \{0, \ldots, n - 1\}$, we define $\mathsf{newGoal}\ drl\ drl'\ h$ to be the equation $T \cong T'[(Y'_j / Y_{h\,j})_{j<n'}]$, where $(Y'_j / Y_{h\,j})_{j<n'}$ is a substitution that maps each variable Y'_j to the variable $Y_{h\,j}$ (more accurately, to the term $\mathsf{Var}\ Y_{h\,j}$).

simul and newGoal will work in tandem in our deduction system as follows: Given a goal $U \cong U'$, we wish to prove U and U' universally bisimilar. For this, we should show that, for any continuation of an instance of U, there exists a bisimilar continuation of an instance of U' (and vice versa, but next we ignore the "vice versa" part). By the completeness of mdr, any transition of an instance of U is given by a derived rule drl in $\mathsf{mdr}_{U'}\, U$. By the soundness of mdr, for finding a transition of an instance of U' that simulates that of U, it would suffice to find for drl a derived rule in drl' which is possible whenever drl is possible. Thus, we first need a map $G :: \mathsf{mdr}_{U'}\, U \Rightarrow \mathsf{mdr}_U\, U'$ (giving the drl' for each $drl \in \mathsf{mdr}_{U'}\, U$), and then, for each drl, a justification of the possibility of $G\ drl$ in terms of that of drl. Now, possibility of (a transition along) a derived rule is given by its (formal) hypotheses and its side conditions. Hence, a justification of the possibility of $G\,drl$ in terms of the possibility of drl can be given by a map from the hypotheses of $G\,drl$ to those of drl that *preserves the sources* (which are variables) and *yields an implication between the side conditions* – this is formally achieved by a function $g :: \prod_{drl \in \mathsf{mdr}_{U'}\, U} \{0, \ldots, \mathsf{length}(\mathsf{theXXs}(G\ \ drl)) - 1\} \Rightarrow \{0, \ldots, \mathsf{length}(\mathsf{theXXs}\ \ drl) - 1\}$ that, together with G, satisfies the conditions defining $\mathsf{simul}\ U\ U'\ G\ g$. Moreover, we have to prove that, for each combination $(drl, G\ \ drl)$, the resulted continuations of the presumptive instances of U and U' are again bisimilar – we obtain a $\mathsf{newGoal}\ drl\ (G\ drl)\ (g\ drl)$ for each such combination (note that generating this new goal has to take into consideration the dispatching of formal hypotheses performed by $g\ drl$, meaning that we also have to substitute some "Ys"). Finally, the incremental nature of our coinduction (inherited from the previous section) shows up: for proving each of the new goals, we may *assume* the old goal, $U \cong U'$.

We are led to the deduction relation $\vdash$ :: **equation** $\underline{\textbf{set}}$ $\Rightarrow$ **equation** $\Rightarrow$ **bool** (with infix notation), defined inductively by the following clauses:

$$\frac{\cdot}{\theta \vdash U \cong U'}\,(\mathsf{Eqnl})\,[\theta \cup \mathsf{bis} \vdash_{\mathrm{eq}} U \cong U']$$

$$\frac{\begin{array}{c}\forall drl \in \mathsf{mdr}_{U'}\, U.\ \theta \cup \{U \cong U'\} \vdash \mathsf{newGoal}\ drl\ (G\ drl)\ (g\ drl)\\ \forall drl' \in \mathsf{mdr}_{U}\, U'.\ \theta \cup \{U \cong U'\} \vdash \mathsf{newGoal}\ drl'\ (G'\ drl')\ (g'\ drl')\end{array}}{\theta \vdash U \cong U'}\,(\mathsf{Coind})\left[\begin{array}{l}\mathsf{simul}\ U\ U'\ G\ g\\ \mathsf{simul}\ U'\ U\ G'\ g'\end{array}\right]$$

In the side-condition at (Eqnl), $\vdash_{\mathrm{eq}}$ is standard equational-logic deduction. We include bis among the hypotheses, because we wish to allow any known facts about bisimilarity to "help" $\vdash$-deduction, including facts obtained by means other than $\vdash$. Again, due to circularity (moving goals to the hypotheses), a rule like (Coind) cannot be sound in itself, but again we have global soundness:

Theorem 3. *If* $\emptyset \vdash U \cong U'$, *then* $(U, U') \in \mathsf{ubis}$.

Proof sketch. We use the soundness of $\Vdash$ (Theorem 1) together with the rules defining $\vdash$ being simulated by those defining $\Vdash$. Namely, we show, by induction on $\vdash$, that $\theta \vdash U \cong U'$ implies $\mathsf{sstvsmCl}(\theta) \Vdash \mathsf{sstvsmCl}(\{(U, U')\})$, where $\mathsf{sstvsmCl} :: \mathbf{rel} \Rightarrow \mathbf{rel}$ gives the *substitutive and symmetric closure of a relation*, i.e., $\mathsf{sstvsmCl}(\xi) = \{(S[\sigma], T[\sigma]).\ \sigma :: \mathbf{var} \Rightarrow \mathbf{term},\ (S,T) \in \xi\ \vee\ (T,S) \in \xi\}$.

If $\theta \vdash U \cong U'$ followed by an application of (Eqnl), then $\mathsf{sstvsmCl}(\theta) \Vdash \mathsf{sstvsmCl}(\{(U, U')\})$ follows applying the $\Vdash$-clause (Ax), since the equational closure coincides with the substitutive symmetric closure of the congruence closure.

Assume now $\theta \vdash U \cong U'$ followed by (Coind), meaning that there exist G, g, G', g' such that: **(i)** $\mathsf{simul}\ U\ U'\ G\ g$; **(ii)** $\forall drl \in \mathsf{mdr}_{U'}\, U.\ \theta \cup \{U \cong U'\} \vdash \mathsf{newGoal}\ drl\ (G\ drl)\ (g\ drl)$; **(iii)** $\mathsf{simul}\ U'\ U\ G'\ g'$; **(iv)** $\forall drl' \in \mathsf{mdr}_{U}\, U'.\ \theta \cup \{U \cong U'\} \vdash \mathsf{newGoal}\ drl'\ (G'\ drl')\ (g'\ drl')$. Then, by the induction hypothesis:
- $\forall drl \in \mathsf{mdr}_{U'}\, U.\ \mathsf{sstvsmCl}(\theta \cup \{U \cong U'\}) \Vdash \mathsf{sstvsmCl}(\{\mathsf{newGoal}\ drl\ (G\ drl)\ (g\ drl)\})$.
- $\forall drl' \in \mathsf{mdr}_{U}\, U'.\ \mathsf{sstvsmCl}(\theta \cup \{U \cong U'\}) \Vdash \mathsf{sstvsmCl}(\mathsf{newGoal}\ drl'\ (G'\ drl')\ (g'\ drl'))$.

Let $\theta' = \mathsf{sstvsmCl}(\{(U, U')\})$ and let $\theta'' = \{\mathsf{newGoal}\ drl\ (G\ drl)\ (g\ drl).\ drl \in \mathsf{mdr}_{U'}\, U\} \cup \{\mathsf{newGoal}\ drl'\ (G'\ drl')\ (g'\ drl').\ drl' \in \mathsf{mdr}_{U}\, U'\}$. The crucial thing to notice is that, since $\mathsf{simul}\ U\ U'\ G\ g$ and $\mathsf{simul}\ U'\ U\ G'\ g'$ hold, $\mathsf{sstvsmCl}(\{(U, U')\}) \subseteq \mathsf{Retr}\ \theta'$ also holds – and the paragraph right before introducing $\vdash$ can be regarded as an informal justification for why this is true. Therefore, θ'' is an "interpolant" for applying the $\Vdash$-clause (Coind). Indeed, applying the $\Vdash$-clause (Split) to (1) and (2), we obtain $\theta' \cup \theta \Vdash \theta''$ and then, by the $\Vdash$-clause (Coind), we obtain $\theta \Vdash \theta'$, as desired. □

Running example (finished). We are now ready to make rigorous the proof of $\forall P :: \mathbf{term}.\ (P|!P, !P) \in \mathsf{bis}$ presented in the introduction. Consider the following four proof trees of depth 0 (later referred to as Pf_1, Pf_2, Pf_3, Pf_4) where we list the side-conditions for (Eqnl) as hypotheses:

$$\frac{\{X|!X \cong !X\} \cup \mathsf{bis} \vdash_{\mathrm{eq}} Y|!X \cong !X|Y}{\{X|!X \cong !X\} \vdash Y|!X \cong !X|Y}(\mathsf{Eqnl})$$
$$\frac{\{X|!X \cong !X\} \cup \mathsf{bis} \vdash_{\mathrm{eq}} X|(!X|Y) \cong !X|Y}{\{X|!X \cong !X\} \vdash X|(!X|Y) \cong !X|Y}(\mathsf{Eqnl})$$
$$\frac{\{X|!X \cong !X\} \cup \mathsf{bis} \vdash_{\mathrm{eq}} X|(!X|(Y_0|Y_1)) \cong !X|(Y_0|Y_1)}{\{X|!X \cong !X\} \vdash X|(!X|(Y_0|Y_1)) \cong !X|(Y_0|Y_1)}(\mathsf{Eqnl})$$
$$\frac{\{X|!X \cong !X\} \cup \mathsf{bis} \vdash_{\mathrm{eq}} Y_0|(!X|Y_1) \cong !X|(Y_0|Y_1)}{\{X|!X \cong !X\} \vdash Y_0|(!X|Y_1) \cong !X|(Y_0|Y_1)}(\mathsf{Eqnl})$$

Then our final proof (tree) is:

$$\frac{Pf_1 \quad Pf_2 \quad Pf_3 \quad Pf_4}{\emptyset \vdash X|!X \cong !X}(\mathsf{Coind})$$

Explanations. At (Coind), we took:
- G to map DRL_1 and DRL_2 to DRL_5, and to map DRL_3 and DRL_4 to DRL_6;
- g DRL_1 and g DRL_2 to be the identity on $\{0\}$, and g DRL_3 and g DRL_4 to be the identity on $\{0, 1\}$;
- G' to map DRL_5 to DRL_1, and to map DRL_6 to DRL_3;
- g' DRL_5 to be the identity on $\{0\}$, and g' DRL_6 to be the identity on $\{0, 1\}$.
(Note that any function G' mapping DRL_5 to either DRL_1 or DRL_2 and DRL_6 to either DRL_3 or DRL_4 together with g' as above would lead to a valid proof.)

Here is why we end up with the above four proof tasks after applying (Coind):

$\mathsf{newGoal}\,\mathsf{DRL}_1(G\,\mathsf{DRL}_1)(g\,\mathsf{DRL}_1) = \mathsf{newGoal}\,\mathsf{DRL}_1\,\mathsf{DRL}_5(\lambda i.\,i) = Y|!X \cong !X|Y$;
$\mathsf{newGoal}\,\mathsf{DRL}_2(G\,\mathsf{DRL}_2)(g\,\mathsf{DRL}_2) = \mathsf{newGoal}\,\mathsf{DRL}_2\,\mathsf{DRL}_5(\lambda i.\,i) = X|(!X|Y) \cong !X|Y$;
$\mathsf{newGoal}\,\mathsf{DRL}_3(G\,\mathsf{DRL}_3)(g\,\mathsf{DRL}_3) = \mathsf{newGoal}\,\mathsf{DRL}_3\mathsf{DRL}_6(\lambda i.\,i) = X|(!X|(Y_0|Y_1)) \cong !X|(Y_0|Y_1)$;
$\mathsf{newGoal}\,\mathsf{DRL}_4(G\,\mathsf{DRL}_4)(g\,\mathsf{DRL}_4) = \mathsf{newGoal}\,\mathsf{DRL}_4\,\mathsf{DRL}_6(\lambda i.\,i) = Y_0|(!X|Y_1) \cong !X|(Y_0|Y_1)$;
$\mathsf{newGoal}\,\mathsf{DRL}_5(G'\mathsf{DRL}_5)(g'\mathsf{DRL}_5) = \mathsf{newGoal}\,\mathsf{DRL}_5\,\mathsf{DRL}_1(\lambda i.\,i) = Y|!X \cong !X|Y$;
$\mathsf{newGoal}\,\mathsf{DRL}_6(G'\mathsf{DRL}_6)(g'\mathsf{DRL}_6) = \mathsf{newGoal}\,\mathsf{DRL}_6\mathsf{DRL}_3(\lambda i.\,i) = X|(!X|(Y_0|Y_1)) \cong !X|(Y_0|Y_1)$.

The side-conditions of (Coind) are immediately checkable. E.g., for $\mathsf{simul}\ (X|!X)\ (!X)\ G\ g$, we need to check the following trivial facts:
- w.r.t. condition (1) (in the definition of simul): that $X = X$.
- w.r.t. condition (2): that each of the following are pairwise equivalent:
— $as\ 0 = b$ and $as\ 0 = b$;
— $\exists c.\ as\ 0 = c \wedge c = b$ and $as\ 0 = b$;
— $\exists c.\ \mathsf{sync}\,(as\ 0)\,(as\ 1)\,c \wedge c = b$ and $\mathsf{sync}\,(as\ 0)\,(as\ 1)\,b$;
— $\exists c.\ as\ 1 = c \wedge \mathsf{sync}\,(as\ 0)\,c\,b$ and $\mathsf{sync}\,(as\ 0)\,(as\ 1)\,b$.

At (Eqnl) in all the four immediate subtrees of the main proof tree, we considered the fact (assumed previously proved) that $\{X|Y \cong Y|X,\ (X|Y)|Z \cong X|(Y|Z)\} \subseteq \mathsf{bis}$, hence what we really used was equational-logic deduction from $\{X|!X \cong !X,\ X|Y \cong Y|X,\ (X|Y)|Z \cong X|(Y|Z)\}$, which easily discharges the equational side-conditions of the axioms, finalizing the proof.

The above proof does not display any non-trivial "dispatch" function g in the (Coind) rule application. In general however, it is not guaranteed that the formal hypotheses of two obtained derived rules (from the two terms of the goal) that one wishes to pair come in the same order, nor that these rules have the same number of hypotheses. (See the proof of commutativity of "|" from App. B in [28].)

5 Concluding Remarks

We have developed and formalized in Isabelle a proof system for process algebra where bisimilarity is proved *incrementally*, while exploring and expanding the goal, without requiring an a priori constructed bisimulation relation. Our results apply to a wide class of process algebras.

Related work. Unique fixpoint induction for CCS and its variants [23,17,25] is an early notion of proof-theoretic circularity for coinduction applicable to situations where circularity is *explicit* in the SOS by means of (guarded) fixpoint equations. We conjecture that unique fixpoint induction in an instance of our incremental coinduction.

We had two major sources of inspiration. First, the idea of *circular coinduction* (CC for short) in the context of algebraic specifications. It was introduced in [14] in the behavioral specification language BOBJ [1], and then also implemented axiomatically in (generic) Isabelle under the "supervision" of the CoCASL specification language [16] and in Maude [9] as the circular coiductive prover CIRC [20,19,31]. A comparison of our proof system with CC is somewhat difficult to sketch, as it has to deal with different technical settings and to balance the advantages of both generality and specialization. To simplify the discussion, we shall implicitly assume a back-and-forth translation between SOS specifications and the coalgebraic and behavioral specifications required by the CC settings. Our proof system is in a sense more general and in a sense more specialized.

It is more general in that it applies to *nondeterministic processes*, not handled by CC (e.g., the running example in this paper is not approachable in CC, not even interactively). On the other hand, CIRC, based on rewriting logic, could employ the results presented here in order to extend CC with nondeterminism.[2] Also, CoCASL as a specification language has the expressive power required to deal with process algebra and nondeterminism, hence to support a version of CC for nondeterministic systems. However, it is the determinism of CC that allows for partial automation, admirably illustrated by CIRC. (Our formalized system, once fine-tuned into a tool, will also allow automation for deterministic, and, to some extent, finitely-branching cases – see below the discussion on future work.)

It is more specialized in that the *deterministic* instances of our setting are more restricted than what CC can handle (in particular, e.g., deterministic lookahead, not approachable here, is unproblematic in CC). On the other hand, our powerful coinduction "up to", underneath arbitrary contexts (not supported by CC) is possible precisely because of this restriction.

[2] Which is not to say our proof system is a minor variation of CC – nondeterminism (technically, the interplay between our rules (Split) and (Coind) from Sec. 3) represented the main difficulty in our soundness proof.

Finally, our coinductive technique is presented in a logical form, as a *proof system*, like in [31], and not as an *algorithm* like in the other cited works on CC. In [31], logical form is achieved through the introduction of so-called *freezing operators*, hard to justify logically – with this respect, our proof system has the advantage of "purity".[3] (Here we should also remark some less related work: circular systems in logical form were also developed in [7,10] for first-order logic and the μ-calculus, respectively.)

The second major source of inspiration was the notion of coinduction proofs up to bisimilarity and arbitrary contexts, introduced in [11,24] and developed in [34,35]. This idea also appears in a general coalgebraic setting in [5] and is illustrated by extensive examples in, e.g., [33]. The convenience of performing unrestricted equational reasoning relies essentially on the "up to" coinduction principle, Theorem 1.

Other related work includes frameworks for *bisimilarity of open terms* in [29,8,4] (also building on the seminal work from [11]), where open terms are considered universally quantified, as we do in this paper for universal bisimilarity. Our soundness result for $\vdash$ w.r.t. universal bisimilarity, Theorem 3, could have been more sharply phrased: on one hand, as a soundness result w.r.t. the notion of *bisimulation under formal hypotheses* from [11,29]; on the other, w.r.t. to the relation from [4] (which is essentially universal bisimilarity in any conservative extension of the SOS system). All works cited in this paragraph discuss non-incremental proof systems, where the desired bisimulation relation needs to be fed by the user.

Descriptions of more or less automatic software tools for proving bisimilarity in process algebra abound in the literature – see [18,21] for overviews. While most of these tools are dedicated to (and optimized for) particular process algebras (and many to *finite-state* systems), ECRINS [12] is based precisely on generic process algebra in de Simone format, meaning that the results of this paper on incremental coinduction apply directly to that setting (and, interestingly, a form of coinduction that "attempts to add more couples to the [previously specified] relation" is indicated in [12] as a direction for future research, to our knowledge not pursued so far). Finally, in Coq [2], the interaction between its general-purpose support for building proofs and its coinductive types (as illustrated, e.g., in [13]) also leads to a form of incremental coinduction whose relationship with our approach is yet to be understood.

Future work. The de Simone SOS format is already fairly general, covering a wide range of process algebras, including CCS-like process algebras with *guarded recursion*, and even de Simone systems under weak bisimilarity (since for them weak bisimilarity can be regarded as strong bisimilarity for trace-based de Simone systems, as suggested in [27]). An extension of our incremental coinductive technique to more general formats such as GSOS [6] or tyft/tyxt [15] is of course

[3] In a sense, what these freezing operators do is to guard *against* coinduction up-to, not sound in general. So again, our logical system achieves convenience because of specialization, i.e., by sacrificing some generality.

desirable. Another direction for generalization is the allowance of bindings in the syntax of terms, including π-calculus-like bindings featuring scope extrusion (thus generalizing HOL-based settings for π-calculus such as [22,30]).

In our proof system for universal bisimilarity, $\vdash$, one has to come up with suitable dispatch functions G, g, G', g' at each application of the coinduction rule (Coind), and therefore (Coind) is not syntax-directed per se, hence not trivially automatable – this is inherent in the hardness of bisimilarity in our general setting. However, (Coind) does allow to decompose the goal symbolically, without asking the user to decide for a global bisimilarity candidate – instead, assisted by the powerful Isabelle classical reasoner and simplifier (able to discharge the typically simple goals resulted from chaining side-conditions), the user can explore various choices of the dispatch functions by analyzing the derived rules. Moreover, for systems with finite number of rules one can write an Isabelle tactic that tries all the combinations of dispatch functions. While this would require exponential time, it may still be feasible for cases of interest, since the time-complexity is a function of the *symbolic* branching of process patterns (determined by logical formulas obtained from side-conditions), and not of the actual branching of processes given by all possible instances of the rules. For the general case, we should aim at organizing our formalization (by means of proof tactics and pretty-printers) into a partly-interactive partly-automatic tool. The advantages of such a tool will of course include the generality of its scope and the fact that, unlike most other tools, it would be (a priori) *formally certified*.

Acknowledgments. We thank the reviewers for the careful and insightful reading of this paper and for pointing out related work. The first author is grateful to Grigore Roşu for many hours of discussion on his circular coinduction technique.

References

1. BOBJ, `http://cseweb.ucsd.edu/groups/tatami/bobj`
2. The Coq proof assistant, `http://coq.inria.fr`
3. Isabelle, `http://www.cl.cam.ac.uk/research/hvg/Isabelle`
4. Aceto, L., Cimini, M., Ingolfsdottir, A.: A bisimulation-based method for proving the validity of equations in GSOS languages. To appear in Electr. Proc. Theor. Comput. Sci.
5. Bartels, F.: Generalised coinduction. Math. Struct. Comp. Sci. 13(2), 321–348 (2003)
6. Bloom, B., Istrail, S., Meyer, A.R.: Bisimulation can't be traced. J. ACM 42(1), 232–268 (1995)
7. Brotherston, J.: Cyclic proofs for first-order logic with inductive definitions. In: Beckert, B. (ed.) TABLEAUX 2005. LNCS (LNAI), vol. 3702, pp. 78–92. Springer, Heidelberg (2005)
8. Bruni, R., de Frutos-Escrig, D., Martí-Oliet, N., Montanari, U.: Bisimilarity congruences for open terms and term graphs via Tile Logic. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 259–274. Springer, Heidelberg (2000)

9. Clavel, M., Durán, F.J., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Quesada, J.F.: The Maude system. In: Narendran, P., Rusinowitch, M. (eds.) RTA 1999. LNCS, vol. 1631, pp. 240–243. Springer, Heidelberg (1999)
10. Dam, M., Gurov, D.: μ-calculus with explicit points and approximations. J. Log. Comput. 12(2), 255–269 (2002)
11. de Simone, R.: Higher-level synchronizing devices in MEIJE-SCCS. Theor. Comput. Sci. 37, 245–267 (1985)
12. Doumenc, G., Madelaine, E., de Simone, R.: Proving process calculi translations in ECRINS: The pureLOTOS → MEIJE example. Technical Report RR1192, INRIA (1990), http://hal.archives-ouvertes.fr/inria-00075367/en/
13. Giménez, E.: An application of co-inductive types in Coq: Verification of the alternating bit protocol. In: Berardi, S., Coppo, M. (eds.) TYPES 1995. LNCS, vol. 1158, pp. 135–152. Springer, Heidelberg (1996)
14. Goguen, J.A., Lin, K., Roşu, G.: Circular coinductive rewriting. In: ASE 2000, pp. 123–132 (2000)
15. Groote, J.F., Vaandrager, F.: Structured operational semantics and bisimulation as a congruence. Inf. Comput. 100(2), 202–260 (1992)
16. Hausmann, D., Mossakowski, T., Schröder, L.: Iterative circular coinduction for coCASL in Isabelle/HOL. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, pp. 341–356. Springer, Heidelberg (2005)
17. Hennessy, M., Lin, H.: Proof systems for message-passing process algebras. Formal Asp. Comput. 8(4), 379–407 (1996)
18. Inverardi, P., Priami, C.: Automatic verification of distributed systems: The process algebra approach. Formal Methods in System Design 8(1), 7–38 (1996)
19. Lucanu, D., Goriac, E.-I., Caltais, G., Roşu, G.: CIRC: A behavioral verification tool based on circular coinduction. In: Kurz, A., Lenisa, M., Tarlecki, A. (eds.) CALCO 2009. LNCS, vol. 5728, pp. 433–442. Springer, Heidelberg (2009)
20. Lucanu, D., Roşu, G.: CIRC: A circular coinductive prover. In: Mossakowski, T., Montanari, U., Haveraaen, M. (eds.) CALCO 2007. LNCS, vol. 4624, pp. 372–378. Springer, Heidelberg (2007)
21. Madelaine, E.: Verification tools from the CONCUR project, http://www-sop.inria.fr/meije/papers/concur-tools
22. Melham, T.F.: A mechanized theory of the pi-calculus in HOL. Nord. J. Comput. 1(1), 50–76 (1994)
23. Milner, R.: A complete inference system for a class of regular behaviours. J. Comput. Syst. Sci. 28(3), 439–466 (1984)
24. Milner, R.: Communication and concurrency. Prentice-Hall, Englewood Cliffs (1998)
25. Monroy, R., Bundy, A., Green, I.: On process equivalence = equation solving in ccs. J. Autom. Reasoning 43(1), 53–80 (2009)
26. Mousavi, M.R., Reniers, M.A., Groote, J.F.: SOS formats and meta-theory: 20 years after. Theor. Comput. Sci. 373(3), 238–272 (2007)
27. Popescu, A.: Weak bisimilarity coalgebraically. In: Kurz, A., Lenisa, M., Tarlecki, A. (eds.) CALCO 2009. LNCS, vol. 5728, pp. 157–172. Springer, Heidelberg (2009)
28. Popescu, A., Gunter, E.L.: Incremental pattern-based coinduction for process algebra and its Isabelle formalization. Technical Report, University of Illinois, http://hdl.handle.net/2142/14858
29. Rensink, A.: Bisimilarity of open terms. Inf. Comput. 156(1-2), 345–385 (2000)

30. Röckl, C., Hirschkoff, D.: A fully adequate shallow embedding of the π-calculus in Isabelle/HOL with mechanized syntax analysis. J. Funct. Program. 13(2) (2003)
31. Roşu, G., Lucanu, D.: Circular coinduction: A proof theoretical foundation. In: Kurz, A., Lenisa, M., Tarlecki, A. (eds.) CALCO 2009. LNCS, vol. 5728, pp. 127–144. Springer, Heidelberg (2009)
32. Rutten, J.J.M.M.: Processes as terms: Non-well-founded models for bisimulation. Math. Struct. Comp. Sci. 2(3), 257–275 (1992)
33. Rutten, J.J.M.M.: Elements of stream calculus (an extensive exercise in coinduction). Electr. Notes Theor. Comput. Sci., 45 (2001)
34. Sangiorgi, D.: On the bisimulation proof method. Math. Struct. Comp. Sci. 8(5), 447–479 (1998)
35. Sangiorgi, D., Walker, D.: The π-calculus. A theory of mobile processes, Cambridge (2001)

Parameterised Multiparty Session Types[⋆]

Nobuko Yoshida, Pierre-Malo Deniélou, Andi Bejleri, and Raymond Hu

Department of Computing, Imperial College London

Abstract. For many application-level distributed protocols and parallel algorithms, the set of participants, the number of messages or the interaction structure are only known at run-time. This paper proposes a dependent type theory for multiparty sessions which can statically guarantee type-safe, deadlock-free multiparty interactions among processes whose specifications are parameterised by indices. We use the primitive recursion operator from Gödel's System $\mathcal{T}$ to express a wide range of communication patterns while keeping type checking decidable. We illustrate our type theory through non-trivial programming and verification examples taken from parallel algorithms and Web services usecases.

1 Introduction

As the momentum around communications-based computing grows, the need for effective frameworks to globally *coordinate* and *structure* the application-level interactions is pressing. The structures of interactions are naturally distilled as *protocols*. Each protocol describes a bare skeleton of how interactions should proceed, through e.g. sequencing, choices and repetitions. In the theory of multiparty session types [3, 5, 13], such protocols can be captured as types for interactions, and type checking can statically ensure runtime safety and fidelity to a stipulated protocol.

One of the particularly challenging aspects of protocol descriptions is the fact that many actual communication protocols are highly *parametric* in the sense that the number of participants and even the interaction structure itself are not fixed at design time. Examples include parallel algorithms such as the Fast Fourier Transform (run on any number of communication nodes depending on resource availability) and Web services such as business negotiation involving an arbitrary number of sellers and buyers. This paper introduces a robust dependent type theory which can statically ensure communication-safe, deadlock-free process interactions which follow parameterised multiparty protocols.

We illustrate the key ideas of our proposed parametric type structures through examples. Let us first consider a simple protocol where participant `Alice` sends a message of type nat to participant `Bob`. To develop the code for this protocol, we start by specifying the global type, which can concisely and clearly describe a high-level protocol for multiple participants [3, 13, 17], as follows (below end denotes protocol termination):

$$G_1 \;=\; \texttt{Alice} \to \texttt{Bob}\colon\langle \mathsf{nat}\rangle.\mathsf{end}$$

Upon agreement on G_1 as a specification for `Alice` and `Bob`, each program can be implemented separately. For type-checking, G_1 is *projected* into end-point session types: one

⋆ The work is partially supported by EPSRC EP/G015635/1 and EP/F003757/1.

from `Alice`'s point of view, $!\langle \texttt{Bob}, \mathsf{nat}\rangle$ (output to `Bob` with nat-type), and another from `Bob`'s point of view, $?\langle \texttt{Alice}, \mathsf{nat}\rangle$ (input from `Alice` with nat-type), against which the respective `Alice` and `Bob` programs are checked to be compliant.

The first step towards generalised type structures for multiparty sessions is to allow modular specifications of protocols using arbitrary compositions and repetitions of interaction units (this is a standard requirement in multiparty contracts [21]). Consider the type $G_2 = \texttt{Bob} \to \texttt{Carol}\colon \langle \mathsf{nat}\rangle.\mathsf{end}$. The designer may wish to compose sequentially G_1 and G_2 together to build a larger protocol:

$$G_3 = G_1; G_2 = \texttt{Alice} \to \texttt{Bob}\colon \langle \mathsf{nat}\rangle.\texttt{Bob} \to \texttt{Carol}\colon \langle \mathsf{nat}\rangle.\mathsf{end}$$

We may also want to iterate the composed protocols n-times, which can be written by $\texttt{foreach}(i < \mathrm{n})\{G_1; G_2\}$, and moreover bind the number of iteration n by a dependent product to build a *family of global specifications*, as in:

$$\Pi n.\texttt{foreach}(i < n)\{G_1; G_2\} \tag{1}$$

Beyond enabling a variable number of exchanges between a fixed set of participants, the ability to parameterise *participant identities* can represent a wide class of the communication topologies found in the literature. For example, the use of indexed participants $\texttt{W}[i]$ (denoting the i-th worker) allows to specify a family of session types such that neither the number of participants nor message exchanges are known before the runtime instantiation of the parameters. The following type and diagram both describe a sequence of messages from $\texttt{W}[n]$ to $\texttt{W}[0]$ (indices decrease in our `foreach`, see § 2):

$$\Pi n.(\texttt{foreach}(i < n)\{\texttt{W}[i+1] \to \texttt{W}[i]\colon \langle \mathsf{nat}\rangle\}) \qquad \begin{tikzcd} \boxed{\text{n}} \arrow[r] & \boxed{\text{n-1}} \arrow[r] & \cdots \arrow[r] & \boxed{0} \end{tikzcd} \tag{2}$$

Here we face an immediate question: *what is the underlying type structure for such parametrisation, and how can we type-check each (parametric) end-point program?* The type structure should allow the projection of a parameterised global type to an end-point type *before* knowing the exact shape of the concrete topology. In (2), if $\mathrm{n} \geq 2$, there are three distinct *communication patterns* inhabiting this specification: the initiator (send only), the $\mathrm{n} - 1$ middle workers (receive and send), and the last worker (receive only). This is no longer the case when $\mathrm{n} = 1$ (there is only the initiator and the last worker) or when $\mathrm{n} = 0$ (no communication). Can we provide a decidable projection and static type-checking by which we can preserve the main properties of the session types such as progress and communication-safety in parameterised process topologies? The key technique proposed in this paper is a projection method from a dependent global type onto a *generic end-point generator* which exactly captures the interaction structures of parameterised end-points and which can represent the class of all possible end-point types.

The main contributions of this paper follow:

– *A new expressive framework to globally specify and program* a wide range of parametric communication protocols (§ 2). We achieve this result by combining dependent type theories derived from Gödel's System $\mathcal{T}$ [18] (for expressiveness) and indexed dependent types from [22] (for tractability to control parameters), with multiparty session types.
– *Decidable and flexible projection methods* based on a generic end-point generator and mergeability of branching types, enlarging the typability (§ 3.1).

$\mathtt{i} ::= i \mid \mathrm{n} \mid \mathtt{i}\ \mathrm{op}\ \mathtt{i}'$	Indices	$G ::=$	Global types
$\mathtt{P} ::= \mathtt{P} \wedge \mathtt{P} \mid \mathtt{i} \leq \mathtt{i}'$	Propositions	$\mid\ \mathtt{p} \to \mathtt{p}' : \langle U \rangle . G$	Message
$I ::= \mathsf{nat} \mid \{i : I \mid \mathtt{P}\}$	Index sorts	$\mid\ \mathtt{p} \to \mathtt{p}' : \{l_k : G_k\}_{k \in K}$	Branching
$\mathcal{P} ::= \mathtt{Alice} \mid \mathtt{Worker} \mid \ldots$	Participants	$\mid\ \mu\mathbf{x}.G$	Recursion
$\mathtt{p} ::= \mathtt{p}[\mathtt{i}] \mid \mathcal{P}$	Principals	$\mid\ \mathbf{R}\ G\ \lambda i : I . \lambda \mathbf{x} . G'$	Primitive recursion
$S ::= \mathsf{nat} \mid \langle G \rangle$	Value type	$\mid\ \mathbf{x}$	Type variable
$U ::= S \mid T$	Payload type	$\mid\ G\ \mathtt{i}$	Application
$\mathrm{K} ::= \{\mathrm{n}_0, \ldots, \mathrm{n}_k\}$	Finite integer set	$\mid\ \mathsf{end}$	Null

$$\begin{array}{lll} \mathbf{R}\ G\ \lambda i : I . \lambda \mathbf{x} . G'\ 0 & \longrightarrow & G \\ \mathbf{R}\ G\ \lambda i : I . \lambda \mathbf{x} . G'\ (\mathrm{n}+1) & \longrightarrow & G'\{\mathrm{n}/i\}\{\mathbf{R}\ G\ \lambda i : I . \lambda \mathbf{x} . G'\ \mathrm{n}/\mathbf{x}\} \end{array}$$

Fig. 1. Global types and type reduction

- *A dependent typing system* that treats the full multiparty session types integrated with dependent types. The resulting static typing system allows decidable type-checking and guarantees type-safety and deadlock-freedom for well-typed processes involved in parameterised multiparty communication protocols (§ 3).
- *Applications* featuring various process topologies, including the complex butterfly network for the parallel FFT algorithm (§ 2.3, § 3.6). As far as we know, this is the first time such a complex protocol is specified by a single *type* and that its implementation can be automatically type-checked to prove communication-safety and deadlock-freedom. We also extend the calculus with a new asynchronous join primitive for session initialisation, applied to Web services use cases [19] (§ 3.6).

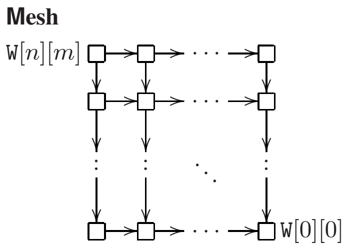
The complete formal definition of our system, including proofs and additional material for examples and implementations can be found in [11].

2 Types and Processes for Parameterised Multiparty Sessions

2.1 Global Types

Global types allow the description of the parameterised conversations of multiparty sessions as a type signature. Our type syntax integrates three different formulations: (1) global types from [3]; (2) dependent types with primitive recursive combinators based on [18]; and (3) parameterised dependent types from a simplified Dependent ML [1, 22].

The grammar of global types $(G, G', \ldots)$ is given in figure 1. *Parameterised principals* $\mathtt{p}, \mathtt{p}', \mathtt{q}, \ldots$ can be indexed by one or more parameters, e.g. $\mathtt{Worker}[5][i+1]$. Index $\mathtt{i}$ ranges over index variables i, j, n, naturals n or arithmetic operations. A global interaction can be a message exchange $(\mathtt{p} \to \mathtt{p}' : \langle U \rangle . G)$, where $\mathtt{p}, \mathtt{p}'$ denote the sending and receiving principals, U the payload type of the message and G the subsequent interaction. Payload types U are either value types S (which contain base type nat and session channel types $\langle G \rangle$), or *end-point types* T (which correspond to the behaviour of one of the session participants and will be explained in § 3) for delegation. Branching $(\mathtt{p} \to \mathtt{p}' : \{l_k : G_k\}_{k \in K})$ allows the session to follow one of the different G_k paths in the interaction (K is a ground and finite set of integers). $\mu\mathbf{x}.G$ is a recursive type where type variable $\mathbf{x}$ is guarded in the standard way.



```
$\Pi n.\Pi m.$
foreach($i < n$){
  foreach($j < m$){
    W$[i+1][j+1] \to$ W$[i][j+1] : \langle \mathsf{nat} \rangle$.
    W$[i+1][j+1] \to$ W$[i+1][j] : \langle \mathsf{nat} \rangle$};
  W$[i+1][0] \to$ W$[i][0] : \langle \mathsf{nat} \rangle$};
foreach($k < m$){W$[0][k+1] \to$ W$[0][k] : \langle \mathsf{nat} \rangle$}
```

Fig. 2. Parameterised multiparty protocol on a mesh topology

The interesting addition is the primitive recursion operator $\mathbf{R}\ G\ \lambda i : I.\lambda \mathbf{x}.G'$ from Gödel's System $\mathcal{T}$ [12] whose reduction semantics is given in figure 1. Its parameters are a global type G, an index variable i with range I, a type variable for recursion $\mathbf{x}$ and a recursion body G'.[1] When applied to an index $\mathtt{i}$, its semantics corresponds to the repetition $\mathtt{i}$-times of the body G', with the index variable i value going down by one at each iteration, from $\mathtt{i}-1$ to 0. The final behaviour is given by G when the index reaches 0. The index sorts comprise the set of natural numbers and its restrictions by predicates $(\mathtt{P}, \mathtt{P}', ..)$ that are, in our case, conjunctions of inequalities. $\mathtt{op}$ represents first-order indices operators (such as $+, -, *$,...). We often omit I and end in our examples. Using $\mathbf{R}$, we define the product, composition, repetition and test operators (seen in § 1):

$$\begin{array}{ll|l} \Pi i.G & = \mathbf{R}\ \mathsf{end}\ \lambda i.\lambda \mathbf{x}.G\{i+1/i\} & \mathtt{foreach}(i<j)\{G\} = \mathbf{R}\ \mathsf{end}\ \lambda i.\lambda \mathbf{x}.G\{\mathbf{x}/\mathsf{end}\}\ j \\ G_1; G_2 & = \mathbf{R}\ G_2\ \lambda i.\lambda \mathbf{x}.G_1\{\mathbf{x}/\mathsf{end}\}\ 1 & \mathsf{if}\ j\ \mathsf{then}\ G_1\ \mathsf{else}\ G_2 = \mathbf{R}\ G_2\ \lambda i.\lambda \mathbf{x}.G_1\ j \end{array}$$

where we assume that $\mathbf{x}$ is not free in G and G_1, and that the leaves of the syntax trees of G_1 and G are end. These definitions rely on a special substitution of each end by $\mathbf{x}$ (for example, $\mathtt{p} \to \mathtt{p}'\{l_1{:}!\langle \mathsf{nat} \rangle; \mathsf{end}, l_2{:}\mathsf{end}\}\{\mathbf{x}/\mathsf{end}\} = \mathtt{p} \to \mathtt{p}'\{l_1{:}!\langle \mathsf{nat} \rangle; \mathbf{x}, l_2{:}\mathbf{x}\}$). The composition operator appends the execution of G_2 to G_1; the repetition operator above repeats G j-times[2]; the boolean values are integers 0 (false) and 1 (true). Similar syntactic sugar is defined for local types and processes. Note that composition and repetition do not necessarily impose sequentiality: only the order of the asynchronous messages and the possible dependency [13] between receivers and subsequent senders controls the sequentiality. A parallel version of the sequence example of (§ 1 (2)) can be written: $\Pi n.(\mathtt{foreach}(i < n)\{\mathtt{W}[n-i] \to \mathtt{W}[n-i-1] : \langle \mathsf{nat} \rangle\})$.

Mesh example. The session presented in figure 2 describes a particular protocol over a standard mesh topology [15]. In this two dimensional array of workers W, each worker receives messages from his left and top neighbours (if they exist) before sending messages to his right and bottom (if they exist). Our session takes two parameters n and m which represent the number of rows and the number of columns. Then we have two iterators that repeat $\mathtt{W}[i+1][j+1] \to \mathtt{W}[i][j+1] : \langle \mathsf{nat} \rangle$ and $\mathtt{W}[i+1][j+1] \to \mathtt{W}[i+1][j] : \langle \mathsf{nat} \rangle$ for all i and j. The communication flow goes from the top-left $\mathtt{W}[n][m]$ and converges towards the bottom-right $\mathtt{W}[0][0]$ in $n+m$ parallel message exchanges.

[1] We distinguish recursion and primitive recursion in order to get decidability results, see § 3.4.
[2] This version of foreach uses decreasing indices. One can write an increasing version [11].

$c ::= y \mid s[\mathtt{p}]$	Channels	$\hat{\mathtt{p}}, \hat{\mathtt{q}} ::= \hat{\mathtt{p}}[\mathrm{n}] \mid \mathcal{P}$	Principal values
$u ::= x \mid a$	Identifiers	$m ::= (\hat{\mathtt{q}},\hat{\mathtt{p}},v) \mid (\hat{\mathtt{q}},\hat{\mathtt{p}},s[\hat{\mathtt{p}}']) \mid (\hat{\mathtt{q}},\hat{\mathtt{p}},l)$	Messages in transit
$v ::= a \mid \mathrm{n}$	Values	$h ::= \epsilon \mid m \cdot h$	Queue types
$e ::= \mathtt{i} \mid v \mid x \mid s[\mathtt{p}] \mid e \ \mathsf{op} \ e'$	Expressions		

$P ::=$	Processes	$\mid \mu X.P$	Recursion
$\mid \bar{u}[\mathtt{p}_0, .., \mathtt{p}_\mathrm{n}](y).P$	Init	$\mid \mathbf{0}$	Inaction
$\mid u[\mathtt{p}](y).P$	Accept	$\mid P \mid Q$	Parallel
$\mid \bar{a}[\mathtt{p}] : s$	Request	$\mid \mathbf{R}\ P\ \lambda i.\lambda X.Q$	Primitive recursion
$\mid c!\langle \mathtt{p}, e\rangle; P$	Value sending	$\mid X$	Process variable
$\mid c?\langle \mathtt{p}, x\rangle; P$	Value reception	$\mid (P\ \mathtt{i})$	Application
$\mid c \oplus \langle \mathtt{p}, l\rangle; P$	Selection	$\mid (\nu s)P$	Session restriction
$\mid c\&\langle \mathtt{p}, \{l_k : P_k\}_{k\in K}\rangle$	Branching	$\mid s{:}h$	Queues

Fig. 3. Syntax for user-defined and run-time processes

2.2 Process Syntax and Semantics

Syntax. The syntax of expressions and processes is given in figure 3, extended from [3], adding the primitive recursion operator and a new request process. Identifiers u can be variables x or channel names a. Values v are either channels a or natural numbers n. Expressions e are built out of indices $\mathtt{i}$, values v, variables x, session end points (for delegation) and operations over expressions. In processes, sessions are asynchronously initiated by $\bar{u}[\mathtt{p}_0, .., \mathtt{p}_\mathrm{n}](y).P$. It spawns, for each of the $\{\mathtt{p}_0, .., \mathtt{p}_\mathrm{n}\}$, a request that is accepted by the participant through $u[\mathtt{p}](y).P$. Messages are sent by $c!\langle \mathtt{p}, e\rangle; P$ to the participant $\mathtt{p}$ and received by $c?\langle \mathtt{q}, x\rangle; P$ from the participant $\mathtt{q}$. Selection $c \oplus \langle \mathtt{p}, l\rangle; P$, and branching $c\&\langle \mathtt{q}, \{l_k : P_k\}_{k\in K}\rangle$, allow a participant to choose a branch from those supported by another. Standard language constructs include recursive processes $\mu X.P$, restriction $(\nu s)P$ and parallel composition $P \mid Q$. The primitive recursion operator $\mathbf{R}\ P\ \lambda i.\lambda X.Q$ takes as parameters a process P, a function taking an index parameter i and a recursion variable X. A queue $s : h$ stores the asynchronous messages in transit.

An *annotated* P is the result of annotating P's bound names and variables as in e.g. $(\nu a : \langle G\rangle)Q$ or $s?(x : \langle G\rangle)Q$ or $\mathbf{R}\ Q\ \lambda i : I.\lambda X.Q'$. We omit the annotations unless needed. We often omit $\mathbf{0}$ and the participant $\mathtt{p}$ from the session primitives. Requests, session hiding and channel queues appear only at runtime, as explained below.

Semantics. The semantics is defined by the reduction relation $\longrightarrow$ presented in figure 4. The standard definition of evaluation contexts (that allow e.g. $\mathtt{W}[3 + 1]$ to be reduced to $\mathtt{W}[4]$) is omitted. The metavariables $\hat{\mathtt{p}}, \hat{\mathtt{q}}, ..$ range over principal values (where all indices have been evaluated). [ZeroR] and [SuccR] are standard and identical to their global type counterparts. The rule [Init] describes the initialisation of a session by its first participant $\bar{a}[\mathtt{p}_0, .., \mathtt{p}_\mathrm{n}](y_0).P_0$. It spawns asynchronous requests $\bar{a}[\hat{\mathtt{p}}_k] : s$ that allow delayed acceptance by the other session participants (rule [Join]). After the connection, the participants share the private session name s, and the queue associated to s (which is initially empty by rule [Init]). The variables $y_\mathtt{p}$ in each participant $\mathtt{p}$ are then replaced with the corresponding session channel, $s[\mathtt{p}]$. A more verbose, but symmetric, version of [Init] (where any participant can start the session, not only $\mathtt{p}_0$) could also be used [11].

$$\mathbf{R}\ P\ \lambda i.\lambda X.Q\ 0 \longrightarrow P \qquad \text{[ZeroR]}$$
$$\mathbf{R}\ P\ \lambda i.\lambda X.Q\ \mathrm{n}+1 \longrightarrow Q\{\mathrm{n}/i\}\{\mathbf{R}\ P\ \lambda i.\lambda X.Q\ \mathrm{n}/X\} \qquad \text{[SuccR]}$$
$$\bar{a}[\hat{\mathtt{p}}_0,..,\hat{\mathtt{p}}_\mathrm{n}](y).P \longrightarrow (\nu s)(P\{s[\hat{\mathtt{p}}_0]/y\} \mid s:\emptyset \mid \bar{a}[\hat{\mathtt{p}}_1]:s \mid ... \mid \bar{a}[\hat{\mathtt{p}}_\mathrm{n}]:s) \qquad \text{[Init]}$$
$$\bar{a}[\hat{\mathtt{p}}_k]:s \mid a[\hat{\mathtt{p}}_k](y_k).P_k \longrightarrow P_k\{s[\hat{\mathtt{p}}_k]/y_k\} \qquad \text{[Join]}$$
$$s[\hat{\mathtt{p}}]!\langle\hat{\mathtt{q}},v\rangle;P \mid s:h \longrightarrow P \mid s:h\cdot(\hat{\mathtt{p}},\hat{\mathtt{q}},v) \qquad \text{[Send]}$$
$$s[\hat{\mathtt{p}}]\oplus\langle\hat{\mathtt{q}},l\rangle;P \mid s:h \longrightarrow P \mid s:h\cdot(\hat{\mathtt{p}},\hat{\mathtt{q}},l) \qquad \text{[Label]}$$
$$s[\hat{\mathtt{p}}]?(\hat{\mathtt{q}},x);P \mid s:(\hat{\mathtt{q}},\hat{\mathtt{p}},v)\cdot h \longrightarrow P\{v/x\} \mid s:h \qquad \text{[Recv]}$$
$$s[\hat{\mathtt{p}}]\&(\hat{\mathtt{q}},\{l_k:P_k\}_{k\in K}) \mid s:(\hat{\mathtt{q}},\hat{\mathtt{p}},l_{k_0})\cdot h \longrightarrow P_{k_0} \mid s:h \quad (k_0\in K) \qquad \text{[Branch]}$$

Fig. 4. Reduction rules

The rest of the session reductions are standard [3, 13]. The output rules [Send] and [Label] push values, channels and labels into the queue of the session s. The rules [Recv] and [Branch] perform the complementary operations. Note that these operations check that the sender and receiver match. Processes are considered modulo structural equivalence, denoted by $\equiv$ (in particular, we note $\mu X.P \equiv P\{\mu X.P/X\}$).

2.3 Processes for Parameterised Multiparty Protocols

We give here the processes corresponding to the interactions described in § 1 and § 2.1, then introduce a parallel implementation of the Fast Fourier Transform algorithm.

Sequence from § 1 (2). The process below generates all participants using a recursor:

$$\begin{array}{lll}\Pi n.(\text{if } n=0 \text{ then} & \mathbf{0} & \\ \phantom{\Pi n.(\text{if } n=0}\ \text{else} & (\mathbf{R} & (\bar{a}[\mathtt{W}[n],..,\mathtt{W}[0]](y).y!\langle\mathtt{W}[n-1],v\rangle;\mathbf{0} \\ & & \mid a[\mathtt{W}[0]](y).y?(\mathtt{W}[1],z);\mathbf{0}) \\ & & \lambda i.\lambda X.(a[\mathtt{W}[i+1]](y).y?(\mathtt{W}[i+2],z);y!\langle\mathtt{W}[i],z\rangle;\mathbf{0} \mid X) \quad n-1)\end{array}$$

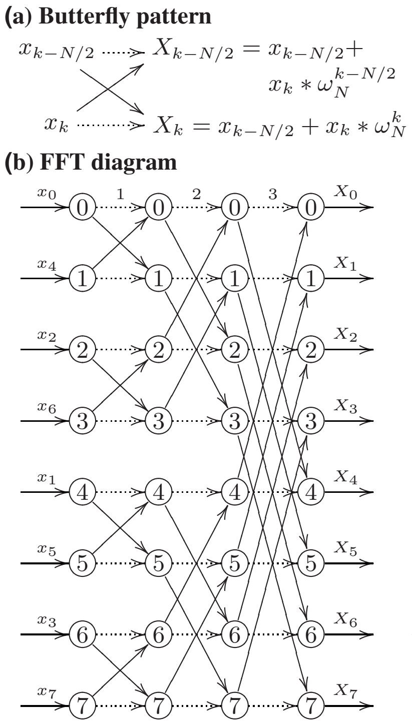
When $n = 0$ no message is exchanged. In the other case, the recursor creates the $n-1$ workers through the main loop and finishes by spawning the initial and final ones.

As an illustration of the semantics, we show the reduction of the above process for $n = 2$. After several applications of the [SuccR] and [ZeroR] rules, we have:

$$\bar{a}[\mathtt{W}[2],\mathtt{W}[1],\mathtt{W}[0]](y).y!\langle\mathtt{W}[1],v\rangle; \mid a[\mathtt{W}[0]](y).y?(\mathtt{W}[1],z); \mid a[\mathtt{W}[1]](y).y?(\mathtt{W}[2],z);y!\langle\mathtt{W}[0],z\rangle;$$

which, with [Init], [Join], [Send], [Recv], gives:

$$\begin{array}{ll}\longrightarrow & (\nu s)(s:\epsilon \mid s[\mathtt{W}[2]]!\langle\mathtt{W}[1],v\rangle; \mid \bar{a}[\mathtt{W}[1]]:s \mid \bar{a}[\mathtt{W}[0]]:s \mid \\ & \quad a[\mathtt{W}[0]](y).y?(\mathtt{W}[1],z); \mid a[\mathtt{W}[1]](y).y?(\mathtt{W}[2],z);y!\langle\mathtt{W}[0],z\rangle;) \\ \longrightarrow & (\nu s)(s:\epsilon \mid s[\mathtt{W}[2]]!\langle\mathtt{W}[1],v\rangle; \mid \bar{a}[\mathtt{W}[1]]:s \mid \\ & \quad s[\mathtt{W}[0]]?(\mathtt{W}[1],z); \mid a[\mathtt{W}[1]](y).y?(\mathtt{W}[2],z);y!\langle\mathtt{W}[0],z\rangle;) \\ \longrightarrow^* & (\nu s)(s:\emptyset \mid s[\mathtt{W}[2]]!\langle\mathtt{W}[1],v\rangle; \mid s[\mathtt{W}[0]]?(\mathtt{W}[1],z); \mid s[\mathtt{W}[1]]?(\mathtt{W}[2],z);s[\mathtt{W}[1]]!\langle\mathtt{W}[0],z\rangle;) \\ \longrightarrow^* & (\nu s)(s:\emptyset \mid s[\mathtt{W}[0]]?(\mathtt{W}[1],z); \mid s[\mathtt{W}[1]]!\langle\mathtt{W}[0],v\rangle;) \\ \longrightarrow^* & \equiv \mathbf{0}\end{array}$$


(a) Butterfly pattern
$x_{k-N/2}$
$X_{k-N/2} = x_{k-N/2} + x_k * \omega_N^{k-N/2}$
x_k
$X_k = x_{k-N/2} + x_k * \omega_N^k$
(b) FFT diagram
x_0 x_4 x_2 x_6 x_1 x_5 x_3 x_7
1 2 3
X_0 X_1 X_2 X_3 X_4 X_5 X_6 X_7

(c) Global type $G =$

```
$\Pi n.$
foreach($i < 2^n$){$i \to i : \langle \mathsf{nat} \rangle$};
foreach($l < n$){
 foreach($i < 2^l$){
  foreach($j < 2^{n-l-1}$){
   foreach($k < 2$){
    foreach($k' < 2$){
     $i * 2^{n-l} + k * 2^{n-l-1} + j$
     $\to i * 2^{n-l} + k' * 2^{n-l-1} + j : \langle \mathsf{nat} \rangle$}}}}}
```

(d) Processes $P(n, \mathtt{p}, x_{\overline{\mathtt{p}}}, y, r_{\mathtt{p}}) =$

```
$y!\langle \mathtt{p}, x_{\overline{\mathtt{p}}} \rangle;$
foreach($l < n$){
  if $\mathsf{bit}_{n-l}(\mathtt{p}) = 0$
   then $y?\langle \mathtt{p}, x \rangle; y!\langle \mathtt{p} + 2^{n-l-1}, x \rangle;$
        $y?\langle \mathtt{p} + 2^{n-l-1}, z \rangle; y!\langle \mathtt{p}, x + z\,\omega_N^{g(l,\mathtt{p})} \rangle;$
   else $y?\langle \mathtt{p}, x \rangle; y!\langle \mathtt{p} - 2^{n-l-1}, x \rangle;$
        $y?\langle \mathtt{p} - 2^{n-l-1}, z \rangle; y!\langle \mathtt{p}, z + x\,\omega_N^{g(l,\mathtt{p})} \rangle;$ };
$y?\langle \mathtt{p}, x \rangle; r_{\mathtt{p}}!\langle 0, x \rangle;$
```

where $g(l, \mathtt{p}) = \mathtt{p} \mod 2^l$

Fig. 5. Fast Fourier Transform on a butterfly network topology

Mesh from figure 2. The mesh example is more complex: when n and m are bigger than 2, there are 9 distinct roles that each have a different pattern of communication. We only list processes for (1) the centre workers $\mathtt{W}[i][j]$ ($0 < i < n$, $0 < j < m$) who are connected in all four directions, (2) the initiator $\mathtt{W}[n][m]$ from the top-left corner. Below, $f(i, j)$ represents the expression computed at the (i, j)-th element.

$$\begin{aligned} P_{\text{centre}}(i,j) &= a[\mathtt{W}[i][j]](y).y?(\mathtt{W}[i+1][j], z_1); y?(\mathtt{W}[i][j+1], z_2);\\ &\qquad y!\langle \mathtt{W}[i-1][j], f(i-1,j)\rangle; y!\langle \mathtt{W}[i][j-1], f(i,j-1)\rangle; \mathbf{0}\\ P_{\text{start}}(n,m) &= \bar{a}[\mathtt{W}[0][0]..\mathtt{W}[n][m]](y).y!\langle \mathtt{W}[n-1][m], f(n-1,m)\rangle; \end{aligned}$$

FFT. We describe a parallel implementation of the Fast Fourier Transform algorithm (more precisely the radix-2 variant of the Cooley-Tukey algorithm [10]).

Figure 5(a) illustrates the recursive principle of the algorithm, called *butterfly*, where two different outputs can be computed in constant time from the results of the same two recursive calls. The complete algorithm is illustrated by the diagram from figure 5(b). It features the application of the FFT on a network of $N = 2^3$ machines on an hypercube network computing the discrete Fourier transform of vector $x_0, \ldots, x_7$. Each row represents a single machine at each step of the algorithm. Each edge represents a value sent to another machine. The dotted edges represent the particular messages that a machine sends to itself to remember a value for the next step. Each machine is successively involved in a butterfly with a machine whose number differs by only one bit.

$$
\begin{array}{llll}
T ::= & \text{End-point types} & \mid \mu\mathbf{x}.T & \text{Recursion} \\
\quad \mid\ !\langle \mathtt{p}, U\rangle; T & \text{Output} & \mid \mathbf{R}\, T\ \lambda i\!:\!I.\lambda\mathbf{x}.T' & \text{Primitive recursion} \\
\quad \mid\ ?\langle \mathtt{p}, U\rangle; T & \text{Input} & \mid \mathbf{x} & \text{Type variable} \\
\quad \mid\ \oplus\langle \mathtt{p}, \{l_k : T_i\}_{k\in K}\rangle & \text{Selection} & \mid T\ \mathtt{i} & \text{Application} \\
\quad \mid\ \&\langle \mathtt{p}, \{l_k : T_i\}_{k\in K}\rangle & \text{Branching} & \mid \mathsf{end} & \text{End}
\end{array}
$$

Fig. 6. End-point types

$$
\begin{array}{rcl|rcl}
\mathtt{p} \to \mathtt{p}' : \langle U\rangle.G \restriction \mathtt{q} & = & \mathsf{if}\ \mathtt{q{=}p{=}p'}\ \mathsf{then}\ !\langle \mathtt{p}, U\rangle; ?\langle \mathtt{p}, U\rangle; G \restriction \mathtt{q} & (\mu\mathbf{t}.G)\restriction \mathtt{p} & = & \mu\mathbf{t}.G \restriction \mathtt{p} \\
 & & \mathsf{else\ if}\ \mathtt{q{=}p}\ \mathsf{then}\ !\langle \mathtt{p}', U\rangle; G \restriction \mathtt{q} & \mathbf{x}\restriction \mathtt{p} & = & \mathbf{x} \\
 & & \mathsf{else\ if}\ \mathtt{q{=}p'}\ \mathsf{then}\ ?\langle \mathtt{p}, U\rangle; G \restriction \mathtt{q} & (G\ \mathtt{i})\restriction \mathtt{p} & = & (G\restriction \mathtt{p})\ \mathtt{i} \\
 & & \mathsf{else}\ G\restriction \mathtt{q} & \mathsf{end}\restriction \mathtt{p} & = & \mathsf{end} \\
\mathtt{p} \to \mathtt{p}' : \{l_k : G_k\}_{k\in K} \restriction \mathtt{q} & = & \mathsf{if}\ \mathtt{q{=}p}\ \mathsf{then}\ \oplus\langle \mathtt{p}', \{l_k : G_k \restriction \mathtt{q}\}_{k\in K}\rangle & & & \\
 & & \mathsf{else\ if}\ \mathtt{q{=}p'}\ \mathsf{then}\ \&\langle \mathtt{p}, \{l_k : G_k \restriction \mathtt{q}\}_{k\in K}\rangle & & & \\
 & & \mathsf{else}\ \sqcup_{k\in K} G_k \restriction \mathtt{q} & & & \\
\mathbf{R}\, G\ \lambda i\!:\!I.\lambda\mathbf{x}.G' \restriction \mathtt{q} & = & \mathbf{R}\, G \restriction \mathtt{q}\ \lambda i\!:\!I.\lambda\mathbf{x}.G' \restriction \mathtt{q} & & &
\end{array}
$$

Fig. 7. Projection of global types to end-point types

Note that the recursive partition over the value of a different bit at each step requires a particular bit-reversed ordering of the input vector: the machine number $\mathtt{p}$ initially receives $x_{\overline{\mathtt{p}}}$ where $\overline{\mathtt{p}}$ denotes the bit-reversal of $\mathtt{p}$. Figure 5(c) gives the global session type describing the interactions between 2^n machines. The first iterator is the initialisation step. Then we have an iteration over variable l for the n successive steps of the algorithm. Figure 5(d) defines the processes that each of the machines runs. Each process returns the final answer at $r_{\mathtt{p}}$.

3 Typing Parameterised Multiparty Interactions

3.1 End-Point Types and End-Point Projections

The syntax of end-point types is given in figure 6. Output expresses the sending to $\mathtt{p}$ of a value or channel of type U, followed by the interactions T. Selection represents the transmission to $\mathtt{p}$ of a label l_k chosen in $\{l_k\}_{k\in K}$ followed by T_k. Input and branching are their dual counterparts. The other types are similar to their global versions.

End-point projection: a generic projection. The relation between end-point types and global types is formalised by the projection relation. Since the actual participant characteristics might only be determined at runtime, we cannot straightforwardly use the definition from [3, 13]. Instead, we rely on the expressive power of the primitive recursive operator: *a generic end-point projection of G onto* $\mathtt{q}$, written $G \restriction \mathtt{q}$, represents the family of all the possible end-point types that a principal $\mathtt{q}$ can satisfy at run-time.

The general endpoint generator is defined in figure 7 using the derived construct if _ then _ else _. The projection $\mathtt{p} \to \mathtt{p}' : \langle U\rangle.G \restriction \mathtt{q}$ leads to a case analysis: if the participant $\mathtt{q}$ is equal to $\mathtt{p}$, then the end-point type of $\mathtt{q}$ is an output of type U to $\mathtt{p}'$; if participant $\mathtt{q}$ is $\mathtt{p}'$ then $\mathtt{q}$ inputs U from $\mathtt{p}'$; else we skip the prefix. The first case corresponds to the possibility for the sender and receiver to be identical. Projecting the branching global type is similarly defined, but for the operator $\sqcup$ explained below. For the other cases (as well as for our derived operators), the projection is homomorphic.

Mergeability and injection of branching types. We first recall the example from [13], which explains that naïve branching projection leads to inconsistent end-point types.

$$\mathtt{W}[0] \to \mathtt{W}[1] : \{\mathsf{ok} : \mathtt{W}[1] \to \mathtt{W}[2] : \langle \mathsf{bool} \rangle,\ \mathsf{quit} : \mathtt{W}[1] \to \mathtt{W}[2] : \langle \mathsf{nat} \rangle\}$$

We cannot project the above type onto $\mathtt{W}[2]$ because, while the branches behave differently, $\mathtt{W}[0]$ makes a choice without informing $\mathtt{W}[2]$ who thus cannot know the type of the expected value. A solution is to define projection only when the branches are identical, i.e. we change the above nat to bool in our example above.

In our framework, this restriction is too strong since each branch may contain different parametric interaction patterns. To overcome this, we propose two methods called *mergeability* and *injection* of branching types. Formally, the mergeability relation $\bowtie$ is the smallest congruence relation over end-point types such that:[3] if $\forall i \in (K \cap J).T_k \bowtie T'_j$ and $\forall i \in (K \setminus J) \cup (J \setminus K).l_k \neq l_j$, then $\&\langle \mathtt{p}, \{l_k : T_k\}_{k\in K}\rangle \bowtie \&\langle \mathtt{p}, \{l_j : T'_j\}_{j\in J}\rangle$. When $T_1 \bowtie T_2$ is defined, we define the injection $\sqcup$ as a partial commutative operator over two types such that $T \sqcup T = T$ for all types and that:

$$\begin{aligned}&\&\langle \mathtt{p}, \{l_k : T_i\}_{k\in K}\rangle \sqcup \&\langle \mathtt{p}, \{l_j : T'_j\}_{j\in J}\rangle \ = \\ &\quad \&\langle \mathtt{p}, \{l_k : T_k \sqcup T'_k\}_{k\in K\cap J} \cup \{l_k : T_k\}_{k\in K\setminus J} \cup \{l_j : T'_j\}_{j\in J\setminus K}\rangle\end{aligned}$$

The mergeability relation states that two types are identical up to their branching types where only branches with distinct labels are allowed to be different. By this extended typing condition, we can modify our previous global type example to add ok and quit labels to notify $\mathtt{W}[2]$. We get:

$$\begin{aligned}\mathtt{W}[0] \to \mathtt{W}[1] : \{&\mathsf{ok} : \mathtt{W}[1] \to \mathtt{W}[2] : \{\mathsf{ok} : \mathtt{W}[1] \to \mathtt{W}[2]\langle \mathsf{bool} \rangle\ \},\\ &\mathsf{quit} : \mathtt{W}[1] \to \mathtt{W}[2] : \{\mathsf{quit} : \mathtt{W}[1] \to \mathtt{W}[2]\langle \mathsf{nat} \rangle\}\}\end{aligned}$$

Then $\mathtt{W}[2]$ can have the type $\&\langle \mathtt{W}[1],\ \{\mathsf{ok} : \langle \mathtt{W}[1], \mathsf{bool}\rangle,\ \mathsf{quit} : \langle \mathtt{W}[1], \mathsf{nat}\rangle\}\rangle$ which could not be obtained through the original projection rule in [3, 13]. This projection is sound up to branching subtyping (cf. Lemma 3.4).

3.2 Type System

This subsection introduces the type system. Because free indices appear both in terms (e.g. participants in session initialisation) and in types, the formal definition of what constitutes a valid term and a valid type are interdependent and both in turn require a careful definition of a valid global type.

Judgements and environments. One of the main differences with previous session type systems is that session environments Δ can contain dependent *process types*. The grammar of environments, process types and kinds are given below.

$$\Delta ::= \emptyset \mid \Delta, c{:}T \qquad \Gamma ::= \emptyset \mid \Gamma, \mathtt{P} \mid \Gamma, u : S \mid \Gamma, i : I \mid \Gamma, X : \tau \qquad \tau ::= \Delta \mid \Pi i{:}I.\tau$$

[3] The idea of mergeability is introduced informally in the tutorial paper [8].

Δ is the *session environment* which associates channels to session types. Γ is the *standard environment* which contains predicates and which associates variables to sort types, service names to global types, indices to index sets and process variables to session types. τ is a *process type* which is either a session environment or a dependent type. We write $\Gamma, u : S$ only if $u \notin dom(\Gamma)$. We use the same convention for others.

Following [22], we assume given in the typing rules two semantically defined judgements: $\Gamma \models \mathtt{P}$ (predicate $\mathtt{P}$ is a consequence of Γ) and $\Gamma \models \mathtt{i} : I$ ($\mathtt{i} : I$ follows from the assumptions of Γ). We also inductively define well-formed types using a kind system [11]. The judgement $\Gamma \vdash U \blacktriangleright \kappa$ means type U has kind κ. Kinds include proper types for global, value, principal, end-point and process types (denoted by Type), and the kind of type families, written by $\Pi i : I.\kappa$. Well-formedness of a term $\mathtt{i}$ and $\mathtt{P}$ in Γ and environments is defined in the standard way [1].

3.3 Typing Processes

We explain here (Figure 8) a selection of the process typing rules. Rules [TNAT] and [TVAR] are standard ($\Gamma \vdash \mathsf{Env}$ means that Γ is well-formed). For participants, we check their typing by [TID] and [TP] in a similar way as [22] where $\Gamma \vdash \kappa$ means kinding κ is well-formed. In [TPREC], we use the abbreviation $[0..\mathtt{j}] = \{i : \mathsf{nat} \mid i \leq \mathtt{j}\}$. Then

$$\frac{\Gamma \vdash \mathsf{Env}}{\Gamma \vdash \mathrm{n} \triangleright \mathsf{nat}}\,[\text{TNAT}] \qquad \frac{\Gamma \vdash \kappa}{\Gamma \vdash \mathtt{Alice} \triangleright \kappa}\,[\text{TID}] \qquad \frac{\Gamma \vdash \mathtt{p} \triangleright \Pi i{:}I.\kappa \quad \Gamma \models \mathtt{i}{:}I}{\Gamma \vdash \mathtt{p}[\mathtt{i}] \triangleright \kappa\{\mathtt{i}/i\}}\,[\text{TP}]$$

$$\frac{\Gamma, i : I^-, X : \tau\{i/j\} \vdash Q \triangleright \tau\{i+1/j\} \quad \Gamma \vdash P \triangleright \tau\{0/j\} \quad \Gamma, j{:}I \vdash \tau \blacktriangleright \kappa}{\Gamma \vdash \mathbf{R}\; P\; \lambda i.\lambda X.Q \triangleright \Pi j{:}I.\tau}\,[\text{TPREC}]$$

$$\frac{\begin{cases}\Gamma \vdash G_1 \equiv G_2 \quad \Gamma \vdash G_1' \equiv G_2' \qquad \text{or} \\ \Gamma \vdash \mathbf{R}\; G_1\; \lambda i{:}I.\lambda \mathbf{x}.G_1'\; \mathrm{n} \equiv \mathbf{R}\; G_2\; \lambda i{:}I.\lambda \mathbf{x}.G_2' \mathrm{n} \text{ with } \Gamma \models I = [0..\mathrm{m}], 0 \leq \mathrm{n} \leq \mathrm{m}\end{cases}}{\Gamma \vdash \mathbf{R}\; G_1\; \lambda i{:}I.\lambda \mathbf{x}.G_1' \equiv_{\text{wf}} \mathbf{R}\; G_2\; \lambda i{:}I.\lambda \mathbf{x}.G_2'}\,[\text{PREC}]$$

$$\frac{\Gamma \vdash \text{whnf}(G_1) \equiv_{\text{wf}} \text{whnf}(G_2)}{\Gamma \vdash G_1 \equiv G_2}\,[\text{WF}] \quad \frac{\Gamma \vdash P \triangleright \tau \quad \Gamma \vdash \tau \equiv \tau'}{\Gamma \vdash P \triangleright \tau'}\,[\text{TEQ}] \quad \frac{\Gamma \vdash P \triangleright \tau \quad \Gamma \vdash \tau \leq \tau'}{\Gamma \vdash P \triangleright \tau'}\,[\text{TSUB}]$$

$$\frac{\Gamma, X : \tau \vdash P \triangleright \tau}{\Gamma \vdash \mu X.P \triangleright \tau}\,[\text{TREC}] \quad \frac{\Gamma, X : \tau \vdash \mathsf{Env}}{\Gamma, X : \tau \vdash X \triangleright \tau}\,[\text{TVAR}] \quad \frac{\Gamma \vdash P \triangleright \Pi i{:}I.\tau \quad \Gamma \models \mathtt{i} \in I}{\Gamma \vdash P\; \mathtt{i} \triangleright \tau\{\mathtt{i}/i\}}\,[\text{TAPP}]$$

$$\frac{\begin{array}{c}\Gamma \vdash u : \langle G\rangle \quad \Gamma \vdash P \triangleright \Delta, y : G \upharpoonright \mathrm{p}_0 \\ \Gamma \vdash \mathtt{p}_i \triangleright \mathsf{nat} \quad \Gamma \models \mathsf{pid}(G) = \{\mathrm{p}_0..\mathrm{p}_\mathrm{n}\}\end{array}}{\Gamma \vdash \bar{u}[\mathrm{p}_0, .., \mathrm{p}_\mathrm{n}](y).P \triangleright \Delta}\,[\text{TINIT}] \quad \frac{\begin{array}{c}\Gamma \vdash u : \langle G\rangle \quad \Gamma \vdash P \triangleright \Delta, y : G \upharpoonright \mathtt{p} \\ \Gamma \vdash \mathtt{p} \triangleright \mathsf{nat} \quad \Gamma \models \mathtt{p} \in \mathsf{pid}(G)\end{array}}{\Gamma \vdash u[\mathtt{p}](y).P \triangleright \Delta}\,[\text{TACC}]$$

$$\frac{\Gamma \vdash a : \langle G\rangle \;\; \Gamma \vdash \mathtt{p} \triangleright \mathsf{nat} \;\; \Gamma \models \mathtt{p} \in \mathsf{pid}(G)}{\Gamma \vdash \bar{a}[\mathtt{p}] : s \triangleright s[\mathtt{p}] : G \upharpoonright \mathtt{p}}\,[\text{TREQ}] \quad \frac{\Gamma \vdash e \triangleright S \quad \Gamma \vdash P \triangleright \Delta, c : T}{\Gamma \vdash c!\langle \mathtt{p}, e\rangle; P \triangleright \Delta, c :!\langle \mathtt{p}, S\rangle; T}\,[\text{TOUT}]$$

Fig. 8. Process typing

we define I^- by $[0..0]^- = \emptyset$ and $[0..\mathtt{i}]^- = [0..\mathtt{i} - 1]$. This rule deals with the changed index range within the recursor body. More precisely, we first check τ's kind. Then we verify for the base case ($j = 0$) that P has type $\tau\{0/j\}$. Last, we check the more complex inductive case: Q should have type $\tau\{i + 1/j\}$ under the environment $\Gamma, i{:}I^-, X{:}\tau\{i/j\}$ where $\tau\{i/j\}$ of X means that X satisfies the predecessor's type (induction hypothesis). [TAPP] is the elimination rule for dependent types.

Since our types include dependent types and recursors, we need a notion of type equivalence. We extend the standard method of [1, §2] with the recursor: [WF] is the main rule defining $G_1 \equiv G_2$ and relies on the existence of a common weak head normal form for the two types and [PREC] says two recursors are equated if either (1) each subgraph is equated by $\equiv$, or (2) they reduce to the same normal forms when applied to a finite number of indices. Rule [TEQ] allows to type processes up-to type equivalence.

[TINIT] types a session initialisation on shared channel u, binding channel y and requiring participants $\{\mathtt{p}_0, .., \mathtt{p}_n\}$. The premise verifies that the type of y is the first projection of the global type G of u and that the participants in G (denoted by $\mathtt{pid}(G)$) can be semantically derived as $\{\mathtt{p}_0, .., \mathtt{p}_\mathtt{n}\}$. [TACC] allows to type the $\mathtt{p}$-th participant to the session initiated on u. The typing rule checks that the type of y is the $\mathtt{p}$-th projection of the global type G of u and that G is fully instantiated. The kind rule ensures that G is fully instantiated (i.e. G''s kind is Type). [TREQ] types the process that waits for an accept from a participant: its type corresponds to the end-point projection of G.

Recursion [TREC], variable ([TVAR]), output ([TOUT]), input, delegation, inaction, branching/selection and the expression typing rules as well as the typing rules for queues are similar to those in [3, 13].

3.4 Properties of Typing

Ensuring termination of type-checking with dependent types is not an easy task since type equivalences are often derived from term equivalences. We rely on the strong normalisation of System $\mathcal{T}$ [12] for the termination proof.

Proposition 3.1 (Termination and Confluence). *The head relation $\longrightarrow$ on global and end-point types (i.e. $G \longrightarrow G'$ and $T \longrightarrow T'$ for closed types in Figure 2) are strong normalising and confluent on well-formed kindings.*

The following lemma is proved by defining the weight of the equality and showing the weight of any premise of a rule is always less than the weight of the conclusion (the weight for a recursor needs to be extended to allow the inductive equality rule).

Proposition 3.2 (Termination for Type-Equality Checking). *Assuming that proving the predicates $\Gamma \models \mathtt{P}$ appearing in type equality derivations is decidable, then type-equality checking of $\Gamma \vdash G \equiv G'$ terminates. Similarly for other types.*

Proposition 3.3 (Termination for Type-Checking). *Assuming that proving the predicates $\Gamma \models \mathtt{P}$ appearing in kinding, equality, projection and typing derivations is decidable, then type-checking of annotated process P, i.e. $\Gamma \vdash P \triangleright \emptyset$ terminates.*

Proof. (Outline) By the standard argument from indexed dependent types [1, 22], for the dependent λ-applications, we do not require equality of terms (i.e. we only need the equality of the indices by the semantic consequence judgements). Hence to eliminate the type equality rule $\lfloor\text{TEQ}\rfloor$, we include the type equality check into $\lfloor\text{TINIT,TREQ,TACC}\rfloor$ (between the global type and its projected session type), and the input rule (between the session type and the type annotating x). Similarly for recursive agents. Since $\alpha \equiv \beta$ (for any type α and β) terminates, these checks always terminate. $\square$

To ensure the termination of $\Gamma \models \mathtt{P}$, several solutions include the restriction of predicates to linear equalities over natural numbers without multiplication (or to other decidable arithmetic subsets) or the restriction of indices to finite domains, cf. [22].

3.5 Type Soundness and Progress

The following lemma states that mergeability is sound with respect to the branching subtyping $\leq$. By this, we can safely replace the third clause $\sqcup_{k\in K} G_k \upharpoonright \mathtt{q}$ of the branching case from the projection definition by $\sqcap\{T \mid \forall k \in K.T \leq (G_k \upharpoonright \mathtt{q})\}$. This allows us to prove subject reduction by including subsumption as done in [13].

Lemma 3.4 (Soundness of mergeability). *Suppose $G_1 \upharpoonright \mathtt{p} \bowtie G_2 \upharpoonright \mathtt{p}$ and $\Gamma \vdash G_i$. Then there exists G such that $G \upharpoonright \mathtt{p} = \sqcap\{T \mid T \leq G_i \upharpoonright \mathtt{p}\ (i = 1, 2)\}$ where $\sqcap$ denotes the maximum element with respect to $\leq$.*

As session environments record channel states, they evolve when communications proceed. This can be formalised by introducing a notion of session environments reduction. These rules are formalised below modulo $\equiv$.

- $\{s[\hat{\mathtt{p}}] :!\langle\hat{\mathtt{q}}, U\rangle; T, s[\hat{\mathtt{q}}] :?\langle\hat{\mathtt{p}}, U\rangle; T'\} \Rightarrow \{s[\hat{\mathtt{p}}] : T, s[\hat{\mathtt{q}}] : T'\}$
- $\{s[\hat{\mathtt{p}}] : \oplus\langle\hat{\mathtt{q}}, \{l_k : T_k\}_{k\in K}\rangle\} \Rightarrow \{s[\hat{\mathtt{p}}] : \oplus\langle\hat{\mathtt{q}}, l_j\rangle; T_j\}$
- $\{s[\hat{\mathtt{p}}] : \oplus\langle\hat{\mathtt{q}}, l_j\rangle; T, s[\hat{\mathtt{q}}] : \&(\mathtt{p}, \{l_k : T_k\}_{k\in K})\} \Rightarrow \{s[\hat{\mathtt{p}}] : T, s[\hat{\mathtt{q}}] : T_j\}$
- $\Delta \cup \Delta'' \Rightarrow \Delta' \cup \Delta''$ if $\Delta \Rightarrow \Delta'$.

The first rule corresponds to the reception of a value or channel by the participant $\hat{\mathtt{q}}$; the second rule treats the case of the choice of label l_j while the third rule propagate these choices to the receiver (participant $\hat{\mathtt{q}}$). Using the above notion we can state type preservation under reductions as follows:

Theorem 3.5 (Subject Congruence and Reduction). *If $\Gamma \vdash P \triangleright \tau$ and $P \longrightarrow^* P'$, then $\Gamma \vdash P' \triangleright \tau'$ for some τ' such that $\tau \Rightarrow^* \tau'$.*

Note that communication safety [13, Theorem 5.5] and session fidelity [13, Corollary 5.6] are corollaries of the above theorem. A notable fact is, in the presence of the asynchronous join primitive, we can still obtain *progress* in a single multiparty session as in [13, Theorem 5.12], i.e. if a program P starts from one session, the reductions at session channels do not get a stuck. Formally we write $\Gamma \vdash^\star P \triangleright \Delta$ if P is typable and with a type derivation where the session typing in the premise and the conclusion of each prefix rule is restricted to at most a singleton. Another element which can hinder progress is when interactions at shared channels cannot proceed. We say P is *well-linked* when for each $P \longrightarrow^* Q$, whenever Q has an active prefix whose subject is a (free or bound) shared channels, then it is always reducible. The proof is similar to [13, Theorem 5.12].[4]

Theorem 3.6 (Progress). *If P is well-linked and without any element from the runtime syntax and $\Gamma \vdash^\star P \triangleright \emptyset$. Then for all $P \longrightarrow^* Q$, either $Q \equiv \mathbf{0}$ or $Q \longrightarrow R$ for some R.*

3.6 Typing Examples

Repetition example - § 1 (1). This example illustrates the repetition of a message pattern. Let $G(n) = \mathtt{foreach}(i < n)\{\mathtt{Alice} \to \mathtt{Bob}\colon\langle \mathsf{nat}\rangle.\mathtt{Bob} \to \mathtt{Carol}\colon\langle\mathsf{nat}\rangle\}$. Following the projection from figure 7, Alice's end-point projection of $G(n)$ is[5]:

$$G(n) \upharpoonright \mathtt{Alice} = \mathbf{R}\ \mathsf{end}\ \lambda i.\lambda \mathbf{x}.!\langle \mathtt{Bob}, \mathsf{nat}\rangle; \mathbf{x}\ n$$

Let $\mathtt{Alice}(n) = \bar{a}[\mathtt{Alice}, \mathtt{Bob}, \mathtt{Carol}](y).(\mathbf{R}\ \mathbf{0}\ \lambda i.\lambda X.y!\langle \mathtt{Bob}, e[i]\rangle; X\ \ n)$ and let $\Delta(n) = \{y : (G(n) \upharpoonright \mathtt{Alice})\}$ and $\Gamma = n : \mathsf{nat}, a : \langle G\rangle$. We can prove that $\Gamma \vdash \mathtt{Alice}(n) \triangleright \emptyset$ from [TINIT] if we have $\Gamma \vdash \mathbf{R}\ \mathbf{0}\ \lambda i.\lambda X.y!\langle \mathtt{Bob}, e[i]\rangle; X\ n \triangleright \Delta(n)$. This, in turn, is given by [TPREC] and [TAPP] from $\Gamma, i : I^-, X : \Delta(i) \vdash y!\langle \mathtt{Bob}, e[i]\rangle; X \triangleright \Delta(i+1)$ and the trivial $\Gamma \vdash \mathbf{0} \triangleright y : \mathsf{end}$. From [TVAR], we have $\Gamma, i : I^-, X : \Delta(i) \vdash X \triangleright \Delta(i)$. We conclude by [TOUT] and weak head normal form equivalence [WF] of the types $\Delta(i+1)$ and $y\ \ :!\langle \mathtt{Bob}, \mathsf{nat}\rangle; (\mathbf{R}\ \mathsf{end}\ \lambda j.\lambda \mathbf{x}.!\langle \mathtt{Bob}, \mathsf{nat}\rangle; \mathbf{x}\ \ i)$. $\mathtt{Bob}(n)$ and $\mathtt{Carol}(n)$ can be similarly typed.

Sequence example - § 1 (2). The sequence example consists of three roles (when $n \geq 2$): the starter $\mathtt{W}[n]$ sends the first message, the final worker $\mathtt{W}[0]$ receives the final message and the middle workers first receive a message and then send another to the next worker. We write below the generic projection for participant $\mathtt{W}[\mathtt{p}]$ (left) and the end-point type that naturally types the processes (right):

$\mathbf{R}\ \mathsf{end}\ \lambda i.\lambda \mathbf{x}.$ $\ \mathsf{if}\ \ \mathtt{p} = \mathtt{W}[i+1]\ \mathsf{then}\ !\langle \mathtt{W}[i], \mathsf{nat}\rangle; \mathbf{x}$ $\ \mathsf{else\ if}\ \ \mathtt{p} = \mathtt{W}[i]\ \mathsf{then}\ ?\langle \mathtt{W}[i+1], \mathsf{nat}\rangle; \mathbf{x}$ $\ \mathsf{else}\ \mathbf{x} \qquad n$	$\mathsf{if}\ \ \mathtt{p} = \mathtt{W}[n]\ \mathsf{then}\ !\langle \mathtt{W}[n-1], \mathsf{nat}\rangle; \mathsf{else}$ $\mathsf{if}\ \ \mathtt{p} = \mathtt{W}[0]\ \mathsf{then}\ ?\langle \mathtt{W}[1], \mathsf{nat}\rangle; \mathsf{else}$ $\mathsf{if}\ \ \mathtt{p} = \mathtt{W}[i]\ \mathsf{then}\ ?\langle \mathtt{W}[i+1], \mathsf{nat}\rangle; !\langle \mathtt{W}[i-1], \mathsf{nat}\rangle;$

In order to type this example, we need to prove the equivalence of these two types. For any instantiation of $\mathtt{p}$ and n, the standard weak head normal form equivalence rule [WF] is sufficient. Proving the equivalence for all $\mathtt{p}$ and n requires either (a) to bound the domain I in which they live, and check all instantiations within this finite domain; or (b) to prove the equivalence through a meta-logic case analysis. In case (a), type checking terminates, while case (b) allows to easily prove strong properties about a protocol's implementation.

[4] We believe a stronger progress property for interleaved multiparty sessions ensured by the interaction typing in [3] can be obtained in this framework, too (since our typing system is an extension from the communication system in [3]).

[5] For readability, in the following examples, we omit from the nested conditionals the impossible cases.

FFT example - Figure 5. We prove type-safety and deadlock-freedom for the FFT processes. Let P_{fft} be the following process:

$$\begin{array}{l}\mathit{\Pi} n.(\nu a)(\mathbf{R}\ \bar{a}[\mathtt{p}_0..\mathtt{p}_{2^n-1}](y).P(2^n-1,\mathtt{p}_0,x_{\overline{\mathtt{p}_0}},y,r_{\mathtt{p}_0})\\ \quad \lambda i.\lambda Y.(\bar{a}[\mathtt{p}_{i+1}](y).P(i+1,\mathtt{p}_{i+1},x_{\overline{\mathtt{p}_{i+1}}},y,r_{\mathtt{p}_{i+1}}) \mid Y)\ 2^n-1)\end{array}$$

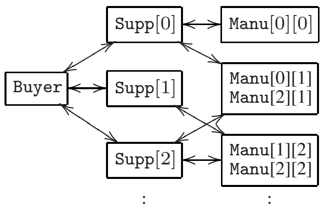
As we reasoned above, each $P(n,\mathtt{p},x_{\overline{\mathtt{p}}},y,r_{\mathtt{p}})$ is straightforwardly typable by an end-point type which is equivalent with the one projected from the global type G from figure 5(c). Automatically checking the equivalence for all n is not easy though: we need to rely on the finite domain restriction using [WF,PREC]. The following theorem says once P_{fft} is applied to a natural number m, its evaluation always terminates with the answer at $r_{\mathtt{p}}$. The proof is by the progress (Theorem 3.5), noting $P_{\text{fft}}\ \mathrm{m}$ is typable by a single, multiparty dependent session (except the answering channel at $r_{\mathtt{p}}$).

Theorem 3.7 (Type safety and deadlock-freedom of FFT). *For all* m, $\emptyset \vdash P_{\mathit{fft}}\ \mathrm{m} \rhd \emptyset$*; and if* $P_{\mathit{fft}}\ \mathrm{m} \longrightarrow^* Q$*, then* $Q \longrightarrow^* (r_0!\langle 0, X_0\rangle \mid \ldots \mid r_{2^{\mathrm{m}}-1}!\langle 0, X_{2^{\mathrm{m}}-1}\rangle)$ *where the* $r_{\mathtt{p}}!\langle 0, X_{\mathtt{p}}\rangle$ *are the actions sending the final values* $X_{\mathtt{p}}$ *on external channels* $r_{\mathtt{p}}$.

Web Service example - Figure 9. We program and type a real-world Web service use case: Quote Request (C-U-002) is the most complex scenario described in [19], the public document authored by the W3C Choreography Description Language Working Group [21]. As described in Figure 9, a buyer interacts with multiple suppliers who in turn interact with multiple manufacturers in order to obtain quotes for some goods or services. The Requirements from Section 3.1.2.2 of [19] include the ability to *reference a global description from within a global description* to support *recursive behaviour* as denoted in STEP 4(b, d): this can be achieved by parameterised multiparty session types.

We write the specification of the usecase program modularly, starting from the first steps of the informal description above. Here, `Buyer` stands for the buyer, $\mathtt{Supp}[i]$ for a supplier, and $\mathtt{Manu}[j]$ for a manufacturer. We alias manufacturers by $\mathtt{Manu}[i][j]$ to express the fact that $\mathtt{Manu}[j]$ is connected to $\mathtt{Supp}[i]$ (a single $\mathtt{Manu}[j]$ can have multiple aliases $\mathtt{Manu}[i'][j]$, see figure 9). Then, we can write global types for each of the steps. STEP 1 is a simple *multicast* (type G_1). For STEP 2, we write first $G_2(i)$, the nested interaction loop between the i-th supplier and its manufacturers (J_i gives all $\mathtt{Manu}[j]$ connected to $\mathtt{Supp}[i]$). Then G_2 can describe the subsequent action within the main loop. For STEP 3, for simplicity we assume the preference is given by the (reverse) ordering of I. The first choice of G_3 corresponds to the two cases of STEP 3. In the innermost branch of G_3, the branches ok, retryStep3 and reject correspond to STEP 4(a), (b) and (c) respectively, while the type variable **t** models STEP 4(d). We can now compose these subprotocols together. The full global type is then $G = \mathit{\Pi} i.\mathit{\Pi}\tilde{J}.(G_1\ ;\ \mu\mathbf{t}.(G_2\ ;\ G_3))$ where we have i suppliers, and $\tilde{J}$ gives the index sets J_i of the $\mathtt{Manu}[j]$s connected with each $\mathtt{Supp}[i]$.

For the end-point projection, we focus on the suppliers' case. The projections of G_1 and G_2 are straightforward. For $G_3 \upharpoonright \mathtt{Supp}[\mathrm{n}]$, we use the branching injection and mergeability theory developed in § 3.1. After the relevant application of $\lfloor\text{TEQ}\rfloor$, we



1. A buyer requests a quote from a set of suppliers.
$G_1 = \mathtt{foreach}(i < n)\{\mathtt{Buyer} \to \mathtt{Supp}[i] : \langle \mathsf{Quote} \rangle\}$

2. All suppliers receive the request to ask their respective manufacturers for a bill of material items. The suppliers interact with their manufacturers to build their quotes for the buyer.
$G_2(i) = \mathtt{foreach}(j : J_i)\{\mathtt{Supp}[i] \to \mathtt{Manu}[i][j] : \langle \mathsf{Item} \rangle. \; \mathtt{Manu}[i][j] \to \mathtt{Supp}[i] : \langle \mathsf{Quote} \rangle\}$
The eventual quote is sent back to the buyer.
$G_2 = \mathtt{foreach}(i : I)\{G_2(i); \mathtt{Supp}[i] \to \mathtt{Buyer} : \langle \mathsf{Quote} \rangle\}$

3. EITHER
   (a) The buyer agrees with one or more of the quotes and places the order(s). OR
   (b) The buyer responds to one or more of the quotes by modifying and sending them back to the relevant suppliers.

4. EITHER
   (a) The suppliers respond to a modified quote by agreeing to it and sending a confirmation message back to the buyer. OR
   (b) The supplier responds by modifying the quote and sending it back to the buyer and the buyer goes back to STEP 3. OR
   (c) The supplier responds to the buyer rejecting the modified quote. OR
   (d) The quotes from the manufacturers need to be renegotiated by the supplier. Go to STEP 2.

$$\begin{array}{l} G_3 = \mathbf{R}\ \mathbf{t}\ \lambda i.\lambda \mathbf{y}.\mathtt{Buyer} \to \mathtt{Supp}[i] : \{ \\ \quad \mathsf{ok} : \quad \mathsf{end} \\ \quad \mathsf{modify} : \mathtt{Buyer} \to \mathtt{Supp}[i] : \langle \mathsf{Quote} \rangle. \\ \qquad \mathtt{Supp}[i] \to \mathtt{Buyer} : \{ \mathsf{ok} : \quad \mathsf{end} \\ \qquad\qquad \mathsf{retryStep3} : \mathbf{y} \\ \qquad\qquad \mathsf{reject} : \quad \mathsf{end}\}\}\ i \end{array}$$

$$\begin{array}{l} G_3 \upharpoonright \mathtt{Supp}[\mathrm{n}] = \&\langle \mathtt{Buyer}, \{ \\ \quad \mathsf{ok} : \quad \mathsf{end} \\ \quad \mathsf{modify} : ?\langle \mathtt{Buyer}, \mathsf{Quote} \rangle; \oplus \langle \mathtt{Buyer}, \{ \\ \qquad \mathsf{ok} : \quad \mathsf{end} \\ \qquad \mathsf{retryStep3} : \mathbf{y} \\ \qquad \mathsf{reject} : \quad \mathsf{end}\}\rangle\}\rangle \end{array}$$

Fig. 9. The Quote Request use case (C-U-002) [19] with the corresponding global types

can obtain the projection written in Figure 9. To tell the other suppliers whether the loop is being reiterated or if it is finished, we can simply insert the following closing notification $\mathtt{foreach}(j \in I \setminus i)\{\mathtt{Buyer} \to \mathtt{Supp}[j] : \{\mathsf{close} :\}\}$ before each end, and a similar retry notification (with label $\mathsf{retryStep3}$) before $\mathbf{x}$. Finally, each end-point type is formed by $(G_1 \upharpoonright \mathtt{Supp}[\mathrm{n}]\,;\, \mu\mathbf{x}.G_2 \upharpoonright \mathtt{Supp}[\mathrm{n}]\,;\; G_3 \upharpoonright \mathtt{Supp}[\mathrm{n}])$. While the global types look sequential, actual typed processes can asynchronously join a session and be executed in parallel (e.g., at STEP 1-2, no synchronisation is needed between $\mathtt{Supp}[i]$).

We have explored the impact of parameterised type structures for communications through implementations of the above use case and of several parallel algorithms in Java with session types [14], including the Jacobi method (with sequence and mesh topologies) and the FFT. We observe (1) a clear coordination of the communication behaviour of each party with the construction of the whole multiparty protocol, thus reducing the programming errors and ensuring deadlock-freedom; and (2) a performance benefit against the original binary session version, reducing the overhead of multiple binary session establishments (see [11]). Full implementation and integration of our theory into [4, 14] is on-going work.

4 Related Work

Dependent types. The first use of primitive recursive functionals for dependent types is in Nelson's $\mathcal{T}^\pi$ [18] for the λ-calculus, which is a finite representation of $\mathcal{T}^\infty$ by Tait and Martin Löf [16, 20]. $\mathcal{T}^\pi$ can type functions previously untypable in ML, and the finite representability of dependent types makes it possible to have a type-reconstruction algorithm. We also use the ideas from the DML's dependent typing system in [1, 22] where type dependency is only allowed for index sorts, so that type-checking can be reduced to a constraint-solving problem over indices. Our design choice to combine both systems gives (1) the simplest formulation of sequences of global and end-point types and processes described by the primitive recursor; (2) a precise specification for parameters appearing in the participants based on index sorts; and (3) a clear integration with the full session types and general recursion, whilst ensuring decidability of type-checking (if the constraint-solving problem is decidable). From the basis of these works, our type equivalence does not have to rely on behavioural equivalence between processes, but only on the strongly normalising *types* represented by recursors. None of these works investigate families of global specifications using dependent types.

Types and contracts for multiparty interactions. The work [7] presented an *executable global processes* for Web interactions based on the binary session types. Our work provides flexible, programmable global descriptions as *types*, offering a progress for parameterised multiparty session, which is not ensured in [7]. Recent formalisms for typing multiparty interactions include [6, 9]. These works treat different aspects of dynamic session structures. *Contracts* [9] can type more processes than session types, thanks to the flexibility of process syntax for describing protocols. However, typable processes themselves in [9] may not always satisfy the properties of session types such as progress: it is proved later by checking whether the type meets a certain form. Hence proving progress with contracts effectively requires an exploration of all possible paths (interleaving, choices) of a protocol. The most complex example of [9, § 3] (a group key agreement protocol from [2]), which is typed as π-processes with delegations, can be specified and articulated by a single parameterised global session type as:

$$\begin{aligned}\Pi n{:}I.(&\mathtt{foreach}(i<n)\{\mathtt{W}[n-i]\to\mathtt{W}[n-i+1]{:}\langle\mathsf{nat}\rangle\};\\ &\mathtt{foreach}(i<n)\{\mathtt{W}[n-i]\to\mathtt{W}[n+1]{:}\langle\mathsf{nat}\rangle.\mathtt{W}[n+1]\to\mathtt{W}[n-i]{:}\langle\mathsf{nat}\rangle\})\end{aligned}$$

Once the end-point process conforms to this specification, we can automatically guarantee communication safety and progress.

Conversation Calculus [6] supports the dynamic joining and leaving of participants. Though the formalism in § 2.2 can operationally capture such dynamic features, the aim of the present work is *not* the type-abstraction of dynamic interaction patterns. Our purpose is to capture, in a single type description, a family of protocols over arbitrary number of participants, to be instantiated at runtime. The parameterisation gives freedom not possible with previous session types: once typed, a parametric process is ensured that its arbitrary well-typed instantiations, in terms of both topologies and process behaviours, satisfy the safety and progress properties of typed processes. Parameterisation, composition and repetition are common idioms in parallel algorithms and choreographic/conversational interactions, all of which are uniformly treatable in our dependent type theory. Here types offer a rigorous structuring principle which can economically abstract rich interaction structures, including parameterised ones.

Acknowledgements. We thank the reviewers, Lasse Nielsen and Dimitris Mostrous for their useful comments for the paper and Kohei Honda for discussions.

References

1. Aspinall, D., Hofmann, M.: Dependent Types. In: Advanced Topics in Types and Programming Languages. MIT Press, Cambridge (2005)
2. Ateniese, G., Steiner, M., Tsudik, G.: Authenticated group key agreement and friends. In: CCS 1998: Proceedings of the 5th ACM conference on Computer and communications security, pp. 17–26. ACM, New York (1998)
3. Bettini, L., et al.: Global progress in dynamically interfered multiparty sessions. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 418–433. Springer, Heidelberg (2008)
4. Bhargavan, K., Corin, R., Deniélou, P.-M., Fournet, C., Leifer, J.: Cryptographic protocol synthesis and verification for multiparty sessions. In: CSF, pp. 124–140 (2009)
5. Bonelli, E., Compagnoni, A.: Multipoint session types for a distributed calculus. In: Barthe, G., Fournet, C. (eds.) TGC 2007 and FODO 2008. LNCS, vol. 4912, pp. 240–256. Springer, Heidelberg (2008)
6. Caires, L., Vieira, H.T.: Conversation types. In: Castagna, G. (ed.) ESOP 2009. LNCS, vol. 5502, pp. 285–300. Springer, Heidelberg (2009)
7. Carbone, M., Honda, K., Yoshida, N.: Structured communication-centred programming for web services. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 2–17. Springer, Heidelberg (2007)
8. Carbone, M., Yoshida, N., Honda, K.: Asynchronous session types: Exceptions and multiparty interactions. In: Bernardo, M., Padovani, L., Zavattaro, G. (eds.) SFM 2009. LNCS, vol. 5569, pp. 187–212. Springer, Heidelberg (2009)
9. Castagna, G., Padovani, L.: Contracts for mobile processes. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009 - Concurrency Theory. LNCS, vol. 5710, pp. 211–228. Springer, Heidelberg (2009)
10. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex fourier series. Mathematics of Computation 19(90), 297–301 (1965)
11. Online Appendix, http://www.doc.ic.ac.uk/~yoshida/dependent/
12. Girard, J.-Y., Lafont, Y., Taylor, P.: Proofs and Types. Cambridge Tracts in Theoretical Computer Science, vol. 7. CUP, Cambridge (1989)
13. Honda, K., Yoshida, N., Carbone, M.: Multiparty Asynchronous Session Types. In: POPL 2008, pp. 273–284. ACM, New York (2008)
14. Hu, R., Yoshida, N., Honda, K.: Session-Based Distributed Programming in Java. In: Vitek, J. (ed.) ECOOP 2008. LNCS, vol. 5142, pp. 516–541. Springer, Heidelberg (2008)
15. Leighton, F.T.: Introduction to parallel algorithms and architectures: arrays, trees, hypercubes. Morgan Kaufmann, San Francisco (1991)
16. Martin-Lőf, P.: Infinite terms and a system of natural deduction. In: Compositio Mathematica, pp. 93–103. Wolters-Noordhoof (1972)
17. Mostrous, D., Yoshida, N., Honda, K.: Global principal typing in partially commutative asynchronous sessions. In: Castagna, G. (ed.) ESOP 2009. LNCS, vol. 5502, pp. 316–332. Springer, Heidelberg (2009)

18. Nelson, N.: Primitive recursive functionals with dependent types. In: Schmidt, D., Main, M.G., Melton, A.C., Mislove, M.W., Brookes, S.D. (eds.) MFPS 1991. LNCS, vol. 598, pp. 125–143. Springer, Heidelberg (1992)
19. Web Services Choreography Requirements (No. 11), `http://www.w3.org/TR/ws-chor-reqs`
20. Tait, W.W.: Infinitely long terms of transfinite type. In: Formal Systems and Recursive Functions, pp. 177–185. North Holland, Amsterdam (1965)
21. Web Services Choreography Working Group. Choreography Description Language, `http://www.w3.org/2002/ws/chor/`
22. Xi, H., Pfenning, F.: Dependent types in practical programming. In: POPL, pp. 214–227 (1999)

On the Relationship between Spatial Logics and Behavioral Simulations[⋆]

Lucia Acciai[1], Michele Boreale[1], and Gianluigi Zavattaro[2]

[1] Dipartimento di Sistemi e Informatica, Università di Firenze, Italy
[2] Dipartimento di Scienze dell'Informazione, Università di Bologna, Italy

Abstract. Spatial logics have been introduced to reason about distributed computation in models for concurrency. We first define a spatial logic for a general class of infinite-state transition systems, the *Spatial Transition Systems* (STS), where a monoidal structure on states accounts for the spatial dimension. We then show that the model checking problem for this logic is undecidable already when interpreted over Petri nets. Next, building on work by Finkel and Schnöbelen, we introduce a subclass of STS, the *Well-Structured* STS (WS-STS), which is general enough to include such models as Petri nets, Broadcast Protocols, CCS and Weighted Automata. Over WS-STS, an interesting fragment of spatial logic - the *monotone* fragment - turns out to be decidable under reasonable effectiveness assumptions. For this class of systems, we also offer a Hennessy-Milner theorem, characterizing the logical preorder induced by the monotone fragment as the largest *spatial-behavioural* simulation. We finally prove that, differently from the corresponding logic, this preorder is in general not decidable, even when confining to effective WS-STS.

1 Introduction

Spatial logics [6, 7] are modal logics for describing the behavior and spatial structure of concurrent systems. Beside propositional and temporal operators, they include spatial operators, the most prominent of which is $\_|\_$, having the following meaning: the formula $\phi_1|\phi_2$ is satisfied by any process that can be decomposed into two processes that satisfy respectively ϕ_1 and ϕ_2. Spatial logics have been applied to several models, such as the pi-calculus [7] and the Ambient Calculus [9].

Starting from the well-known correspondence between the Hennessy-Milner logic and bisimulation [19], a rich literature has been dedicated to the study of the relationship between modal logics and behavioural equivalences or preorders. In the realm of spatial logics, this study has been undertaken in [20, 24], in the case of the Ambient Calculus, and in [5, 8], in the case of the CCS and pi-calculus. A discussion on these works, which are strongly related to our study, is deferred to the concluding section. The objective of the present paper is to start such an investigation in a *general* setting. To this aim, we introduce the notion of *spatial transition system* (STS): a possibly infinite-state transition system, endowed with a monoidal structure on states representing the spatial

[⋆] Research partly supported by the EU within the FET-GC2 initiative, project SENSORIA. The third author is partially supported by the EU integrated project HATS.

dimension. We introduce a very simple spatial logic $\mathcal{L}$, consisting of atomic predicates, the and/or/not logical operators, a behavioral modality indicating the possibility to reach a state with a given property, and the spatial operator described above. We then interpret $\mathcal{L}$ over STS's, relating the spatial operator to the monoidal structure. On the one hand, STS are general enough to include models such as the Calculus of Communicating Systems (CCS, [22]), Petri nets (and variants thereof, such as reset nets, transfer nets and broadcast protocols), and weighted automata. On the other hand, even if $\mathcal{L}$ is very simple, it is enough expressive to describe interesting properties, such as the impossibility for a CCS process to reach a state where a race condition on a channel arises, or *one-safety* on Petri nets (i.e. all places in all reachable markings contain at most one token).

Our first result is that model checking of $\mathcal{L}$ is indeed in general undecidable in infinite state systems, even for quite simple STS like Petri nets. This leads us to considering the negation-free fragment of $\mathcal{L}$, written $\mathcal{L}_0$ and introduce the class of *Well-Structured Spatial Transition Systems* (WS-STS). The latter builds on the class of Well-Structured Transition Systems introduced by Finkel and Schnöbelen [17]; in particular, the existence of a well-quasi order on states that is compatible with both the transition and the monoidal structure of the system plays a crucial role. The class of WS-STS is still general enough to include all the models mentioned above. We prove that $\mathcal{L}_0$ is decidable for the class of WS-STS, subject to some reasonable effectiveness conditions.

Next, we characterize the logical preorder induced by $\mathcal{L}_0$, that is, the preorder that relates s to t whenever t satisfies all the $\mathcal{L}_0$-formulae satisfied by s. We present a coinductive characterization of the logical preorder in terms of a (weak) simulation, enriched with constraints on the spatial properties of s and t and the basic predicates they satisfy (a Hennessy-Milner theorem for $\mathcal{L}_0$). This simulation, that we call *spatial-behavioral* preorder (SBS), is, in fact, the largest well-quasi order that is compatible with the spatial and transition structure of the system.

Finally, we show that, differently from $\mathcal{L}_0$, this preorder is in general not decidable, even restricting to effective WS-STS.

Structure of the paper. In Section 2 we define STS and SBS and introduce some concrete instances that will be used throughout the paper. In Section 3 we introduce $\mathcal{L}$ and its fragment $\mathcal{L}_0$, the considered spatial logics, and we prove the undecidability of $\mathcal{L}$ in Petri nets and CCS. Section 4, after introducing some background material and defining the class of effective WS-STS, discusses the decidability of $\mathcal{L}_0$. In Section 5 we prove that the preorder induced by $\mathcal{L}_0$ coincides with the largest SBS, and in Section 6 we prove that the latter is not decidable in WS-STS in general. A few remarks on further and related work conclude the paper in Section 7. Due to lack of space, some proofs have been left out of this short version (they can be found in [1]).

2 Spatial Transition Systems

In this section we introduce spatial transition systems, we provide some instances that will be used as running examples throughout the paper, and we introduce a natural extension of weak simulation for spatial transition systems.

2.1 Basic Definitions

Recall that a *transition system* (TS) is a structure $(S, \rightarrow)$ where S, ranged over $s, t, \ldots$, is a set of states and $\rightarrow \subseteq S \times S$ is a set of transitions. We let $\rightarrow^*$ be the reflexive and transitive closure of $\rightarrow$. The set of *immediate predecessors* and *predecessors* of a state $s \in S$ are defined respectively as $\mathsf{Pred}(s) = \{t \,|\, t \rightarrow s,\ t \in S\}$ and $\mathsf{Pred}^*(s) = \{t \,|\, t \rightarrow^* s,\ t \in S\}$. The definitions of Pred and Pred^* are extended to sets of states as expected. In the following, we let At be a finite set of *atomic predicates* ranged over $\mathrm{p}, \mathrm{p}', \ldots$. We let $\mathcal{P}(\mathrm{At})$ denote the powerset of At. Recall that a *monoid* $(M, \oplus, 0_M)$ is a semigroup with 0_M as an identity element.

Definition 1 (spatial transition system). *A* spatial transition system *(*STS*) is a tuple* $\mathcal{S} = (S, \rightarrow, \oplus, \mathrm{O})$ *where: (1)* $(S, \rightarrow)$ *is a transition system, (2)* $(S, \oplus, 0_S)$ *is a monoid for some* $0_S \in S$*, and (3)* $\mathrm{O} : S \rightarrow \mathcal{P}(\mathrm{At})$ *is an* observation function.

The relationship among the transition system, the monoid and the observation function is given by the following *spatial-behavioral simulation*.

Definition 2 (spatial-behavioral simulation). *A* spatial-behavioral simulation *(*SBS*) over a* STS $\mathcal{S} = (S, \rightarrow, \oplus, \mathrm{O})$ *is a binary relation* $\mathcal{R} \subseteq S \times S$ *such that whenever* $s \mathcal{R} t$ *then:*

1. *whenever* $s \rightarrow s'$ *then there exists* $t' \in S$ *such that* $t \rightarrow^* t'$ *and* $s' \mathcal{R} t'$*;*
2. *whenever* $s = s_1 \oplus s_2$ *then there are* $t_1, t_2 \in S$ *such that* $t = t_1 \oplus t_2$ *and* $s_i \mathcal{R} t_i$*,* $i = 1, 2$*;*
3. $\mathrm{O}(s) \subseteq \mathrm{O}(t)$.

The largest SBS*, denoted* $\sqsubseteq$*, is a preorder over* S*, called* spatial-behavioral preorder.

2.2 Concrete Instances of STS

Calculus of Communicating Systems. The fragment of CCS with input guarded replication[1] instead of recursion can be turned into a STS as detailed in [4], that is, working modulo structural congruence and identifying $\oplus$ and its identity with parallel composition $|$ and $\mathbf{0}$, respectively. Moreover, the relation $\preceq$ introduced in Definition 14 of [4] can be easily proved to be a spatial-behavioral simulation w.r.t. this STS.

Affine Well-Structured Nets. We consider a generalization of Petri nets (PN) introduced by Finkel et al.: as discussed in Section 7.2 of [16], PN, Double PN, Generalized Transfer PN (thus also Broadcast Protocols [12]) and Reset PN are all instances of affine WSN. Let $\mathbb{N}^p$, for $p \geq 1$, be ranged over by $n, m, \ldots$. Let $m(i)$ denote the i^{th} component of m. The preorder $\preceq \subseteq \mathbb{N}^p \times \mathbb{N}^p$ is defined point-wise as expected: $n \preceq m$ if and only if for each $i = 1, \ldots, p$, $n(i) \leq m(i)$. A *well-structured net* (WSN) is a triple $N = (\mathbb{N}^p, F, \preceq)$, where $\mathbb{N}^p$ is the set of states (in the PN terminology, *markings* on the *places* $1, \ldots, p$) and F is a finite set of *partial* functions $f, f', \ldots : \mathbb{N}^p \rightarrow \mathbb{N}^p$, whose domain is an upward-closed subset of $\mathbb{N}^p$ – that is, whenever $m \preceq n$ and $m \in \mathrm{dom}(f)$ then $n \in \mathrm{dom}(f)$, for each $f \in F$. An *affine* WSN is a WSN where for each function $f \in F$ there is a square matrix $A \in \mathbb{N}^{p \times p}$

[1] Intuitively, the replicated process $!a.P$ corresponds to an infinite number of copies of $a.P$ in parallel: $a.P | \cdots | a.P | \cdots$.

and a vector $B \in \mathbb{Z}^p$ such that for each $m \in \mathrm{dom}(f)$, $f(m) = A \cdot m + B$ with m, B seen as column vectors. Let us fix an affine WSN N. Clearly, $(\mathbb{N}^p, +, 0^p)$, with $+$ indicating sum of vectors, is a monoid. Define At to be $\{1, \dots, p\}$ and, for any $n \in \mathbb{N}^p$, $\mathrm{O}(n) = \{i \mid n(i) \neq 0\}$. Finally, define the transition relation: $n \rightarrow_F m$ iff $f(n) = m$ for some $f \in F$ (see [16]). Clearly, $(\mathbb{N}^p, \rightarrow_F, +, \mathrm{O})$ is a STS for which $\preceq$ is a SBS (due to the fact that each $f \in F$ is monotone). Moreover, $\preceq$ is also the spatial-behavioral preorder (i.e. the largest SBS).

There are interesting variations of this construction, corresponding to choosing different observation functions O. For instance, one can leave the set At unspecified, fix a labelling function from places to sets of atomic predicates, $l : \{1, ..., p\} \rightarrow \mathcal{P}(\mathrm{At})$, and then let $\mathrm{O}(m) = \bigcup_{i \in 1..p : m(i)>0} l(i)$. Yet another possibility is observing enabled transitions in the current state: assuming $F = \{f_i \mid i \in I\}$, this is obtained by letting $\mathrm{At} = I$ and $\mathrm{O}(m) = \{i \in I \mid m \in \mathrm{dom}(f_i)\}$. These variations too give rise to STS for which $\preceq$ is a SBS, but in general *not* the largest one.

Weighted Automata. Recall that a *semiring* $\mathbb{K}$ is a structure $(K, +, \times, 0, 1)$ such that $(K, +, 0)$ is a commutative monoid, $(K, \times, 1)$ is a (not necessarily commutative) monoid, $\times$ distributes over $+$ both to the left and to the right and 0 annihilates both to the left and to the right (i.e., $0 \times a = a \times 0 = 0$ for each $a \in K$). Now let us fix a semiring $\mathbb{K}$ and consider the following preorder over K: $a \preceq b$ iff there is $c \in K$ s.t. $a + c = b$. With these definitions, the construction illustrated above for Affine Nets carries over formally unchanged when replacing both $\mathbb{N}$ and $\mathbb{Z}$ by $\mathbb{K}$. Doing so, we cast (a generalization of) *Weighted Automata* (WA, [21]) into the framework of STS[2]. Concrete instances are: $\mathbb{K} = \mathbb{N}$, $\mathbb{K} = \mathbb{Q}^+$ (positive rationals) and $\mathbb{K} = \mathbb{R}^+$ (positive reals, hence e.g. finite-state Markov chains). Another instance is the $(\max, +)$ (a.k.a. tropical semiring [25]) used in quantitative evaluation of discrete-time systems [3]. The latter is defined over $\mathbb{N} \cup \{\infty\}$, by letting "+" to be max and "$\times$" to be +.

3 The Logics $\mathcal{L}$ and $\mathcal{L}_0$

3.1 Definitions and Examples

Definition 3. *The set $\mathcal{L}$ of logic formulae $\phi, \psi, \dots$ is given by the following syntax, where* $\mathrm{p} \in \mathrm{At}$*:* $\phi ::= \mathrm{p} \mid \phi|\phi \mid \diamond^* \phi \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi$.

The set of logical operators includes spatial modalities (atomic predicates $\mathrm{p} \in \mathrm{At}$, the composition operator "|"), dynamic connectives (the eventuality modality $\diamond^*$), and the usual boolean connectives ($\neg, \wedge, \vee$). The interpretation of $\mathcal{L}$ over a STS is given below.

$$[\![\mathrm{p}]\!] = \{s \in S \mid \mathrm{p} \in \mathrm{O}(s)\} \qquad [\![\diamond^*\phi]\!] = \mathsf{Pred}^*([\![\phi]\!]) \qquad [\![\neg\phi]\!] = S \setminus [\![\phi]\!]$$
$$[\![\phi_1 \vee \phi_2]\!] = [\![\phi_1]\!] \cup [\![\phi_2]\!] \qquad [\![\phi_1 \wedge \phi_2]\!] = [\![\phi_1]\!] \cap [\![\phi_2]\!]$$
$$[\![\phi_1|\phi_2]\!] = \{s_1 \oplus s_2 \mid s_j \in [\![\phi_j]\!] \text{ for } j = 1,2\}$$

Connectives are interpreted as expected. In particular, satisfiability of the basic predicates is given by O while satisfiability of the composition operator relies on the possibility of decomposing states via $\oplus$. In what follows we usually write $s \models \phi$ if $s \in [\![\phi]\!]$.

[2] Concretely, states $s, t, ...$ of the WA are elements of $\mathbb{K}^{p \times 1}$, and $s \rightarrow t$ iff $t = A \cdot s$, where $A \in \mathbb{K}^{p \times p}$ is the transition matrix of the WA.

Following [2], we say that a formula is *monotone* if it does not contain any occurrence of $\neg$. We let $\mathcal{L}_0$ denote the subset of monotone formulae. A formula is *anti-monotone* if it is of the form $\neg\phi$ with ϕ monotone.

Example 1. The following formulae for CCS depends on generic names a and b. The anti-monotone formula $\phi = \neg\Diamond^*(\overline{a} \wedge \overline{b})$ says that the output on a and b are forever mutually exclusive; e.g. $\overline{a}+\overline{b} \not\models \phi$. This is different from $\psi = \neg\Diamond^*(\overline{a}|\overline{b})$, still anti-monotone, saying that it will never be the case that there are two independent threads in the system offering an output on a and one on b; e.g. $\overline{a}+\overline{b} \models \psi$, while $\overline{a}|\overline{b} \not\models \psi$.

Similar properties can be defined in the case of WSN. Let $i, j \in \{1,\dots,p\}$. The formula $\neg\Diamond^*(i \wedge j)$ says that no marking is reachable where place i and place j are non-empty. This is equivalent to $\neg\Diamond^*(i\,|\,j)$ if $i \neq j$, while $\neg\Diamond^* \bigvee_{l=1}^{p}(l\,|\,l)$ says that the net is *one-safe*. Switching the alternative interpretation where enabled transitions are observed, the formula $\neg\Diamond^*(i \wedge j)$ would say that no marking is reachable where both f_i and f_j are enabled. This is now *stronger* than $\neg\Diamond^*(i\,|\,j)$, saying that no marking is reachable where f_i and f_j can both fire simultaneously. The logic can also be used to define properties that go beyond pure coverability, such as $(\Diamond^* a)|(\Diamond^* b)$, saying that, from the current state, two non conflicting transitions on a and b can be reached. These formulae also make sense in the case of WA. As an example, in a Markov chain, $\neg\Diamond^*(i\,|\,j)$ says that it is not possible to reach a state where both transition i and transition j have non-zero probability.

3.2 Undecidability of $\mathcal{L}$

We now prove that the logic $\mathcal{L}$ turns out to be undecidable already for Petri nets PN, one of the simplest instantiations of affine WSN in which the matrix A, used in the definitions of the functions f, always corresponds with the identity matrix.

The proof is by reduction from the *containment problem* for PN. An instance of the containment problem consists of two PN Σ_1, Σ_2 with the same number of places, and a bijection g between the sets of places of Σ_1 and Σ_2 (g, called *renaming* in the following, is extended to a bijection between markings in the obvious way). The problem consists of checking whether for every reachable marking m of Σ_1, $g(m)$ is reachable in Σ_2. Rabin showed that this problem is undecidable (the proof is in [18]). Following the approach used in [13] to prove the undecidability of the modal μ-calculus for PN, we reduce the containment problem to the problem of model checking a given formula in a PN.

Assume that the number of places of Σ_1 and Σ_2 is p. We now define a PN $[[\Sigma_1,\Sigma_2]] = (\mathbb{N}^{2p+4}, F, \leq)$ where $\leq$ is the usual ordering on naturals extended to vectors. The states of $[[\Sigma_1,\Sigma_2]]$ are vectors of length $2p+4$: the first p elements are used to represent states of Σ_1, the subsequent p elements are used to represent states of Σ_2, while the last 4 elements are used to divide the computations in four distinct phases. The first phase is a simulation of a computation of Σ_1, the second phase is the passage through a specific observable state, the third phase is a simulation of Σ_2, and the last phase is used to check whether the markings reached by the simulations of Σ_1 and Σ_2 correspond up-to renaming.

Formally, the computations of $[[\Sigma_1,\Sigma_2]]$ are controlled by F containing the three classes of functions defined below, where we use $x \cdot y$ to denote the juxtapositions of the vectors x and y:

- F contains the functions $f_1(x) = x + 0^{2p} \cdot (-1,1,0,0)$, $f_2(x) = x + 0^{2p} \cdot (0,-1,1,0)$, and $f_3(x) = x + 0^{2p} \cdot (0,0,-1,1)$;
- for each $f(x) = x + B$ in Σ_1 (resp. Σ_2), F contains a corresponding $f(x) = x + B \cdot 0^{p+4}$ (resp. $f(x) = x + 0^p \cdot B \cdot 0^4$) defined only if $x(2p+1) > 0$ (resp. if $x(2p+3) > 0$);
- for each place s in $\{1,\cdots,p\}$, F contains a function $f(x) = x + B_s$, defined only if $x(2p+4) > 0$, where B_s contains -1 in the positions s and $p + g(s)$, and 0 elsewhere.

Assuming that the initial states of Σ_1 and Σ_2 are respectively m_1 and m_2, we will consider for $[[\Sigma_1,\Sigma_2]]$ the state $m_1 \cdot m_2 \cdot (1,0,0,0)$.

We now discuss the meaning of the three classes of functions defined above. The first three functions dictate the passage from one phase to the subsequent one. In the second class of functions, those of the form $f(x) = x + B \cdot 0^{p+4}$ (resp. $f(x) = x + 0^p \cdot B \cdot 0^4$) are used to mimic the computations of Σ_1 (resp. Σ_2) during the first (resp. the third) phase. The third class of functions reduces in a synchronized manner the values in the places s and $p + g(s)$. In this way, if a state with all the first $2p$ elements equal to 0 is reached, we can conclude that the markings reached during the simulations of the computations of Σ_1 and Σ_2 correspond up-to renaming. We now prove a proposition that formalizes the correctness of the reduction: in the statement of the proposition we exploit the fact that every state x reachable in $[[\Sigma_1,\Sigma_2]]$ such that $x(2p+2) > 0$ is an intermediary state between the simulation of a computation of Σ_1 (performed while the element in position $2p+1$ is 1) and the subsequent simulation of a computation of Σ_2 (performed while the element in position $2p+3$ is 1).

Proposition 1. *Given two Petri nets Σ_1, Σ_2 with initial markings m_1 and m_2, respectively, we have that they satisfy the containment problem iff for every computation of $[[\Sigma_1,\Sigma_2]]$ starting from $m_1 \cdot m_2 \cdot (1,0,0,0)$ and leading to a state x such that $x(2p+2) > 0$, then the computation can be extended in order to reach a state in which the first $2p$ elements are all equal to 0.*

In the light of this theorem we conclude that two Petri nets Σ_1 and Σ_2, with initial markings m_1 and m_2, satisfy the containment problem iff for the PN $[[\Sigma_1,\Sigma_2]]$ we have

$$m_1 \cdot m_2 \cdot (1,0,0,0) \models \neg\diamond^*\neg(\neg(2p+2) \vee (\diamond^* \textstyle\bigwedge_{i\in\{1,\cdots,2p\}} \neg i))$$

from which we get the following result.

Theorem 1. *The model checking of $\mathcal{L}$ is undecidable for* PN.

It is worth noting that the logic $\mathcal{L}$ is undecidable also for the fragment of CCS introduced in Section 2.2. This can be proved as a corollary of an undecidability result in [4]. In that paper (see Section 3) weak bisimulation is proved to be undecidable, for this fragment of CCS, presenting a nondeterministic modeling of Random Access Machines (RAMs) [23], a well known register based Turing complete formalism. The encoding is nondeterministic because it can give rise to computations that do not correspond to the computation of the modeled RAM. Nevertheless, those computations generate subprocesses that performs infinite computations presenting the barb w' infinitely often. So we have that a RAM R terminates iff the corresponding encoding $[[R]]$ in CCS has a computation leading to a *halt* instruction – in the encoding in [4] *halt* instructions present the barb $\overline{w}$ – but without subprocesses left by wrong computations, that is, iff $[[R]] \models \diamond^*(\overline{w} \wedge (\neg\diamond^* w'))$.

4 Decidability of $\mathcal{L}_0$

In this section we show the existence of a significant sub-class of Spatial Transition Systems, that we call effective Well-Structured Spatial Transition Systems (WS-STS), for which the monotone fragment $\mathcal{L}_0$ turns out to be decidable. Basically, WS-STS enhance classical WSTS [17] by taking into account the spatial structure given by the monoid $(S,\oplus,0)$.

4.1 Well-Structured Spatial Transition Systems

Recall that a *quasi-ordering* (QO) (aka *preorder*) over S is a reflexive and transitive relation over S. A *well-quasi-ordering* (WQO) is a QO $\preceq$ over S such that for any infinite sequence $s_1, s_2, \ldots$ in S there exists indexes $i < j$ such that $s_i \preceq s_j$; in other words, S does not have infinite *antichains*. For any QO $\preceq$ over S and $T \subseteq S$, we say $s \in T$ is a *minimal* element of T if for each $t \in T$, $s \preceq t$; we let $\mathrm{Min}(T)$ denote the set of minimal elements of T. A well-structured spatial transition system is a STS equipped with a QO that is compatible with $\to$, $\oplus$ and O.

Definition 4 (well-structured spatial transition system). *A* well-structured spatial transition system (WS-STS) *is a* STS $\mathcal{S} = (S,\to,\oplus,\mathrm{O})$ *equipped with a* WQO $\preceq$ *over* S *satisfying the following conditions: (1)* $\preceq$ *is a* SBS*, (2) whenever* $s \preceq s'$ *then for each* t $s\oplus t \preceq s' \oplus t$ *and* $t \oplus s \preceq t \oplus s'$*, and (3)* $0_S \in \mathrm{Min}(S)$.

Example 2. The concrete instances of STS provided in Section 2.2 are WS-STS. Theorem 6 of [4] proves that $\preceq$ is a WQO over CCS, while clauses (1), (2) and (3) can be easily checked.For what concerns affine WSN, clearly $\preceq$ is a WQO because it is defined as the pointwise extension of $\le$ over $\mathbb{N}$, which is a WQO (Dickson's Lemma [10]). Clause (1) of Definition 4 has been discussed in Section 2.2 and clauses (2) and (3) can be easily checked. Hence, $(\mathbb{N}^p, F, +, \mathrm{O})$ is a WS-STS for any variation of O. Essentially the same reasoning applies to weighted automata. Concrete instances where $\preceq$ is a WQO, hence the construction yields a WS-STS, are $\mathbb{K} = \mathbb{N}$, $\mathbb{K} = \mathbb{R}^+$ and $\mathbb{K}$ =(max,+).

4.2 Decidability of $\mathcal{L}_0$ for Effective WS-STS

Before presenting the technical machinery needed to define effective WS-STS and to prove the decidability of $\mathcal{L}_0$ for this particular class of WS-STS, we present the following lemma stating that $\models$ on monotone formulae is compatible with both SBS and $\oplus$.

Lemma 1. *Let* $(S,\to,\oplus,\mathrm{O})$ *be a* WS-STS *and* $s,t \in S$*. Let* ϕ *be a monotone formula. (1) If* $s \sqsubseteq t$ *and* $s \models \phi$ *then* $t \models \phi$*. (2) If* $s \models \phi$ *then, for each* t *also* $s \oplus t \models \phi$*.*

Let us introduce some auxiliary notations and results first. Given any s in a set X preordered by $\preceq$, we let its *upward-closure* to be $\uparrow s = \{t \mid s \preceq t\}$. This notation is extended to any set $I \subseteq X$ as expected. A set I is *upward-closed* if $\uparrow I = I$. A *finite basis* of an upward-closed set I is a finite set $B \subseteq X$ such that $\uparrow B = I$. If X is a WQO, any upward-closed set I has a finite basis. Indeed, it is enough to choose from $\mathrm{Min}(I)$ one representative of each equivalence class induced by $\preceq$: there must be finitely many such classes, otherwise one could form an antichain. Now, in any WS-STS and for any monotone ϕ, $[\![\phi]\!]$ is upward closed w.r.t. $\preceq$ (Lemma 1(1)). Hence the existence of a finite basis of $[\![\phi]\!]$ is guaranteed. Now we have to show how to build one such basis.

For the rest of the section, let us fix a WS-STS $\mathcal{S}$. We assume three functions b, pb^* and mub, yielding finite bases for certain upward closed sets. Specifically: for each atomic predicate p, $\mathsf{b}(\mathrm{p})$ yields a finite basis of $[\![\mathrm{p}]\!]$, that is $\uparrow \mathsf{b}(\mathrm{p}) = [\![\mathrm{p}]\!]$; for each finite $I \subseteq S$, $\mathsf{pb}^*(I)$ yields a finite basis of $\mathsf{Pred}^*(\uparrow I)$; and for each $s_1, s_2 \in S$, $\mathsf{mub}(s_1, s_2)$ yields a finite basis of $\uparrow s_1 \cap \uparrow s_2$, in other words a set of minimal upper bounds for s_1 and s_2. For the time being, we make no assumption about the effectiveness of these functions. Building on them, a finite basis of $[\![\phi]\!]$, written $\mathsf{Fb}(\phi)$, is defined below by induction on ϕ.

$$\mathsf{Fb}(\mathrm{p}) = \mathsf{b}(\mathrm{p}) \qquad \mathsf{Fb}(\diamond^*\phi) = \mathsf{pb}^*(\mathsf{Fb}(\phi)) \qquad \mathsf{Fb}(\phi_1 \vee \phi_2) = \mathsf{Fb}(\phi_1) \cup \mathsf{Fb}(\phi_2)$$
$$\mathsf{Fb}(\phi_1 \wedge \phi_2) = \bigcup_{s_1 \in \mathsf{Fb}(\phi_1),\ s_2 \in \mathsf{Fb}(\phi_2)} \mathsf{mub}(s_1, s_2) \qquad \mathsf{Fb}(\phi_1 | \phi_2) = \bigcup_{s_1 \in \mathsf{Fb}(\phi_1),\ s_2 \in \mathsf{Fb}(\phi_2)} \{s_1 \oplus s_2\}$$

Proposition 2. *Let ϕ be monotone. Then $\mathsf{Fb}(\phi)$ is a finite basis of* $[\![\phi]\!]$.

In order to prove decidability, we need to argue now about effectiveness of b, pb^* and mub. In particular, we shall rely on Finkel and Schnöbelen's result below that establishes effectiveness of pb^* under certain conditions. Let us define the *pred-basis* of a state s, $\mathsf{pb}(s)$, as the finite basis of $\uparrow \mathsf{Pred}(\uparrow s)$:

$$\uparrow \mathsf{pb}(s) = \uparrow \mathsf{Pred}(\uparrow s) = \{t \,|\, t \succeq \rightarrow \succeq s\}.$$

Let us say that a WS-STS has an *effective pred-basis* if $\mathsf{pb}(\cdot)$ is computable. We say a WSTS $\mathcal{S}$ is *effective* if it has effective pred-basis and $\preceq$ is decidable.

Proposition 3 (Proposition 3.5 [17]). *If $\mathcal{S}$ is an effective* WSTS*, then it is possible to effectively compute a finite basis of $\mathsf{Pred}^*(\uparrow I)$, for any finite $I \subseteq S$. That is, there exists a computable pb^* function for $\mathcal{S}$.*

We say a WS-STS $\mathcal{S}$ is *effective* if it is effective as a WSTS and $\oplus$, $\mathsf{b}(\cdot)$ and $\mathsf{mub}(\cdot,\cdot)$ are computable. By the above proposition and the definition of Fb, we have the following result. The wanted result follows as a corollary.

Proposition 4. *Let $\mathcal{S}$ be an effective* WS-STS. *Then $\mathsf{Fb}(\phi)$ can be effectively computed, for any monotone ϕ.*

Corollary 1 (decidability). *Let $\mathcal{S}$ be an effective* WS-STS. *For any s and (anti-)monotone ϕ, it is decidable whether $s \models \phi$.*

4.3 Decidability in Concrete Instances

Let us discuss effectiveness of the WS-STS introduced in Section 2. In each case, the nontrivial part is actually defining effective pred-basis pb and mub functions. Effectiveness of CCS as a WS-STS can be proved along the lines of [2] (note that the definition of mub turns out to be nontrivial).

Let us now briefly consider affine WSN. Each affine function is recursive, hence effectiveness of the pred-basis follows from Theorem 4.2 of [16]. Next, for any $m, n \in \mathbb{N}^p$,

$\mathsf{mub}(m,n) = l \in \mathbb{N}^p$ is defined thus: $l(i) = \max\{n(i), m(i)\}$, for each $i = 1,\ldots,p$. Indeed l is computable and $n,m \leq l$. Moreover, by definition, whenever $n \leq k$ and $m \leq k$ then $l \leq k$, hence $\uparrow l = \uparrow n \cap \uparrow m$. These definitions of course apply also to the alternative version of WS-STS with enabled transitions as atomic predicates.

In the case of WA, one must ensure in the first place the effectiveness of the pred-basis, that strictly depends on the specific semiring; we leave this problem for future investigations. As an example, the results in [26] seem to indicate that an effective pred-basis exists in the case of tropical semirings.

5 A Hennessy-Milner Theorem for $\mathcal{L}_0$

In this section we prove that, under certain conditions, the logical preorder induced by the monotone fragment $\mathcal{L}_0$ coincides with the largest SBS. The proof goes along the lines of the classical theorem for bisimulation and the Hennessy-Milner logic [22]. However, the proof requires extra care, as it has to work also for non-*image-finite* processes. Indeed, the condition of image-finiteness, customary in process calculi when dealing with "weak" relations, makes little sense in our setting. When building distinguishing formulas for SBS-unrelated states, this fact will lead us to considering an in general infinite number of derivatives. A similar issue is raised by the monoidal structure of the system. In the end, we will be able to prove the result for a rather general class of WS-STS that enjoy certain monotonicity conditions. In the actual proof, though, we will have to resort to certain continuity arguments in order to cope with the issues outlined above. The technical device to do this is the notion of *complete* WSTS of [14, 15]. Intuitively, in a complete WS-STS, ascending chains of states always converge to limit points, and the limit operation commutes with transitions, sum and observation.

A few preliminary definitions are in order. Let $(X,\leq)$ be a poset. Recall that a set $D \subseteq X$ is *directed* if any two elements in D have an upper-bound in D. A *dcpo* is a poset $(X,\leq)$ where any directed set $D \subseteq X$ has a least upper bound (lub) in X, denoted $\bigvee D$. A dcpo is *algebraic* if any element $x \in X$ is the lub of the set of finite elements $\leq x$ (recall that $y \in X$ is *finite* if for every directed D, whenever $y \leq \bigvee D$ then $y \leq d$ for some $d \in D$). The set of finite elements of any poset X is denoted by $\mathrm{fin}(X)$. Let X, Y be two preordered sets. A partial function $f : X \to Y$ is *monotone* if $\mathrm{dom}(f)$ is upward closed in X and whenever $x \leq x'$ in $\mathrm{dom}(f)$ then $f(x) \leq f(x')$ in Y. When X and Y are dcpo, we say f is *continuous* if f is monotone, its domain is *Scott-closed* (that is, upward-closed, and such that for any directed $D \subseteq X$, $\bigvee D \in \mathrm{dom}(f)$ implies $D \cap \mathrm{dom}(f) \neq \emptyset$) and, for any directed D, $f(\bigvee D) = \bigvee f(D)$. Finally, we say f is *finitary* if $f(\mathrm{fin}(X)) \subseteq \mathrm{fin}(Y)$.

Following [14], we say a WS-STS is *functional* if its transition relation $\to$ can be decomposed as the union of finitely many transitions functions: $\to = \cup_{i=1}^{n} \delta_i$, where for $i = 1,...,n$, $\delta_i : S \to S$ is a partial monotone function. Note that in a functional WS-STS, monotonicity of $\oplus$ and O (the latter w.r.t. set inclusion in $\mathcal{P}(\mathrm{At})$) follows by definition. Complete WS-STS however require something more than monotonicity. Let us record that for any finite set I, $\mathcal{P}(I)$, partially ordered by set inclusion, is trivially an (algebraic) dcpo.

Definition 5 (complete WS-STS). *A* complete WS-STS *is a functional* WS-STS $(S, \cup_{i=1}^{n} \delta_i, \oplus, \mathrm{O})$ *such that* $(S,\leq)$ *is an algebraic dcpo, each* $\delta_i : S \to S$ *is a finitary* *continuous partial function and* $\oplus : S \times S \to S$ *and* $\mathrm{O} : S \to \mathcal{P}(\mathrm{At})$ *are finitary continuous total functions.*

For each state s in a transition system, define $\mathsf{Post}^*(s)$ to be the set of states reachable from s: $\mathsf{Post}^*(s) \overset{\triangle}{=} \{s' | s \to^* s'\}$. Note that a complete WS-STS in our sense is also a complete WSTS in the sense of [14]. Hence we have the following result from [14] about the *cover* of s, that is, the downward closure of $\mathsf{Post}^*(s)$.

Lemma 2 ([14], Proposition 6.1). *In any complete* WSTS, *hence in any complete* WS-STS, *for any state s there is a finite set $F \subseteq \mathsf{Post}^*(s)$ such that $\downarrow \mathsf{Post}^*(s) = \downarrow F$.*

The following result is a generalization of the previous one to the spatial component.

Lemma 3. *Let $\mathcal{S}$ be a complete* WS-STS. *For any state s, consider the set $D_s = \{(t,t') | s = t \oplus t'\}$. Then there is a finite $F \subseteq D_s$ s.t. $\downarrow D_s = \downarrow F$ in $S \times S$.*

We also need inductively defined approximants of the spatial-behavioral similarity.

Definition 6 (approximants of similarity). *Let $\mathcal{S}$ be a* WS-STS. *We let $(\sqsubseteq_i)_{i \in \mathbb{N}}$ be the sequence of preorders on S defined by induction on i as follows:*

a) $s \sqsubseteq_0 t$ *always;*
b) $s \sqsubseteq_{i+1} t$ *if: (1) whenever $s \to s'$ then there exists t' s.t. $t \to^* t'$ and $s' \sqsubseteq_i t'$; (2) whenever $s = s_1 \oplus s_2$ then there exist t_1, t_2 s.t. $t = t_1 \oplus t_2$ and $s_j \sqsubseteq_i t_j$ for $j = 1,2$; (3)* $\mathrm{O}(s) \subseteq \mathrm{O}(t)$.

We let $\sqsubseteq_\omega$ be $\bigcap_{i \in \mathbb{N}} \sqsubseteq_i$.

The preorder $\sqsubseteq_\omega$ can be proved to coincide with $\sqsubseteq$; the proof relies on arguments similar to those used in the proof of Theorem 2 below.

Proposition 5. *On a complete* WS-STS, $\sqsubseteq$ *and* $\sqsubseteq_\omega$ *coincide.*

Let $\mathcal{S}$ be a spatial transition system. The *logical preorder* $\sqsubseteq_{\mathcal{L}}$ over states is defined as: $s \sqsubseteq_{\mathcal{L}_0} t$ if and only if for each formula $\phi \in \mathcal{L}_0$, $s \models \phi$ implies $t \models \phi$. Here is the first version of the result we are after.

Theorem 2 (Hennessy-Milner type theorem for complete WS-STS). *On a complete* WS-STS, $\sqsubseteq$ *and* $\sqsubseteq_{\mathcal{L}_0}$ *concide.*

Proof: The inclusion $\sqsubseteq \subseteq \sqsubseteq_{\mathcal{L}_0}$ holds for any WS-STS (Lemma 1(1)). As for the opposite inclusion, it is convenient to work with $\sqsubseteq_\omega$ (Proposition 5). The proof then is a variant of the one in [22]. In particular, we prove the contrapositive statement, that $s \not\sqsubseteq_\omega t$ implies $s \not\sqsubseteq_{\mathcal{L}} t$. Assume that there is an index i s.t. $s \not\sqsubseteq_i t$: we show the existence of a formula ϕ s.t. $s \models \phi$ and $t \not\models \phi$. The proof is by induction on i. Assume $i > 0$. Now, $s \not\sqsubseteq_i t$ means that one of the clauses (1–3) of Definition 2 is violated.

Let us examine (1) first: there is s' s.t. $s \to s'$ and for no $t' \in \mathsf{Post}^*(t)$ it holds that $s' \sqsubseteq_{i-1} t'$. Consider the cover of t, $\downarrow \mathsf{Post}^*(t) = \downarrow F$ for some finite set $F \subseteq \mathsf{Post}^*(t)$ (Lemma 2). It is not difficult to show that for any $u \in F$, $s' \not\sqsubseteq_{i-1} u$. By induction hypothesis, there exists then ϕ_u s.t. $s' \models \phi_u$ and $u \not\models \phi_u$. Moreover, for each $t' \in \downarrow u$, $t' \not\models \phi_u$ either (a consequence of Lemma 1). Consider now $\phi = \diamond^* (\bigwedge_{u \in F} \phi_u)$. By construction $s \models \phi$, but $t \not\models \phi$.

The case where (2) is violated is handled similarly relying on Lemma 3. In particular, $s = s_1 \oplus s_2$ for some s_1 and s_2 s.t. for each pair (t_1, t_2) satisfying $t = t_1 \oplus t_2$, either $s_1 \not\sqsubseteq_{i-1} t_1$ or $s_2 \not\sqsubseteq_{i-1} t_2$. Let us write as $\{(t_1^j, t_2^j) | j \in J\}$ (J finite) the set F given by Lemma 3. By induction, for each $j \in J$ there exists either ϕ_1^j satisfied by s_1 but not by t_1^j, or ϕ_2^j satisfied by s_2 but not by t_2^j. In the former case let $\phi_2^j \triangleq \mathit{true}$, in the latter case $\phi_1^j \triangleq \mathit{true}$. Consider now $\phi = (\bigwedge_{j \in J} \phi_1^j) | (\bigwedge_{j \in J} \phi_2^j)$. By construction, $s \models \phi$, but $t \not\models \phi$. Finally, the case (3) is obvious. □

Next, we extend the above result to the class of functional (not necessarily complete) WS-STS's. Let us introduce some terminology first. For any complete WS-STS $\mathcal{S} = (S, \cup_{i=1}^n \delta_i, \oplus, \mathrm{O})$, let us denote by $\mathrm{fin}(\mathcal{S})$ the functional WSTS $(\mathrm{fin}(S), \cup_{i=1}^n \delta_{i,\mathrm{fin}}, \oplus_{\mathrm{fin}}, \mathrm{O}_{\mathrm{fin}})$, where $\mathrm{fin}(S)$ inherits the WQO of S and $\delta_{i,\mathrm{fin}}$, $\oplus_{\mathrm{fin}}$, $\mathrm{O}_{\mathrm{fin}}$ are the restrictions of δ_i, $\oplus$ and O, respectively, to $\mathrm{fin}(S)$. Let us denote by $\sqsubseteq_{\mathrm{fin}}$ the spatial-behavioral similarity defined over $\mathrm{fin}(\mathcal{S})$ (the subscript $_{\mathrm{fin}}$ will be omitted when no ambiguity arises).

Two functional WS-STS's are *isomorphic* if there is an embedding between them that is a bijection and commutes with the functions δ_i ($i = 1, ..., n$, for one and the same n), $\oplus$ and O, as expected. Clearly, any isomorphism preserves, in both directions, both $\sqsubseteq$ and $\models$, hence $\sqsubseteq_{\mathcal{L}_0}$. Now, given a functional WS-STS $\mathcal{S}$ there is a canonical way of building a complete WS-STS $\hat{\mathcal{S}}$ such that $\mathrm{fin}(\hat{\mathcal{S}})$ is isomorphic to $\mathcal{S}$: one takes the *ideal completion* of $\mathcal{S}$, where the set of states, $\hat{S}$, is the set of all ideals (that is, directed, downward closed subsets) of S ordered by set inclusion, and $\hat{\delta}_i$, $\hat{\oplus}$, $\hat{\mathrm{O}}$ are the unique continuous extensions of the corresponding (monotone) functions of $\mathcal{S}$. The isomorphism between $\mathcal{S}$ and $\mathrm{fin}(\hat{\mathcal{S}})$ is given by the function $\hat{\cdot} : S \to \mathrm{fin}(\hat{S})$ that sends each $s \in S$ into $\downarrow s \in \mathrm{fin}(\hat{S})$. A further technical ingredient is needed so as to ensure that $\hat{S}$ is well-ordered: the WQO S we start with must be a ω^2*-wqo*. Intuitively, ω^2*-wqo* strengthens the condition of WQO in the sense that one can always extract an infinite ascending chain $x_{i,j} < x_{j,k} < x_{k,l} < \cdots$, with $i < j < k < l$, out of any family of elements $\{x_{m,n}\}_{m \in I, n \in J}$. We refer the reader to [14] for the formal definition of this concept and further details of this construction; here, we just stress that only pathological instances of non-ω^2-wqo exist, and they have no computational relevance. With these definitions and facts at hand, the main result of the section is an easy consequence of Theorem 2.

Theorem 3 (Hennessy-Milner type theorem, functional case). *Let $\mathcal{S}$ be a functional* WS-STS *equipped with a ω^2-wqo. Then $\sqsubseteq$ and $\sqsubseteq_{\mathcal{L}_0}$ coincide over $\mathcal{S}$.*

Example 3. Both WSN (not necessarily affine) and WA are functional by definition. As mentioned in Section 2.2, $\sqsubseteq$ coincides with $\preceq$ in the interpretation of the observation function given by $\mathrm{O}(m) = \{i | m(i) \neq 0\}$. In the other two cases discussed in Section 2, the result is more interesting, as $\sqsubseteq$, hence $\sqsubseteq_{\mathcal{L}_0}$, is a much coarser relation.

6 Undecidability of the Spatial-Behavioral Preorder

The spatial-behavioral preorder $\sqsubseteq$ is in general undecidable in (effective) WS-STS even if it is the preorder induced by the decidable logic $\mathcal{L}_0$. The proof is by reduction from the *boundedness* problem for reset nets (RN) [11]. RN correspond to the subset of affine well-structured nets in which the matrix A, used in the definitions of the functions f, contains only the value 0 excluding some value equal to 1 in the main diagonal. This model can be seen as an extension of PN in which transitions can remove all the tokens in some given places. Given a RN and an initial state, the boundedness problem consists of checking whether the set of reachable states is finite. This problem is proved to be undecidable for RN in [11].

Consider a reset net Σ with p places and initial marking m. It is not restrictive to assume that there is no function $f(x) = Ax + B$ defined for $x = 0^p$. We define an affine WSN $[[\Sigma]] = (\mathbb{N}^{p+4}, F, \preceq)$ (where $\preceq$ is the usual ordering on naturals extended to vectors) such that there exist two states $s_1 = 0^p \cdot (0,0,1,0)$ and $s_2 = m \cdot (1,0,0,0)$ of $[[\Sigma]]$ such that $s_1 \sqsubseteq s_2$ iff Σ is unbounded. In the constructed WSN we will consider a very simple atomic predicate *notEmpty* and the interpretation exploiting the following labelling function $l : \{1,...,p+4\} \to \mathcal{P}(\mathrm{At})$, such that $l(i) = \{notEmpty\}$, for $i = p+2$ and $i = p+4$, and $l(i) = \emptyset$, otherwise. The idea that we follow in the definition of $[[\Sigma]]$ is as follows:

- The state $s_1 = 0^p \cdot (0,0,1,0)$ can give rise to computations of length n, for every n, that first increment by one the value in position $p+3$ until the value n is reached, and then such value is moved in position $p+4$ yielding the state $0^p \cdot (0,0,0,n)$.
- The state $s_2 = m \cdot (1,0,0,0)$ can mimic every computation $m \to^* m'$ in Σ. Every time a transition is performed, the value in position $p+1$ is increased by one, thus the state $m' \cdot (k,0,0,0)$ is reached assuming that $k-1$ steps have been simulated. An additional transition can move all the tokens in the marking m' in position $p+2$ and set to 0 the value in position $p+1$, thus yielding the state $0^p \cdot (0,\#m',0,0)$ where $\#m'$ denotes the total number of tokens in the marking m' .

The final states $0^p \cdot (0,0,0,n)$, for every n, and $0^p \cdot (0,\#m',0,0)$, for every marking m' reachable in Σ, of the computations starting from s_1 and s_2 are related by the atomic predicate *notEmpty* (able to observe the values in positions $p+2$ and $p+4$). We will prove that this relationship guarantees that $s_1 \sqsubseteq s_2$ iff Σ is unbounded.

We formally introduce $[[\Sigma]]$ defining its set F that contains the following functions:

- f_1 is a function that increments by one the value in position $p+3$ if it is greater than 1. Namely, $f_1(x) = Ax + B$, defined only if $x(p+3) > 0$, where A is the identity matrix and $B = 0^p \cdot (0,0,1,0)$.
- f_2 is a function that moves the value in position $p+3$ to position $p+4$. Namely, $f_2(x) = Ax + B$ where $A(i, j) = 0$ for every i and j, excluding $A(p+4, p+3) = 1$, and $B = 0^{p+4}$.
- A set of functions that simulate the computation steps of Σ (when the value in position $p+1$ is greater than 1) and increment the value in position $p+1$. Namely, for every function $f(x) = Ax + B$ in the definition of Σ, we consider a function $f'(x') = A'x' + B'$, defined only if $x' = x \cdot (1,0,0,0)$ and f is defined for x, where $A'(i, j) = A(i, j)$ for every $1 \le i, j \le p$ and $A'(i, j) = 0$ for every $p+1 \le i, j \le p+4$, and $B' = B \cdot (1,0,0,0)$.
- f_3 is a function that moves all the tokens in the marking reached after the simulated computation of Σ in position $p+2$. Namely, $f_3(x) = Ax + B$ where $A(i, j) = 0$ for every i and j, excluding $A(p+2, j) = 1$ for $1 \le j \le p$, and $B = 0^{p+4}$.

– f_4 is a function that permits to restart the simulation of Σ if the value in position $p+1$ is not 0. Namely, $f_4(x) = Ax + B$, defined only if $x(p+1) > 0$, where $A(i, j) = 0$ for every i and j, and $B = m \cdot (1,0,0,0)$.

We now prove the correctness of the reduction.

Theorem 4. *Let Σ be a reset net with inital marking m. Consider the affine* WSN *system $[[\Sigma]]$, and the states $s_1 = 0^p \cdot (0,0,1,0)$ and $s_2 = m \cdot (1,0,0,0)$. We have that $s_1 \sqsubseteq s_2$ iff Σ is unbounded.*

Proof: We first consider the *if* part. Assume that $s_1 \sqsubseteq s_2$. Given a natural number n, we will prove that Σ has a computation starting from m and leading to a marking with at least n tokens, from which the unboundedness of Σ follows. Consider a computation of $[[\Sigma]]$ of length n starting from s_1 and leading to $0^p \cdot (0,0,0,n)$. As $s_1 \sqsubseteq s_2$ we have that $[[\Sigma]]$ has a computation starting from s_2 and leading to a state $0^p \cdot (0,n',0,0)$ with $n' \geq n$. As the computations starting from the state s_2 mimic computations of Σ before moving all the tokens in the reached marking in position $p+2$, we have that also in Σ there is a computation from m leading to a marking with n' tokens.

We now consider the *only-if* part. Assume that Σ is unbounded. In this case we have that Σ has an infinite computation $m = m_1 \to m_2 \to \cdots \to m_i \to \cdots$ with the following property: there exists an infinite increasing sequence of indexes $1 = l_1, l_2, \ldots, l_j, \ldots$ such that if $j < j'$ then $\#m_{l_j} < \#m_{l_{j'}}$. We now prove that $s_1 \sqsubseteq s_2$ showing the existence of a spatial-behavioral simulation $\mathcal{R}$ between states of $[[\Sigma]]$ such that $(s_1, s_2) \in \mathcal{R}$. Let $\mathcal{R}$ be the relation including the following pairs:

1. $(0^p \cdot (0,0,k,0),\ m_{l_j} \cdot (k',0,0,0)) \in \mathcal{R}$ for every $0 < k \leq k' \leq l_j$, where m_{l_j} is taken from the computation of Σ, $m_{l_1} \to^+ m_{l_2} \to^+ \cdots \to^+ m_{l_j} \to^+ \cdots$, with $\#m_{l_j} < \#m_{l_{j'}}$ for every $j < j'$, described above,
2. $(0^p \cdot (0,0,0,k),\ 0^p \cdot (0,k',0,0)) \in \mathcal{R}$ for every $k \leq k'$,
3. $(0^{p+4},\ 0^{p+4}) \in \mathcal{R}$,
4. $(0^p \cdot (0,0,k,0),\ 0^p \cdot (k,0,0,0)) \in \mathcal{R}$ for every $k > 0$.

The relation $\mathcal{R}$ is a spatial-behavioural simulation as it satisfies the three conditions in the Definition 2. There are only two non-trivial conditions to be checked. The first one is that the pairs $(0^p \cdot (0,0,k,0),\ 0^p \cdot (k,0,0,0))$ satisfy condition (1). This holds because of the function f_4 that allows the second state $0^p \cdot (k,0,0,0)$ to restart the simulation of the computation $m_1 \to m_2 \to \cdots \to m_i \to \cdots$. The second one is that the pairs $(0^p \cdot (0,0,k,0),\ m_{l_j} \cdot (k',0,0,0))$ satisfy condition (2). In this case we observe that $0^p \cdot (0,0,k,0) = t_1 + t_2$ iff $t_1 = 0^p \cdot (0,0,k_1,0)$ and $t_2 = 0^p \cdot (0,0,k_2,0)$ with $k = k_1 + k_2$. If $k_1 = k$ and $k_2 = 0$ we simply observe that $m_{l_j} \cdot (k',0,0,0) = m_{l_j} \cdot (k',0,0,0) + 0^{p+4}$. If $k_1 < k$ we observe that $m_{l_j} \cdot (k',0,0,0) = 0^p \cdot (k_1,0,0,0) + m_{l_j} \cdot (k' - k_1,0,0,0)$ and that $(0^p \cdot (0,0,k_1,0),\ 0^p \cdot (k_1,0,0,0)) \in \mathcal{R}$ and $(0^p \cdot (0,0,k_2,0),\ m_{l_j} \cdot (k' - k_1,0,0,0)) \in \mathcal{R}$. We complete the proof observing that $(s_1, s_2) \in \mathcal{R}$ as $(0^p \cdot (0,0,1,0),\ m \cdot (1,0,0,0)) \in \mathcal{R}$ due to the first item of the definition of $\mathcal{R}$ and because $m = m_1$. □

In the light of this theorem and knowing from [11] that the boundedness problem in reset nets is undecidable, we can conclude that the spatial-behavioral simulation $\sqsubseteq$ is undecidable for affine WSN.

7 Conclusion

We have investigated connections between spatial logic and simulation relations, and related decidability issues, in a general setting of spatial transition systems. One of our results states the coincidence, under certain assumptions, of the logical preorder and of the largest SBS. In the setting of the Ambient Calculus and Ambient Logic, similar results have been achieved by Sangiorgi & al. in [20, 24]. On the one hand, the clauses of our SBS are reminiscent of their *intensional bisimilarity*. On the other hand, their completeness proof relies on techniques very different from ours; in particular, the presence in the logic of an adjunct of | helps them in defining characteristic formulae for processes in a syntax-driven way, which can have no counterpart in our framework. (Un)decidability of the Ambient Logic is also investigated in [20]. Caires and Lozes study the power of the adjunct in [8]; they too offer Hennessy-Milner theorems based on characteristic formulae and undecidability results relatively to a small fragment of CCS, but they consider a strong, rather than a weak next-step modality as we do. In the setting of the pi-calculus, Caires [5] offers a Hennessy-Milner theorem and decidability results for a Spatial Logic without adjunct, again considering strong modalities and relying on characteristic formulae.

The reader may observe that, while decidability of the monotone spatial logic $\mathcal{L}_0$ holds of course for all instances of the framework, the undecidability results are proved in the setting of Affine Well-Structured Nets (WSN). As for future work, we plan to investigate this issue further, so as to obtain new decidability results, or even abstract undecidability results that holds for a whole sub-class of models. Concerning the first direction, we observe that the decidability of SBS seems connected to the problem of effective computation of the finite *clover* set (see [15]) for $\downarrow \mathsf{Post}^*(s)$, and this turns out to be effective if we move from reset/transfer Petri nets to Petri nets. Another issue left open by our study is, in the case of WA, the characterization of semirings for which a pred-basis is effectively computable.

Acknowledgment. We thank the anonymous referees for their useful comments on the submitted version of this paper. The second author wishes to thank Jean Goubault-Larrecq for stimulating discussions on complete WSTS.

References

1. Acciai, L., Boreale, M., Zavattaro, G.: On the relationship between spatial logics and behavioral simulations. Tech. rep. (2010), http://rap.dsi.unifi.it/~acciai
2. Acciai, L., Boreale, M.: Deciding safety properties in infinite-state pi-calculus via behavioural types. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 31–42. Springer, Heidelberg (2009)
3. Baccelli, F., Cohen, G., Olsder, G.J., Quadrat, J.P.: Synchronization and linearity. Wiley, Chichester (1992)
4. 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)

5. Caires, L.: Behavioural and Spatial Observations in a Logic for the pi-Calculus. In: Walukiewicz, I. (ed.) FOSSACS 2004. LNCS, vol. 2987, pp. 72–89. Springer, Heidelberg (2004)
6. Caires, L., Cardelli, L.: A spatial logic for concurrency (part II). Theor. Comput. Sci. 322(3), 517–565 (2004)
7. Caires, L., Cardelli, L.: A spatial logic for concurrency (part I). Inf. Comput. 186(2), 194–235 (2003)
8. Caires, L., Lozes, E.: Elimination of Quantifiers and Undecidability in Spatial Logics for Concurrency. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 240–257. Springer, Heidelberg (2004)
9. Cardelli, L., Gordon, A.D.: Anytime, Anywhere: Modal Logics for Mobile Ambients. In: Proc. of POPL, pp. 365–377 (2000)
10. Dickson, L.E.: Finiteness of the odd perfect and primitive abundant numbers with r distinct prime factors. Amer. Journal Math 35, 413–422 (1913)
11. Dufourd, E.C., Finkel, A., Schnöebelen, P.: Reset Nets Between Decidability and Undecidability. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 103–115. Springer, Heidelberg (1998)
12. Esparza, J., Finkel, A., Meyr, R.: On the Verification of Broadcast Protocols. In: Proc. of LICS, pp. 352–359 (1999)
13. Esparza, J.: On the Decidability of Model Checking for Several μ-calculi and Petri Nets. In: Tison, S. (ed.) CAAP 1994. LNCS, vol. 787, pp. 115–129. Springer, Heidelberg (1994)
14. Finkel, A., Goubault-Larrecq, J.: Forward Analysis for WSTS, Part I: Completions. In: Proc. of STACS, Dagstuhl Seminar Proceedings 09001, pp. 433–444 (2009)
15. Finkel, A., Goubault-Larrecq, J.: Forward Analysis for WSTS, Part II: Complete WSTS. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 188–199. Springer, Heidelberg (2009)
16. Finkel, A., McKenzie, P., Picaronny, C.: A Well-Structured Framework for Analysing Petri Net Extensions. Information and Computation 195(1-2), 1–29 (2004)
17. Finkel, A., Schnöebelen, P.: Well-Structured Transition Systems Everywhere! Theoretical Computer Science 256(1-2), 63–92 (2001)
18. Hack, M.H.T.: Decidability questions for Petri nets. Ph.D Thesis. MIT (1976)
19. Hennessy, M., Milner, R.: On Observing Nondeterminism and Concurrency. In: de Bakker, J.W., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 299–309. Springer, Heidelberg (1980)
20. Hirschkoff, D., Lozes, E., Sangiorgi, D.: Separability, Expressiveness, and Decidability in the Ambient Logic. In: Proc. of LICS, pp. 423–432 (2002)
21. Kuich, W., Salomaa, A.: Semirings, Automata, Languages. Monographs in Theoretical Computer Science, EATCS Series, vol. 5. Springer, Heidelberg (1986)
22. Milner, R.: Communication and concurrency. Prentice-Hall, Englewood Cliffs (1989)
23. Minsky, M.: Computation: Finite and Infinite Machines, 1st edn. Prentice-Hall, Inc., Englewood Cliffs (1967)
24. Sangiorgi, D.: Extensionality and Intensionality of the Ambient Logics. In: Proc. of POPL, pp. 4–13 (2001)
25. Simon, I.: Limited subset of a Free Monoid. In: Proc. of FOCS, pp. 143–150 (1978)
26. Valk, R., Jantzen, M.: The residue of vector sets with applications to decidability problems in Petri nets. Acta Informatica 21, 643–674 (1985)

An Easy Completeness Proof for the Modal μ-Calculus on Finite Trees

Balder ten Cate[1,*] and Gaëlle Fontaine[2,*]

[1] University of California, Santa Cruz
[2] ILLC, Universiteit van Amsterdam

Abstract. We give a complete axiomatization for the modal μ-calculus on finite trees. While the completeness of our axiomatization already follows from a more powerful result by Igor Walukiewicz in [11], our proof is easier and uses very different tools, inspired from model theory. We show that our approach generalizes to certain axiomatic extensions, and to the extension of the μ-calculus with graded modalities. We hope that the method might be helpful for other completeness proofs as well.

The μ-calculus is an extension of modal logic with a fixpoint operator. In 1983, Dexter Kozen suggested an axiomatization and showed completeness for the aconjunctive fragment of the μ-calculus (see, e.g., [7]). It took more than ten years to prove completeness. This proof is due to Igor Walukiewicz [11] and is quite involved. It uses tableaux and the notion of disjunctive formula. We propose here a simpler proof in a particular case. More precisely, we prove the completeness of the Kozen axiomatization $\mathbf{K}^{\mu}$ extended with the axiom $\mu x.\Box x$ with respect to the class of finite tree models. Finite trees are a fundamental data structure in computer science, and logics on finite trees have received considerable attention in recent literature, motivated by applications in areas such as XML [1,8].

Our argument consists of three steps. The first step consists of defining a notion of rank which plays the same role as the modal depth for modal formulas. One of the main properties of the rank is the following. In order to know whether a formula φ of rank n is true at a node w, it is enough to know which proposition letters are true at w and which formulas of rank at most n are true at the successor nodes of w. Another key property of the rank is that there are only finitely many formulas of a given rank (up to logical equivalence).

The second step is to prove completeness of the μ-calculus with respect to generalized models, which are basically Kripke models augmented with a set of admissible subsets, in the style of Henkin semantics for second order logic.

* The first author was supported by the Netherlands Organization for Scientific Research (NWO), under grant 639.021.508 and by ERC Advanced Grand Webdam on Foundation of Web data management. The second author is supported by VICI grant 639.073.501 of the NWO. We are grateful to Alexandru Baltag and Yde Venema for their comments on earlier drafts. We also thank the anonymous referees for helpful remarks.

The last step is inspired by the work of Kees Doets (see, e.g., [3]). Let us call a node in a generalized model n-good if there is a node in a finite tree model which satisfies exactly the same formulas of rank at most n. Using an induction principle, we show that every node in a generalized model satisfying $\mu x.\Box x$ is n-good. It is here that we use the main property of the rank. Finally, putting this together with the completeness for generalized models, we obtain completeness for the class of finite tree models.

This argument can also be applied to some extensions of the logic $\mathbf{K}^\mu + \mu x.\Box x$. More precisely, we prove that when we add finitely many shallow axioms (as defined in [10]), we obtain a complete axiomatization for the corresponding class of finite trees. We also show that we can adapt our proof to show completeness for the graded μ-calculus extended with the axiom $\mu x.\Box x$. Let us also mention that a similar method has been used for other completeness proofs in [6].

The paper is organized as follows. In section 1, we recall what is the Kozen axiomatization for the μ-calculus $\mathbf{K}^\mu$ and what is the intended semantics. In section 2, we define the notion of rank for a formula. In section 3, we give a definition for the generalized models and we show completeness of $\mathbf{K}^\mu$ with respect to the class of generalized models. In section 4, we use Kees Doets' argument to obtain completeness of $\mathbf{K}^\mu + \mu x.\Box x$ with respect to the class of finite tree models. In the last two sections, we give some examples of extensions of $\mathbf{K}^\mu + \mu x.\Box x$ to which we can apply our method in order to prove completeness.

1 Syntax, Semantics and Axiomatization

We introduce the language and the Kripke semantics for the μ-calculus. We also recall the axiomatization given by Dexter Kozen.

Definition 1. The μ-formulas over a set $Prop$ of proposition letters and a set Var of variables are given by

$$\varphi \ ::= \ \top \mid p \mid x \mid \varphi \vee \varphi \mid \neg\varphi \mid \Diamond\varphi \mid \mu x.\varphi,$$

where p ranges over the set $Prop$ and x ranges over the set Var of variables. In $\mu x.\varphi$, we require that the variable x appears only under an even number of negations in φ. We will assume that Var is infinite.

As usual, we let $\phi \wedge \psi$, $\Box\varphi$ and $\nu x.\varphi$ be abbreviations for $\neg(\neg\varphi \vee \neg\psi)$, $\neg\Diamond\neg\varphi$ and $\neg\mu x.\neg[\neg x/x]$. The notions of *subformula*, *bound variable*, *free variable* and *substitution* are defined in the usual way. If φ and ψ are μ-formulas and if p is a proposition letter, we denote by $\varphi[\psi/p]$ the formula obtained by replacing in φ each occurrence of p by ψ. Similarly, if x is a variable, we define $\varphi[\psi/x]$.

A *μ-sentence* is a formula in which all the variables are bound. A μ-formula is a *modal formula* if it does not contain any subformula of the form $\mu x.\varphi$.

Definition 2. A *Kripke frame* is a pair (W, R), where W is a set and R a binary relation on W. A *Kripke model* is a triple (W, R, V) where (W, R) is a Kripke frame and $V : Prop \to \mathcal{P}(W)$ a valuation. If (w, v) belongs to R, we say that w is a *predecessor* of v and v is a *successor* of w.

Given a formula φ, a Kripke model $\mathcal{M} = (W, R, V)$ and an assignment $\tau : Var \to \mathcal{P}(W)$, we define a subset $[\![\varphi]\!]_{\mathcal{M},\tau}$ that is interpreted as the set of points at which φ is true. The subset is defined by induction in the usual way. We only recall that

$$[\![\mu x.\varphi]\!]_{\mathcal{M},\tau} = \bigcap\{U \subseteq W : [\![\varphi]\!]_{\mathcal{M},\tau[x:=U]} \subseteq U\},$$

where $\tau[x := U]$ is the assignment τ' such that $\tau'(x) = U$ and $\tau'(y) = \tau(y)$, for all $y \neq x$. Observe that the set $[\![\mu x.\varphi]\!]_{\mathcal{M},\tau}$ is the least fixpoint of the map $\varphi_x : \mathcal{P}(W) \to \mathcal{P}(W)$ defined by $\varphi_x(U) := [\![\varphi]\!]_{\mathcal{M},\tau[x:=U]}$, for all $U \subseteq W$.

If $w \in [\![\varphi]\!]_{\mathcal{M},\tau}$, we write $\mathcal{M}, w \Vdash_\tau \varphi$ and we say that φ is *true* at w under the assignment τ. If φ is a sentence, we simply write $\mathcal{M}, w \Vdash \varphi$.

A formula φ is *true* in $\mathcal{M}$ under an assignment τ if for all $w \in W$, we have $\mathcal{M}, w \Vdash_\tau \varphi$. In this case, we write $\mathcal{M} \Vdash_\tau \varphi$. A set Φ of formulas is *true* in a model $\mathcal{M}$ under an assignment τ, notation: $\mathcal{M} \Vdash_\tau \Phi$, if for all φ in Φ, φ is true in $\mathcal{M}$ under τ. Finally, if (W, R) is a Kripke frame and for all valuations V and all assignments τ, φ is true in (W, R, V) under the assignment τ, we say that φ is *valid* in (W, R) and we write $(W, R) \Vdash \varphi$.

Definition 3. The axiomatization of the Kozen system $\mathbf{K}^\mu$ consists of the following axioms and rules

propositional tautologies,	
If $\vdash \varphi \to \psi$ and $\vdash \varphi$, then $\vdash \psi$	(Modus ponens),
If $\vdash \varphi$, then $\vdash \varphi[p/\psi]$	(Substitution),
$\vdash \Box(p \to q) \to (\Box p \to \Box q)$	(K-axiom),
If $\vdash \varphi$, then $\vdash \Box\varphi$	(Necessitation),
$\vdash \varphi[x/\mu x.\varphi] \to \mu x.\varphi$	(Fixpoint axiom),
If $\vdash \varphi[x/\psi] \to \psi$, then $\vdash \mu x.\varphi \to \psi$	(Fixpoint rule),

where x is not a bound variable of φ and no free variable of ψ is bound in φ.

Definition 4. If Φ is a set of modal formulas, we write $\mathbf{K} + \Phi$ for the smallest set of modal formulas which contains the propositional tautologies, the K-axiom and is closed under the Modus Ponens, Substitution and Necessitation rules. We say that $\mathbf{K} + \Phi$ is the *extension* of $\mathbf{K}$ by Φ. Note that if Φ is empty, we simply write $\mathbf{K}$.

Next, if Φ is a set of modal formulas, we denote by $\mathbf{K} +_r \Phi$ the smallest set of formulas which contains both $\mathbf{K}$ and Φ and is closed under the Modus Ponens and Necessitation rules. We call $\mathbf{K} +_r \Phi$ the *restricted extension* of $\mathbf{K}$ by Φ.

Finally, if Φ is a set of μ-formulas, we write $\mathbf{K}^\mu + \Phi$ for the smallest set of formulas which contains both $\mathbf{K}^\mu$ and Φ and is closed under the Modus Ponens, Substitution, Necessitation and Fixpoint rules. We say that $\mathbf{K}^\mu + \Phi$ is the *extension* of $\mathbf{K}^\mu$ by Φ.

Definition 5. Let (W, R) be a Kripke frame. A point r in W is a *root* if for all w in W, there is a sequence $w_0, \ldots, w_n$ such that $w_0 = r$, $w_n = w$ and (w_i, w_{i+1}) belongs to R, for all $i \in \{0, \ldots, n-1\}$. The frame (W, R) is a *tree* if it has a root, every point distinct from the root has a unique predecessor and there is no sequence $w_0, \dots, w_{n+1}$ in W such that $w_{n+1} = w_0$ and (w_i, w_{i+1}) belongs to R, for all $i \in \{0, \dots, n\}$.

The frame (W, R) is a *finite tree* if it is a tree and W is finite. Finally, a *finite tree Kripke model* is a Kripke model (W, R, V) such that (W, R) is a finite tree.

Proposition 1. *Let $\mathcal{M} = (W, R, V)$ be a Kripke model. The formula $\mu x.\Box x$ is true at a point w in $\mathcal{M}$ iff there is no infinite sequence $w_0, w_1 \dots$ in W such that $w_0 = w$ and (w_i, w_{i+1}) belongs to R, for all $i \in \mathbb{N}$.*

In particular, the formula $\mu x.\Box x$ is true in $\mathcal{M}$ iff there is no infinite sequence $w_0, w_1, \dots$ such that (w_i, w_{i+1}) belongs to R, for all $i \in \mathbb{N}$. That is, iff $\mathcal{M}$ is conversely well-founded.

We prove the completeness of the logic $\mathbf{K}^\mu + \mu x.\Box x$ with respect to the class of finite tree Kripke models. That is, a formula φ is provable in $\mathbf{K}^\mu + \mu x.\Box x$ iff it is valid in any finite tree Kripke model. In fact, this result can be derived from the completeness result proved by Igor Walukiewicz in [11]. We will give more details at the end of Section 4.

2 Rank of a Formula

The goal of this section is to come up with a definition of rank that would be the analogue of the depth of a modal formula. For modal logics, it is not hard to see that the truth of an arbitrary formula φ at some world w only depends of the truth of the proposition letters at w and of the truth of formulas ψ at the successors of w, where the depth of ψ is at most the depth of φ. In our proof, we will need something similar for the μ-calculus.

The most natural idea would be to look at the nesting depth of modal and fixpoint operators. However, this definition does not have the required properties. The notion of rank that we develop here is in fact related to the closure of a formula, which has been introduced by Michael Fischer and Robert Ladner in [5].

Definition 6. The *closure* $Cl(\varphi)$ of a formula φ is the smallest set of formulas such that

$$\begin{array}{l|l} \varphi \in Cl(\varphi), & \text{if } \psi \vee \chi \in Cl(\varphi), \text{ then } \{\psi, \chi\} \subseteq Cl(\varphi), \\ \text{if } \neg\psi \in Cl(\varphi), \text{ then } \psi \in Cl(\varphi), & \text{if } \mu x.\psi \in Cl(\varphi), \text{ then } \psi[x/\mu x.\psi] \in Cl(\varphi). \\ \text{if } \Diamond\psi \in Cl(\varphi), \text{ then } \psi \in Cl(\varphi), & \end{array}$$

It is also proved in [7] that the closure $Cl(\varphi)$ of a formula φ is finite. In order to define the rank, we also need to recall the notion of the depth of a formula.

Definition 7. The *depth* $d(\varphi)$ of a formula φ is defined by induction as follows

$$\begin{array}{l|l} d(\top) = d(p) = d(x) = 0, & d(\varphi \vee \psi) = max\{d(\varphi), d(\psi)\}, \\ d(\neg\varphi) = d(\varphi), & d(\Diamond\varphi) = d(\mu x.\varphi) = d(\varphi) + 1. \end{array}$$

Definition 8. The *rank* of a formula φ is defined as follows

$$rank(\varphi) = max\{d(\psi) \mid \psi \in Cl(\varphi)\}.$$

Remark that since $Cl(\varphi)$ is finite, $rank(\varphi)$ is always a natural number. All we will use later are the following properties of the rank.

Proposition 2. *If the set Prop of proposition letters is finite, then for all natural numbers k, there are only finitely many sentences of rank k (up to logical equivalence).*

Proof. Fix a natural number k. Note first that if $rank(\varphi) = k$, then in particular, $d(\varphi) \leq k$. Hence, it is enough to show that there only finitely many sentences of depth below k (up to logical equivalence). If $d(\varphi) \leq k$, we may assume that the only variables occurring in φ are some $x_1, \dots, x_k$. It is routine to prove by induction on l that there are finitely many formulas of depth l with variables $x_1, \dots, x_k$, up to logical equivalence. $\square$

Proposition 3. *The rank is closed under boolean combination. That is, for any n, a boolean combination of formulas of rank at most n is a formula of rank at most n.*

Proposition 4. *Every formula φ is provably equivalent to a boolean combination of proposition letters and formulas of the form $\Diamond\psi$, with $rank(\psi) \leq rank(\varphi)$.*

Proof. A formula is guarded if every bound variable is in the scope of a modal operator. Each formula φ is provably equivalent to a guarded formula of rank less or equal to the rank of φ (see, e.g., [11]) . Therefore, let φ be a guarded formula. We define a map G by induction as follows:

$$\begin{array}{l|l} G(\top) = \top, & G(p) = p, \text{ if } p \text{ is a free variable of } \varphi, \\ G(\neg\psi) = \neg G(\psi), & G(\psi \vee \psi') = G(\psi) \vee G(\psi'), \\ G(\Diamond\psi) = \Diamond\psi, & G(\mu x.\psi) = G(\psi[x/\mu x.\psi]). \end{array}$$

We note that G is not defined for a bound variable x of φ. Using the fact that φ is guarded, one can show that the computation of $G(\varphi)$ is well-defined and does terminate. It is not hard to see that $G(\varphi)$ is equivalent to φ. We remark now that if ψ belongs to $Cl(\varphi)$, then $Cl(\psi)$ is a subset of $Cl(\varphi)$. It follows that $G(\varphi)$ is a boolean combination of proposition letters and formulas of the form $\Diamond\psi$, with $rank(\psi) \leq rank(\varphi)$. $\square$

3 Completeness for Generalized Models

We introduce generalized models which are the analogue for the μ-calculus of the general models for second order logic. We prove completeness of $\mathbf{K}^\mu$ with respect to the class of generalized models.

Definition 9. Consider a quadruple $\mathcal{M} = (W, R, V, \mathbb{A})$ where (W, R) is a Kripke frame, $\mathbb{A}$ is a subset of $\mathcal{P}(W)$ and $V : Prop \to \mathbb{A}$ a valuation. A set which belongs to $\mathbb{A}$ is called *admissible*.

We define the truth of a formula φ under an assignment $\tau : Var \to \mathbb{A}$ by induction. All the clauses are the same as usual, except the one defining the truth of $\mu x.\varphi$. Normally, we define the set $[\![\mu x.\varphi]\!]_{\mathcal{M},\tau}$ as the least pre-fixpoint of the map φ_x (see Definition 2). But here, we define it as the intersection of all the admissible pre-fixpoints of φ_x.

$$\begin{aligned}
&[\![\top]\!]_{\mathcal{M},\tau} = W,\\
&[\![p]\!]_{\mathcal{M},\tau} = V(p),\\
&[\![x]\!]_{\mathcal{M},\tau} = \tau(x),\\
&[\![\neg\varphi]\!]_{\mathcal{M},\tau} = W \backslash [\![\varphi]\!]_{\mathcal{M},\tau},\\
&[\![\varphi \vee \psi]\!]_{\mathcal{M},\tau} = [\![\varphi]\!]_{\mathcal{M},\tau} \cup [\![\psi]\!]_{\mathcal{M},\tau},\\
&[\![\Diamond\varphi]\!]_{\mathcal{M},\tau} = \{w \in W : \exists v \in W \text{ s.t. } wRv \text{ and } v \in [\![\varphi]\!]_{\mathcal{M},\tau}\},\\
&[\![\mu x.\varphi]\!]_{\mathcal{M},\tau} = \bigcap\{U \in \mathbb{A} : [\![\varphi]\!]_{\mathcal{M},\tau[x:=U]} \subseteq U\},
\end{aligned}$$

where $\tau[x := U]$ is the assignment τ' such that $\tau'(x) = U$ and $\tau(y) = \tau(y)$, for all $y \neq x$. If $w \in [\![\varphi]\!]_{\mathcal{M},\tau}$, we write $\mathcal{M}, w \Vdash_\tau \varphi$ and we say that φ is *true* at w under the assignment τ. If φ is a sentence, we simply write $\mathcal{M}, w \Vdash \varphi$. A formula φ is *true* in $\mathcal{M}$ under an assignment τ if for all $w \in W$, we have $\mathcal{M}, w \Vdash_\tau \varphi$. In this case, we write $\mathcal{M} \Vdash_\tau \varphi$.

The quadruple $\mathcal{M} = (W, R, V, \mathbb{A})$ is a *generalized model* if for all formulas φ and all assignments $\tau : Var \to \mathbb{A}$, the set $[\![\varphi]\!]_{\mathcal{M},\tau}$ belongs to $\mathbb{A}$. A triple $\mathcal{F} = (W, R, \mathcal{A})$ is a *generalized frame* if for every valuation $V : Prop \to \mathbb{A}$, the quadruple $(W, R, V, \mathbb{A})$ is a generalized model.

If $\mathcal{F} = (W, R, \mathcal{A})$ is a generalized frame, we call (W, R) the *underlying Kripke frame* of $\mathcal{F}$. A formula φ is *valid* in a generalized frame $\mathcal{F} = (W, R, \mathcal{A})$, notation: $\mathcal{F} \Vdash \varphi$, if for all valuations $V : Prop \to \mathbb{A}$ and all assignments $\tau : Var \to \mathbb{A}$, the formula φ is true in $(W, R, V, \mathbb{A})$ under the assignment τ.

Any Kripke model $M = (W, R, V)$ can be seen as the generalized model $M' = (W, R, V, \mathcal{P}(W))$. It follows easily from our definition that for all formulas φ and all points $w \in W$,

$$M, w \Vdash \varphi \quad \text{iff} \quad M', w \Vdash \varphi.$$

Now we show that for all sets Φ of formulas, the logic $\mathbf{K}^\mu + \Phi$ is complete with respect to a particular generalized model. If we were only interested in showing that $\mathbf{K}^\mu + \mu x.\Box x$ is complete, we would restrict ourselves to prove that $\mathbf{K}^\mu + \mu x.\Box x$ is complete with respect to a particular generalized model. But later, we will also show completeness for some extensions of $\mathbf{K}^\mu$ and it will become handy to suppose that Φ contains other additional axioms.

First we introduce some definitions and recall some results of modal logic.

Theorem 1 ([9]). *Let Φ be a set of modal formulas. There exists a model $\mathcal{M}$ such that for all modal formulas φ, φ is provable in $\mathbf{K} +_r \Phi$ iff $\mathcal{M} \Vdash \varphi$.*

This theorems says that every (restricted) extension of $\mathbf{K}$ is complete with respect to a class of Kripke models. However, all of these extensions might not be complete with respect to a class of Kripke frames. Indeed, there is no guarantee that the formulas in Φ are valid in the frame corresponding to the model given by Theorem 1.

Definition 10. Let *Prop* be a set of proposition letters and *Var* a set of variables. We let μFL be the set of sentences of the form $\mu x.\varphi$ or $\nu x.\varphi$, for some μ-formula φ over *Prop*. We denote by $Prop^+$ the set $Prop \cup \{p_\varphi : \varphi \in \mu FL\}$.

If φ is a μ-formula over $Prop^+$, we define $s(\varphi)$ as the formula obtained by replacing each proposition letter of the form p_ψ ($\psi \in \mu FL$), by the formula ψ. We call $s(\varphi)$ the *source* of φ.

Next, if φ is a μ-formula over $Prop^+$, we say that a modal formula ψ over $Prop^+$ is the *replacement* of φ if ψ is obtained by replacing in the formula $s(\varphi)$ all maximal subformulas χ in μFL, by the proposition letter p_χ. In this case, we use the notation $repl(\varphi)$. Finally, if Σ is a set of μ-formulas over *Prop*, we let $repl(\Sigma)$ be the set $\{repl(\varphi) : \varphi \in \Sigma\}$.

For example, let φ be the formula $\Diamond(\mu x.p_{\nu y.x \wedge y})$. Then $s(\varphi)$ is the formula $\Diamond(\mu x.\nu y.(x \wedge y))$ and $repl(\varphi)$ is the formula $\Diamond p_{\mu x.\nu y.(x\wedge y)}$. We remark also that for all formulas ψ, $repl(\psi)$ is a modal formula over $Prop^+$.

Now we will prove that for all sets of formulas Φ, the logic $\mathbf{K}^\mu + \Phi$ is complete with respect to the class of generalized models which make Φ true. An easy way to show this would be to do a standard canonical model construction (inspired by the one used for the completeness of the modal logic $\mathbf{K}$).

However, we give here another proof. The idea is to use the replacement map introduced previously in order to translate the completeness result for modal logic into a completeness result for generalized Kripke models. This proof might seem a bit more tedious. In fact, it will make our result easier to extend to other settings (like graded μ-calculus).

Theorem 2. *Let Φ be a set of μ-formulas over a set Prop. There is a generalized model $\mathcal{M} = (W, R, V, \mathbb{A})$ such that for all sentences φ, φ is provable in $\mathbf{K}^\mu + \Phi$ iff $\mathcal{M} \Vdash \varphi$.*

In particular, the logic $\mathbf{K}^\mu + \Phi$ is complete with respect to the class of generalized models which make Φ true. That is, for all sentences φ, φ is provable in $\mathbf{K}^\mu + \Phi$ iff for all generalized models $\mathcal{M}$ such that $\mathcal{M} \Vdash \Phi$, we have $\mathcal{M} \Vdash \varphi$.

Proof. By Theorem 1, there is a Kripke model $\mathcal{N} = (W, R, V^+)$ (over $Prop^+$) such that for all modal formulas α over $Prop^+$, α is provable in $\mathbf{K} +_r repl(\mathbf{K}^\mu + \Phi)$ iff $\mathcal{N} \Vdash \alpha$. Now let $\mathbb{A}$ be the set $\{[\![\delta]\!]_\mathcal{N} : \delta$ modal formula over $Prop^+\}$. We define $\mathcal{M}$ as the quadruple $(W, R, V^+, \mathbb{A})$.

First, we show that for all μ-formulas φ over $Prop^+$, all v in W and all assignments $\tau : Var \to \mathbb{A}$, we have

$$\mathcal{N}, v \Vdash_\tau repl(\varphi) \quad \text{iff} \quad \mathcal{M}, v \Vdash_\tau \varphi. \tag{1}$$

The proof is by induction on the complexity of ψ. We skip the details by lack of space. Next, we prove that for all μ-sentences φ (over *Prop*), we have

$$\mathcal{M} \Vdash \varphi \quad \text{iff} \quad \varphi \text{ is provable in } \mathbf{K}^\mu + \Phi.$$

For the direction from left to right, suppose that φ is not provable in $\mathbf{K}^\mu + \Phi$. This implies that $repl(\varphi)$ is not provable in $\mathbf{K} +_r repl(\mathbf{K}^\mu + \Phi)$. Therefore, the formula $repl(\varphi)$ is not true in $\mathcal{N}$. By equivalence (1), this means that φ is not true in $\mathcal{M}$.

For the direction from right to left, assume that φ is provable in $\mathbf{K}^\mu + \Phi$. It is routine to show that for all generalized models $\mathcal{M}'$ such that $\mathcal{M}' \Vdash \Phi$, we have that $\mathcal{M}' \Vdash \varphi$. Moreover, using equivalence (1) together with the fact that $repl(\Phi)$ is true in $\mathcal{N}$, we obtain that Φ is true in $\mathcal{M}$. Putting everything together, we get that φ is true in $\mathcal{M}$.

To finish the proof, it remains to show that $\mathcal{M}$ is a generalized model. That is, for all μ-formulas φ over *Prop*, the set $[\![\varphi]\!]_\mathcal{M}$ belongs to $\mathbb{A}$. Fix a μ-formula φ over *Prop*. By equivalence (1), the set $[\![\varphi]\!]_\mathcal{M}$ is equal to $[\![repl(\varphi)]\!]_\mathcal{N}$. By definition of $\mathbb{A}$, this set belongs to $\mathbb{A}$.

4 Completeness for Finite Tree Models

In the style of Kees Doets [3], we prove completeness of $\mathbf{K}^\mu + \mu x.\Box x$ with respect to the class of finite tree Kripke models. The argument is as follows. First, we say that a point w in a generalized model is n-good if there is a point v in a finite tree Kripke model such that no formula of rank at most n can distinguish w from v. Next, we show that "being n-good" is a property that can be expressed by a formula γ_n of rank at most n. Afterwards, we prove that each point (in a generalized model) satisfying $\mu x.\Box x$, is n-good. Finally, using completeness for generalized models, we obtain completeness of $\mathbf{K}^\mu + \mu x.\Box x$ with respect to the class of finite tree Kripke models.

In this section, we will assume that the set *Prop* of proposition letters is finite. Often we write "finite tree" instead of "finite tree Kripke model".

Definition 11. Fix a natural number n. Let $\mathcal{M}$ and $\mathcal{M}'$ be two generalized models. A world $w \in \mathcal{M}$ is *rank n-indistinguishable* to a world $w' \in \mathcal{M}'$ if for all formulas φ of rank at most n, we have

$$\mathcal{M}, w \Vdash \varphi \quad \text{iff} \quad \mathcal{M}', w' \Vdash \varphi.$$

In case this happens, we write $(\mathcal{M}, w) \sim_n (\mathcal{M}', w')$. Finally, we say that $w \in \mathcal{M}$ is *n-good* if there exists a finite tree $\mathcal{N}$ and some $v \in \mathcal{N}$ such that $(\mathcal{M}, w) \sim_n (\mathcal{N}, v)$.

Definition 12. Let n be a natural number and let Φ_n be the set of formulas of rank at most n. For any generalized model $\mathcal{M}$ and any $w \in \mathcal{M}$, we define the *n-type* $\theta_n(w)$ as the set of formulas in Φ_n which are true at w.

By Proposition 2, Φ_n is finite (up to logical equivalence) and in particular, there are only finitely many distinct n-types.

Lemma 1. *Let n be a natural number. There exists a formula γ_n of rank n such that for any generalized model $\mathcal{M}$ and any $w \in \mathcal{M}$, we have*

$$\mathcal{M}, w \Vdash \gamma_n \quad \textit{iff} \quad (\mathcal{M}, w) \textit{ is } n\textit{-good.}$$

Proof. Let n be a natural number and let γ_n be the formula defined by

$$\gamma_n = \bigvee\{\bigwedge \theta_n(w) \mid w \text{ is } n\text{-good}\},$$

where w a point in a generalized model $\mathcal{M}$ and $\bigwedge \theta_n(w)$ is shorthand for $\bigwedge\{\varphi : \varphi \in \theta_n(w)\}$. Since there are only finitely many distinct n-types, the formula γ_n is well-defined. Moreover, from Proposition 3, it follows that the rank of γ_n is n.

It remains to check that γ_n has the required properties. It is immediate to see that if a point w in a generalized model is n-good, then γ_n is true at w. For the other direction, assume that γ_n is true at a point w in a generalized model $\mathcal{M}$. Therefore, there is a point w' in a generalized model $\mathcal{M}'$ such that w' is n-good and $\theta_n(w')$ is true at w. Since w' is n-good, there is a point v in a model $\mathcal{N}$ such that w' and v are rank n-indistinguishable. Using the fact that w and w' have the same n-type, we obtain that w and v are also rank n-indistinguishable. That is, w is n-good. □

Lemma 2. *For all natural numbers n, $\vdash_{\mathbf{K}^\mu} \Box\gamma_n \to \gamma_n$.*

Proof. Let n be a natural number. By Theorem 2, it is enough to show that the formula $\Box\gamma_n \to \gamma_n$ is valid in all generalized models. Let $\mathcal{M}$ be a generalized model and let $w \in \mathcal{M}$. We have to show $\mathcal{M}, w \Vdash \Box\gamma_n \to \gamma_n$. So suppose $\mathcal{M}, w \Vdash \Box\gamma_n$. If w is a reflexive point, we immediately obtain $\mathcal{M}, w \Vdash \gamma_n$ and this finishes the proof. Assume now that w is irreflexive. We have to prove that $(\mathcal{M}, w)$ is n-good. That is, we have to find a finite tree $\mathcal{N}$ and some $v \in \mathcal{M}$ such that $(\mathcal{M}, w) \sim_n (\mathcal{N}, v)$.

Now for any successor u of w, we have $\mathcal{M}, u \Vdash \gamma_n$. Therefore, $(\mathcal{M}, u)$ is n-good and there exists a finite tree $\mathcal{M}_u = (W_u, R_u, V_u)$ and some $w_u \in W_u$ such that $(\mathcal{M}, u) \sim_n (\mathcal{M}_u, w_u)$. Without loss of generality, we may assume that w_u is the root of $\mathcal{M}_u$.

The idea is now to look at the disjoint union of these models and to add a root v (that would be rank n-indistinguishable from w). However, this new model might not be a finite tree (w might have infinitely many successors). The solution is to restrict ourselves to finitely many successors of w. More precisely, for each n-type, we pick at most one successor of w.

So let U be a set of successors of w such that for any successor u of w, there is exactly one point u' of U satisfying $\theta_n(u) = \theta_n(u')$. Remark that since there are only finitely many distinct n-types, U is finite. Let $\mathcal{N} = (W, R, V)$ be the model defined by

$$W = \{v\} \cup \biguplus\{W_u : u \in U\},$$

$$R = \{(v, w_u) : u \in U\} \cup \bigcup\{R_u : u \in U\},$$

$$V(p) = \begin{cases} \{v\} \cup \bigcup\{V_u(p) : u \in U\} & \text{if } \mathcal{M}, w \Vdash p, \\ \bigcup\{V_u(p) : u \in U\} & \text{otherwise,} \end{cases}$$

for all proposition letters p. Since U is finite, $\mathcal{N}$ is a finite tree. Thus, it is enough to check that for any formula φ of rank at most n, we have

$$\mathcal{M}, w \Vdash \varphi \quad \text{iff} \quad \mathcal{N}, v \Vdash \varphi.$$

By Proposition 4, φ is provably equivalent to a boolean combination of proposition letters and formulas of the form $\Diamond\psi$, where $rank(\psi)$ is at most n. Thus, it is enough to show that w and v satisfy exactly the same proposition letters and the same formulas $\Diamond\psi$ with $rank(\psi) \leq n$.

By definition of V, it is immediate that w and v satisfy the same proposition letters. Now let ψ be a formula of rank at most n. We have to show that

$$\mathcal{M}, w \Vdash \Diamond\psi \quad \text{iff} \quad \mathcal{N}, v \Vdash \Diamond\psi.$$

For the direction from left to right, suppose that $\mathcal{M}, w \Vdash \Diamond\psi$. Thus, there exists a successor u of w such that $\mathcal{M}, u \Vdash \psi$. By definition of U, there is $u' \in U$ such that $(\mathcal{M}, u) \sim_n (\mathcal{M}, u')$. Thus, $(\mathcal{M}, u) \sim_n (\mathcal{M}_{u'}, w_{u'})$ and in particular, $\mathcal{M}_{u'}, w_{u'} \Vdash \psi$. By definition of R, it follows that $\mathcal{N}, v \Vdash \Diamond\psi$. The direction from right to left is similar. $\square$

Proposition 5. *For all natural numbers n, $\vdash_{\mathbf{K}^\mu} \mu x.\Box x \to \gamma_n$.*

Proof. By Lemma 2, we know that $\Box\gamma_n \to \gamma_n$ is provable in $\mathbf{K}^\mu$. By the Fixpoint rule, we obtain that $\mu x.\Box x \to \gamma_n$ is provable in $\mathbf{K}^\mu$. $\square$

Theorem 3. $\mathbf{K}^\mu + \mu x.\Box x$ *is complete with respect to the class of finite tree Kripke models.*

Proof. For any finite tree $\mathcal{M}$, we have $\mathcal{M} \Vdash \mathbf{K}^\mu$ and $\mathcal{M} \Vdash \mu x.\Box x$. Thus, it is sufficient to show that if φ is not provable in $\mathbf{K}^\mu + \mu x.\Box x$, there exists a finite tree N such that $N \nVdash \varphi$. Let φ be such a formula. In particular, $\nvdash_{\mathbf{K}^\mu} \mu x.\Box x \to \varphi$. By Theorem 2, we have $\mathcal{M}, w \nVdash \mu x.\Box x \to \varphi$, for some generalized model $\mathcal{M}$ and some $w \in \mathcal{M}$.

Let n be the rank of φ. By Theorem 2 and Proposition 5, we get that $\mathcal{M}, w \Vdash \mu x.\Box x \to \gamma_n$. Since $\mathcal{M}, w \Vdash \mu x.\Box x$, it follows that $\mathcal{M}, w \Vdash \gamma_n$. Therefore, there exists a finite tree $\mathcal{N}$ and some $v \in \mathcal{N}$ such that $(\mathcal{M}, w) \sim_n (\mathcal{N}, v)$. Since $\mathcal{M}, w \nVdash \varphi$, we have $\mathcal{N}, v \nVdash \varphi$.

As mentioned before, this result also follows from the completeness of $\mathbf{K}^\mu$ showed by Igor Walukiewicz in [11]. We briefly explain how to derive Theorem 3 from the completeness of $\mathbf{K}^\mu$. Recall that in [11], Igor Walukiewicz showed that a sentence φ is provable in $\mathbf{K}^\mu$ iff it is valid in all trees.

Suppose that a sentence φ is not provable in $\mathbf{K}^\mu + \mu x.\Box x$. In particular, the formula $\mu x.\Box x \to \varphi$ is not provable in $\mathbf{K}^\mu$. It follows from the completeness of $\mathbf{K}^\mu$ that there is a model $\mathcal{M} = (W, R.V)$ and a point w in W such that (W, R) is a tree and $\mu x.\Box x \to \varphi$ is not true at w. We may assume that w is the root of (W, R).

Since $\mu x.\Box x$ is true at w and since w is the root, it follows from Fact 1 that the tree (W, R) is conversely well-founded. Let n be the rank of φ. Now, if a point v in W has more than one successor of a given n-type θ, we can pick one successor of n-type θ and delete all the other successors of n-type θ. This would not modify the fact that φ is not true at w. By doing this operation inductively and using the fact that (W, R) is well-founded, we can prove that the tree (W, R) may be assumed to be finite. Therefore, there is a finite tree (W, R) in which φ is not valid.

5 Adding Shallow Axioms to $\mathbf{K}^\mu + \mu x.\Box x$

By slightly modifying our method, it is also possible to prove that when we extend the logic $\mathbf{K}^\mu + \mu x.\Box x$ with axioms that are shallow (defined below), we obtain complete axiomatizations for the corresponding class of finite trees.

Definition 13 ([10]). A formula is *Prop-free* if it is a sentence that does not contain any proposition letter. A formula is *propositional* if it is a sentence of the μ-calculus that contains neither $\Diamond$ nor μ.

A formula is *shallow* if no occurrence of a proposition letter is in the scope of a fixpoint operator and each occurrence of a proposition letter is in the scope of at most one modality. In other words, the shallow formulas is the language defined by

$$\varphi \ ::= \ \psi \mid \Diamond\psi \mid \varphi \vee \varphi \mid \neg\varphi,$$

where ψ is either a *Prop*-free formula or a propositional formula.

For example, $\Diamond p \rightarrow \Box p$ is a shallow formula. Other examples are formulas expressing that each point has at most two successors $(\Diamond p \wedge \Diamond(q \vee \neg p) \rightarrow \Box(p \vee q))$, or that each point has at most one blind successor $(\Diamond(p \wedge \Box\bot) \wedge \Box(\Box\bot \rightarrow p))$.

The remaining of the section is devoted to the proof of the following completeness result. Recall that a formula φ defines a class $\mathcal{C}$ of finite trees if $\mathcal{C}$ is exactly the class of trees which make φ valid.

Theorem 4. *Let φ be a shallow formula. Then the logic $\mathbf{K}^\mu + \mu x.\Box x + \varphi$ is complete with respect to the class of finite trees defined by φ.*

In order to prove this result, as for the logic $\mathbf{K}^\mu + \mu x.\Box x$, we first show that the logic is complete with respect to a class of generalized frames. To do so, we combine the completeness of the logic $\mathbf{K}^\mu$ with respect to the class of generalized frames together with a property of shallow formulas, which was proved in [10]. We first recall this property.

Definition 14. A generalized frame $\mathcal{F} = (W, R, \mathbb{A})$ is *differentiated* if for all $w, v \in W$ with $w \neq v$, there exists $A \in \mathbb{A}$ such that $w \in A$ and $v \notin A$.

A generalized model $\mathcal{F} = (W, R, \mathbb{A})$ is *tight* if for all $w, v \in W$ such that $(w, v) \notin R$, there exists $A \in \mathbb{A}$ such that $v \in A$ and for all $u \in A$, $(w, u) \notin R$.

A generalized frame is *refined* if it is differentiated and tight.

Definition 15. A formula φ is *persistent* with respect to refined frames if for all refined frame $\mathcal{F}$ such that $\mathcal{F} \Vdash \varphi$, the formula φ is valid on the underlying Kripke frame of $\mathcal{F}$.

Theorem 5 ([10]). *Every shallow formula is persistent with respect to refined frames.*

Theorem 6. *Let φ be a shallow formula. The logic $\mathbf{K}^\mu + \varphi$ is complete with respect to the class of generalized frames whose underlying Kripke frames make φ valid.*

Proof. By Theorem 2, we know that the logic $\mathbf{K}^\mu + \varphi$ is complete with respect to a generalized model $\mathcal{N} = (W, R, V, \mathbb{A})$. Moreover, it follows from the proof of this theorem, that we may assume $\mathbb{A}$ to be the set $\{[\![\psi]\!]_\mathcal{N} : \psi \text{ sentence }\}$. Therefore, it is enough to show that in the underlying Kripke frame (W, R), φ is valid.

First we prove that φ is valid in the generalized Kripke frame $(W, R, \mathbb{A})$. Let $V' : Prop \to \mathbb{A}$ be a valuation and let $\mathcal{N}'$ be the generalized model $(W, R, V', \mathbb{A})$. We have to show that $\mathcal{N} \Vdash \varphi$. It follows from the definition of $\mathbb{A}$ that for all proposition letters p, there is a formula φ_p such that $V(p) = [\![\varphi_p]\!]_\mathcal{N}$. Now it is routine to show that

$$\mathcal{N}' \Vdash \varphi \quad \text{iff} \quad \mathcal{N} \Vdash \psi,$$

where ψ is a formula obtained by replacing each proposition letter p occurring in φ by the formula φ_p. Using Theorem 2, we obtain that $\mathcal{N} \Vdash \psi$ iff ψ belongs to the logic $\mathbf{K}^\mu + \varphi$. Clearly, ψ belongs to $\mathbf{K}^\mu + \varphi$ since this logic is closed under substitution and this finished the proof that φ is valid in the generalized Kripke frame $(W, R, \mathbb{A})$.

Now, in Theorem 1, we could make the extra assumption that the model $\mathcal{M} = (W_\mathcal{M}, R_\mathcal{M}, V_\mathcal{M})$ is such that the generalized frame $(W_\mathcal{M}, R_\mathcal{M}, \mathbb{A}_\mathcal{M})$ is refined, where $\mathbb{A}_\mathcal{M}$ is the set $\{[\![\psi]\!]_\mathcal{M} : \psi \text{ modal formula }\}$. This is a standard result and follows from the proof of Theorem 1. By looking at the proof of Theorem 2, we can see that this implies that the generalized frame $(W, R, \mathbb{A})$ (as defined in the first paragraph of this proof) is refined. Recall also that we proved that φ is valid in this generalized frame. It follows from Theorem 5 that the formula φ is valid in the Kripke frame (W, R).

Definition 16. Let φ be a formula and let $\mathcal{M}$ be a generalized model. A point $w \in \mathcal{M}$ is *n-good for φ* if there exist a finite tree $\mathcal{G}$, a Kripke model $\mathcal{N}$ based on $\mathcal{G}$ such that $\mathcal{G} \Vdash \varphi$ and $(\mathcal{M}, w) \sim_n (\mathcal{N}, v)$, for some $v \in \mathcal{G}$.

Lemma 3. *Let φ be a formula and let n be a natural number strictly greater than the rank of φ. There exists a formula δ_n of rank n such that for all generalized models $\mathcal{M}$ and all $w \in \mathcal{M}$, we have*

$$\mathcal{M}, w \Vdash \gamma_n \quad \textit{iff} \quad (\mathcal{M}, w) \textit{ is } n\textit{-good for } \varphi.$$

Proof. Let γ_n be the formula given by Lemma 1. We can define δ_n as the formula $\gamma_n \wedge \mu x.\varphi \wedge \Box x$.

The proof of the next lemma is an easy adaptation of the proof of Lemma 2. Details are omitted.

Lemma 4. *Let φ be a shallow formula and let n be a natural number strictly greater than the rank of φ. If δ_n is the formula given by Lemma 3, then $\vdash_{\mathbf{K}^\mu + \varphi} \Box\delta_n \to \delta_n$.*

Proposition 6. *Let φ be a shallow formula and let n be a natural number strictly greater than the rank of φ. If δ_n is the formula given by Lemma 3, $\vdash_{\mathbf{K}^\mu} \mu x.\Box x \to \gamma_n$.*

Proof. By Lemma 4, we know that $\Box\delta_n \to \delta_n$ is provable in $\mathbf{K}^\mu + \varphi$. By the Fixpoint rule, we obtain that $\mu x.\Box x \to \delta_n$ is provable in $\mathbf{K}^\mu + \varphi$.

Theorem 7. *Let φ be a shallow formula. The logic $\mathbf{K}^\mu + \mu x.\Box x + \varphi$ is complete with respect to the class of finite tree defined by φ.*

Proof. It is easy to see that every formula of the logic $\mathbf{K}^\mu + \mu x.\Box x + \varphi$ is valid on all finite trees of the class defined by φ. Thus, it is sufficient to show that if ψ is not provable in $\mathbf{K}^\mu + \mu x.\Box x + \varphi$, there exists a finite tree $\mathcal{G}$ such that $\mathcal{G} \Vdash \varphi$ and $\mathcal{G} \nVdash \psi$. Let ψ be such a formula. In particular, $\nvdash_{\mathbf{K}^\mu + \varphi} \mu x.\Box x \to \psi$. By Theorem 6, there exist a generalized frame $\mathcal{F}$ and generalized model $\mathcal{M}$ based on $\mathcal{F}$ such that $\mathcal{F} \Vdash \varphi$ and $\mathcal{M}, w \nVdash \mu x.\Box x \to \varphi$, for some $w \in \mathcal{F}$.

Let n be a natural number strictly greater than the rank of φ and greater or equal to the rank of ψ. By Theorem 6 and Proposition 6, we get that $\mathcal{M}, w \Vdash \mu x.\Box x \to \delta_n$. Since $\mathcal{M}, w \Vdash \mu x.\Box x$, it follows that $\mathcal{M}, w \Vdash \delta_n$. Therefore, there exist a finite tree $\mathcal{G}$, a Kripke model $\mathcal{N}$ based on $\mathcal{G}$ and $v \in \mathcal{G}$ such that $\mathcal{G} \Vdash \varphi$ and $(\mathcal{M}, w) \sim_n (\mathcal{N}, v)$. Since $\mathcal{M}, w \nVdash \psi$, we have $\mathcal{N}, v \nVdash \psi$.

6 Graded μ-Calculus

By adapting the definition of rank for the graded μ-formulas, we can use the same proof to show that the graded μ-calculus together with the axiom $\mu x.\Box x$ is complete with respect to the class of finite trees. We start by recalling the definition of the graded μ-calculus.

Definition 17. The formulas of the graded μ-calculus are given by

$$\varphi \ ::= \ \top \mid p \mid x \mid \varphi \vee \varphi \mid \neg\varphi \mid \Diamond^k \varphi \mid \mu x.\varphi,$$

where p ranges over the set *Prop* of proposition letters, x ranges over the set Var of variables and k is a natural number. In $\mu x.\varphi$, we require that the variable x appears only under an even number of negations in φ. As before, we assume that Var is infinite and a *graded modal formula* is a formula which does not have any subformula of the form $\mu x.\varphi$.

For all natural numbers k, we let $\Box^k\varphi$ be an abbreviation for $\neg\Diamond^k\neg\varphi$. Moreover, for all $k > 0$, we denote by $\Diamond^{!k}\varphi$ the formula $\neg\Diamond^k\varphi \wedge \Diamond^{k-1}\varphi$. We also let $\Diamond^{!0}\varphi$ be the formula $\neg\Diamond^0\varphi$.

Definition 18. Given a formula φ, a Kripke model $\mathcal{M} = (W, R, V)$ and an assignment $\tau : Var \to \mathcal{P}(W)$, we define a subset $[\![\varphi]\!]_{\mathcal{M},\tau}$ that is interpreted as the set of points at which φ is true. The subset is defined by induction as before, with the extra requirement that

$$\begin{aligned}[\![\Diamond^k\varphi]\!]_{\mathcal{M},\tau} = \{w \in W : \exists\, U \subseteq [\![\varphi]\!]_{\mathcal{M},\tau} \cap \{u \in W : wRu\} \\ \text{s.t. } U \text{ has at least } k+1 \text{ elements}\}.\end{aligned}$$

The notions of *truth* and *validity* are defined as in Definition 2.

Definition 19. The axiomatization of the system $\mathbf{GK}^\mu$ consists of the following axioms and rules

$$\begin{array}{lr}
\text{propositional tautologies,} & \\
\text{If } \vdash \varphi \to \psi \text{ and } \vdash \varphi, \text{ then } \vdash \psi & \text{(Modus ponens),} \\
\text{If } \vdash \varphi, \text{ then } \vdash \varphi[p/\psi] & \text{(Substitution),} \\
\text{If } \vdash \varphi, \text{ then } \vdash \Box^0\varphi & \text{(Necessitation),} \\
\vdash \varphi[x/\mu x.\varphi] \to \mu x.\varphi & \text{(Fixpoint rule),} \\
\Diamond^{k+1}p \to \Diamond^k p & \text{(axiom } G1\text{),} \\
\Box^0(p \to q) \to (\Diamond^n p \to \Diamond^n q) & \text{(axiom } G2\text{),} \\
\Diamond^{!0}(p \wedge q) \to ((\Diamond^{!k}p \wedge \Diamond^{!l}q) \to \Diamond^{!k+l}(p \vee q)) & \text{(axiom } G3\text{),} \\
\vdash \varphi[x/\mu x.\varphi] \to \mu x.\varphi & \text{(Fixpoint axiom),} \\
\text{If } \vdash \varphi[x/\psi] \to \psi, \text{ then } \vdash \mu x.\varphi \to \psi & \text{(Fixpoint rule),}
\end{array}$$

where x is not a bound variable of φ and no free variable of ψ is bound in φ.

The logic **GK** is the smallest set of formulas which contains the propositional tautologies, the axioms $G1$, $G2$ and $G3$ and is closed under the Substitution, the Modus ponens and the Necessitation rules.

Theorem 8 ([4]). *The logic* **GK** *is complete with respect to a single model. That is, there is a Kripke model $\mathcal{M}$ such that a graded modal formula is provable in* **GK** *iff it is true in $\mathcal{M}$.*

Theorem 9. *The logic $\mathbf{GK}^\mu + \mu x.\Box^0 x$ is complete with respect to the class of finite trees. That is, a graded μ-formula is provable in $\mathbf{GK}^\mu + \mu x.\Box^0 x$ iff it is valid in all finite trees.*

Proof (sketch). The structure of the proof is the same as the one for the proof of Theorem 3. So first, we need to define a notion of rank for graded μ-formulas. As before, we start by defining the closure and the depth of a formula. The closure of graded formula is defined as in Definition 6, except that we replace $\Diamond$ by $\Diamond^k$. The depth of a graded μ-formula is defined by induction as follows

$$\begin{array}{l|l}
d(\top) = d(p) = d(x) = 0, & d(\varphi \vee \psi) = max\{d(\varphi), d(\psi)\}, \\
d(\neg\varphi) = d(\varphi), & d(\Diamond^k\varphi) = d(\varphi) + k + 1, \\
d(\mu x.\varphi) = d(\varphi) + 1. &
\end{array}$$

Finally, we can define the rank of a graded μ-formula as in Definition 8.

The second step is to prove completeness of $\mathbf{GK}^\mu$ with respect to the class of generalized frames. Using Theorem 8, it is possible to show this by using a proof that is completely similar to the proof of Theorem 2. We do not give details.

We can do so easily this step because we gave a proof of Theorem 2 which uses directly the completeness result for the modal case, instead of adapting the canonical model construction for the modal case to the μ-calculus. Indeed, the canonical model construction for graded modal logic is rather difficult and it would not be immediate to adapt it when fixpoints are added.

The last step is to show the completeness of $\mathbf{GK}^\mu + \mu x.\Box^0 x$ with respect to the class of finite trees. This is done by extending all the notions and results of section 4 to the setting of the graded μ-calculus. It is immediate how to proceed.

7 Further Work

We believe that this method could be adapted to other cases. In the last section, we considered the fixpoint version of graded modal logic. Graded modal logic is an extension of modal logic with some sort of counting. We could look at fixpoint versions of modal logic extended with richer form of counting. An example would be Presburger modal logic (see, e.g., [2]). Finally, we would like to mention that Stéphane Demri (p.c.) raised the question to which class of coalgebras this proof could be adapted.

References

1. Bojańczyk, M.: Effective characterizations of tree logics. In: PODS 2008: Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 53–66. ACM, New York (2008)
2. Demri, S., Lugiez, D.: Presburger modal logic is only PSPACE-complete. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 541–556. Springer, Heidelberg (2006)
3. Doets, K.: Monadic Π^1_1-theories of Π^1_1-properties. Notre Dame Journal of Formal Logic 30(2) (1989)
4. Fattorosi-Barnaba, M., Cerrato, C.: Graded modalities I. Studia Logica, 47 (1988)
5. Fischer, M., Ladner, R.: Propositional dynamic logic of regular programs. Journal of Computer and System Sciences 18(2) (1979)
6. Gheerbrant, A., ten Cate, B.: Complete axiomatizations of MSO, FO(TC1) and FO(LFP1) on finite trees. In: LFCS (2009)
7. Kozen, D.: Results on the propositional μ-calculus. In: Lee, I., Smolka, S.A. (eds.) CONCUR 1995. LNCS, vol. 962, Springer, Heidelberg (1995)
8. Libkin, L.: Logics for unranked trees: An overview. Logical Methods in Computer Science 2(3) (2006)
9. de Rijke, M., Blackburn, P., Venema, Y.: Modal Logic. Cambridge University Press (1991)
10. ten Cate, B.: Model theory for extended modal languages. PhD thesis, University of Amsterdam, ILLC Dissertation Series DS-2005-01 (2005)
11. Walukiewicz, I.: A note on the completeness of Kozen's axiomatization of the propositional μ-calculus. The Bulletin of Symbolic Logic 2(3) (1996)

When Model-Checking Freeze LTL over Counter Machines Becomes Decidable[⋆]

Stéphane Demri[1] and Arnaud Sangnier[2]

[1] LSV, ENS Cachan, CNRS, INRIA Saclay IdF, France
[2] Dipartimento di Informatica, Università di Torino, Italy

Abstract. We study the decidability status of model-checking freeze LTL over various subclasses of counter machines for which the reachability problem is known to be decidable (reversal-bounded counter machines, vector additions systems with states, flat counter machines, one-counter machines). In freeze LTL, a register can store a counter value and at some future position an equality test can be done between a register and a counter value. Herein, we complete an earlier work started on one-counter machines by considering other subclasses of counter machines, and especially the class of reversal-bounded counter machines. This gives us the opportuniy to provide a systematic classification that distinguishes determinism vs. nondeterminism and we consider subclasses of formulae by restricting the set of atomic formulae or/and the polarity of the occurrences of the freeze operators, leading to the flat fragment.

1 Introduction

Counter machines. Counter machines are ubiquitous computational models that provide a natural class of infinite-state transition systems, suitable for modeling various applications such as broadcast protocols [17], time granularities [10] and programs with pointer variables [6], to quote a few examples. They are also known to be closely related to data logics for which decision procedures can be designed relying on those for counter machines, see e.g. remarkable examples in [5,3]. When dealing with this class of models, most interesting reachability problems are undecidable but subclasses leading to decidability have been designed including reversal-bounded counter machines [25], one-counter machines [26], flat counter machines [18] and vector addition systems with states (see e.g. [32]).

Model-checking with Freeze LTL. In order to verify properties on counter machines, we aim at comparing counter values and we shall use the so-called *freeze* operator. The freeze quantifier in real-time logics has been introduced in the logic TPTL, see e.g. [1]. The formula $x \cdot \phi(x)$ binds the variable x to the time t of the current state: $x \cdot \phi(x)$ is semantically equivalent to $\phi(t)$. This variable-binding mechanism, quite natural when rephrased in first-order logic, is present in various logical formalisms including for example hybrid logics [22,2], freeze LTL [14] and predicate λ-abstraction [30]. Freeze

⋆ A. Sangnier is financed by a postdoctoral fellowship from DGA/ENS Cachan, France. Work also supported by the Agence Nationale de la Recherche, grant ANR-06-SETIN-001.

LTL is a powerful extension of LTL that allows to store counter values in registers. Infinitary satisfiability restricted to one register is already undecidable [14] just as model-checking for nondeterministic one-counter machines [15], which is quite unexpected since one-counter machines seem to be harmless operational models. Moreover, there is some hope that model-checking happens to be more tractable than satisfiability since more constraints are requested on models viewed as runs.

Our contribution. We carry on with the quest started in [15] to determine which classes of counter machines admit decidable model-checking with freeze LTL. In the paper, we consider the above-mentioned classes of counter machines for which the reachability problem is decidable. We provide an exhaustive analysis completing [15]; some results are obtained by adequately adapting known results to our framework or by designing simple reductions. However, at each position, we may have to deal with more than one counter values. Our main technical contributions allow us to establish the following results with a special focus on reversal-bounded counter machines.

- Model-checking freeze LTL (written $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow})$) over deterministic vector addition systems with states and deterministic reversal-bounded counter machines is decidable (see Corollary 11). However, $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow})$ over reversal-bounded counter machines is undecidable, even when restricted to one register (see Theorem 7).
- $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow})$ restricted to flat formulae over reversal-bounded counter machines is decidable (see Corollary 17) as well as the restriction to positively flat formulae over one-counter machines (see Theorem 18), partly by taking advantage of recent results about parameterized one-counter machines [23].

A complete summary can be found in Section 8. As a nice by-product of the classification we made, we show a tight relationship between reachability problems for parameterized counter machines and model-checking counter machines over the flat fragment of freeze LTL (see Section 7.2). Besides, we believe that the principles underlying our undecidability proof for $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow})$ over reversal-bounded counter machines could be reused for other problems on such counter machines.

Because of lack of space, omitted proofs can be found in [16].

2 Standard Classes of Counter Machines

In this section, we recall standard definitions about various classes of counter machines. We write $\mathbb{N}$ [resp. $\mathbb{Z}$] for the set of natural numbers [resp. integers]. Given a dimension $n \geq 1$ and $k \in \mathbb{Z}$, we write $\mathbf{k}$ to denote the vector with all values equal to k and $\mathbf{e}_i$ to denote the unit vector for $i \in \{1, \ldots, n\}$. We recall that a semilinear set of $\mathbb{N}^n$ is a finite union of linear sets. We often refer to Presburger arithmetic which consists of first-order logic over the structure $\langle \mathbb{N}, 0, \leq, + \rangle$ (and more generally over $\langle \mathbb{Z}, 0, \leq, + \rangle$) [31]. It is known that a subset of $\mathbb{N}^k$ is semilinear if and only if it is definable by a formula in Presburger arithmetic [20].

2.1 Counter Machines

A *counter machine* M is defined as a tuple $\langle n, Q, \Delta, q_0 \rangle$ where $n \geq 1$ is the *dimension* of M, Q is a finite set of *control states*, $\Delta \subseteq Q \times G \times A \times Q$ is a finite set of *transitions* where $G = \{\texttt{zero}, \texttt{true}\}^n$ is the finite set of *guards* and $A = \{-1, 0, 1\}^n$ is the finite set of *actions*, and $q_0 \in Q$ is the *initial* control state. Given a counter machine M, we introduce the *transition system* $TS(M) = \langle Q \times \mathbb{N}^n, \rightarrow \rangle$ where $Q \times \mathbb{N}^n$ is the set of *configurations* and $\rightarrow \subseteq (Q \times \mathbb{N}^n) \times (Q \times \mathbb{N}^n)$ is the *transition relation*: for $\langle q, \mathbf{v} \rangle, \langle q', \mathbf{v}' \rangle \in Q \times \mathbb{N}^n$, we have $\langle q, \mathbf{v} \rangle \rightarrow \langle q', \mathbf{v}' \rangle \overset{\text{def}}{\Leftrightarrow}$ there exists a transition $t = \langle q, \mathbf{g}, \mathbf{a}, q' \rangle \in \Delta$ such that: $\mathbf{v}' = \mathbf{v} + \mathbf{a}$ and for $1 \leq c \leq n$, $\mathbf{g}(c) = \texttt{zero}$ implies $\mathbf{v}(c) = 0$. We write $\xrightarrow{*}$ to denote the reflexive and transitive closure of $\rightarrow$ and the reachability set of M is $\mathrm{Reach}(M) \overset{\text{def}}{=} \{\langle q, \mathbf{v} \rangle \mid \langle q_0, \mathbf{0} \rangle \xrightarrow{*} \langle q, \mathbf{v} \rangle\}$. Observe that this reachability set implicitly depends on the initial configuration $\langle q_0, \mathbf{0} \rangle$: this is all what we need in the sequel. A finite (resp. infinite) *run* in $TS(M)$ is a finite (resp. infinite) sequence $\rho = \langle q_0, \mathbf{0} \rangle \rightarrow \langle q_1, \mathbf{v}_1 \rangle \rightarrow \ldots$. A counter machine M is *deterministic* (also known as *single-path*) whenever for each $\langle q, \mathbf{v} \rangle \in \mathrm{Reach}(M)$, there is at most one configuration $\langle q', \mathbf{v}' \rangle$ such that $\langle q, \mathbf{v} \rangle \rightarrow \langle q', \mathbf{v}' \rangle$. In the sequel, we shall use Minsky machines that form a special class of deterministic 2-counter machines.

We present below two types of decision problems when $\mathcal{C}$ is a class of counter machines. The *reachability problem* for the class $\mathcal{C}$ is defined as follows: given a machine $M \in \mathcal{C}$ and a configuration $\langle q, \mathbf{v} \rangle$, does $\langle q_0, \mathbf{0} \rangle \xrightarrow{*} \langle q, \mathbf{v} \rangle$? Similarly, the *generalized repeated reachability problem* for the class $\mathcal{C}$ is defined as follows: given a counter machine $M \in \mathcal{C}$ and N sets $F_1, \ldots, F_N$ of control states, is there a run of M such that for $1 \leq i \leq N$, there is a control state in F_i that is repeated infinitely often?

1CM. One-counter machines are naturally defined as counter machines of dimension one. Various logical formalisms have been introduced to specify the behavior of one-counter machines, including Freeze LTL [15] and EF logic [21]. When one-counter machines are enriched by a finite alphabet (so that transitions are labelled), the universality problem is undecidable [26], witnessing that this simple operational model can lead to natural undecidable problems.

VASS. Vector addition systems with states (a.k.a. VASS) are known to be equivalent to Petri nets, see e.g. [32], and they correspond to counter machines without zero-tests, i.e. each guard has no component equal to zero. To be precise, we are a bit less liberal than the usual definition since we only consider actions in $\{-1, 0, 1\}^n$ (instead of $\mathbb{Z}^n$) but this does not make a real difference for all the developments made in this paper.

Flat counter machines. A directed graph $G = \langle V, E \rangle$ (with $V \subseteq E \times E$) is said to be *flat* whenever each vertex belongs to at most one cycle (path for which the initial and final vertices coincide). A counter machine $\langle n, Q, \Delta, q_0 \rangle$ is *flat* whenever (1) between two control states there is at most one transition and (2) the directed graph $\langle Q, \{\langle q, q' \rangle \in Q^2 : \langle q, \mathbf{g}, \mathbf{a}, q' \rangle \in \Delta\} \rangle$ is flat. Reachability problems have been considered for flat counter machines in [4,18]; for instance it is proved that flat counter machines have an effectively computable semilinear set [4,18], see also [8].

2.2 Reversal-Bounded Counter Machines

The class of *reversal-bounded* counter machines has been introduced in [25] by considering the following restriction: each counter performs only a bounded number of alternations between increasing and decreasing mode. This class of counter machines is particularly interesting because it has been shown that each reversal-bounded counter machine has a semilinear reachability set which can be effectively computed. We present now a more general class of counter machines proposed in [19]. Given a bound $b \in \mathbb{N}$, we consider the number of alternations between increasing and decreasing mode when the value of the considered counter is above b; if for each counter this number of alternations is bounded by a constant $k \in \mathbb{N}$, we say that the counter machine is k-reversal-b-bounded. From now on, we say that a counter machine M is *reversal-bounded* if there exist $k, b \in \mathbb{N}$ such that M is k-reversal-b-bounded and in the sequel, when reversal-bounded counter machines are part of the instances of some decision problems, we assume that they come with their k and b. As mentioned in [19], the above-defined class of reversal-bounded counter machines contains those defined in [25] and it also contains the counter machines for which the set of reachable configurations is finite. To make the distinction, we will call the machines introduced in [25] *Ibarra* reversal-bounded counter machines.

In [19], the authors prove that the reachability problem is decidable for reversal-bounded counter machines (in fact their reachability set is also an effectively computable semilinear set) and in [33] it is proved that the generalized repeated reachability problem for this class of machines is also decidable when considering only one set of control states to be repeated infinitely often. The proof of this last result relies on the fact that this problem is decidable for Ibarra reversal-bounded counter machines [11]. Note that we can easily reduce the generalized reachability problem with $N \geq 1$ sets of control states to its restriction to only one set (the same way the emptiness problem for generalized Bûchi automata can be reduced to the emptiness problem for Büchi automata).

Theorem 1. *The generalized repeated reachability problem for reversal-bounded counter machines is decidable.*

3 LTL with the Freeze Operator

In this section, we present a variant of temporal logic LTL with registers (also known as Freeze LTL) in order to reason about runs from counter machines. In [15], LTL with registers is used to specify properties about one-counter machines. The datum stored in a register is the current counter value and equality tests are performed between a register value and the current counter value. When dealing with counter machines, a register can store the value of a counter c and test it later against the value of counter c' with possibly $c \neq c'$. Below, we present different ways to restrict the equality tests between registers and counters.

Given a finite set Q of control states (possibly empty) and $n \geq 1$, the formulae of the logic $\mathrm{LTL}^{\downarrow}[Q, n]$ are defined as follows:

$$\phi ::= q \;|\; \uparrow_r^c \;|\; \neg\phi \;|\; \phi \wedge \phi \;|\; \phi \vee \phi \;|\; \phi \mathtt{U} \phi \;|\; \phi \mathtt{R} \phi \;|\; \mathtt{X}\phi \;|\; \downarrow_r^c \phi$$

where $q \in Q$, $c \in \{1, \ldots, n\}$ and $r \in (\mathbb{N} \setminus \{0\})$. Intuitively, the modality $\downarrow_r^c$ is used to store the value of the counter c into the register r; the atomic formula $\uparrow_r^c$ holds true if the value stored in the register r is equal to the current value of the counter c. An occurrence of $\uparrow_r^c$ within the scope of some freeze quantifier $\downarrow_r^c$ is bound by it; otherwise it is free. A sentence is a formula with no free occurrence of any $\uparrow_r^c$.

Models of $\mathrm{LTL}^{\downarrow}[Q, n]$ are runs of transition systems from counter machines of dimension n and with a set of control states containing Q. Given a counter machine $\langle n, Q', \Delta, q_0 \rangle$ with $Q \subseteq Q'$ and a run ρ, we write $|\rho|$ to denote its *length* in $\omega + 1$ and the ith configuration ($0 \leq i < |\rho|$) is denoted by $\langle q_i, \mathbf{v}_i \rangle$. A *counter valuation* f is a finite partial map from $\mathbb{N} \setminus \{0\}$ to $\mathbb{N}$. Note that whenever $f(r)$ is undefined, the atomic formula $\uparrow_r^c$ is interpreted as false. Given a run ρ and a position $0 \leq i < |\rho|$, the satisfaction relation $\models$ is defined as follows (Boolean clauses are omitted):

$$\begin{array}{rcl}
\rho, i \models_f q & \overset{\text{def}}{\Leftrightarrow} & q_i = q \\
\rho, i \models_f \uparrow_r^c & \overset{\text{def}}{\Leftrightarrow} & r \in \mathrm{dom}(f) \text{ and } f(r) = \mathbf{v}_i(c) \\
\rho, i \models_f \mathtt{X}\phi & \overset{\text{def}}{\Leftrightarrow} & i + 1 < |\rho| \text{ and } \rho, i + 1 \models_f \phi \\
\rho, i \models_f \phi_1 \mathtt{U} \phi_2 & \overset{\text{def}}{\Leftrightarrow} & \text{for some } i \leq j < |\rho|,\ \rho, j \models_f \phi_2 \\
 & & \text{and for all } i \leq j' < j, \text{ we have } \rho, j' \models_f \phi_1 \\
\rho, i \models_f \phi_1 \mathtt{R} \phi_2 & \overset{\text{def}}{\Leftrightarrow} & \text{for all } i \leq j < |\rho|,\ \rho, j \models_f \phi_2 \\
 & & \text{or for some } i \leq j < |\rho|,\ \rho, j \models_f \phi_1 \\
 & & \quad \text{and for all } i \leq k \leq j,\ \rho, k \models_f \phi_2 \\
\rho, i \models_f \downarrow_r^c \phi & \overset{\text{def}}{\Leftrightarrow} & \rho, i \models_{f[r \mapsto \mathbf{v}_i(c)]} \phi
\end{array}$$

$f[r \mapsto \mathbf{v}_i(c)]$ denotes the register valuation equal to f except that the register r is mapped to $\mathbf{v}_i(c)$. In the sequel, we omit the subscript "f" in $\models_f$ when sentences are involved. We use the standard abbreviations for the temporal operators ($\mathtt{G}$, $\mathtt{F}$, ...) and for the Boolean operators and constants ($\Rightarrow$, $\top$, $\bot$, ...).

We defined below fragments of $\mathrm{LTL}^{\downarrow}[Q, n]$ by restricting the use of the freeze operators. The *strict* fragment, written $\mathrm{LTL}^{\downarrow,s}[Q, n]$, consists in associating a unique counter to each register (to store and to test). More precisely, a formula ϕ in $\mathrm{LTL}^{\downarrow,s}[Q, n]$ verifies the following syntactic property: if $\downarrow_r^c \psi$ is a subformula of ϕ, then ϕ has not subformulae of the form either $\uparrow_r^{c'}$ or $\downarrow_r^{c'} \psi'$ with $c \neq c'$. We also write $\mathrm{LTL}[Q]$ to denote the fragment of $\mathrm{LTL}^{\downarrow}[Q, n]$ in which the atomic formulae of the form $\uparrow_r^c$ are forbidden (and therefore $\downarrow_r^c$ becomes also useless).

Model-checking problems. The infinitary (existential) model-checking problem over counter machines, written $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow}[\cdot, \cdot])$, is defined as follows: given a counter machine $M = \langle n, Q', \Delta, q_0 \rangle$ and a sentence $\phi \in \mathrm{LTL}^{\downarrow}[Q, n]$ with $Q \subseteq Q'$, is there an infinite run ρ such that $\rho, 0 \models \phi$? If the answer is "yes", we write $M \models^{\omega} \phi$. The subproblem of $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow}[\cdot, \cdot])$ with formulae restricted to $\mathrm{LTL}^{\downarrow,s}[Q, n]$ is written $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow,\mathrm{s}}[\cdot, \cdot])$. Given $n \geq 1$, we write $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow}[\cdot, \mathrm{n}])$ to denote the subproblem of $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow}[\cdot, \cdot])$ with counter machines of dimension at most n. Similarly, we write $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow}[\emptyset, \cdot])$ to denote the subproblem of $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow}[\cdot, \cdot])$ with no atomic formula made of control states. Similar notations are used with other fragments of $\mathrm{LTL}^{\downarrow}[Q, n]$. In this existential version of model checking, this problem can be viewed as a variant of satisfiability in which satisfaction of a formula can be only witnessed within a specific class of data words, namely the runs of the counter machine. Note that results for the universal version of model checking will follow easily from those for the existential version when considering fragments closed under negation or deterministic counter machines.

Flat formulae. We say that the occurrence of a subformula in a formula is *positive* if it occurs under an even number of negations, otherwise it is *negative*. Let $\mathcal{L}$ be a fragment of $\mathrm{LTL}^{\downarrow}[Q, n]$. The *flat fragment of* $\mathcal{L}$, written flat-$\mathcal{L}$, is the restriction of $\mathcal{L}$ where, for any occurrence of $\phi_1 \mathtt{U} \phi_2$ [resp. $\phi_2 \mathtt{R} \phi_1$], if it is positive then the freeze operator $\downarrow$ does not occur in ϕ_1, and if it is negative then the freeze operator $\downarrow$ does not occur in ϕ_2. A formula is *positively flat* when it is flat and no occurrence of the freeze operator $\uparrow$ occurs in the scope of an odd number of negations. For example, the formula below belongs to the positively flat fragment and it states that sometimes there is a value of the counter 1 such that (1) infinitely often counter 2 takes that value if and only if infinitely often counter 3 takes that value and (2) from some future position, the counter 4 has always that value: $\mathtt{F} \downarrow_1^1 [(\mathtt{GF} \uparrow_1^2 \Leftrightarrow \mathtt{GF} \uparrow_1^3) \wedge \mathtt{FG} \uparrow_1^4]$. Considering flat fragments remains a standard means to regain decidability: for instance flat fragments of LTL variants have been studied in [9,7] (see also in [27, Section 5] the design of a flat logical temporal language for model-checking pushdown machines). Section 7 shall illustrate that flatness can lead to decidability but this is not always the case.

4 Preliminary Results

In this section, we present preliminary results that will be helpful to strenghten forthcoming results and we present results for flat counter machines and one-counter machines based on existing works. We shall study the effects of restricting the set of atomic formulae, for instance by allowing only atomic formulae that are control states [resp. that are of the form $\uparrow_r^c$].

4.1 Purification, or How to Get Rid of Control States

Control states can be viewed as an internal piece of information about the counter machines and therefore, it is interesting to understand whether the absence of control states among the set of atomic formulae (called herein *purification*) makes a difference. Lemma 2 below roughly shows that control states can be always encoded by patterns for various classes of counter machines.

Lemma 2. *Given a counter machine* $M = \langle n, Q, \Delta, q_0 \rangle$ *and a sentence* ϕ *in* $\mathrm{LTL}^{\downarrow}[Q, n]$*, one can build in logspace a counter machine* $M_P = \langle n+1, Q_P, \Delta_P, q_0 \rangle$ *and a formula* $\phi_P \in \mathrm{LTL}^{\downarrow}[\emptyset, n+1]$ *such that* $M \models^{\omega} \phi$ *iff* $M_P \models^{\omega} \phi_P$*. Moreover,* M *is deterministic [resp. reversal-bounded, flat] iff* M_P *is deterministic [resp. reversal-bounded, flat] and* $\phi \in \mathrm{LTL}^{\downarrow,s}[Q, n]$ *iff* $\phi_P \in \mathrm{LTL}^{\downarrow,s}[\emptyset, n+1]$.

The proof consists in introducing an additional counter whose behavior in M_P encodes the control states from M. The reduction in the proof of Lemma 2 does not preserve the number of counters; however, a purification lemma can be also established for the class of one-counter machines as shown in [15]. By the way, the construction in [15] could be also adapted to encode control states by patterns however, it does not preserve reversal-boundedness.

4.2 Restricting the Atomic Formulae to Control States

Before considering decidability issues with the freeze operator, it is legitimate to wonder what happens when the atomic formulae are restricted to control states. We show below that for all subclasses of counter machines considered in this paper, this restriction leads to decidability (for flat counter machines, the proof is postponed to the next subsection). Basically, the proof is a consequence of the two following properties: LTL formulae can be translated into equivalent Büchi automata (see e.g. [35]) and repeated reachability problem is decidable for the concerned subclasses of counter machines.

Theorem 3. $\mathrm{MC}^{\omega}(\mathrm{LTL}[\cdot])$ *restricted to one-counter machines, VASS, and reversal-bounded counter machines is decidable.*

4.3 Existing Results for Two Subclasses

In this paper, we wish to provide a complete classification with respect to the above-mentioned subclasses. The two following results are known results recasted in our context. First, we observe that $\mathrm{LTL}^{\downarrow}[Q,n]$ can be viewed as a fragment of the temporal logic FOCTL*(Pr) [12] which extends the logic CTL* by allowing the use of Presburger formulae as atomic propositions to describe sets of configurations for a counter machine. Since model-checking FOCTL*(Pr) over flat counter machines is decidable [12], we establish the following theorem.

Theorem 4. $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow}[\cdot,\cdot])$ *restricted to flat counter machines is decidable.*

Moreover, in [15], the authors obtain the following results concerning the model-checking of LTL with registers over one-counter machines.

Theorem 5. *[15] (I)* $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow}[\cdot,1])$ *is undecidable. (II)* $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow}[\cdot,1])$ *restricted to deterministic one-counter machines is* PSPACE*-complete.*

5 Nondeterministic Counter Machines

Herein, we consider the model-checking problems over $\mathrm{LTL}^{\downarrow}[Q,n]$ for nondeterministic counter machines. We have seen that for the class of one-counter machines the problem is undecidable (see Theorem 5(I)) whereas it is decidable for flat counter machines (see Theorem 4). First, we observe that zero-tests can be easily encoded in $\mathrm{LTL}^{\downarrow}[Q,n]$ by first storing the initial value of counters in some register r_0 and then performing a zero-test on counter c with the atomic formula $\uparrow^{c}_{r_0}$.

Theorem 6. $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow}[\cdot,\cdot])$ *restricted to VASS and to positively flat formulae with at most one register is undecidable.*

The proof is based on a simple encoding of zero-tests. For what concerns reversal-bounded counter machines, we have the following result:

Theorem 7. $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow}[\cdot,4])$ *restricted to reversal-bounded counter machines and to formulae with at most one register is undecidable.*

To prove this result, we present a reduction from the halting problem for Minsky machines; note that a similar reduction is used in [28] in order to prove that in reversal-bounded counter machines extended with equality tests between distinct counters, the reachability problem is undecidable.

Proof. (sketch) Let $M = \langle 2, Q, \Delta, q_0 \rangle$ be a Minsky machine (deterministic counter machine with two counters) and $q_F \in Q$ be a final control state with no transition from it. Without any loss of generality, we can assume that if $\langle q, \mathbf{g}, \mathbf{a}, q' \rangle \in \Delta$ performs a decrementation, then the transition is of the form $\langle q, \mathtt{true}, -\mathbf{e}_c, q' \rangle$ for some $c \in \{1,2\}$. Moreover, for $q, q' \in Q$, the set $\{\langle \mathbf{g}, \mathbf{a} \rangle : \langle q, \mathbf{g}, \mathbf{a}, q' \rangle \in \Delta\}$ contains at most one element. Let us build the reversal-bounded counter machine $M = \langle 4, Q', \Delta', (q_0)_\emptyset \rangle$ as follows:

- $Q' = \{q_X : q \in Q,\ X \subseteq \{1,2\}\}$ (X records on which counter of M zero-test is needed next),
- Δ' is the smallest set of transitions satisfying the conditions below:
  - for $X \subseteq \{1,2\}$, $\langle (q_0)_\emptyset, \mathbf{true}, \mathbf{0}, (q_0)_X \rangle \in \Delta'$,
  - for all $\langle q, \mathbf{g}, \mathbf{a}, q' \rangle \in \Delta$, we have $\langle q_1, \mathbf{true}, \mathbf{a}', q'_1 \rangle \in \Delta'$ assuming that
    - $q_1 = q_X$ with $X = \{c \in \{1,2\} : \mathbf{g}(c) = \mathtt{zero}\}$,
    - for $c \in \{1,2\}$,
      - $\mathbf{a}(c) = 1$ implies $\mathbf{a}'(c) = 1$ and $\mathbf{a}'(c+2) = 0$,
      - $\mathbf{a}(c) = -1$ implies $\mathbf{a}'(c) = 0$ and $\mathbf{a}'(c+2) = 1$,
      - $\mathbf{a}(c) = 0$ implies $\mathbf{a}'(c) = \mathbf{a}'(c+2) = 0$.
  - for $X \subseteq \{1,2\}$, $\langle (q_F)_X, \mathbf{true}, \mathbf{0}, (q_F)_X \rangle \in \Delta$ (final loops).

By construction, the counter machine M' is reversal-bounded since the four counters only increase. The idea behind this construction is that the first [resp. second] and the third [resp. fourth] counters of M' respectively count the number of incrementations and decrementations of the first [resp. second] counter of M. No zero-test is performed in M'; in order to simulate a zero-test in M, we would need to test equality between two counters, which is not allowed in our models. Consequently, we encode these equality tests by formulae.

Let us build a formula ϕ in $\mathrm{LTL}^{\downarrow}[Q', 4]$ such that $M' \models^\omega \phi$ iff the control state q_F can be reached from the initial configuration of M. We consider the following auxiliary formulae ($c \in \{1,2\}$):

$$\phi_c \stackrel{\text{def}}{=} \bigvee_{q \in Q} \ \bigvee_{\{c\} \subseteq X \subseteq \{1,2\}} q_X \quad \text{and} \quad \phi_q \stackrel{\text{def}}{=} \bigvee_{X \subseteq \{1,2\}} q_X.$$

We are now in position to define ϕ:

$$\phi \stackrel{\text{def}}{=} \mathtt{F}\phi_{q_F} \wedge \bigwedge_{c \in \{1,2\}} \mathtt{G}(\phi_c \Rightarrow \downarrow_1^c \uparrow_1^{c+2}) \wedge \bigwedge_{c \in \{1,2\}} \mathtt{G}(\bigwedge_{\langle q, \mathbf{true}, -\mathbf{e}_c, q' \rangle \in \Delta} q_\emptyset \wedge \mathtt{X}\phi_{q'} \Rightarrow \downarrow_1^c \neg \uparrow_1^{c+2})$$

It remains to show that $M' \models^\omega \phi$ iff the control state q_F can be reached in M. □

The result of Theorem 7 can be refined by showing the undecidability of the strict fragment $\mathrm{MC}^\omega(\mathrm{LTL}^{\downarrow,\mathrm{s}}[\cdot, 4])$ restricted to reversal-bounded counter machines. Observe that we shall modify the above developments while we are dealing with a strict fragment for which each register is associated with a unique counter.

Theorem 8. $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow,\mathrm{s}}[\cdot,4])$ *restricted to reversal-bounded counter machines is undecidable.*

The proof takes advantage of a refinement in the contruction of the counter machine M' from the proof of Theorem 7 and it is interesting for its own sake. So far, it is still open whether the problem is Σ_1^1-hard since we are "only" able to reduce the halting problem to it.

6 Deterministic Counter Machines

In this section, we restrict ourselves to classes of deterministic counter machines. A class $\mathcal{C}$ of deterministic counter machines has the *PA-property* $\overset{\mathrm{def}}{\Leftrightarrow}$ for each counter machine $M \in \mathcal{C}$, one can effectively build a formula $\phi_M(x_0,\ldots,x_{n+1})$ in Presburger arithmetic such that for all $j_0,\ldots,j_{n+1} \in \mathbb{N}$, $\langle j_0, \langle j_1,\ldots,j_n\rangle\rangle$ is the j_{n+1}th configuration of the unique run of M iff $\langle j_0,\ldots,j_{n+1}\rangle \models \phi_M(x_0,\ldots,x_{n+1})$ (assuming that M has dimension n and its set of control states is viewed as a finite subset of $\mathbb{N}$).

We show below that model-checking restricted to counter machines can be sometimes reduced to the decidable satisfiability problem for Presburger arithmetic.

Lemma 9. *Let $\mathcal{C}$ be a class of deterministic counter machines. If $\mathcal{C}$ has the PA-property, then the model-checking problem* $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow}[\cdot,\cdot])$ *over counter machines in $\mathcal{C}$ is decidable.*

The proof of Lemma 9 is based on an internalization of the satisfaction relation in Presburger arithmetic.

Lemma 10. *Deterministic reversal-bounded counter machines and deterministic VASS have the PA-property.*

Corollary 11. $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow}[\cdot,\cdot])$ *is decidable when restricted to deterministic reversal-bounded counter machines and deterministic VASS.*

Checking whether a VASS is deterministic can be decided by using instances of the covering problem (the problem is actually PSPACE-complete [24]). Checking whether a reversal-bounded counter machine is deterministic is also decidable adding a counter which counts each step and using the fact that the reachability set can be expressed in Presburger arithmetic. By contrast, checking whether a counter machine is reversal-bounded is undecidable [19].

7 Flat Freeze LTL

In this section, we consider the restriction of the model-checking problem to flat formulae only. By Theorem 4, we already know that $\mathrm{MC}^{\omega}(\mathrm{flat}-\mathrm{LTL}^{\downarrow}[\cdot,\cdot])$ restricted to flat counter machines is decidable and that $\mathrm{MC}^{\omega}(\mathrm{flat}-\mathrm{LTL}^{\downarrow}[\cdot,\cdot])$ restricted to VASS is undecidable (the proof of Theorem 6 involves only flat formulae). It is worth observing that flat $\mathrm{LTL}^{\downarrow}[Q,n]$ strictly contains $\mathrm{LTL}[Q]$, and therefore we refine below decidability results from Section 4.2.

7.1 A Detour to Counter Machines with Parameterized Tests

We introduce here parameterized counter machines in order to solve later model-checking problems restricted to flat formulae. First, let us fix some definitions. A *counter machine with parameterized tests* (shortly *parameterized* counter machine) is defined as a counter machine $M = \langle n, Q, \Delta, q_0, Z \rangle$ extended with a finite set Z of integer variables such that the guards $\mathbf{g}$ are among $(\{\mathtt{zero}, \mathtt{true}\} \cup \{=(z), \neq(z), >(z), <(z) \mid z \in Z\})^n$. A *concretization* C of M is a map $\mathrm{C} : Z \rightarrow \mathbb{N}$. Given a parameterized counter machine M and a concretization C, we introduce the transition system $TS(M, \mathrm{C}) = \langle Q \times \mathbb{N}^n, \rightarrow \rangle$ where $\rightarrow \subseteq (Q \times \mathbb{N}^n) \times (Q \times \mathbb{N}^n)$ is defined as follows: for $\langle q, \mathbf{v} \rangle, \langle q', \mathbf{v}' \rangle \in Q \times \mathbb{N}^n$, we have $\langle q, \mathbf{v} \rangle \rightarrow \langle q', \mathbf{v}' \rangle \overset{\text{def}}{\Leftrightarrow}$ there exists a transition $t = \langle q, \mathbf{g}, \mathbf{a}, q' \rangle \in \Delta$ such that $\mathbf{v}' = \mathbf{v} + \mathbf{a}$, and for $1 \leq c \leq n$, $\mathbf{g}(c)$ equals $\mathtt{zero}$ implies $\mathbf{v}(c) = 0$, $\mathbf{g}(c)$ is equal to $=(z)$ implies $\mathbf{v}(c) = \mathrm{C}(z)$, $\mathbf{g}(c)$ is equal to $\neq(z)$ implies $\mathbf{v}(c) \neq \mathrm{C}(z)$, $\mathbf{g}(c)$ is equal to $>(z)$ implies $\mathbf{v}(c) > \mathrm{C}(z)$ and, $\mathbf{g}(c)$ is equal to $<(z)$ implies $\mathbf{v}(c) < \mathrm{C}(z)$. A finite [resp. infinite] *run* in $TS(M, \mathrm{C})$ is a finite [resp. infinite] sequence $\rho = \langle q_0, \mathbf{0} \rangle \rightarrow \langle q_1, \mathbf{v}_1 \rangle \rightarrow \ldots$. The *parameterized reachability problem* for counter machines is defined as follows: given a parameterized counter machine M and a configuration $\langle q, \mathbf{v} \rangle$, is there a concretization C such that $\langle q_0, \mathbf{0} \rangle \xrightarrow{*} \langle q, \mathbf{v} \rangle$ in $TS(M, \mathrm{C})$? Even if the parameterized reachability problem is obviously undecidable, we will see in this section that some restrictions lead to decidability. We will say that a parameterized counter machine is *Ibarra reversal-bounded* if the classical counter machine obtained by replacing each parameterized test by $\mathtt{true}$ is Ibarra reversal-bounded. We have then the following result.

Theorem 12. *[28] The parameterized reachability problem for Ibarra reversal-bounded parameterized counter machines is decidable.*

If a parameterized counter machine has no guard of the form either $\neq(z)$ or $<(z)$, we say it is *restricted*. In [23], parametric one-counter machines are defined as extensions of one-counter machines extended with actions consisting in incrementing or decrementing the unique counter with some parameterized integer constants. In [23], it is shown that the reachability problem for this class of one-counter machines is decidable. Here is a corollary.

Lemma 13. *The parameterized reachability problem for restricted parameterized one-counter machines is decidable.*

The proof of Lemma 13 consists in substituting each test of the form $=(z)$ by the following sequence of instructions: decrement by z, perform a zero-test and increment by z. In order to encode the test $>(z)$, we use the same technique except that we do not introduce a zero-test between the decrementation (in fact we also add a decrementation by 1 and an incrementation by 1) and the incrementation. Note that this method does not work if we allow guards of the form either $\neq(z)$ or $<(z)$, because the value of the counter cannot be negative, hence the decidability of the parameterized reachability problem for one-counter machines remains an open problem.

We introduce here a new problem which is needed to reduce the considered model-checking problem. The *parameterized generalized repeated reachability problem* for parameterized counter machines is defined as follows: given a parameterized counter machine M, N sets $F_1, \ldots, F_N$ of control states, are there a concretization C and an infinite run of $TS(M, \mathrm{C})$ such that for $1 \leq i \leq N$, one control state in F_i is repeated infinitely often?

From the previous theorem and lemma, we deduce the following corollary.

Corollary 14. *The parameterized generalized repeated reachability problem is decidable when considering Ibarra reversal-bounded parameterized counter machines and restricted parameterized one-counter machines.*

7.2 Flat Formulae and Parameterized Counter Machines

For $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow}[\cdot, \cdot])$ restricted to flat formulae, we have the following result.

Theorem 15. *There is a reduction from* $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow}[\cdot, \cdot])$ *restricted to flat formulae to the parameterized generalized repeated reachability problem for counter machines.*

Proof. (sketch) Let $M = \langle n, Q, \Delta, q_0 \rangle$ be a counter machine and ϕ be a flat sentence belonging to $\mathrm{LTL}^{\downarrow}[Q, n]$. Without any loss of generality, we can assume that ϕ is in negation normal form (which means that all the occurrences of negation appear only in front of atomic formulae). Moreover, we can assume that if $\downarrow_r^c \psi$ and $\downarrow_{r'}^{c'} \psi$ are distinct occurrences of subformulae in ϕ, then $r \neq r'$ (this may just linearly increase the number of registers). Consequently, if $\psi_1 \mathtt{U} \psi_2$ [resp. $\psi_1 \mathtt{R} \psi_2$] is a subformula of ϕ, then the freeze operator $\downarrow$ cannot occur in ψ_1 [resp. ψ_2]. We shall effectively build a parameterized counter machine $M' = \langle n, Q', \Delta', q_0, Z' \rangle$ and sets $F_1, \ldots, F_N \subseteq Q'$ for which there is a concretization C and an infinite run of $TS(M', \mathrm{C})$ such that for $1 \leq i \leq N$, one control state in F_i is repeated infinitely often iff $M \models^{\omega} \phi$.

Let us fix some notations. As usual, the formula ϕ can be encoded as a finite tree whose leaves are labelled by atomic formulae and internal nodes are labelled by (Boolean, temporal or freeze) connectives. Each node of the formula tree corresponds naturally to a subformula and the set of nodes can be viewed as a finite prefix-closed subset $\mathrm{occ}(\phi) \subseteq (\mathbb{N} \setminus \{0\})^*$ (finite sequence of natural numbers). Each element in $\mathrm{occ}(\phi)$ corresponds to the occurrence of a subformula in ϕ; hence two occurrences may correspond to the same subformula. The use of occurrences instead of subformulae is motivated by the need to provide formal and clear statements in which occurrences are crucial. For each occurrence $u \in \mathrm{occ}(\phi)$, we write $\phi(u)$ to denote the corresponding subformula in ϕ; for instance $\phi(\epsilon) = \phi$. Moreover, when u is a prefix of u', written $u \leq_{\mathrm{pre}} u'$, we know that $\phi(u')$ is a subformula of $\phi(u)$. We write $\mathrm{occ}^{\downarrow}(\phi)$ [resp. $\mathrm{occ}^{\uparrow}(\phi)$] to denote the set of occurrences corresponding to formulae whose outermost connective is of the form $\downarrow_r^c$ [resp. $\uparrow_r^c$]. Let $m = \mathrm{card}(\mathrm{occ}^{\downarrow}(\phi))$. Observe that if $m = 0$, then we are in the case of $\mathrm{MC}^{\omega}(\mathrm{LTL}[\cdot])$ which has been treated in Section 4.2. In the sequel, we assume that $m > 0$. Given $u \in \mathrm{occ}^{\uparrow}(\phi)$ with $\phi(u) = \uparrow_r^c$, we write $\mathrm{bind}(u)$ to denote the longest prefix of u (with respect to $\leq_{\mathrm{pre}}$) in $\mathrm{occ}^{\downarrow}(\phi)$ such that $\phi(\mathrm{bind}(u))$ is of the form $\downarrow_r^{c'} \psi$ (i.e., with the same register). An *atom* X is a subset of $\mathrm{occ}(\phi)$ satisfying the conditions below (we abusively use subformulae to denote occurrences corresponding to formulae with the appropriate outermost connective): (1) if $\psi_1 \wedge \psi_2 \in X$, then $\psi_1, \psi_2 \in X$; (2) for all atomic formulae $\psi \in X$, $\{\psi, \neg\psi\} \not\subseteq X$; (3) if $\psi_1 \vee \psi_2 \in X$, then either $\psi_1 \in X$ or $\psi_2 \in X$; (4) if $\downarrow_r^c \psi \in X$, then $\psi \in X$. The set of *atoms* of ϕ is denoted by $\mathrm{AT}(\phi)$. A pair of atoms $\langle X, X' \rangle$ is said to be *one-step consistent* iff the conditions below hold true: (I) if $\psi_1 \mathtt{U} \psi_2 \in X$, then either $\psi_2 \in X$ or ($\psi_1 \in X$ and $\psi_1 \mathtt{U} \psi_2 \in X'$); (II) if $\psi_1 \mathtt{R} \psi_2 \in X$, then $\psi_2 \in X$ and ($\psi_1 \in X$ or $\psi_1 \mathtt{R} \psi_2 \in X'$); (III) if $\mathtt{X}\psi \in X$, then $\psi \in X'$; (IV) No atom X'' strictly included in X' satisfies the conditions (I)–(III) (by replacing X' by X''). We will now describe the construction of the parameterized counter machine M' which will use m integer variables $z_1, \ldots, z_m$. Intuitively, each integer variable will be used to store the value of a register. In order to make explicit this dependancy, we shall use a one-to-one map $reg : \mathrm{occ}^{\downarrow}(\phi) \rightarrow \{1, \ldots, m\}$. We define also a function $counter : \mathrm{occ}^{\downarrow}(\phi) \cup \mathrm{occ}^{\uparrow}(\phi) \rightarrow \{1, \ldots, n\}$ that indicates the counter involved in the subformula. Given $u \in \mathrm{occ}^{\downarrow}(\phi)$ such that $\phi(u) = \downarrow_r^c \psi$, we have $counter(u) = c$ and given $u \in \mathrm{occ}^{\uparrow}(\phi)$ such that $\phi(u) = \uparrow_r^c$, we have $counter(u) = c$. The set Q' of control states is equal to $\{q_0\} \uplus Q \times \mathrm{AT}(\phi)$ plus some auxiliary control states that are introduced to perform tests. The relation Δ' is defined as follows. First, $\langle q_0, \mathbf{true}, \mathbf{0}, \langle q_0, Y \rangle \rangle \in \Delta'$ whenever $\epsilon \in Y$ and no atom strictly included in Y contains ϵ (init). Then, for each transition $\langle q, \mathbf{g}, \mathbf{a}, q' \rangle \in \Delta$ there is in Δ' the following sequence of transitions $\langle q, Y \rangle \cdots q_1^{aux} \cdots q_T^{aux} \xrightarrow{\mathbf{g},\mathbf{a}} \langle q', Y' \rangle$ assuming that:

1. $\mathrm{occ}^{\downarrow}(\phi) \cap Y$ contains T_1 elements, say $u_1, \ldots, u_{T_1}$; $\mathrm{occ}^{\uparrow}(\phi) \cap Y$ contains T_2 elements, say $u_{T_1+1}, \ldots, u_{T_1+T_2}$; $\{u \in Y \mid u{\cdot}1 \in \mathrm{occ}^{\uparrow}(\phi)$ and $\phi(u)$ is a negation$\}$ contains T_3 elements, say $u_{T_1+T_2+1}, \ldots, u_{T_1+T_2+T_3}$ with $T = T_1 + T_2 + T_3$,
2. $\langle Y, Y' \rangle$ is a one-step consistent pair,
3. $\{\phi(u) : u \in Y\} \cap Q \subseteq \{q\}$ and $\neg q \notin \{\phi(u) : u \in Y\}$,
4. for $i \in \{1, \ldots, T_1\}$ [resp. $i \in \{1, \ldots, T_2\}$, $i \in \{1, \ldots, T3\}$], before reaching the control state q_i^{aux} [resp. $q_{T_1+i}^{aux}$, $q_{T_1+T_2+i}^{aux}$], there exists a transition testing equality [resp. equality, inequality] between z_k and the counter $counter(u_i)$ [resp. $counter(u_{T_1+i})$, $counter(u_{T_1+T_2+i})$] with the identity $k = reg(u_i)$ [resp. $k = reg(\mathrm{bind}(u_{T_1+i}))$, $k = reg(\mathrm{bind}(u_{T_1+T_2+i}))$].

Finally, let $u_1, \ldots, u_N$ be the occurrences in $\mathrm{occ}(\phi)$ such that the outermost temporal connective of $\phi(u_i)$ is the until operator $\mathtt{U}$. Then, for $1 \leq i \leq N$, $F_i = \{\langle q, Y \rangle : u_i \notin Y$ or $(u_i \cdot 2) \in Y\}$. It remains to show that $M \models^{\omega} \phi$ iff there exist a concretization C and an infinite run of $TS(M', \mathrm{C})$ such that for $1 \leq i \leq N$, one control state in F_i is repeated infinitely often. □

7.3 Decidability Results

Remark that if the counter machine M is Ibarra reversal-bounded, then the parameterized counter machine M' built from M and the flat formula ϕ is Ibarra reversal-bounded. Using Corollary 14 and Theorem 15, we conclude that $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow}[\cdot,\cdot])$ restricted to Ibarra reversal-bounded counter machines and to flat formulae is decidable. Furthermore this can be extended to the class of reversal-bounded counter machines, using Lemma 16 below.

Lemma 16. *There is an exponential-time reduction from* $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow}[\cdot,\cdot])$ *restricted to reversal-bounded counter machines into* $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow}[\cdot,\cdot])$ *restricted to Ibarra reversal-bounded counter machines. Furthermore this reduction preserves flatness of the formulae.*

Corollary 17. $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow}[\cdot,\cdot])$ *restricted to reversal-bounded counter machines and to flat formulae is decidable.*

Finally, assume the formula ϕ is a positively flat formula (see Section 3). For all atoms $Y \in \mathrm{AT}(\phi)$, the set $\{u \in Y \mid u \cdot 1 \in \mathrm{occ}^{\uparrow}(\phi)$ and $\phi(u)$ is a negation$\}$ is empty. So, in the construction of M' from M and ϕ, we only use parameterized tests of the form $=(z)$. Hence, if M is a one-counter machine and ϕ is a positively flat formula, we deduce that M' is a restricted parameterized one-counter machine. Using Corollary 14 and Theorem 15, we get the result below.

Theorem 18. $\mathrm{MC}^{\omega}(\mathrm{LTL}^{\downarrow}[\cdot,\cdot])$ *restricted to one-counter machines and to positively flat formulae is decidable.*

In order to extend Theorem 18 to the full flat fragment, one needs to perform inequality tests in parameterized one-counter machines, which is so far unclear how to perform while preserving decidabibility of the corresponding parameterized reachability problem. This generalization is left as an open problem.

8 Concluding Remarks

In this paper, we have studied the decidability status of model-checking freeze LTL over various subclasses of counter machines for which the reachability problem is known to be decidable. Our most remarkable technical contributions concern reversal-bounded counter machines and flat formulae. Besides, we have established an original link between reachability problems for parameterized counter machines and model-checking counter machines over the flat fragment of freeze LTL. The table below contains a summary of the main results (**D** stands for decidability, **U** for undecidability) in which the columns referred to restriction either on the counter machines or on the formulae. Sometimes, an additional restriction between parentheses is indicated in order to emphasize that the result holds true for a stricter fragment. Bibliographical references in the table indicate that the related result is mainly due to the referred work.

	Det.	**NDet.**	**Flat formulae**	**No $\uparrow^c_r$**
RB	**D** **Cor. 11**	**U (strictness)** **Theo. 8**	**D** **Cor. 17**	**D** **[11]**
1CM	**PSPACE-C.** **[15]**	**U (1 reg.)** **[15]**	**open | D for pos. flatness** **Theo. 18**	**PSPACE-C.** **[34,13]**
Flat CM	**D**	**D** **Theo. 4**	**D**	**D**
VASS	**D** **Cor. 11**	**U (1 reg.)** **Theo. 6**	**U** **Theo. 6**	**D** **[29]**

Here are a few rules of thumb: determinism, flat counter machines and no freeze lead to decidability. However, flat formulae often guarantee decidability (except for VASS) whereas reversal-boundedness can lead to decidability (but the restriction with a single register leads to undecidability). Finally, throwing away the atomic formulae made of control states does not help for decidability. Even though we have established various decidability results, the complexity of the decision problems is far from being known, mainly because we use reductions to Presburger arithmetic. Such characterizations are part of future work.

Acknowledgments. We would like to thank the anonymous referees for their comments and suggestions on a preliminary version.

References

1. Alur, R., Henzinger, T.: A really temporal logic. JACM 41(1), 181–204 (1994)
2. 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)
3. Björklund, H., Bojańczyk, M.: Bounded depth data trees. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 862–874. Springer, Heidelberg (2007)
4. Boigelot, B.: Symbolic methods for exploring infinite state spaces. PhD thesis, Université de Liège (1998)
5. Bojańczyk, M., Muscholl, A., Schwentick, T., Segoufin, L., David, C.: Two-variable logic on words with data. In: LICS 2006, pp. 7–16. IEEE, Los Alamitos (2006)
6. 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)
7. Bouyer, P., Markey, N., Ouaknine, J., Worrell, J.: The cost of punctuality. In: LICS 2007, pp. 109–118. IEEE, Los Alamitos (2007)
8. Bozga, M., Iosif, R., Lakhnech, Y.: Flat parametric counter automata. Fundamenta Informaticae 91(2), 275–303 (2009)
9. Comon, H., Cortier, V.: Flatness is not a weakness. In: Clote, P.G., Schwichtenberg, H. (eds.) CSL 2000. LNCS, vol. 1862, pp. 262–276. Springer, Heidelberg (2000)
10. Dal Lago, U., Montanari, A., Puppis, G.: On the equivalence of automaton-based representations of time granularities. In: TIME 2007, pp. 82–93. IEEE, Los Alamitos (2007)
11. Dang, Z., Ibarra, O., San Pietro, P.: Liveness verification of reversal-bounded multicounter machines with a free counter. In: Hariharan, R., Mukund, M., Vinay, V. (eds.) FSTTCS 2001. LNCS, vol. 2245, pp. 132–143. Springer, Heidelberg (2001)
12. Demri, S., Finkel, A., Goranko, V., van Drimmelen, G.: Towards a model-checker for counter systems. In: Graf, S., Zhang, W. (eds.) ATVA 2006. LNCS, vol. 4218, pp. 493–507. Springer, Heidelberg (2006)
13. Demri, S., Gascon, R.: The effects of bounding syntactic resources on Presburger LTL. Journal of Logic and Computation 19(6), 1541–1575 (2009)
14. Demri, S., Lazić, R., Nowak, D.: On the freeze quantifier in constraint LTL: decidability and complexity. I&C 205(1), 2–24 (2007)
15. Demri, S., Lazić, R., Sangnier, A.: Model checking freeze LTL over one-counter automata. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 490–504. Springer, Heidelberg (2008)

16. Demri, S., Sangnier, A.: When Model-Checking Freeze LTL over Counter Machines Becomes Decidable. Research report, LSV, ENS Cachan (2010)
17. Esparza, J., Finkel, A., Mayr, R.: On the verification of broadcast protocols. In: LICS 1999, pp. 352–359. IEEE, Los Alamitos (1999)
18. Finkel, A., Leroux, J.: How to compose Presburger accelerations: Applications to broadcast protocols. In: FST&TCS 2002. LNCS, vol. 2256, pp. 145–156. Springer, Heidelberg (2002)
19. Finkel, A., Sangnier, A.: Reversal-bounded counter machines revisited. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 323–334. Springer, Heidelberg (2008)
20. Ginsburg, S., Spanier, E.H.: Semigroups, Presburger formulas, and languages. Pacific Journal of Mathematics 16(2), 285–296 (1966)
21. Göller, S., Mayr, R., To, A.: On the computational complexity of verifying one-counter processes. In: LICS 2009, pp. 235–244. IEEE, Los Alamitos (2009)
22. Goranko, V.: Hierarchies of modal and temporal logics with references pointers. Journal of Logic, Language, and Information 5, 1–24 (1996)
23. Haase, C., Kreutzer, S., Ouaknine, J., Worrell, J.: Reachability in succinct and parametric one-counter automata. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009 - Concurrency Theory. LNCS, vol. 5710, pp. 369–383. Springer, Heidelberg (2009)
24. Howell, R., Jančar, P., Rosier, L.: Completeness results for single-path Petri nets. I&C 106(2), 253–265 (1993)
25. Ibarra, O.: Reversal-bounded multicounter machines and their decision problems. JACM 25(1), 116–133 (1978)
26. Ibarra, O.: Restricted one-counter machines with undecidable universe problems. Mathematical Systems Theory 13(181), 181–186 (1979)
27. Ibarra, O., Dang, Z.: On removing the stack from reachability constructions. In: Eades, P., Takaoka, T. (eds.) ISAAC 2001. LNCS, vol. 2223, pp. 244–256. Springer, Heidelberg (2001)
28. Ibarra, O., Su, J., Dang, Z., Bultan, T., Kemmerer, R.: Counter machines and verification problems. TCS 289(1), 165–189 (2002)
29. Jančar, P.: Decidability of a temporal logic problem for Petri nets. TCS 74(1), 71–93 (1990)
30. Lisitsa, A., Potapov, I.: Temporal logic with predicate λ-abstraction. In: TIME 2005, pp. 147–155. IEEE, Los Alamitos (2005)
31. Presburger, M.: Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In: Comptes Rendus du Premier Congrès de Mathématiciens des Pays Slaves, Warsaw, pp. 92–101 (1929)
32. Reutenauer, C.: The Mathematics of Petri nets. Masson and Prentice (1990)
33. Sangnier, A.: Vérification de systèmes avec compteurs et pointeurs. Thèse de doctorat, LSV, ENS Cachan, France (2008)
34. Sistla, A., Clarke, E.: The complexity of propositional linear temporal logic. JACM 32(3), 733–749 (1985)
35. Vardi, M., Wolper, P.: Reasoning about infinite computations. I&C 115, 1–37 (1994)

Model Checking *Is* Static Analysis of Modal Logic

Flemming Nielson and Hanne Riis Nielson

DTU Informatics, Technical University of Denmark
{nielson,riis}@imm.dtu.dk

Abstract. Flow Logic is an approach to the static analysis of programs that has been developed for functional, imperative and object-oriented programming languages and for concurrent, distributed, mobile and cryptographic process calculi. In this paper we extend it to deal with modal logics and prove that it can give an exact characterisation of the semantics of formulae in a modal logic. This shows that model checking can be performed by means of state-of-the-art approaches to static analysis and allow us to conclude that the problems of model checking and static analysis are reducible to each other. In terms of computational complexity we show that model checking by means of static analysis gives the same complexity bounds as are known for traditional approaches to model checking.

1 Introduction

Model checking [10,2] is a successful approach to the validation of properties expressed in modal logics with respect to models expressed as transition systems. The transitions may originate from descriptions of hardware or more recently of software systems. Much work focuses on transition systems that are finite and have no structured data although extensions to infinite systems and structured data (e.g. allowing cryptographic terms) exist.

Static analysis [9,13] is in a similar way a successful approach to the validation of properties of programming languages. Originally used in the development of compilers it has spread to uses in editors in software development environments, program validation, program understanding and is also being applied to distributed formalisms such as process calculi. A number of "schools" exist, including Data Flow Analysis, Type and Effect Systems, and Abstract Interpretation to name but a few.

Flow Logic [17] is a particular approach to static analysis that borrows methods and techniques from Abstract Interpretation, Data Flow Analysis and Constraint Based Analysis while presenting the analysis in a style more reminiscent of Type Systems. One of the hallmarks of Flow Logic it that it makes a clear distinction between *(i)* the specification of the analysis, *(ii)* whether or not a proposed analysis result is indeed correct with respect to the semantics, and *(iii)* the computation of the best analysis result. The logical format used for presenting specifications focuses on ensuring the implementability of the analyses — often this is possible in low polynomial (cubic) time. Over the years Flow Logic has proved to be a robust approach able to deal with a wide variety of programming paradigms (e.g. [17]) and calculi of computation (e.g. [4,14,18]).

The problem. The interplay between model checking and static analysis has intrigued many researchers for many years. Early comparisons focused on static analyses over-approximating the solution set while model checking similarly under-approximating the solution set (in case of non-termination). Successive enhancements within model checking and static analysis add to the complexity of understanding their interplay.

A number of papers have taken the view that static analysis *is* model checking of formulae in suitable modal logics. Possibly the first paper is [22] that focused on how to understand the construction of complex data flow equations through their characterisation in a modal μ-calculus. In [20] it is shown how abstract interpretation can be used to cast further light on the construction and this is extended in [21] where an Action Computation Tree Logic is used to express the logical properties. Finally, [11] describe a Java based software system utilising some of these ideas.

The contribution. In this paper we add to the conceptual understanding by showing that model checking really amounts to a static analysis of the modal formulae. To be concrete we choose an *Action Computation Tree Logic* (ACTL [12]) and develop a static analysis in the Flow Logic approach by means of *Alternation-free Least Fixed Point Logic* (ALFP [15]) as used in the Succinct Solver [16] — indeed this is the first paper to extend Flow Logic to deal with modal logic.

At the conceptual level we believe that the developments of Steffen and Schmidt on the one hand, and our present contribution on the other, jointly show the close relationship between model checking and static analysis – to the extent that they seem to be reducible to each other. Clearly we cannot formulate this as precisely as reducibility among computational problems for the good reason that the notions of model checking and static analysis have not been clearly formalised and there may even be lack of consensus on where the exact boundaries are.

Overview. In Section 2 we give the necessary background information on labelled transition systems and Action Computation Tree Logic. Section 3 then presents the essence of the Flow Logic approach to static analysis including the Alternation-free Least Fixed Point Logic that often plays a key role in the implementations of Flow Logics. Our main contribution is in Section 4 where we present a precise static analysis of Action Computational Tree Logic within Flow Logic and prove its correctness – thereby giving further insights on the relationship between model checking and static analysis. We conclude in Section 5.

2 Modal Logic

Labelled Transition Systems. A *labelled transition system* (LTS) has the form $(S, \mathcal{A}, \to)$ where S is a non-empty set of states, $\mathcal{A}$ is a non-empty set of actions and $\to \subseteq S \times \mathcal{A} \times S$ is the transition relation. We shall write $s \to^a s'$ whenever $(s, a, s') \in \to$. A transition system is *finite* whenever both S and $\mathcal{A}$ are finite. A transition system is *finitely branching* whenever the sets $\mathsf{sp}(s) = \{(a, s') \mid s \to^a s'\}$ are finite for all choices of s; clearly a finite transition system is also finitely branching.

A *path* π is a *maximal* sequence $(s_i \to^{a_i} s_{i+1})_i$ such that $s_i \to^{a_i} s_{i+1}$ for all $i \geq 0$. This means that a path is either infinite or ends in a state that is *stuck*; the latter means that there are no outgoing transitions for any action. Sometimes we write the path in the form $(s_i \to^{a_i} s_{i+1})_{0 \leq i < n}$ to make it clear that $i \geq 0$ and that the path has length $n \in \{0, \cdots, \infty\}$; it follows that if $n \neq \infty$ then s_n is stuck. If s_0 is a stuck state then there is exactly one path $(s_i \to^{a_i} s_{i+1})_i$ starting in s_0, namely the empty path.

One can arrange to avoid stuck states by adding a new state s_{stuck} and ensure that all stuck states, as well as s_{stuck}, have a transition to s_{stuck} for some possibly new action; in this case all paths will be infinite and this is the usual assumption in model checking where transition systems are supposed to be Kripke structures. However, this is not usually the case for transition systems generated from programming languages or process calculi; consequently we have opted for a treatment where a path may end in a stuck state.

Action Computation Tree Logic. We shall use a variant of the modal logic *Action Computation Tree Logic* (ACTL) [12] to express properties of paths in labelled transition systems. It is defined by:

$$\begin{aligned} \phi ::= {} & \mathsf{true} \mid \mathsf{false} \mid bp \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi \mid \phi_1 \Rightarrow \phi_2 \\ \mid {} & \mathbf{EX}_{\Omega}\ \phi \mid \mathbf{AX}_{\Omega}\ \phi \mid \mathbf{E}[\phi_1\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2] \mid \mathbf{A}[\phi_1\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2] \end{aligned}$$

The subscripts $\Omega \subseteq \mathcal{A}$ are sets of actions used to restrict the transitions taken. The base predicates bp (so far left unspecified) denote sets of states used to restrict the target state of the transitions taken. The **E** modality quantifies over the existence of paths, the **A** modality quantifies over all paths, **X** focuses on the next step and **U** is an until modality; we have chosen[1] a version of **U** that focuses on eventually taking an action in a certain set Ω_2 to reach a state satisfying a certain property ϕ_2. The interpretation of the modal operators is given in Table 1; when expressing the interpretation of a formula in the state s_0 and next choosing a path $(s_i \to^{a_i} s_{i+1})_{0 \leq i < n}$ it is intended that the state s_0 is indeed the first state in the path $(s_i \to^{a_i} s_{i+1})_{0 \leq i < n}$; we use this convention throughout the paper.

[1] This means that we do *not* obtain CTL by merely setting all Ω, Ω_1, Ω_2 to $\mathcal{A}$. Other choices of **U** are clearly possible without jeopardising our development. The present choice is from [8].

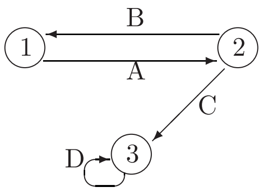
Table 1. Satisfaction relation for modal operators of ACTL: $s \models \phi$

$$\begin{array}{rcl}
s_0 \models \mathbf{EX}_\Omega\ \phi & \underline{\text{iff}} & \exists (s_i \to^{a_i} s_{i+1})_{0 \le i < n} : n > 0 \wedge a_0 \in \Omega \wedge s_1 \models \phi \\
s_0 \models \mathbf{AX}_\Omega\ \phi & \underline{\text{iff}} & \forall (s_i \to^{a_i} s_{i+1})_{0 \le i < n} : n > 0 \wedge a_0 \in \Omega \wedge s_1 \models \phi \\
s_0 \models \mathbf{E}[\phi_1\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2] & \underline{\text{iff}} & \exists (s_i \to^{a_i} s_{i+1})_{0 \le i < n} : \exists k < n : \\
& & \quad [\bigwedge_{0 \le i < k} (a_i \in \Omega_1 \wedge s_{i+1} \models \phi_1) \wedge (a_k \in \Omega_2 \wedge s_{k+1} \models \phi_2)] \\
s_0 \models \mathbf{A}[\phi_1\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2] & \underline{\text{iff}} & \forall (s_i \to^{a_i} s_{i+1})_{0 \le i < n} : \exists k < n : \\
& & \quad [\bigwedge_{0 \le i < k} (a_i \in \Omega_1 \wedge s_{i+1} \models \phi_1) \wedge (a_k \in \Omega_2 \wedge s_{k+1} \models \phi_2)]
\end{array}$$

The modalities **AF**, **EF**, **AG** and **EG** are derivable in this logic[2] in the standard way. The **AG** modality is of special interest to us; its derived interpretations is:

$$s_0 \models \mathbf{AG}_\Omega\ \phi \quad \underline{\text{iff}} \quad \forall (s_i \to^{a_i} s_{i+1})_{0 \le i < n} : \bigwedge_{0 \le i < n} [a_i \in \Omega \Rightarrow s_{i+1} \models \phi]$$

Example 1. Consider a transition system with states $S = \{1, 2, 3\}$, actions $\mathcal{A} = \{A, B, C, D\}$ and transition relation $\to$ given by the diagram to the left:



ϕ	$\{s \mid s \models \phi\}$
$\mathbf{EX}_\mathcal{A}$ goal	$\{2,3\}$
$\mathbf{AX}_\mathcal{A}$ goal	$\{3\}$
$\mathbf{E}[\text{true}\ {}_\mathcal{A}\mathbf{U}_\mathcal{A}\ \text{goal}]$	$\{1,2,3\}$
$\mathbf{A}[\text{true}\ {}_\mathcal{A}\mathbf{U}_\mathcal{A}\ \text{goal}]$	$\{3\}$
$\mathbf{AG}_\mathcal{A}$ goal	$\{3\}$
$\mathbf{AG}_{\{C\}}$ goal	$\{1,2,3\}$

The table to the right shows the validity of some formulae when goal is the base predicate that is only satisfied in the state 3. □

Model Checking. Model checking is the problem of obtaining efficient ways of computing the set $\mathsf{mc}_\phi = \{s \mid s \models \phi\}$ of states that satisfy a given modal formula. In global model checking these algorithms usually proceed in a syntax directed manner on the formula ϕ where it is assumed that the sets of satisfying states have been computed for all sub-formulae. For the base clauses the construction is immediate; for example, $\mathsf{mc}_{\mathsf{true}} = S$ and $\mathsf{mc}_{\mathsf{false}} = \emptyset$. For the clauses involving propositional operators the construction is rather straightforward; for example, $\mathsf{mc}_{\phi_1 \wedge \phi_2} = \mathsf{mc}_{\phi_1} \cap \mathsf{mc}_{\phi_2}$ and $\mathsf{mc}_{\phi_1 \Rightarrow \phi_2} = \mathsf{mc}_{\phi_2} \cup (S \setminus \mathsf{mc}_{\phi_1})$. For the modal operators the construction is somewhat more complex and we refer to standard textbooks like [2] for an overview of existing methods and techniques.

The worst case time complexity of model checking an ACTL formula ϕ of size $|\phi|$ is $\mathcal{O}((|T| + |S|)\,|\phi|)$ where the transition system $(S, \mathcal{A}, \to)$ has a state space of size $|S|$ and a transition relation $\to$ of size $|T|$ and where we assume that the number of base predicates and actions are constant. This result has been adapted from the information given in [2] about the time complexity of model checking CTL (which essentially is ACTL without consideration about the actions).

[2] As above we do not get the CTL equivalents simply by setting Ω to $\mathcal{A}$.

3 Static Analysis

Flow Logic. The aim of static analysis is to estimate the computational behaviour of programs or systems. The idea is to capture the information of interest by elements of complete lattices; the details of the complete lattices depend on the actual property of interest. In the Flow Logic approach *logical judgements* are used to *specify* when the analysis information correctly captures the information of the program or system. The judgements are defined by a set of rules that, in general, must be interpreted co-inductively; for syntax-directed definitions this coincides with the traditional inductive interpretations.

The *correctness* of the analysis is often established as a subject reduction result – much as one will do for a type system. To ensure that the analysis is *implementable* it is customary to establish a Moore family, or model intersection property. This is one of the important points where Flow Logic distinguishes itself from traditional type systems as it ensures that we can determine a best analysis result for all programs – at the same time the result provides a strong link between Flow Logic and Abstract Interpretation.

Example 2. To illustrate the approach we shall review a small example based on the λ-calculus; we refer to [17] for more details. So let the expressions $e \in \mathsf{Exp}$ be given by

$$e ::= x \mid \lambda x.e \mid e_1\, e_2$$

where $x \in \mathsf{Var}$ denotes a variable. The notion of free variables $\mathsf{fv}(e)$ are defined as usual and we shall write $v \in \mathsf{Val}$ for the set of expressions given by $v ::= x \mid \lambda x.e$.

We shall assume that the semantics is given by a labelled transition system where $S = \mathsf{Exp}$ is the set of states, $\mathcal{A} = \mathsf{Val} \times \mathsf{Val}$ is the set of actions (recording the function being applied and its argument) and the transition relation $\rightarrow$ is given by the following axioms and rules

$$(\lambda x.e)\, v \rightarrow^{(\lambda x.e, v)} e[v/x] \qquad \frac{e_1 \rightarrow^{\beta} e_1'}{e_1\ e_2 \rightarrow^{\beta} e_1'\ e_2} \qquad \frac{e_2 \rightarrow^{\beta} e_2'}{e_1\ e_2 \rightarrow^{\beta} e_1\ e_2'}$$

where $e[v/x]$ is the expression obtained by replacing all free occurrences of x in e by v. Note that the actions are used to record the redex of the evaluation step.

A typical analysis for the λ-calculus is a *control flow analysis*. The aim is to approximate the potential values for each sub-expression and to do so it is necessary also to approximate the potential values of the variables. In order to obtain an analysis that is as precise as possible we shall assume that the variables have been α-renamed apart and furthermore we shall assign unique labels $\ell \in \mathsf{Lab}$ to each sub-expression of the expression of interest; thus expressions (and values) now take the form e^{ℓ}. The labels only serve as pointers into the syntax; they simply allow us to pinpoint the various sub-expressions so that we can speak about their values and thus have no semantic significance. So in particular we have

$$((\lambda x.e^{\ell'})^{\ell_1}\, v^{\ell_2})^{\ell} \rightarrow^{((\lambda x.e^{\ell'})^{\ell_1}, v^{\ell_2})} (e[v/x])^{\ell}$$

Table 2. Flow Logic for the λ-calculus: $\mathcal{C}, \mathcal{R} \vdash e^{\ell}$

$\mathcal{C}, \mathcal{R} \vdash x^{\ell}$	iff	$\mathcal{R}(x) \subseteq \mathcal{C}(\ell)$
$\mathcal{C}, \mathcal{R} \vdash (\lambda x.e^{\ell'})^{\ell}$	iff	$(\mathcal{C}, \mathcal{R} \vdash e^{\ell'}) \ \wedge \ (\lambda x.e^{\ell'}) \in \mathcal{C}(\ell)$
$\mathcal{C}, \mathcal{R} \vdash (e_1^{\ell_1}\ e_2^{\ell_2})^{\ell}$	iff	$(\mathcal{C}, \mathcal{R} \vdash e_1^{\ell_1}) \ \wedge \ (\mathcal{C}, \mathcal{R} \vdash e_2^{\ell_2}) \ \wedge$ $\forall (\lambda x.e_0^{\ell_0}) \in \mathcal{C}(\ell_1) : (\mathcal{C}(\ell_2) \subseteq \mathcal{R}(x) \wedge \mathcal{C}(\ell_0) \subseteq \mathcal{C}(\ell))$

The two complete lattices of interest in our analysis are:

– $\mathcal{C} : \mathsf{Lab} \to \mathcal{P}(\mathsf{Val})$: the expression labelled ℓ may evaluate to a value in $\mathcal{C}(\ell)$,
– $\mathcal{R} : \mathsf{Var} \to \mathcal{P}(\mathsf{Val})$: the variable x may evaluate to a value in $\mathcal{R}(x)$.

The analysis is then defined by judgements of the form

$$\mathcal{C}, \mathcal{R} \vdash e^{\ell}$$

as shown in Table 2. The first clause expresses that any value of x is also a possible value of the expression x^{ℓ} and hence any information contained in $\mathcal{R}$ about x must also be contained in $\mathcal{C}(\ell)$. The second clause insists that the body of the λ-expression must be analysable and additionally that the λ-expression itself is a possible value of the expression $(\lambda x.e^{\ell'})^{\ell}$. The clause for function application first insists that the operator as well as the operand must be analysable. Additionally it says that if the operator (i.e. $e_1^{\ell_1}$) evaluates to a λ-expression with formal parameter x and body $e_0^{\ell_0}$ then any value that the actual parameter (i.e. the operand $e_2^{\ell_2}$) might evaluate to is also a possible value of x and any value that the body $e_0^{\ell_0}$ might evaluate to is also a possible value of the overall function application $(e_1^{\ell_1}\ e_2^{\ell_2})^{\ell}$.

Semantic correctness of this simple analysis amounts to ensuring that the judgement $\mathcal{C}, \mathcal{R} \vdash e^{\ell}$ correctly captures the evaluation of the λ-expression e. We can express this as a *subject reduction result*:

Proposition. If $\mathcal{C}, \mathcal{R} \vdash e^{\ell}$ and $e^{\ell} \to^{\beta} e'^{\ell}$ then $\mathcal{C}, \mathcal{R} \vdash e'^{\ell}$.
We can also express the *adequacy* of the analysis by stating that $\mathcal{C}$ correctly captures the function applications taking place:

Proposition. If $\mathcal{C}, \mathcal{R} \vdash e^{\ell}$ and $e^{\ell} \to^{(v_1^{\ell_1}, v_2^{\ell_2})} e'^{\ell}$ then $v_1 \in \mathcal{C}(\ell_1)$ and $v_2 \in \mathcal{C}(\ell_2)$.
Linking back to ACTL we can use the above propositions to show that

$$\mathbf{AG}_{\{(v_1^{\ell_1}, v_2^{\ell_2})\}}\ (v_1 \in \mathcal{C}(\ell_1) \wedge v_2 \in \mathcal{C}(\ell_2))$$

is a true formula that describes the overall adequacy of the analysis. It is observations like this one that has lead some researchers to suggest that static analysis is an instance of model checking [22,20,21].

In addition to proving semantic correctness of the analysis we shall want it to satisfy a *Moore family*, or a *model intersection* property. Recall, that a Moore family is a subset of a complete lattice that is closed under greatest lower bounds.

Proposition. $\{(\mathcal{C},\mathcal{R}) \mid \mathcal{C},\mathcal{R} \vdash e^{\ell}\}$ is a Moore family for all e^{ℓ}.
This result ensures that all expressions can be analysed and have a least, or best, analysis result – this links into the framework of Abstract Interpretation. □

Alternation-free Least Fixed Point Logic. The Moore family result guarantees the existence of best analysis results but in itself it does not provide any mechanism for constructing the analysis result. Here *Alternation-free Least Fixed Point Logic* (ALFP [15]) has proved very useful for obtaining efficient implementations. It is defined by:

$$\begin{array}{lll}
v & ::= & c \mid x \mid f(v_1,\ldots,v_k) \\
pre & ::= & R(v_1,\ldots,v_k) \mid \neg R(v_1,\ldots,v_k) \mid pre_1 \wedge pre_2 \mid pre_1 \vee pre_2 \\
 & \mid & \forall x : pre \mid \exists x : pre \\
cl & ::= & R(v_1,\ldots,v_k) \mid \mathsf{true} \mid cl_1 \wedge cl_2 \mid pre \Rightarrow cl \mid \forall x : cl
\end{array}$$

The clauses are interpreted over a non-empty universe $\mathcal{U}$; indeed, a constant c is an element of $\mathcal{U}$, a function f has arity $\mathcal{U}^* \rightarrow \mathcal{U}$, a variable x ranges over $\mathcal{U}$, and a relation R is a subset of $\mathcal{U}^*$.

The interpretation of the logic is given in terms of satisfaction relations

$$(\varrho,\sigma)\ \underline{\mathsf{sat}}\ pre \qquad \text{and} \qquad (\varrho,\sigma)\ \underline{\mathsf{sat}}\ cl$$

where ϱ is an interpretation of relations and σ is an interpretation of variables which we extend to operate on values by setting $\sigma(v) = v$. The interpretation is standard. We shall say that a clause is *closed* if it contains no free variables. For closed clauses the interpretation σ of the variables is of no importance. Fixing a specific interpretation σ_0 we thus have that $(\varrho,\sigma)\ \underline{\mathsf{sat}}\ cl$ agrees with $(\varrho,\sigma_0)\ \underline{\mathsf{sat}}\ cl$ whenever cl is closed.

The presence of negation in preconditions require some care in order to ensure the existence of least models. An *occurrence* of a relation R in a clause is a sub-formula of the form $R(v_1,\cdots,v_k)$. It is a *negative use* if it has the form $\neg\, R(v_1,\cdots,v_k)$; this necessarily occurs in a precondition, i.e. to the left of an implication. It is a *positive use* if it is not a negative use but occurs in a precondition, i.e. to the left of an implication. All other occurrences are *definitions* and often occur to the right of an implication.

Example 3. Using the equality predicate EQ the clause

$$\forall s : \forall a : \forall s' : T(s,a,s') \wedge \neg EQ(s,s') \Rightarrow R(s,a,s')$$

defines the relation R to be the subset of the labelled transition system represented by T that contains no self-loops. The occurrence of T is a positive use, the occurrence of EQ a negative use and the occurrence of R is a definition. □

A clause cl is *stratified* if there is a number r, an assignment of numbers called ranks $\mathsf{rank}_R \in \{0,\cdots,r\}$ to each relation R, and a way to write cl in the form $\bigwedge_{0\leq i\leq r} cl_i$ such that the following holds for all clauses:

Table 3. Flow Logic in ALFP for the λ-calculus: $\boldsymbol{C}, \boldsymbol{R} \vdash e^{\ell}$

$$\begin{array}{rcl}
\boldsymbol{C}, \boldsymbol{R} \vdash x^{\ell} & \underline{\text{iff}} & \forall v : R_x(v) \Rightarrow C_{\ell}(v) \\
\boldsymbol{C}, \boldsymbol{R} \vdash (\lambda x.e^{\ell'})^{\ell} & \underline{\text{iff}} & (\boldsymbol{C}, \boldsymbol{R} \vdash e^{\ell'}) \wedge C_{\ell}(\lambda x.e^{\ell'}) \\
\boldsymbol{C}, \boldsymbol{R} \vdash (e_1^{\ell_1}\, e_2^{\ell_2})^{\ell} & \underline{\text{iff}} & (\boldsymbol{C}, \boldsymbol{R} \vdash e_1^{\ell_1}) \wedge (\boldsymbol{C}, \boldsymbol{R} \vdash e_2^{\ell_2}) \wedge \\
& & [\forall (\lambda x.e_0^{\ell_0}) : [\forall v : C_{\ell_1}(\lambda x.e_0^{\ell_0}) \wedge C_{\ell_2}(v) \Rightarrow R_x(v)] \wedge \\
& & \qquad [\forall v' : C_{\ell_0}(v') \Rightarrow C_{\ell}(v')]]
\end{array}$$

- if cl_i contains a definition of R then[3] $\mathsf{rank}_R \geq i$;
- if cl_i contains a positive use of R then $\mathsf{rank}_R \leq i$; and
- if cl_i contains a negative use of R then $\mathsf{rank}_R < i$.

Formulae without any occurrences of negation are clearly stratified as one may simply choose $r = 0$. The clause of Example 3 is stratified: simply let EQ and T have rank 0 and let R have rank 1.

Subject to the choice of ranks made above we can define a lexicographic ordering, $\sqsubseteq$, on the the interpretations of relations, ϱ, as follows: $\varrho_1 \sqsubseteq \varrho_2$ if there exists a rank $i \in \{0, \cdots, r\}$ such that (1) $\varrho_1(R) = \varrho_2(R)$ whenever $\mathsf{rank}_R < i$, (2) $\varrho_1(R) \subseteq \varrho_2(R)$ whenever $\mathsf{rank}_R = i$, and (3) either $i = r$ or $\varrho_1(R) \subset \varrho_2(R)$ for some R with $\mathsf{rank}_R = i$. This turns the set of interpretations of relations into a complete lattice. From [15] we then have:

Theorem 1. *The set* $\{\varrho \mid (\varrho, \sigma_0)\ \underline{\mathsf{sat}}\ cl\}$ *is a Moore Family, i.e. is closed under greatest lower bounds, whenever* cl *is closed and stratified; the greatest lower bound* $\sqcap \{\varrho \mid (\varrho, \sigma_0)\ \underline{\mathsf{sat}}\ cl\}$ *is the* least *model of* cl*.*

More generally, given ϱ_0 *the set* $\{\varrho \mid (\varrho, \sigma_0)\ \underline{\mathsf{sat}}\ cl \wedge \varrho_0 \sqsubseteq \varrho\}$ *is a Moore Family and* $\sqcap \{\varrho \mid (\varrho, \sigma_0)\ \underline{\mathsf{sat}}\ cl \wedge \varrho_0 \sqsubseteq \varrho\}$ *is the least model.*

Intuitively, ALFP has been defined to be the *largest subset* of predicate logic that allows us to prove the existence of least models; in particular, clauses contain no disjunction or existential quantification.

Example 4. Following the overall methodology of Flow Logic we shall now reformulate the analysis of Example 2 in ALFP. The idea is to introduce a predicate R_x for each of the variables x and corresponding to $\mathcal{R}(x)$ and similarly a predicate C_ℓ for each of the labels ℓ of expressions and corresponding to $\mathcal{C}(\ell)$. The analysis is then rephrased to use judgements of the form

$$\boldsymbol{C}, \boldsymbol{R} \vdash e^{\ell}$$

and the definition of Table 2 is transformed to be within ALFP as shown in Table 3. In this case we do not need the full power of ALFP; indeed we are within the Datalog fragment [5,1]. $\square$

[3] This generalises the condition $\mathsf{rank}_i = i$ considered in [15].

The Succinct Solver. The least model guaranteed by Theorem 1 can be constructed efficiently as summarised in the following result from [15].

Theorem 2. *Under the assumptions of Theorem 1 the least model given by* $\sqcap\{\varrho \mid (\varrho,\sigma_0)\ \underline{\mathsf{sat}}\ cl \wedge \varrho_0 \sqsubseteq \varrho\}$ *is computable in time* $\mathcal{O}(|\varrho_0| + |\mathcal{U}|^K|cl|)$ *whenever* $|\varrho_0|$ *is the size of* ϱ_0 *and* $|\mathcal{U}|$ *is the size of the (necessarily finite) universe* $\mathcal{U}$ *and finally* K *is the maximal nesting depth of quantifiers within* cl*.*

The Succinct Solver [15,16] computes the least model guaranteed by Theorem 1 and has a worst case time complexity as given by Theorem 2. For many clauses the worst case time complexity does not manifest itself and the Succinct Solver operates in such a way that it may then exhibit a running time substantially lower than the worst case time complexity. Indeed, [15] gives a formula estimating the less than worst case time complexity on a given clause.

In essence the Succinct Solver deals with stratification by computing the relations in increasing order on their rank and therefore the negations present no obstacles. It combines the top-down solving approach of Le Charlier and van Hentenryck [6] with the propagation of differences [7], an optimisation technique for distributive frameworks which is also known in the area of deductive databases [3] or as reduction of strength transformations for program optimisation [19]. Programmed using functional programming its efficient operation is due to the disciplined use of continuations and memoisation as well as arbitrarily branching prefix trees as a universal data-structure for storing relations and for organising sets of waiting consumers. Whenever a new tuple is inserted into a relation this makes it easy to access the continuation for traversing the parts of the clause that might be influenced by the tuple.

Example 5. The analysis specified in Table 3 can be solved in time $\mathcal{O}(N^3)$ where N is the size of the λ-expression analysed; to see this note that the maximal nesting depth of quantifiers is 2 and that the clause and universe have (within a constant factor) the same size as the λ-expression. □

4 Flow Logic for Modal Logic

Returning to ACTL we shall now use the Flow Logic approach to define for each formula ϕ a relation R_ϕ characterising those states where the formula ϕ holds. Much as in the above analysis of the λ-calculus we shall use judgements of the form $\boldsymbol{R} \vdash \phi$ to define the relations R_ϕ of interest. Since the same sub-formula may occur several times in ϕ we shall identify each sub-formula with a unique label ℓ; in examples and explanations where this problem does not arise we shall dispense with the labels.

We assume that

- for each basic predicate bp we have a corresponding relation P_{bp} on states,
- for each subset Ω of $\mathcal{A}$ we have a relation Ω on actions, and
- the transition relation $\rightarrow$ is presented by a ternary relation T.

Table 4. ACTL in ALFP: $\boldsymbol{R} \vdash \phi^{\ell}$ for defining $R_{\phi^{\ell}}$ (part 1)

$\boldsymbol{R} \vdash \mathsf{true}^{\ell}$	<u>iff</u>	$[\forall s : R_{\mathsf{true}^{\ell}}(s)]$
$\boldsymbol{R} \vdash \mathsf{false}^{\ell}$	<u>iff</u>	true
$\boldsymbol{R} \vdash bp^{\ell}$	<u>iff</u>	$[\forall s : P_{bp}(s) \Rightarrow R_{bp^{\ell}}(s)]$
$\boldsymbol{R} \vdash (\phi_1^{\ell_1} \wedge \phi_2^{\ell_2})^{\ell}$	<u>iff</u>	$\boldsymbol{R} \vdash \phi_1^{\ell_1} \wedge \boldsymbol{R} \vdash \phi_2^{\ell_2} \wedge [\forall s : R_{\phi_1^{\ell_1}}(s) \wedge R_{\phi_2^{\ell_2}}(s) \Rightarrow R_{(\phi_1^{\ell_1} \wedge \phi_2^{\ell_2})^{\ell}}(s)]$
$\boldsymbol{R} \vdash (\phi_1^{\ell_1} \vee \phi_2^{\ell_2})^{\ell}$	<u>iff</u>	$\boldsymbol{R} \vdash \phi_1^{\ell_1} \wedge \boldsymbol{R} \vdash \phi_2^{\ell_2} \wedge [\forall s : R_{\phi_1^{\ell_1}}(s) \vee R_{\phi_2^{\ell_2}}(s) \Rightarrow R_{(\phi_1^{\ell_1} \vee \phi_2^{\ell_2})^{\ell}}(s)]$
$\boldsymbol{R} \vdash (\neg \phi^{\ell'})^{\ell}$	<u>iff</u>	$\boldsymbol{R} \vdash \phi^{\ell'} \wedge [\forall s : (\neg R_{\phi^{\ell'}}(s)) \Rightarrow R_{(\neg \phi^{\ell'})^{\ell}}(s)]$
$\boldsymbol{R} \vdash (\phi_1^{\ell_1} \Rightarrow \phi_2^{\ell_2})^{\ell}$	<u>iff</u>	$\boldsymbol{R} \vdash \phi_1^{\ell_1} \wedge \boldsymbol{R} \vdash \phi_2^{\ell_2} \wedge [\forall s : \neg R_{\phi_1^{\ell_1}}(s) \vee R_{\phi_2^{\ell_2}}(s) \Rightarrow R_{(\phi_1^{\ell_1} \Rightarrow \phi_2^{\ell_2})^{\ell}}(s)]$

Table 5. ACTL in ALFP: $\boldsymbol{R} \vdash \phi^{\ell}$ for defining $R_{\phi^{\ell}}$ (part 2)

$\boldsymbol{R} \vdash (\mathbf{EX}_{\Omega}\, \phi^{\ell'})^{\ell}$	<u>iff</u>	$\boldsymbol{R} \vdash \phi^{\ell'} \wedge$ $[\forall s : [\exists a : \exists s' : T(s,a,s') \wedge \Omega(a) \wedge R_{\phi^{\ell'}}(s')]$ $\Rightarrow R_{(\mathbf{EX}_{\Omega}\, \phi^{\ell'})^{\ell}}(s)]$
$\boldsymbol{R} \vdash (\mathbf{AX}_{\Omega}\, \phi^{\ell'})^{\ell}$	<u>iff</u>	$\boldsymbol{R} \vdash \phi^{\ell'} \wedge$ $[\forall s : [\forall a : \forall s' : \neg T(s,a,s') \vee (\Omega(a) \wedge R_{\phi^{\ell'}}(s'))] \wedge$ $[\exists a : \exists s' : T(s,a,s')] \Rightarrow R_{(\mathbf{AX}_{\Omega}\, \phi^{\ell'})^{\ell}}(s)]$
$\boldsymbol{R} \vdash (\mathbf{E}[\phi_1^{\ell_1}\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2^{\ell_2}])^{\ell}$	<u>iff</u>	$\boldsymbol{R} \vdash \phi_1^{\ell_1} \wedge \boldsymbol{R} \vdash \phi_2^{\ell_2} \wedge$ $[\forall s : [\exists a : \exists s' : T(s,a,s') \wedge \Omega_2(a) \wedge R_{\phi_2^{\ell_2}}(s')]$ $\Rightarrow R_{(\mathbf{E}[\phi_1^{\ell_1}\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2^{\ell_2}])^{\ell}}(s)] \wedge$ $[\forall s : [\exists a : \exists s' : T(s,a,s') \wedge \Omega_1(a) \wedge R_{\phi_1^{\ell_1}}(s') \wedge$ $R_{(\mathbf{E}[\phi_1^{\ell_1}\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2^{\ell_2}])^{\ell}}(s')] \Rightarrow R_{(\mathbf{E}[\phi_1^{\ell_1}\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2^{\ell_2}])^{\ell}}(s)]$
$\boldsymbol{R} \vdash (\mathbf{A}[\phi_1^{\ell_1}\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2^{\ell_2}])^{\ell}$	<u>iff</u>	$\boldsymbol{R} \vdash \phi_1^{\ell_1} \wedge \boldsymbol{R} \vdash \phi_2^{\ell_2} \wedge$ $[\forall s : [\ [\exists a : \exists s' : T(s,a,s')] \wedge$ $[\forall a : \forall s' : \neg T(s,a,s') \vee [\Omega_2(a) \wedge R_{\phi_2^{\ell_2}}(s')] \vee$ $[\Omega_1(a) \wedge R_{\phi_1^{\ell_1}}(s') \wedge R_{(\mathbf{A}[\phi_1^{\ell_1}\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2^{\ell_2}])^{\ell}}(s')]\ ]$ $\Rightarrow R_{(\mathbf{A}[\phi_1^{\ell_1}\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2^{\ell_2}])^{\ell}}(s)]$

The interpretations of these predicates are fixed and the intention is to define the judgements $\boldsymbol{R} \vdash \phi$ such that $s \models \phi$ holds whenever $R_{\phi}(s)$ holds in the least model satisfying $\boldsymbol{R} \vdash \phi$. The definition is given in Tables 4 and 5 and is explained below.

The relation R_{true} corresponding to the ACTL formula true should hold for all states and we express this by the ALFP clause $\forall s : R_{\mathsf{true}}(s)$. The relation R_{false} corresponding to false does not hold on any state so the ALFP clause does not insist on R_{false} holding on any states; hence we simply use the ALFP clause true to reflect this meaning that in the least model there are no states where R_{false} holds. For basic predicates bp we have to make use of the predefined predicate P_{bp} and impose the constraint that if P_{bp} holds on some state s then so does the relation R_{bp}.

The four clauses for the propositional operators $\wedge$, $\vee$, $\neg$ and $\Rightarrow$ follow the same pattern so let us just explain one of them, namely conjunction $\phi_1 \wedge \phi_2$. Here the conjuncts $\boldsymbol{R} \vdash \phi_1$ and $\boldsymbol{R} \vdash \phi_2$ ensure that the relations $R_{\phi'}$ corresponding to sub-expressions of ϕ_1 and ϕ_2 correctly record which states are acceptable. The third conjunct $\forall s : R_{\phi_1}(s) \wedge R_{\phi_2}(s) \Rightarrow R_{\phi_1 \wedge \phi_2}(s)$ then caters for the relation $R_{\phi_1 \wedge \phi_2}$. Note that in the case of negation and implication we have *negative uses* of relations; we shall return to the stratification issues related to this later.

Turning to the modal operators of Table 5 let us first consider $\mathbf{EX}_{\Omega}\,\phi$. The first conjunct ensures that the sub-formula ϕ is handled correctly. The second conjunct captures the semantics of the **EX** construct: for all states s there must be an action a and a state s' such that there is a transition in the corresponding transition system (i.e. $T(s, a, s')$), that a is in Ω (i.e. $\Omega(a)$) and furthermore ϕ holds in s', i.e. that $R_{\phi}(s')$ holds. If these conditions are satisfied then it also must be the case that $R_{\mathbf{EX}_{\Omega}\,\phi}$ holds on s.

The clause for $\mathbf{AX}_{\Omega}\,\phi$ is slightly more complicated since we must cater for the possibility of stuck states in the transition system. The clause in Table 5 reflects that two conditions must be satisfied in order for the formula to hold on a given state s. First it must be the case that any transition going out of s must make use of an action from Ω and must lead to a state satisfying ϕ. The other conjunct ensures that s is not a stuck state. Only if both conditions are fulfilled the ALFP clause imposes the requirement that $R_{\mathbf{AX}_{\Omega}\,\phi}$ holds on s. Note that also in this case we have a *negative use* of a relation. Furthermore we are stepping outside the Datalog [5,1] fragment of ALFP in that we use universal quantification in a precondition of an implication.

The clause for $\mathbf{E}[\phi_1\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2]$ captures two possibilities. There might be a transition from s using an action from Ω_2 and resulting in a state satisfying R_{ϕ_2} and if so then $R_{\mathbf{E}[\phi_1\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2]}$ should also hold on s. Alternatively there might be a transition out of s using an action from Ω_1 and in this case it has to lead to a state satisfying not only R_{ϕ_1} but also $R_{\mathbf{E}[\phi_1\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2]}$ – and only if this is the case we impose the condition that $R_{\mathbf{E}[\phi_1\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2]}$ also holds on s. It is here worth pointing out that the ALFP clause defining the relation of interest is recursive in the sense that it contains both *uses* and *definitions* of the relation.

Finally let us consider the clause for $\mathbf{A}[\phi_1\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2]$. Here the first component of the premise of the implication insists that the state s of interest is not a stuck state. The second component then requires that a transition out of s either makes use of an Ω_2 action and then the next state satisfies R_{ϕ_2} or it makes use of an Ω_1 action and then the next state satisfies R_{ϕ_1} as well as $R_{\mathbf{A}[\phi_1\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2]}$. If these premises are all satisfied then the clause expresses that $R_{\mathbf{A}[\phi_1\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2]}$ should hold on s. This is by far the most complex of the clauses as it makes use of universal quantification in preconditions, has *negative uses* of relations and defines the relation of interest in a recursive manner as discussed above.

In order to complete the presentation of the Flow Logic of Tables 4 and 5 we should like to establish that the ALFP clause constructed for an ACTL formula is indeed *closed* and *stratified*. We shall do so below and this will allow us to make use of Theorem 1 to show that least models exists – and that the Succinct Solver can be used to compute them.

Example 6. Consider the ACTL formula $\mathbf{AF}_{\mathcal{A}}\ \mathsf{goal}$ expressing that the transition system of interest eventually will reach a goal state. It is equivalent to $\mathbf{A}[\mathsf{true}\ {}_{\mathcal{A}}\mathbf{U}_{\mathcal{A}}\ \mathsf{goal}]$ (omitting annotations) and using the definition in Tables 4 and 5 we obtain the following ALFP clause defining the predicate $R_{\mathbf{A}[\mathsf{true}\ {}_{\mathcal{A}}\mathbf{U}_{\mathcal{A}}\ \mathsf{goal}]}$:

$$\begin{array}{l} [\forall s : R_{\mathsf{true}}(s)] \wedge \\ [\forall s : P_{\mathsf{goal}}(s) \Rightarrow R_{\mathsf{goal}}(s)] \wedge \\ [\forall s : [\,[\exists a : \exists s' : T(s,a,s')] \wedge \\ \qquad [\forall a : \forall s' : \neg T(s,a,s') \vee R_{\mathsf{goal}}(s') \vee R_{\mathbf{A}[\mathsf{true}_{\mathcal{A}}\mathbf{U}_{\mathcal{A}}\ \mathsf{goal}]}(s') \Rightarrow R_{\mathbf{A}[\mathsf{true}_{\mathcal{A}}\mathbf{U}_{\mathcal{A}}\ \mathsf{goal}]}(s)]]] \end{array}$$

Here we exploit that $\mathcal{A}$ holds on all actions and that R_{true} holds on all states. □

Remark. It is instructive to note that we *cannot* simply define $\boldsymbol{R} \vdash \mathbf{AG}_{\Omega}\,\phi$ (omitting annotations) by the conjunction of $\boldsymbol{R} \vdash \phi$ and

$$[\forall s : [\forall a : \forall s' : \neg T(s,a,s') \vee (R_{\mathbf{AG}_{\Omega}\,\phi}(s') \wedge (\neg\Omega(a) \vee R_{\phi}(s')))] \Rightarrow R_{\mathbf{AG}_{\Omega}\,\phi}(s)]$$

since the above computes the least fixed point rather than the intended greatest fixed point – which is important for transition systems with loops. Rather we need to compute the staged relations $\boldsymbol{R} \vdash \neg\mathbf{E}[\mathsf{true}\ {}_{\mathcal{A}}\mathbf{U}_{\Omega}\ \neg\phi]$ and due to the use of negation this requires stratification.

Example 7. To illustrate this observation, consider the transition system of Example 1 and the formula $\mathbf{AG}_{\{C\}}\ \mathsf{goal}$ (omitting annotations). The above definition gives rise to the recursive equation

$$R = \{s \mid \forall s' : \forall a : s \rightarrow^{a} s' \Rightarrow (R(s') \wedge (\Omega(a) \Rightarrow s' \models \mathsf{goal}))\}$$

The least solution is $R = \emptyset$ whereas the correct solution is $R_{\mathbf{AG}_{\{C\}}\ \mathsf{goal}} = \{1,2,3\}$. □

Post-order labelling. It is easy to see that the clauses $\boldsymbol{R} \vdash \phi^{\ell}$ defined by Tables 4 and 5 are indeed *closed* for any ACTL formula ϕ^{ℓ}. To show that the clauses are *stratified* we shall fix the labelling scheme that so far has been unspecified.

To this end we assume that all sub-formula of an ACTL formula are annotated with their number in a *post-order traversal* of the formula. To be specific we shall say that a formula ϕ is (i,j)-*annotated* for positive integers i and j if the lowest annotation occurring within ϕ is i and the highest is j – since we consider a post-order traversal the annotation of the formula ϕ itself will necessarily be j. As an example the (5,7)-annotated version of the formula $\mathbf{A}[\mathsf{true}\ {}_{\mathcal{A}}\mathbf{U}_{\mathcal{A}}\ \mathsf{goal}]$ is $\mathbf{A}[\mathsf{true}^{5}\ {}_{\mathcal{A}}\mathbf{U}_{\mathcal{A}}\ \mathsf{goal}^{6}]^{7}$.

Proposition 1. *For every ACTL formula ϕ there exists an $(1,\ell)$-annotated formula ϕ^{ℓ} for some ℓ. Furthermore, the ALFP clause $\boldsymbol{R} \vdash \phi^{\ell}$ generated by Tables 4 and 5 from an $(1,\ell)$-annotated ACTL formula ϕ^{ℓ} is closed and stratified.*

To see the latter assign the relations P_{bp}, Ω and T the rank 0. We may then prove by induction on the annotated sub-formulae ϕ'^{j} of ϕ^{ℓ} that ϕ'^{j} is closed and (i, j)-annotated for some i. Furthermore, the ALFP clause $\boldsymbol{R} \vdash \phi'^{j}$ will only contain definitions of relations of rank in $\{i, \cdots, j\}$, it will only contain uses of relations of rank $\{0\} \cup \{i, \cdots, j\}$ and all negative uses will involve relations of rank $\{0\} \cup \{i, \cdots, j-1\}$ and thus it is a stratified formula. We shall say that an ALFP clause is (i, j)-*stratified* when this is the case.

Correctness and precision. We are now able to establish our main theorem expressing the *correctness* of the static analysis of the modal logic with respect to the interpretation of the modal logic; this is along the same lines as the correctness result exhibited for the analysis of the λ-calculus in Example 2. Additionally, the theorem expresses that the analysis is *precise*; this is not usually the case for static analyses but does indeed hold for the so-called *collecting semantics* analyses that often are the starting point for developing coarser analyses by Abstract Interpretation.

For the remainder of this section consider some transition system $(S, \mathcal{A}, \rightarrow)$ and a ranking as described above; we shall restrict ourselves to a finitely branching transition system. Furthermore assume that $\mathcal{U} = S$, that $\varrho_0(\Omega) = \Omega$ for all $\Omega \subseteq \mathcal{A}$, that $\varrho_0(T) = \{(s, a, s') \mid s \rightarrow^{a} s'\}$, and that $\varrho_0(P_{bp}) = \{s \mid bp \text{ holds on } s\}$. Then we have the following main result that is proved by structural induction on the formulae:

Theorem 3. *Consider an (i, j)-annotated formula ϕ^{j} in ACTL and the least model ϱ of $\boldsymbol{R} \vdash \phi$ such that $\varrho \sqsupseteq \varrho_0$. We then have:*

- *Correctness of ϱ: if $s \models \phi^{j}$ then $\varrho(R_{\phi^{j}})(s)$.*
- *Precision of ϱ: if $\varrho(R_{\phi^{j}})(s)$ then $s \models \phi^{j}$.*

It follows from Proposition 1 and Theorem 3 that an implementation of the static analysis of ACTL by means of the Succinct Solver constitutes a *model checker* for ACTL.

Complexity. We may estimate the worst case time complexity of model checking as performed using our static analysis of modal logic. For this assume that S has size $|S|$ and the transition relation $\rightarrow$ has size $|T|$. Next consider an ACTL formula ϕ of size $|\phi|$; it is immediate that the ALFP clause $\boldsymbol{R} \vdash \phi$ has size $\mathcal{O}(|\phi|)$ and nesting depth 3. According to Theorem 2 the worst case time complexity is $\mathcal{O}(|T| + \sum_{bp} |P_{bp}| + 2^{|\mathcal{A}|} + |S|^3 |\phi|)$ where bp ranges over all base predicates.

To compare with more traditional approaches to model checking we shall assume that both the number of base predicates and the number of action labels (as opposed to actions) is bounded by some constant. In this case we obtain the worst case time complexity $\mathcal{O}(|T| + |S|^2 |\phi|)$ because in this case the calculations of nesting depth can safely ignore quantifications over actions. Indeed using a more refined reasoning than that of Theorem 2 we obtain $\mathcal{O}((|T| + |S|)\,|\phi|)$ because it is clear that the "double quantifications" over states in Table 5 really correspond to traversing all possible transitions rather than all pairs of states.

Thus our alternative model checking algorithm has the same complexity as classical model checking algorithms [2]. And indeed, in our experience the Succinct Solver operates in such a manner that it will attain this worst case time complexity.

5 Conclusion

The interplay between static analysis and model checking has intrigued researchers for many years – leading to a growing feeling that the methods are connected and can be used to strengthen each other.

At the conceptual level a number of papers have argued that various forms of static analysis are instances of model checking [22,20,21]. Our contribution completes the circle by showing the close relationship between static analysis and model checking – to the extent that one may conjecture that they are reducible to each others.

At the practical level this gives increased faith in exploring the methods and techniques developed within each of the approaches to strengthen the other. This is at the core of Research Theme 1 in MT-LAB (see below) where it is hypothesised that "static analysis and model checking fundamentally solve the same problem but using a different repertoire of techniques that must be combined in order to produce more powerful analysis techniques." In future work we hope to identify concrete methods and techniques that can be transferred between the two approaches to their mutual benefit.

Acknowledgements. This work has been supported by MT-LAB, a VKR Centre of Excellence (`http://www.mt-lab.dk`). The authors would like to thank Kim Guldstrand Larsen for provoking us to write this paper.

References

1. Apt, K., Blair, H., Walker, A.: A theory of declarative programming. In: Foundations of Deductive Databases and Logic Programming, pp. 89–148. Morgan-Kaufman, San Francisco (1988)
2. Baier, C., Katoen, J.-P.: Principles of Model Checking. MIT Press, Cambridge (2008)
3. Balbin, I., Ramamohanarao, K.: A generalization of the differential approach to recursive query evaluation. Journal of Logic Programming 4(3), 259–262 (1987)
4. Bodei, C., Buchholtz, M., Degano, P., Nielson, F., Riis Nielson, H.: Static validation of security protocols. Journal of Computer Security 13, 347–390 (2005)
5. Chandra, A., Harel, D.: Computable queries for relational data bases. Journal of Computer and System Sciences 21(2), 156–178 (1980)
6. Le Charlier, B., Van Hentenryck, P.: A universal top-down fixpoint algorithm. Technical report, CS-92-25, Brown University (1992)
7. Fecht, C., Seidl, H.: Propagating differences: An efficient new fixpoint algorithm for distributive constraint systems. Nordic Journal of Computing 5(4), 304–329 (1998)

8. Hankin, C., Nielson, F., Riis Nielson, H.: Advice from Belnap policies. In: Computer Security Foundations Symposium. IEEE Computer Society, Los Alamitos (2009)
9. Hecht, M.S.: Flow Analysis of Computer Programs. North Holland, Amsterdam (1977)
10. Clarke Jr., E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
11. Lamprecht, A.-L., Margaria, T., Steffen, B.: Data-flow analysis as model checking within the jABC. In: Mycroft, A., Zeller, A. (eds.) CC 2006. LNCS, vol. 3923, pp. 101–104. Springer, Heidelberg (2006)
12. De Nicola, R., Vaandrager, F.W.: Action versus state based logics for transition systems. In: Guessarian, I. (ed.) LITP 1990. LNCS, vol. 469, pp. 407–419. Springer, Heidelberg (1990)
13. Nielson, F., Riis Nielson, H., Hankin, C.: Principles of Program Analysis, 2nd edn. Springer, Berlin (2005)
14. Nielson, F., Riis Nielson, H., Hansen, R.R.: Validating firewalls using flow logics. Theoretical Computer Science 283(2), 381–418 (2002)
15. Nielson, F., Riis Nielson, H., Seidl, H.: A succinct solver for ALFP. Nordic Journal of Computing 9, 335–372 (2002)
16. Nielson, F., Riis Nielson, H., Sun, H., Buchholtz, M., Hansen, R.R., Pilegaard, H., Seidl, H.: The Succinct Solver Suite. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 251–265. Springer, Heidelberg (2004)
17. Riis Nielson, H., Nielson, F.: Flow Logic: a multi-paradigmatic approach to static analysis. In: Mogensen, T.Æ., Schmidt, D.A., Sudborough, I.H. (eds.) The Essence of Computation. LNCS, vol. 2566, pp. 223–244. Springer, Heidelberg (2002)
18. Riis Nielson, H., Nielson, F., Pilegaard, H.: Spatial analysis of BioAmbients. In: Giacobazzi, R. (ed.) SAS 2004. LNCS, vol. 3148, pp. 69–83. Springer, Heidelberg (2004)
19. Paige, R.: Symbolic finite differencing - part i. In: Jones, N.D. (ed.) ESOP 1990. LNCS, vol. 432, pp. 36–56. Springer, Heidelberg (1990)
20. Schmidt, D.A.: Data flow analysis is model checking of abstract interpretations. In: POPL 1998, pp. 38–48 (1998)
21. Schmidt, D.A., Steffen, B.: Program analysis *as* model checking of abstract interpretations. In: Levi, G. (ed.) SAS 1998. LNCS, vol. 1503, pp. 351–380. Springer, Heidelberg (1998)
22. Steffen, B.: Data flow analysis as model checking. In: Ito, T., Meyer, A.R. (eds.) TACS 1991. LNCS, vol. 526, pp. 346–365. Springer, Heidelberg (1991)

Counting CTL

François Laroussinie[1,⋆], Antoine Meyer[2], and Eudes Petonnet[1]

[1] LIAFA, Université Paris Diderot – Paris 7 & CNRS UMR 7089, France
{Francois.Laroussinie,Eudes.Petonnet}@liafa.jussieu.fr
[2] LIGM, Université Paris Est – Marne-la-Valle & CNRS UMR 8049, France
Antoine.Meyer@univ-mlv.fr

Abstract. This paper presents a range of quantitative extensions for the temporal logic CTL. We enhance temporal modalities with the ability to constrain the number of states satisfying certain sub-formulas along paths. By selecting the combinations of Boolean and arithmetic operations allowed in constraints, one obtains several distinct logics generalizing CTL. We provide a thorough analysis of their expressiveness and of the complexity of their model-checking problem (ranging from P-complete to undecidable).

1 Introduction

Among the existing approaches to the formal verification of automated systems, model checking [7,17] aims at automatically establishing the validity of a certain formal specification (modeled as a formula in a suitable logic) over the system under study (modeled for instance as a finite automaton). This set of techniques is now well-established and successful, with many real-world applications.

To formalize the specification of temporal properties, for instance in the case of reactive systems, temporal logics (TL) were proposed thirty years ago [16] and widely studied since. They are today used in many model-checking tools. There exists a wide variety of temporal logics, differing for instance by the models over which formulas are interpreted or by the kind of available temporal modalities. Two well-known examples are LTL in the linear-time framework (where formulas are interpreted over infinite runs) and CTL for the branching-time case (where formulas are interpreted over states of Kripke structures). See [8] for a survey of classical temporal logics for systems specification.

Temporal logics have been extended in various ways in order to increase their expressive power. For example, while LTL and CTL only contain future operators, it is also possible to consider past-time modalities to express properties of the past of a run. One can also extend temporal logics with regular expressions (see for instance [19,10]). Other extensions were proposed to handle *quantitative* aspects of systems. For example, some logics can contain timing constraints to specify that some event P_1 has to occur less than 10 time units before another event P_2. Such temporal logics, like TCTL [2,9], have been especially studied in

⋆ Partly supported by ANR project DOTS and project QUASIMODO (FP7-ICT).

the framework of timed model checking. Another quantitative extension consists in *probabilistic* logics where one can specify probability bounds over the truth of some property (see for example [4]).

We propose several extensions of CTL with constraints over the number of states satisfying certain sub-formulas along runs. For example, considering a model for an ATM, we can express the property "whenever the PIN is locked, at least three erroneous attempts have been made" by: $\neg\mathsf{EF}_{[\sharp\mathtt{error}\leq 2]}\mathtt{lock}$ (one cannot reach a state where the PIN is locked but less than two errors have occurred). Similarly, $\neg\mathsf{EF}_{[\sharp\mathtt{error}\geq 3]}\mathtt{money}$ states that three mistakes forbid cash retrieval. We use subscripts on the temporal modality (as in TCTL) to associate a constraint with the run for which the modality holds. Note that these two properties could be clearly stated in CTL by nesting E_U_ modalities, but the resulting formulas would probably be too big to be easily handled by the user of a model checker. For each extension we consider, we study its expressiveness compared to CTL. In some cases, there is no formal gain of expressiveness because there exist natural translations to obtain equivalent CTL formulas, but these extensions are often *exponentially more succinct* than CTL: they allow writing concise specifications that would require formulas of exponentially larger size in CTL. In other cases, we show that adding some constraints increases the expressive power of CTL.

We consider the model checking problem for various sets $\mathcal{C}$ of constraints, and denote by $\mathsf{CCTL}_{\mathcal{C}}$ the corresponding extension of CTL. We show that polynomial-time algorithms exist when considering Until modalities with constraints of the form[1] $\sum_i \sharp\varphi_i \sim c$ with $\sim \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{N}$. Additionally allowing Boolean combinations of such constraints or integer coefficients in the sum (or both) makes model checking Δ_2^{P}-complete. We also consider the case of diagonal constraints $\sharp\varphi - \sharp\psi \sim c$ and their more general form $\sum_i \pm\sharp\varphi_i \sim c$ with $c \in \mathbb{Z}$ and show that model checking can be done in polynomial time. However, allowing Boolean combinations of such constraints leads to undecidability. Finally we define a version of CCTL with freeze variables and show that it induces a complexity blow-up: model checking becomes PSPACE-complete.

Several existing works provide related results. In [10], an extension of LTL with a kind of regular expressions containing quantitative constraints over the number of occurrences of sub-expressions is presented. This extension yields algorithms whose time complexity is exponential in the size of formulas and the *value* of integer constants. In [11], extensions of CTL are defined including parameters in constraints. One of these formalisms, namely GPCTL, allows one to express properties with constraints defined as positive Boolean combinations of sums of the form $\sum_i P_i \leq c$ where every P_i is an atomic proposition. Model-checking E_U_ formulas with such a constraint is shown to be NP-complete and a polynomial algorithm is given for a restricted logic (with parameters). In [20], a branching-time temporal logic with general counting constraints (using a variant of freeze variables) is defined to specify event-driven real-time systems. To obtain

[1] For complexity results, we always assume that integer constants are encoded in binary.

decidability, they restrict the analysis to systems verifying some bounded progress condition. In [6,5], extensions of LTL and CTL with Presburger constraints over the number of states satisfying some formulas are considered, for some class of infinite state processes. The complexity of these problems is much higher than the cases we are concerned with. Finally there also exist timed extensions of CTL interpreted over Kripke structures (see for instance [9]).

The paper is organized as follows. In Section 2, we introduce the definitions of the main formalisms we will use. In Section 3, we show that several of our proposed logics are not more expressive than the classical CTL, yet exponentially more succinct. In Section 4, we address the model-checking problem and provide exact complexity results for most of the logics we introduce. Finally we present in Section 5 a different logic with freeze variables, together with the complexity of its model-checking problem.

2 Definitions

2.1 Models

Let AP be a set of atomic propositions. In branching-time temporal logics, formulas are interpreted over states of Kripke structures.

Definition 1. *A* Kripke structure *(or KS) $\mathcal{S}$ is a tuple $\langle Q, R, \ell\rangle$ where Q is a finite set of states, $R \subseteq Q \times Q$ is a total accessibility relation*[2] *and $\ell : Q \to 2^{AP}$ is a labelling of states with atomic propositions.*

A run ρ of $\mathcal{S}$ is an infinite sequence of states $q_0q_1q_2\ldots$ such that $(q_i, q_{i+1}) \in R$ for every i. We use $\rho(i)$ to denote the state q_i and $\rho_{|i}$ to denote the prefix $q_0 \cdots q_i$ of ρ. $\mathsf{Runs}(q)$ denotes the set of runs starting from some state $q \in Q$. We write $\sigma \leq \rho$ when σ is a prefix of ρ.

We will also consider *Durational Kripke Structures* (DKS), where an integer duration is associated with every transition. Thus for a DKS $\mathcal{S} = \langle Q, R, \ell\rangle$, we have $R \subseteq Q \times \mathbb{N} \times Q$. The duration of a transition is also called a weight or a cost. We use DKS$^{0/1}$ to denote the DKSs where the durations belong to $\{0, 1\}$. The notion of duration is naturally extended to finite runs of DKSs.

2.2 Counting CTL

We define several extensions of CTL able to express constraints over the number of times certain sub-formulas are satisfied along a run.

Definition 2. *Given a set of atomic propositions AP and a set of constraints $\mathcal{C}$, we define:*

$$\mathsf{CCTL}_{\mathcal{C}} \ni \varphi, \psi \ ::= \ P \mid \varphi \wedge \psi \mid \neg\varphi \mid \mathsf{E}\varphi\mathsf{U}_{[C]}\psi \mid \mathsf{A}\varphi\mathsf{U}_{[C]}\psi$$

[2] By *total* relation, we mean a relation $R \subseteq Q \times Q$ such that $\forall p \in Q, \exists q \in Q, (p, q) \in R$.

where $P \in \mathit{AP}$ and $C \in \mathcal{C}$. The sets of constraints we consider are defined as follows, with $l, k \in \mathbb{N}$, $k' \in \mathbb{Z}$ and $\sim \in \{<, \leq, =, \geq, >\}$. First we have the sets of atomic *constraints:*

$$\begin{array}{rll}
\mathcal{C}_0 \ni C & ::= \sharp\varphi \sim k & \textit{with } \varphi \in \mathit{CCTL}_{\mathcal{C}_0} \\
\mathcal{C}_1 \ni C & ::= (\Sigma_{i=1}^{l} \sharp\varphi_i) \sim k & \textit{with } \varphi_i \in \mathit{CCTL}_{\mathcal{C}_1} \\
\alpha\mathcal{C}_1 \ni C & ::= (\Sigma_{i=1}^{l} \alpha_i \cdot \sharp\varphi_i) \sim k & \textit{with } \varphi_i \in \mathit{CCTL}_{\alpha\mathcal{C}_1}, \alpha_i \in \mathbb{N} \\
\mathcal{C}_2 \ni C & ::= (\sharp\varphi - \sharp\psi) \sim k' & \textit{with } \varphi, \psi \in \mathit{CCTL}_{\mathcal{C}_2} \\
\mathcal{C}_3 \ni C & ::= (\Sigma_{i=1}^{l} \pm \cdot \sharp\varphi_i) \sim k' & \textit{with } \varphi_i \in \mathit{CCTL}_{\mathcal{C}_3} \\
\alpha\mathcal{C}_3 \ni C & ::= (\Sigma_{i=1}^{l} \alpha_i \cdot \sharp\varphi_i) \sim k' & \textit{with } \varphi_i \in \mathit{CCTL}_{\alpha\mathcal{C}_3}, \alpha_i \in \mathbb{Z}
\end{array}$$

Let $\mathcal{L}_a$ be the set $\{\mathcal{C}_0, \mathcal{C}_1, \alpha\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \alpha\mathcal{C}_3\}$. We also consider the set of constraints $\mathcal{B}(\mathcal{C})$ for every $\mathcal{C} \in \mathcal{L}_a$, defined as the set of Boolean combinations of atomic constraints in $\mathcal{C}$ with sub-formulas in $\mathit{CCTL}_{\mathcal{B}(\mathcal{C})}$. We use $\mathcal{L}_b$ for $\{\mathcal{B}(\mathcal{C}) \mid \mathcal{C} \in \mathcal{L}_a\}$. Finally $\mathcal{L}_{cons} \overset{def}{=} \mathcal{L}_a \cup \mathcal{L}_b$.

We make use of the standard abbreviations $\vee, \Rightarrow, \Leftrightarrow, \bot, \top$, as well as the additional modalities $\mathsf{EF}_{[C]}\varphi \overset{\text{def}}{=} \mathsf{E}\top\mathsf{U}_{[C]}\varphi$, $\mathsf{AF}_{[C]}\varphi \overset{\text{def}}{=} \mathsf{A}\top\mathsf{U}_{[C]}\varphi$, and their duals $\mathsf{AG}_{[C]}\varphi \overset{\text{def}}{=} \neg\mathsf{EF}_{[C]}\neg\varphi$ and $\mathsf{EG}_{[C]}\varphi \overset{\text{def}}{=} \neg\mathsf{AF}_{[C]}\neg\varphi$. Any formula occurring in a constraint C associated with a modality in Φ is considered as a sub-formula of Φ. The size $|\Phi|$ of Φ takes the size of these constraints and their sub-formulas into account, assuming that integer constants are encoded in *binary* (unless explicitly stated otherwise). The DAG-size of Φ is the total number of distinct sub-formulas of Φ. As model-checking algorithms compute only once the truth value of a sub-formula, this is generally more relevant to the complexity of model-checking.

The semantics of $\mathsf{CCTL}_{\mathcal{C}}$ formulas (with $\mathcal{C} \in \mathcal{L}_{cons}$) is defined over Kripke structures as follows:

Definition 3. *The following clauses define the conditions for a state q of some KS $\mathcal{S} = \langle Q, R, \ell\rangle$ to satisfy a $\mathit{CCTL}_{\mathcal{C}}$ formula φ – written $q \models_{\mathcal{S}} \varphi$ – by induction over the structure of φ (we omit Boolean modalities):*

$$\begin{array}{lll}
q \models_{\mathcal{S}} \mathit{E}\varphi\mathit{U}_{[C]}\psi & \textit{iff } \exists\rho \in \mathsf{Runs}(q),\ \exists i \geq 0, & \rho(i) \models_{\mathcal{S}} \psi,\ \rho_{|i-1} \models_{\mathcal{S}} C, \\
& & \textit{and } \forall 0 \leq j < i,\ \rho(j) \models_{\mathcal{S}} \varphi \\
q \models_{\mathcal{S}} \mathit{A}\varphi\mathit{U}_{[C]}\psi & \textit{iff } \forall\rho \in \mathsf{Runs}(q),\ \exists i \geq 0, & \rho(i) \models_{\mathcal{S}} \psi,\ \rho_{|i-1} \models_{\mathcal{S}} C, \\
& & \textit{and } \forall 0 \leq j < i,\ \rho(j) \models_{\mathcal{S}} \varphi
\end{array}$$

Let $\mathcal{C} \in \mathcal{L}_{cons}$ be a set of constraints and C be a constraint in $\mathcal{C}$, the semantics of $\rho_{|i} \models_{\mathcal{S}} C$ is based on the interpretation of $\sharp\varphi$ over $\rho_{|i}$, denoted by $|\rho_{|i}|_{\varphi}$ and defined as: $|\rho_{|i}|_{\varphi} \overset{def}{=} |\{j \mid 0 \leq j \leq i \wedge \rho(j) \models_{\mathcal{S}} \varphi\}|$. Given these values, C is interpreted in a natural way.

In the following we omit the subscript $\mathcal{S}$ for $\models$ when no confusion is possible. We use $\equiv$ to denote the standard equivalence between formulas.

Remark 1. The classical X operator ("neXt") can be expressed in $\mathsf{CCTL}_{\mathcal{C}_0}$ as $\mathsf{EX}\varphi \equiv \mathsf{EF}_{[\sharp\top=1]}\varphi$, and that basic constraints in $\mathcal{C}_0$ can be expressed in $\mathcal{C}_2$ because

$\sharp\varphi \equiv \sharp\varphi - \sharp\bot$. Moreover, for $\mathsf{CCTL}_{\mathcal{B}(\mathcal{C})}$ with $\mathcal{C} \in \mathcal{L}_a$, since $\varphi \mathsf{U}_{[C]} \psi \equiv \mathsf{F}_{[C \wedge \sharp(\neg\varphi)=0]} \psi$ the modality F is sufficient to define U; thus such a logic $\mathsf{CCTL}_{\mathcal{B}(\mathcal{C})}$ can also be built from atomic propositions using Boolean operators and modalities $\mathsf{EF}_{[C]}\varphi$ and $\mathsf{AF}_{[C]}\varphi$ (or $\mathsf{EG}_{[C]}\varphi$). Note that all these translations are succinct (linear in the size of formulas) and do not have any impact on complexity results.

Remark 2. The related temporal logic TCTL [2], whose semantics are defined over *timed* models, allows one to label temporal modalities with duration constraints. For instance, one may write $\varphi \mathsf{U}_{<k} \psi$ to express the fact that φ is consistently true until, before k time units have elapsed, ψ eventually holds. When all transitions in a durational Kripke structure have duration 1 (i.e. the duration of any run is equal to its length), TCTL (or RTCTL in [9]) formulas can be directly coded into the logic $\mathsf{CCTL}_{\mathcal{C}_0}$ by only using the sub-formula $\top$ inside constraints. A similar coding is also possible when one uses a proposition *tick* to mark time elapsing as in [15].

2.3 Examples of CCTL Formulas

Consider a model for an ATM, whose atomic propositions include `money`, `reset` and `error`, with the obvious meaning. To specify that it is not possible to get money when three mistakes are made in the same session (*i.e.* with no intermediate reset), we can use the formula $\mathsf{AG}\big(\neg\mathsf{EF}_{[\sharp\mathtt{error} \geq 3 \wedge \sharp\mathtt{reset}=0]}\mathtt{money}\big)$ that belongs to $\mathsf{CCTL}_{\mathcal{B}(\mathcal{C}_0)}$. Note that this could also be expressed by the $\mathsf{CCTL}_{\mathcal{C}_0}$ formula $\mathsf{AG}\big(\neg\mathsf{E}(\neg\mathtt{reset})\mathsf{U}_{[\sharp\mathtt{error}\geq 3]}\mathtt{money}\big)$.

Consider a mutual exclusion algorithm with n processes trying to reach their critical section. We can specify that it verifies the bounded waiting property with bound 10 (*i.e.* when a process P tries to reach its CS, then at most 10 other processes can reach theirs before P does) by the following $\mathsf{CCTL}_{\mathcal{B}(\mathcal{C}_1)}$ formula: $\mathsf{AG}\bigwedge_i \big(\mathtt{request}_i \Rightarrow \neg\mathsf{EF}_{[\sum_{j\neq i}\sharp\mathtt{CS}_j > 10 \wedge \sharp\mathtt{CS}_i = 0]}\top\big)$.

$\mathsf{AG}_{[\sharp\mathtt{send} - \sharp\mathtt{receive} < 0]}\bot$ belongs to $\mathsf{CCTL}_{\mathcal{C}_2}$ and states that along any finite run, the number of `receive` events cannot exceed the number of `send` events.

Quantitative constraints can also be useful for fairness properties. For example $\mathsf{AGAF}_{[\bigwedge_i 5 \leq \sharp\varphi_i \leq 10]}\top$ expresses that the φ_i's occur infinitely often along every run (as stated with the CTL formula $\bigwedge_i(\mathsf{AGAF}\varphi_i)$) but it also ensures some constraint on the number of states satisfying the φ_i's along every execution: for example, it is not possible to have a sub-run where φ_1 holds for 11 states and φ_2 holds only for 4 states.

Note that with $\mathsf{CCTL}_{\alpha\mathcal{C}_3}$ one can express properties over the ratio of two kinds of states along a run. For example, $\mathsf{EF}_{[100\cdot\sharp\mathtt{error} - \sharp\top < 0]}P$ is true when there is a path leading to P such that the rate of `error` states is less than 1 percent. Thus constraints of the form "$\frac{\sharp P}{\sharp P'} \sim k$" can easily be expressed with this logic.

Finally note that we can use any temporal formula inside a constraint (and not only atomic propositions). For example, $\mathsf{AG}(\mathsf{EF}_{[\sharp(\mathsf{EX}\mathtt{alarm})\leq 5]}\mathtt{init})$ states that it is always possible to reach `init` with a path along which at most 5 states have a successor satisfying `alarm`.

These examples illustrate the ability of CCTL formulas to state properties over the portion of a run leading to some state. A similar kind of properties could also be expressed with past-time modalities (like S or F^{-1}), but unlike these modalities our constraints cannot easily describe the ordering of events in the past: they "only" allow to count the number of occurrences of formulas. We will see in the next sections that our extensions do not always induce a complexity blow-up, while model-checking $\mathsf{CTL}+\mathsf{F}^{-1}$ is known to be PSPACE-complete [14].

3 CCTL Expressiveness and Succinctness

When comparing two logics, the first question which comes to mind is the range of properties they can be used to define, in other words their *expressiveness*. When they turn out to be equally expressive, a natural way to distinguish them is then to ask *how concisely* each logic can express a given property. This is referred to as *succinctness*, and is also relevant when studying the complexity of model-checking for instance, since it may considerably influence the size of a formula required to express a given property, and hence the time required to model-check it. In this section we study the expressiveness of the different logics defined in the previous section, and provide partial results and comments about their respective succinctness with respect to CTL.

3.1 Expressiveness

We first show that only allowing boolean combinations and positive sums in constraints does not allow CCTL to express more properties than CTL.

Proposition 4. *Any $\mathsf{CCTL}_{\mathcal{B}(\alpha\mathcal{C}_1)}$ formula can be translated into CTL.*

Proof (sketch). Let Φ be a $\mathsf{CCTL}_{\mathcal{B}(\alpha\mathcal{C}_1)}$ formula of the form $\mathsf{EF}_{[C]}\varphi$ whose constraint C contains n counting arguments $\sharp\varphi_1$ to $\sharp\varphi_n$, each preceded by a multiplicative constant α_i, and m atomic constraints. We inductively translate Φ to CTL by building a family of formulas whose intended meaning (up to technical details) is as follows:

– If constraint C holds with $\sharp\varphi_i = 0$ for all i, then φ may be true immediately.
– Otherwise, successively check for every i which of the φ_i hold in the current state, updating constraint C along the way by decreasing by α_i the constant to which φ_i is compared in Φ.
– Once all φ_i have been scanned, proceed to the next state and reevaluate C for the new values of the constants.

Each of these steps corresponds to a sub-formula of the form $\Phi_{C',i,b}$ in the CTL translation of Φ, where C' is the current constraint to be checked, i is the index of the formula φ_i being scanned, and b is a boolean flag used to enforce termination, $\Phi_{C,0,\perp}$ being the translation of Φ itself. By counting the number of distinct $\Phi_{C',i,b}$, one can show that the DAG-size of $\Phi_{C,0,\perp}$ is $O(n.k^m)$, where k is the maximal constant appearing in Φ. A similar argument holds for formulas of the form $\mathsf{EG}_{[C]}\varphi$ with the same resulting DAG-size. □

Note that the upper bound for the above translation is parametric, and can be interpreted for all variants of CCTL below $\mathsf{CCTL}_{\mathcal{B}(\alpha\mathcal{C}_1)}$. An example of this translation on a $\mathsf{CCTL}_{\alpha\mathcal{C}_1}$ formula is given in the next section. In contrast to this result, introducing subtractions in constraints yields a strict increase in expressiveness.

Proposition 5. *The $\mathsf{CCTL}_{\mathcal{C}_2}$ formula $\varphi = \mathsf{AG}_{[\sharp A - \sharp B < 0]}\bot$ cannot be translated into CTL.*

Proof (sketch). Formula φ (already seen in Sec. 2 with different atomic propositions) states that the number of B-labeled states cannot exceed the number of A-labeled states along any path. As shown by [3] and also presented in [18], the set of models of any CTL formula can be recognized by a finite alternating tree automaton. From such an automaton, one can easily build a finite alternating automaton over words, whose accepted language is the set of all prefixes of branches in models of the formula, seen as words over 2^{AP}.

Suppose there exists a CTL formula φ' equivalent to φ, and let $\mathcal{A}$ be the alternating tree automaton accepting its set of models. As stated above, from $\mathcal{A}$ one can derive a finite alternating automaton recognizing the set of all words over $2^{\{A,B\}}$ labeling a finite prefix of a branch in a model of φ, namely words whose prefixes contain at most as many B's as A's. Since this language is clearly not regular, this leads to a contradiction. $\square$

3.2 Succinctness

Our extensions of CTL come with three main sources of possible concision, which appear to be orthogonal : the encoding of constants in binary, the possibility to use boolean combinations in the constraints, and the use of sums. However, only the first two prove out to yield an exponential improvement in succinctness :

Proposition 6. *For every formula $\Phi \in \mathsf{CCTL}_{\mathcal{C}_1}$ with unary encoding of integers, there exists an equivalent CTL formula of DAG-size polynomial in $|\Phi|$.*

This proposition is a direct consequence of Prop. 4 where m, the number of atomic constraints, is set to 1. For instance, to translate $\Phi = \mathsf{EF}_{[\Sigma_{i=1}^{n}\sharp \mathsf{p}_i = K]}\varphi$, we define $\forall 0 \leq k \leq K$ the family of CTL formulas:

$$\begin{array}{lrl}
\forall 1 \leq i \leq n, 0 \leq j < n, & \Phi_{i,j,k} & = (\mathsf{p}_i \wedge \Phi_{i+1,j+1,k}) \vee (\neg \mathsf{p}_i \wedge \Phi_{i+1,j,k}) \\
\forall 1 \leq j \leq k, & \Phi_{n+1,j,k} & = \mathsf{EXE}(\neg \bigvee_{i=1}^{n} \mathsf{p}_i)\mathsf{U}((\bigvee_{i=1}^{n} \mathsf{p}_i) \wedge \Phi_{1,0,k-j}) \\
 & \Phi_{n+1,k,k} & = \mathsf{EXE}(\neg \bigvee_{i=1}^{n} \mathsf{p}_i)\mathsf{U}\varphi \\
\forall j > k, & \Phi_{n+1,j,k} & = \bot
\end{array}$$

By construction, we have $\Phi \equiv \Phi_{1,0,K}$. The size of this family is $O(n.k)$, thus the DAG-size of $\Phi_{1,0,K}$ is also polynomial in $|\Phi|$, even if its literal size is exponential. This example relies on the fact that constants are encoded in unary, to measure the impact of the addition operation in constraints. We now look at the succinctness gap due to the binary encoding of constants.

Proposition 7. *$\mathsf{CCTL}_{\mathcal{C}_0}$ can be exponentially more succinct than CTL.*

Proof. In [15], it is shown that the logic TCTL, when interpreted over Kripke structures with a special atomic proposition *tick* used to mark the elapsing of time, can be exponentially more succinct than CTL. More precisely, the TCTL formulas $\mathsf{EF}_{<n}A$ and $\mathsf{EF}_{>n}A$, which are of size $O(\log(n))$ since n is encoded in binary, do not admit any equivalent CTL formula of temporal height (and hence also size) less than n. These formulas express the existence of a path where A eventually holds and less (resp. more) than n clock ticks are seen until then. They are clearly equivalent to the $O(\log(n))$-size $\mathsf{CCTL}_{\mathcal{C}_0}$ formulas $\mathsf{EF}_{[\sharp tick<n]}A$ and $\mathsf{EF}_{[\sharp tick>n]}A$ respectively. □

This exhibits a first aspect in which CCTL logics can be exponentially more succinct than CTL. However, as expressed in the next proposition, another orthogonal feature of the logic may yield a similar blow-up.

Proposition 8. *$\mathsf{CCTL}_{\mathcal{B}(\mathcal{C}_0)}$ with unary encoding of integers can be exponentially more succinct than CTL.*

Proof. It was shown by [18,1] that any CTL formula φ equivalent to the CTL^+ formula $\psi = \mathsf{E}(\mathsf{F}P_0 \wedge \ldots \wedge \mathsf{F}P_n)$ must be of length exponential in n. It turns out ψ is equivalent to the $\mathsf{CCTL}_{\mathcal{B}(\mathcal{C}_0)}$ formula $\psi' = \mathsf{EF}_{[\bigwedge_i \sharp P_i \geq 1]}\top$, which entails the result. Note that ψ' only contains the constant 1, which means that this gap cannot be imputed to the binary encoding. □

The intuitive reason for this blow-up is that a CTL formula expressing the property that atomic propositions P_1 to P_n are each seen at least once along a path would have to keep track of all possible interleavings of occurrences of P_i's.

To summarize, we showed that two different aspects of the extensions of CTL presented in this paper, while not increasing the overall expressiveness of the logic, may yield exponential improvements in succinctness We still have to study similar succinctness properties of the remaining CCTL fragments with respect to CTL and to each other.

4 Model Checking

4.1 Model Checking $\mathsf{CCTL}_{\mathcal{C}_0}$ and $\mathsf{CCTL}_{\mathcal{C}_1}$

It turns out that model-checking $\mathsf{CCTL}_{\mathcal{C}_1}$ is polynomially equivalent to model-checking $\mathsf{CCTL}_{\mathcal{C}_0}$ (or CTL), as both problems are P-complete.

Theorem 9. *The model-checking problem for $\mathsf{CCTL}_{\mathcal{C}_1}$ is P-complete.*

Proof. P-hardness comes from the P-hardness of CTL model-checking. For membership in P, we provide polynomial-time procedures to deal with the subformulas $\mathsf{E}\psi\mathsf{U}_{[C]}\psi'$ and $\mathsf{A}\psi\mathsf{U}_{[C]}\psi'$ with $C \overset{\text{def}}{=} \Sigma_{i=1}^{l}\sharp\varphi_i \sim k$. Consider a Kripke structure $\mathcal{S} = (Q, R, \ell)$, and inductively assume that the truth values of ψ, ψ'

and φ_i over each state of $\mathcal{S}$ are known: these sub-formulas will be seen as atomic propositions in the following.

To each state q occurring along a path, we associate a cost $|q|_C = |\{i \mid q \models \varphi_i\}|$, and note that the *value* of $|q|_C$ is in $O(|C|)$. This cost is additively extended to paths in the usual way. Deciding the truth value of the path formula $\psi \mathsf{U}_{[C]} \psi'$ over any path ρ verifying $\psi \mathsf{U} \psi'$ then amounts to checking whether there exists a finite prefix $\rho' q$ of ρ such that $|\rho'|_C \sim k$, $q \models \psi'$ and $\forall i \leq |\rho'|, \rho'(i) \models \varphi$.

We reduce this problem to the model-checking of a TCTL formula over a $\mathrm{DKS}^{0/1}$ (DKS with 0/1-durations) for which there exists a polynomial-time algorithm [15]. We build from $\mathcal{S}$ a $\mathrm{DKS}^{0/1}$ $\mathcal{S}' = (Q', R', \ell')$ as follows: for each state $q \in Q$ with $|q|_C = n$, Q' contains $n+1$ additional states $q_0, \ldots, q_n$. R' is then defined as $\{q \xrightarrow{0} q_0 \mid q \in Q\} \cup \{q_i \xrightarrow{1} q_{i+1} \mid q \in Q, i < |q|_C\} \cup \{q_n \xrightarrow{0} q' \mid (q,q') \in R, n = |q|_C\}$. Finally, we set $\ell'(q_i) = \varnothing$ for all $q_i \in Q' \setminus Q$ and $\ell'(q) = \ell(q) \cup \{\mathsf{ok}\}$ for all $q \in Q' \cap Q$, where ok is a new atomic predicate.

To each path $\rho = q\sigma q'$ in $\mathcal{S}$, we associate the path $\tilde{\rho} = qq_0 \ldots q_n \tilde{\sigma} q'$ in $\mathcal{S}'$. It can now be shown by induction on run lengths that ρ satisfies $\psi \mathsf{U}_{[C]} \psi'$ if and only if $\tilde{\rho}$ satisfies the TCTL path formula $(\mathsf{ok} \Rightarrow \psi) \mathsf{U}_{[\sim k]} (\mathsf{ok} \wedge \psi')$. □

Since $\mathsf{CCTL}_{\mathcal{C}_0}$ includes CTL and is included in $\mathsf{CCTL}_{\mathcal{C}_1}$, we get:

Corollary 1. *The model-checking problem for $\mathsf{CCTL}_{\mathcal{C}_0}$ is* P*-complete.*

4.2 Model-Checking $\mathsf{CCTL}_{\mathcal{B}(\mathcal{C}_0)}$ and $\mathsf{CCTL}_{\alpha\mathcal{C}_1}$

We now establish the Δ_2^{P}-completeness of model-checking for the fragments $\mathsf{CCTL}_{\mathcal{B}(\mathcal{C}_0)}$, $\mathsf{CCTL}_{\alpha\mathcal{C}_1}$ and $\mathsf{CCTL}_{\mathcal{B}(\alpha\mathcal{C}_1)}$. Let us first recall the definition of the complexity class Δ_2^{P}, one of the classes of the polynomial hierarchy.

Definition 10. $\Delta_2^{\mathsf{P}} = \mathsf{P}^{\mathsf{NP}}$ *is the class of problems solvable in polynomial time with access to an oracle for some* NP*-complete problem.*

We now prove Δ_2^{P}-hardness of the model-checking problem for $\mathsf{CCTL}_{\mathcal{B}(\mathcal{C}_0)}$.

Theorem 11. *The model-checking problem for $\mathsf{CCTL}_{\mathcal{B}(\mathcal{C}_0)}$ is Δ_2^{P}-hard.*

Proof. We proceed by reduction from the Δ_2^{P}-complete problem SNSAT [12].

Given p families of variables $X_1, \ldots X_p$ with $X_i = \{x_i^1, ..., x_i^m\}$ and a set $Z = \{z_1, \ldots, z_p\}$ of p variables, an instance $\mathcal{I}$ of SNSAT is defined as a collection of p propositional formulas $\varphi_1, \ldots, \varphi_p$ under 3-conjunctive normal form (3-CNF), where each φ_i involves variables in $X_i \cup \{z_1, ..., z_{i-1}\}$, and the values of each z_i is defined as $z_i \overset{\text{def}}{=} \exists X_i.\ \varphi_i(z_1, ..., z_{i-1}, X_i)$. The instance $\mathcal{I}$ is positive iff the value of z_p is $\top$. We denote by $v_{\mathcal{I}}$ the unique valuation of variables in Z induced by $\mathcal{I}$.

From $\mathcal{I}$, we define the KS described in Figure 1. Every state is labeled by its name, and in addition every state $\bar{z}_i$ is labeled by some new atomic proposition $P_{\bar{z}}$. We use X to denote the set $X_1 \cup \cdots \cup X_p$ and $\mathcal{V}$ for $X \cup Z$. A path ρ from q_p to q_F describes the valuation v_ρ such that $v_\rho(y) = \top$ if ρ visits state y and $\perp$ if it visits $\bar{y}$ for every variable y in $\mathcal{V}$. We use a $\mathsf{CCTL}_{\mathcal{B}(\mathcal{C}_0)}$ formula to ensure

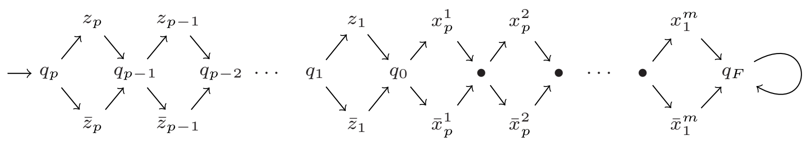


Fig. 1. Kripke structure associated to an SNSAT problem

that v_ρ coincides with $v_\mathcal{I}$ over Z, that is: $v_\rho(z_i) = \top$ iff $v_\mathcal{I}(z_i) = \top$ for any $i \in \{1, \ldots, p\}$.

Let $\widetilde{\varphi_i}$ be the formula φ_i where every occurrence of the *literal* x is replaced by $\sharp x = 1$. We define the $\mathsf{CCTL}_{\mathcal{B}(\mathcal{C}_0)}$ formula Ψ_0 as $\top$ and for every $1 \leq k \leq p$, Ψ_k as $\mathsf{EX}\big(\mathsf{E}(P_{\bar{z}} \Rightarrow \neg\Psi_{k-1})\mathsf{U}_{[C_k]}q_F\big)$, with $C_k \stackrel{\text{def}}{=} \bigwedge_{l \leq k} \big((\sharp z_l = 1) \Rightarrow \widetilde{\varphi_l}\big) \wedge \bigwedge_{j=1}^{k} \big((\sharp q = j) \Rightarrow \widetilde{\varphi_j}\big)$. The first part of the constraint C_k aims at ensuring that $v_\rho(z_l) = \top$ is witnessed by a valuation for $\{z_1, \ldots, z_{l-1}\} \cup X^l$ satisfying φ_l. The second part ensures the formula φ_j is satisfied by v_ρ when Ψ_k is interpreted from z_j or $\bar{z}_j$ (*i.e.* when the number of qs along the path leading to q_F is j). The formula Ψ_j holds for a state q_i with $i \leq j$ when $v_\mathcal{I}(z_i)$ is $\top$. The embedding of Ψ_{j-1} inside Ψ_j is used to ensure that going through a $\bar{z}_m$ with $i \geq m$ is always necessary w.r.t. $\mathcal{I}$ (*i.e.* there is no way to satisfy the corresponding φ_m):

Lemma 12. *For any $i = 1, \ldots, p$ and $i \leq j \leq p$, we have: $z_i \models \Psi_j \Leftrightarrow v_\mathcal{I}(z_i) = \top$ and $\bar{z}_i \not\models \Psi_j \Leftrightarrow v_\mathcal{I}(z_i) = \bot$*

Now it is sufficient to check whether q_0 satisfies Ψ_p or not, and then deduce the truth value of $v_\mathcal{I}(z_p)$. □

Theorem 13. *The model-checking problem for $\mathsf{CCTL}_{\alpha\mathcal{C}_1}$ is Δ_2^P-hard.*

Proof. We provide a reduction from the model checking problem for TCTL specifications over Durational Kripke structures. TCTL formulas allow to deal with the cost (or duration) of paths (*i.e.* the sum of the weight of every transition occurring along the path). This problem is Δ_2^P-complete [13]. Let $\mathcal{S} = (Q, R_\mathcal{S}, \ell)$ be a DKS. Let W be the set of weights occurring in $\mathcal{S}$. We define the Kripke structure $\mathcal{S}' = (Q', R_{\mathcal{S}'}, \ell')$ as follows:

- $Q' \stackrel{\text{def}}{=} Q \cup \{(q, d, q') \mid \exists (q, d, q') \in R_\mathcal{S}\}$,
- for any $(q, d, q') \in R_\mathcal{S}$, we add $(q, (q, d, q'))$ and $((q, d, q'), q')$ in $R_{\mathcal{S}'}$; and
- $\ell' : Q' \to 2^{\mathsf{AP}'}$ with $\mathsf{AP}' \stackrel{\text{def}}{=} \mathsf{AP} \cup \{\mathsf{ok}\} \cup \{P_d \mid d \in W\}$ – we assume $\mathsf{ok}, P_d \notin \mathsf{AP}$. And we have: $\ell'(q) \stackrel{\text{def}}{=} \ell(q) \cup \{\mathsf{ok}\}$ for any $q \in Q$, and $\ell'(q, d, q') = \{P_d\}$.

Now we can easily see that $q \models_\mathcal{S} \Phi$ with $\Phi \in \mathsf{TCTL}$ is equivalent to $q \models_{\mathcal{S}'} \widetilde{\Phi}$ where $\widetilde{P} \stackrel{\text{def}}{=} P$, $\widetilde{\neg\psi} \stackrel{\text{def}}{=} \neg\widetilde{\psi}$, $\widetilde{\varphi \wedge \psi} \stackrel{\text{def}}{=} \widetilde{\varphi} \wedge \widetilde{\psi}$, $\widetilde{\mathsf{E}\varphi\mathsf{U}_{\sim c}\psi} \stackrel{\text{def}}{=} \mathsf{E}(\mathsf{ok} \Rightarrow \widetilde{\varphi})\mathsf{U}_{[C(\sim c)]}(\mathsf{ok} \wedge \widetilde{\psi})$ and $\widetilde{\mathsf{A}\varphi\mathsf{U}_{\sim c}\psi} \stackrel{\text{def}}{=} \mathsf{A}(\mathsf{ok} \Rightarrow \widetilde{\varphi})\mathsf{U}_{[C(\sim c)]}(\mathsf{ok} \wedge \widetilde{\psi})$ with $C(\sim c) \stackrel{\text{def}}{=} \sum_{d \in W} d \cdot \sharp P_d \sim c$. □

Theorem 14. *The model-checking problem for $\mathsf{CCTL}_{\mathcal{B}(\alpha\mathcal{C}_1)}$ is in Δ_2^P.*

Proof (sketch). Let $\mathcal{S} = \langle Q, R, \ell \rangle$ be a KS. For this proof, we only need to provide NP procedures to deal with sub-formulas of the form $\mathsf{EF}_{[C]}\varphi$ and $\mathsf{EG}_{[C]}\varphi$. First let $\{C_1, \ldots, C_l\}$ be the set of $\alpha\mathcal{C}_1$ constraints occurring in C. Each C_i is of the form $\sum_{j \leq l_i} \alpha^i_j \cdot \sharp\varphi^i_j \sim_i d_i$. And let $d_{\max}$ be the maximal integer constant occurring in C. Now we can present the algorithms:

– $\Phi \overset{\text{def}}{=} \mathsf{EF}_{[C]}\psi$: If $q \models \Phi$, then there exists a run ρ starting from q and leading to some q' such that (1) $q' \models \psi$ and (2) ρ without q' satisfies the constraint C. First note that we can assume that the length of ρ is bounded with respect to the model and formula: a sequence of $|Q|$ states contributes for at least 1 to some linear expressions in C and then the length of ρ is in $O(|Q|.2^{|C|})$ due to the binary encoding of the constants. An easy NP algorithm consists in guessing the Parikh image of the transitions in ρ, which can be represented in polynomial size. Moreover it is possible to check (in polynomial time) that q' satisfies ψ, ρ without q' satisfies C, and F_ρ corresponds to a path in $\mathcal{S}$.
– $\Phi \overset{\text{def}}{=} \mathsf{EG}_{[C]}\psi$: For this case we have to find an infinite path ρ satisfying the property "if the current prefix satisfies the constraint C, then the next state has to satisfy ψ". Every constraint $C_i \in \alpha\mathcal{C}_1$ in C may change its truth value at most twice along ρ. Therefore ρ can be decomposed in a bounded number of parts over which the truth value of every constraint is constant. As previously, the length of every part is bounded and its Parikh image can be encoded in polynomial size. Moreover it is possible to ensure that the juxtaposition of all ρ_m is correct. □

A direct corollary of Theorems 11, 13 and 14 is:

Corollary 2. *The model-checking problem for* $\mathsf{CCTL}_{\mathcal{C}}$ *is* Δ^{P}_2*-complete for each* $\mathcal{C} \in \{\alpha\mathcal{C}_1, \mathcal{B}(\mathcal{C}_0), \mathcal{B}(\mathcal{C}_1), \mathcal{B}(\alpha\mathcal{C}_1)\}$.

4.3 Diagonal Constraints

We now show that even if diagonal constraints lead to strictly more expressive logics than CTL, model checking $\mathsf{CCTL}_{\mathcal{C}_2}$ and $\mathsf{CCTL}_{\mathcal{C}_3}$ is not more difficult than model checking CTL itself.

Theorem 15. *The model-checking problem for* $\mathsf{CCTL}_{\mathcal{C}_2}$ *is* P*-complete.*

Proof (sketch). P-hardness comes from that of $\mathsf{CCTL}_{\mathcal{C}_0}$ model-checking. Using the fact that $\mathsf{A}\varphi'\mathsf{U}_{[C]}\psi' \equiv \mathsf{AF}_{[C \wedge \sharp\neg\varphi'=0]}\psi' \equiv \neg\mathsf{EG}_{[C \wedge \sharp\neg\varphi'=0]}\neg\psi'$, to show membership in P, we only need to provide polynomial-time procedures to verify sub-formulas of the form $\mathsf{E}\varphi'\mathsf{U}_{[C]}\psi'$ and $\mathsf{EG}_{[C \wedge \sharp\varphi'=0]}\psi'$ with $C \overset{\text{def}}{=} \sharp\varphi - \sharp\psi \sim k$. Consider a Kripke structure $\mathcal{S} \overset{\text{def}}{=} (Q, R, \ell)$. As previously, we associate a "cost" to each state $q \in Q$. In this case however, $|q|_C$ can only be -1, 0 or 1 depending on the truth values of φ and ψ. Inductively assume that the truth values of φ, ψ, φ' and ψ' over each state of $\mathcal{S}$ are known: these sub-formulas will be seen as atomic propositions in the following. We distinguish the two main cases below:

– $\Phi \stackrel{\text{def}}{=} \mathsf{E}\varphi'\mathsf{U}_{[C]}\psi'$: We consider the weighted and directed graph $G_{\mathcal{S}} = (V, E)$ representing the transition relation of $\mathcal{S}$ restricted to states verifying the formula $\mathsf{E}\varphi'\mathsf{U}\psi'$, where edges are weighted by the cost of their source state and where only edges whose source verifies φ' are considered.

  If $C \stackrel{\text{def}}{=} \sharp\varphi - \sharp\psi \leq k$, then the formula holds true on state q if and only if there exists a state q' such that $q' \models_{\mathcal{S}} \psi'$ and either an elementary (i.e. acyclic) path ρ in $G_{\mathcal{S}}$ of weight less than k from q to q', or a path from q to some state q'' appearing on a negative-weight cycle, and from q'' to q'. Using a classical reachability algorithm over $G_{\mathcal{S}}$, one can determine the existence of such paths in polynomial time.

  If $C \stackrel{\text{def}}{=} \sharp\varphi - \sharp\psi = k$ with $k \geq 0$, we will compute the relation R_k over V^2 denoting the existence of a run of weight k between states q and q' and simply test whether $(q, q') \in R_k$ for some q' verifying ψ'. Using dichotomy and simple fixed-point computations, we are able to compute R_k in time polynomial in $|\Phi|$, *i.e.* logarithmic in k The treatment of negative weights is omitted.

– $\Phi \stackrel{\text{def}}{=} \mathsf{EG}_{[C \wedge \sharp\varphi'=0]}\psi'$: We use the weighted and directed graph $G_{\mathcal{S}}$ representing the transition relation of $\mathcal{S}$ where edges are weighted by the cost of their source state, to build a new Kripke structure $\mathcal{S}'$ and a classical CTL formula Ψ such that $\mathcal{S}$ satisfies Φ if and only if $\mathcal{S}'$ satisfies Ψ. □

By combining the techniques used in the previous construction with those used in the proof to Theorem 9, we obtain a similar result for the logic $\mathsf{CCTL}_{\mathcal{C}_3}$.

Corollary 3. *The model-checking problem for* $\mathsf{CCTL}_{\mathcal{C}_3}$ *is* P*-complete.*

Proof (sketch). In this setting, each state contributes to the cost of a path by a certain positive or negative number whose absolute value is bounded by some integer d. Similarly to the technique used in the proof of Theorem 9, the idea is to build a durational Kripke structure, this time with weights in $\{-1, 0, 1\}$, by adding intermediate states. Once this DKS is built, relations R_i, R_i^+ and R_i^- may be computed as previously, and the satisfaction of the formula under consideration tested. □

Theorem 16. *The model-checking problem for* $\mathsf{CCTL}_{\mathcal{B}(\mathcal{C}_2)}$ *is undecidable.*

Proof (sketch). This is done by reduction from the halting problem of a two-counter machine $\mathcal{M}$ with counters C and D. We define a Kripke structure $\mathcal{S}_{\mathcal{M}}$ with one state to simulate each of $\mathcal{M}$'s instructions, plus some auxiliary states. We use labels φ_X^+ and φ_X^- with $X \in \{C, D\}$ to witness increments and decrements, and additional labels ok_X, ko_X to simulate the positive test "$X = 0$": whenever the counter's value is assumed to be zero, and before simulating the next instruction, the run goes through an auxiliary state labeled ko_X whose unique successor is labeled ok_X. Hence along any run in $\mathcal{S}_{\mathcal{M}}$, a prefix satisfies $\sharp\mathsf{ko}_X > \sharp\mathsf{ok}_X$ right after counter X was deemed equal to zero, and only then. By counting occurrences of these predicates, one can write a $\mathsf{CCTL}_{\mathcal{B}(\mathcal{C}_2)}$ formula expressing the fact that M is correctly simulated by $\mathcal{S}_{\mathcal{M}}$ and never halts. □

5 Freeze Variables

Instead of using counting constraints associated with temporal modalities, we now consider *freeze variables* and explicit constraints inside formulas.

Definition 17. *Given a set of atomic propositions* AP *and a set of variables* V, *we define:* $\mathsf{CCTL}^{\mathsf{fv}} \ni \varphi, \psi ::= P \mid \varphi \wedge \psi \mid \neg\varphi \mid z[\psi].\varphi \mid C \mid \mathsf{E}\varphi\mathsf{U}\psi \mid \mathsf{A}\varphi\mathsf{U}\psi$ *where* $P \in \mathsf{AP}$ *and* C *is a constraint* $\sum_{i=1}^{l} \alpha_i \cdot z_i \sim c$ *with* $z_i \in V$, $\alpha_i \in \mathbb{N}$, $c \in \mathbb{N}$ *and* $\sim \in \{<, \leq, =, \geq, >\}$.

Intuitively $z[\psi].\varphi$ means that (1) the variable z is reset to zero and associated with the sub-formula ψ (*i.e.* z will evolve like $\sharp\psi$ in the future) and (2) given this semantics for z, φ holds for the current state. We say that an occurrence of some variable z is *free* in φ when this occurrence does not appear in the scope of a reset operator "."; a formula without any free variable is *closed*. For example, the $\mathsf{CCTL}_{\mathcal{B}(\mathcal{C}_0)}$ formula $\mathsf{EF}_{[\sharp P \leq 5 \wedge \sharp P' > 2]} P''$ can be expressed in $\mathsf{CCTL}^{\mathsf{v}}$ as the formula $z[P].z'[P'].\mathsf{EF}(z \leq 5 \wedge z' > 2 \wedge P'')$.

A $\mathsf{CCTL}^{\mathsf{v}}$ formula φ is interpreted in a state of a KS extended with a valuation for any free variable in φ and an environment associating a sub-formula to any free variable. We use dom to denote the domain of such functions and $\perp$ to represent undefined values. Given a function f and $x \in \mathsf{dom}(f)$, we use $f[x \leftarrow a]$ to denote the function mapping x to a, and every element y to $f(y)$ if $y \neq x$. Finally let $\mathsf{SubF}(\varphi)$ be the set of all φ sub-formulas.

Given a valuation $v : V \to \mathbb{N} \cup \{\perp\}$ for a set of variables occurring in a $\mathsf{CCTL}^{\mathsf{v}}$ formula φ, and an environment $\varepsilon : V \to \mathsf{SubF}(\varphi) \cup \{\perp\}$ such that $\mathsf{dom}(v) = \mathsf{dom}(\varepsilon)$, and given a finite run π, we define the valuation $(v +_\varepsilon \pi)$ as follows: $(v +_\varepsilon \pi)(z) \overset{\text{def}}{=} \perp$ if $z \notin \mathsf{dom}(v)$, and otherwise $(v +_\varepsilon \pi)(z) \overset{\text{def}}{=} v(z) + |\{j \mid 0 \leq j \leq |\pi| \wedge \pi(j) \models \varepsilon(z)\}|$. The semantics of $\mathsf{CCTL}^{\mathsf{v}}$ is defined as follows:

Definition 18. *The following clauses define when a state* q *of some KS* $\mathcal{S} = \langle Q, R, \ell \rangle$ *and a valuation* v *satisfy a* $\mathsf{CCTL}^{\mathsf{v}}$ *formula* φ *in an environment* ε *– written* $(q, v) \models_{\mathcal{S},\varepsilon} \varphi$ *– by induction over the structure of* φ *(we omit the cases of Boolean modalities):*

$$
\begin{aligned}
(q,v) &\models_{\mathcal{S},\varepsilon} \mathsf{E}\varphi\mathsf{U}\psi \;\; \textit{iff} \;\; \exists\rho \in \mathsf{Runs}(q), v \models_{\rho,\varepsilon} \varphi\mathsf{U}\psi \\
(q,v) &\models_{\mathcal{S},\varepsilon} \mathsf{A}\varphi\mathsf{U}\psi \;\; \textit{iff} \;\; \forall\rho \in \mathsf{Runs}(q), v \models_{\rho,\varepsilon} \varphi\mathsf{U}\psi \\
(q,v) &\models_{\mathcal{S},\varepsilon} z[\psi].\varphi \;\; \textit{iff} \;\; (q, v[z \leftarrow 0]) \models_{\mathcal{S},\varepsilon[z\leftarrow\psi]} \varphi \\
(q,v) &\models_{\mathcal{S},\varepsilon} \Sigma_{i=1}^{l}\alpha_i \cdot z_i \sim c \;\; \textit{iff} \;\; \Sigma_{i=1}^{l}\alpha_i \cdot v(z_i) \sim c
\end{aligned}
$$

where $v \models_{\rho,\varepsilon} \varphi\mathsf{U}\psi$ *iff* $\exists i \geq 0$, $(\rho(i), v +_\varepsilon \rho_{|i-1}) \models_{\mathcal{S},\varepsilon} \psi$ *and* $\forall 0 \leq j < i, (\rho(j), v +_\varepsilon \rho_{|j-1}) \models_{\mathcal{S},\varepsilon} \varphi$.

Theorem 19. *Model checking closed* $\mathsf{CCTL}^{\mathsf{v}}$ *formulas is* PSPACE*-complete.*

Proof. PSPACE-hardness can be proved by a reduction from QBF. PSPACE-membership is obtained by considering a non-deterministic algorithm working in polynomial space to decide whether a $\mathsf{CCTL}^{\mathsf{v}}$ formula holds for a state q within a KS $\mathcal{S}$. The main idea is to encode a configuration (q, v, ε) in polynomial size: this is possible for v since we just have to record the value for the counter z up to $d_{\max} + 1$ where $d_{\max}$ is the maximal constant used in a constraint with z. In order to verify $\mathsf{E}\varphi\mathsf{U}\psi$ – we assume that φ and ψ have already been treated – we guess, from the current configuration (q, v, ε), the next configuration (q', v', ε) and then we verify that there is a transition in $\mathcal{S}$ leading from q to q' such that the valuation v is updated with v' w.r.t. the environment ε. Then it remains to verify that either ψ or φ holds for (q', v') (and in the latter case, guess a new configuration *etc.*). The same holds for EG. The operator $z[\psi].\varphi$ changes the environment ε and resets z to zero. And for any configuration one can decide the truth value of a constraint C. □

6 Conclusion

In several cases (up to $\mathcal{B}(\alpha\mathcal{C}_1)$ constraints) the logics we introduce are not more expressive than CTL, but can concisely express properties which would be difficult to write in that logic. In particular, $\mathsf{CCTL}_{\mathcal{C}_0}$ and $\mathsf{CCTL}_{\mathcal{B}(\mathcal{C}_0)}$ can be exponentially more succinct than CTL. As for the remaining fragments, even though $\mathsf{CCTL}_{\mathcal{C}_2}$ is strictly more expressive than CTL, model-checking remains polynomial up to $\mathsf{CCTL}_{\mathcal{C}_3}$ (complexity results are summarized in Figure 2). Further work on CCTL will include completing the study of succinctness of its fragments with respect to each other and to other logics, looking for an upper complexity bound for the model-checking of $\mathsf{CCTL}_{\alpha\mathcal{C}_3}$, as well as investigating new kinds of constraints and extensions to LTL and CTL*.

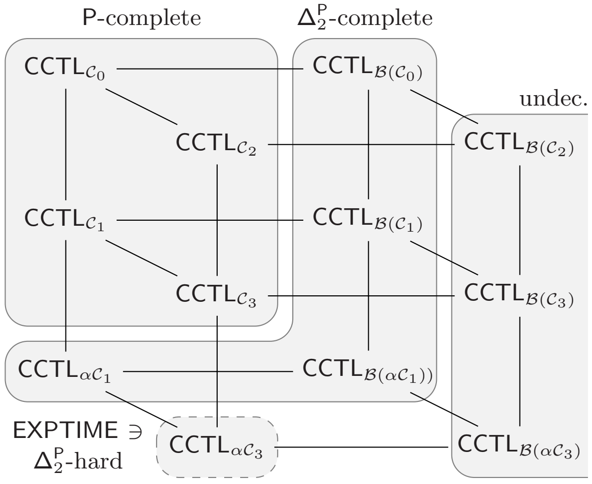

P-complete
Δ_2^{P}-complete
undec.
$\mathsf{CCTL}_{\mathcal{C}_0}$
$\mathsf{CCTL}_{\mathcal{B}(\mathcal{C}_0)}$
$\mathsf{CCTL}_{\mathcal{C}_2}$
$\mathsf{CCTL}_{\mathcal{B}(\mathcal{C}_2)}$
$\mathsf{CCTL}_{\mathcal{C}_1}$
$\mathsf{CCTL}_{\mathcal{B}(\mathcal{C}_1)}$
$\mathsf{CCTL}_{\mathcal{C}_3}$
$\mathsf{CCTL}_{\mathcal{B}(\mathcal{C}_3)}$
$\mathsf{CCTL}_{\alpha\mathcal{C}_1}$
$\mathsf{CCTL}_{\mathcal{B}(\alpha\mathcal{C}_1))}$
EXPTIME $\ni$
Δ_2^{P}-hard
$\mathsf{CCTL}_{\alpha\mathcal{C}_3}$
$\mathsf{CCTL}_{\mathcal{B}(\alpha\mathcal{C}_3)}$

Fig. 2. Summary of model-checking complexity results

References

1. Adler, M., Immerman, N.: An $n!$ lower bound on formula size. ACM Transactions on Computational Logic 4(3), 296–314 (2003)
2. Alur, R., Courcoubetis, C., Dill, D.L.: Model-checking in dense real-time. Inf. Comput. 104(1), 2–34 (1993)
3. Bernholtz, O., Vardi, M., Wolper, P.: An automata-theoretic approach to branching-time model-checking. In: Dill, D.L. (ed.) CAV 1994. LNCS, vol. 818, pp. 142–155. Springer, Heidelberg (1994)
4. Bianco, A., de Alfaro, L.: Model checking of probabilistic and nondeterministic systems. In: Thiagarajan, P.S. (ed.) FSTTCS 1995. LNCS, vol. 1026, pp. 499–513. Springer, Heidelberg (1995)
5. Bouajjani, A., Echahed, R., Habermehl, P.: On the verification problem of nonregular properties for nonregular processes. In: Proc. 10th LICS, pp. 123–133. IEEE Comp. Soc. Press, Los Alamitos (1995)
6. Bouajjani, A., Echahed, R., Habermehl, P.: Verifying infinite state processes with sequential and parallel composition. In: Proc. 22nd POPL, pp. 95–106 (1995)
7. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Kozen, D. (ed.) Logic of Programs 1981. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982)
8. Emerson, E.A.: Temporal and modal logic. In: Handbook of Theoretical Computer Science, vol. B, ch. 16, pp. 995–1072. Elsevier, Amsterdam (1990)
9. Emerson, E.A., Mok, A.K., Sistla, A.P., Srinivasan, J.: Quantitative temporal reasoning. Real-Time Systems 4(4), 331–352 (1992)
10. Emerson, E.A., Trefler, R.J.: Generalized quantitative temporal reasoning: An automata-theoretic approach. In: Bidoit, M., Dauchet, M. (eds.) CAAP 1997, FASE 1997, and TAPSOFT 1997. LNCS, vol. 1214, pp. 189–200. Springer, Heidelberg (1997)
11. Emerson, E.A., Trefler, R.J.: Parametric quantitative temporal reasoning. In: Proc. 14th LICS, pp. 336–343. IEEE Comp. Soc. Press, Los Alamitos (1999)
12. Laroussinie, F., Markey, N., Schnoebelen, P.: Model checking CTL^+ and $FCTL$ is hard. In: Honsell, F., Miculan, M. (eds.) FOSSACS 2001. LNCS, vol. 2030, pp. 318–331. Springer, Heidelberg (2001)
13. Laroussinie, F., Markey, N., Schnoebelen, P.: Efficient timed model checking for discrete-time systems. Theor. Comput. Sci. 353(1-3), 249–271 (2006)
14. Laroussinie, F., Schnoebelen, P.: Specification in CTL+Past for verification in CTL. Inf. Comput. 156(1/2), 236–263 (2000)
15. Laroussinie, F., Schnoebelen, P., Turuani, M.: On the expressivity and complexity of quantitative branching-time temporal logics. Theor. Comput. Sci. 297(1-3), 297–315 (2003)
16. Pnueli, A.: The temporal logic of programs. In: Proc. 18th FOCS, pp. 46–57. IEEE Comp. Soc. Press, Los Alamitos (1977)
17. Queille, J.-P., Sifakis, J.: Specification and verification of concurrent systems in CESAR. In: Dezani-Ciancaglini, M., Montanari, U. (eds.) Programming 1982. LNCS, vol. 137, pp. 337–351. Springer, Heidelberg (1982)
18. Wilke, T.: CTL^+ is exponentially more succinct than CTL. In: Pandu Rangan, C., Raman, V., Sarukkai, S. (eds.) FST TCS 1999. LNCS, vol. 1738, pp. 110–121. Springer, Heidelberg (1999)
19. Wolper, P.: Temporal logic can be more expressive. Inf. and Control 56(1/2), 72–99 (1983)
20. Yang, J., Mok, A.K., Wang, F.: Symbolic model checking for event-driven real-time systems. ACM Transactions on Programming Languages and Systems 19(2), 386–412 (1997)

Algorithmic Metatheorems for Decidable LTL Model Checking over Infinite Systems

Anthony Widjaja To and Leonid Libkin

LFCS, School of Informatics, University of Edinburgh
{anthony.w.to,libkin}@ed.ac.uk

Abstract. By algorithmic metatheorems for a model checking problem P over infinite-state systems we mean generic results that can be used to infer decidability (possibly complexity) of P not only over a specific class of infinite systems, but over a large family of classes of infinite systems. Such results normally start with a powerful formalism F of infinite-state systems, over which P is undecidable, and assert decidability when is restricted by means of an extra "semantic condition" C. We prove various algorithmic metatheorems for the problems of model checking LTL and its two common fragments LTL($\mathbf{F}_{\mathrm{s}}, \mathbf{G}_{\mathrm{s}}$) and LTL$_{\mathrm{det}}$ over the expressive class of word/tree automatic transition systems, which are generated by synchronized finite-state transducers operating on finite words and trees. We present numerous applications, where we derive (in a unified manner) many known and previously unknown decidability and complexity results of model checking LTL and its fragments over specific classes of infinite-state systems including pushdown systems; prefix-recognizable systems; reversal-bounded counter systems with discrete clocks and a free counter; concurrent pushdown systems with a bounded number of context-switches; various subclasses of Petri nets; weakly extended PA-processes; and weakly extended ground-tree rewrite systems. In all cases, we are able to derive optimal (or near optimal) complexity. Finally, we pinpoint the exact locations in the arithmetic and analytic hierarchies of the problem of checking a relevant semantic condition and the LTL model checking problems over all word/tree automatic systems.

1 Introduction

The study of model checking over infinite-state systems is now an active research area. This can be justified by the plethora of real-world scenarios that can be more conveniently modeled using infinite-state systems rather than finite-state systems, e.g., those that typically arise as programs with unbounded data structures (including stacks, lists, and FIFO queues), numeric variables, and clocks. To make sense of the problem of verifying infinite-state systems, the systems under consideration need to have some finite representations, e.g., timed automata, pushdown automata, counter machines, Turing machines, and so forth. Unlike in the case of finite systems, model checking even the most basic properties, such as safety and liveness, is already undecidable over infinite-state systems in general. For this reason, one either adopts non-Turing-powerful formalisms which admit decidability or resorts to semi-algorithms for general formalisms. Examples of formalisms that admit certain decidable model checking problems include pushdown automata [9], higher-order pushdown automata [20], Petri nets [9], timed automata [3], lossy channel systems [1, 4], certain subclasses of counter machines [11, 21, 24], and classes of term/tree rewrite systems [9, 25, 31], to name a few. On the other hand, various notions of finite-state transducers on words/trees [2, 7, 12, 15, 29, 34] and certain extensions of counter machines [6, 11] have emerged as popular general (Turing-powerful) frameworks of infinite-state model checking, for some of which semi-algorithmic approaches to verifying basic model checking properties, including safety and liveness, have been proposed (e.g. see [2, 6, 8] and references therein).

The vast literature of infinite-state model checking in the past decade or so can be extremely daunting and easily obscure proof patterns that can be *reused with ease* to obtain model checking decidability results for seemingly unrelated formalisms of infinite-state systems. This issue motivates the study of *algorithmic metatheorems* for infinite-state model checking, which are generic results that can be used in a "plug-and-play" manner for inferring decidability of certain model checking tasks over *a large family* of formalisms of infinite-state systems, instead of doing so for *a single* formalism at a time. Of course the concept of algorithmic metatheorems is not new; the classical decidability of S2S can be viewed as such for MSO model-checking, due to its wide applicability via the method of interpretations. Other results of this nature include results on flat counter machines [11, 24], well-structured transition systems [4, 17], and the extension of the S2S result to Caucal hierarchy [33]. In the finite case, algorithmic metatheorems are used extensively to obtain good algorithmic bounds for evaluating logical formulas over finite structures [19].

In this paper we study generic algorithmic metatheorems for designing efficient algorithms for model checking LTL, together with two of its commonly considered fragments LTL($\mathbf{F}_s, \mathbf{G}_s$) [31, 32] and LTL$_{\text{det}}$ [27], over infinite-state systems. Our choice of logic is justified by the fact that LTL, LTL($\mathbf{F}_s, \mathbf{G}_s$), and LTL$_{\text{det}}$ can express frequently checked properties including safety and liveness, and that their model checking problems have been frequently studied in the setting of infinite-state systems (e.g. [8, 9, 22, 31]). We will use as our framework the expressive class of *word/tree automatic transition systems* [7, 15, 34], which are generated by *synchronized rational transducers* [12, 15] over finite words and finite ranked trees. Such systems subsume many important decidable formalisms, including many which we previously mentioned and others, and still possess desirable closure and decidability properties (e.g. see [7, 12]), many of which are not satisfied by the general class of rational transducers on words [29]. Since verifying safety and liveness are in general undecidable over automatic transition systems [7], we will study various "semantic restrictions" for ensuring decidability of LTL, LTL($\mathbf{F}_s, \mathbf{G}_s$), or LTL$_{\text{det}}$, without unnecessarily sacrificing *applicability* and *algorithmic efficiency.*

Contributions. We identify semantic conditions on word/tree automatic transition systems that let us conclude decidability (and complexity) of model-checking.

We start with a condition **(C1)** stating that the reachability relation is effectively computable and given by a synchronized rational word/tree transducer. There are many examples of classes of systems satisfying this condition (e.g. see Table 1 in Section 4). Another condition **(C2)** says that a class of systems is closed under products with finite systems. We show that under **(C1)** and **(C2)**, LTL model-checking is decidable with good complexity bounds: exponential in the formula, and polynomial in the size of the input automatic presentation of the system, assuming an oracle for computing the reachability relation.

While many classes of systems satisfy **(C1)**, extending it to products (condition **(C2)**) could be problematic. Thus, we study various weakening of **(C2)** that could be used to obtain metatheorems for *fragments* of LTL. In this paper, we look at the following fragments: (1) LTL($\mathbf{F}_s, \mathbf{G}_s$) with only strict **F** and **G** operators, and (2) LTL$_{\text{det}}$ of [27]. We show that restricting to **(C2)** to closure under products with dag-like finite systems, or dropping **(C2)** altogether at the expense of a slightly worse complexity bound, decidability (and complexity) results for LTL($\mathbf{F}_s, \mathbf{G}_s$) and LTL$_{\text{det}}$ model checking could be retained. We also look at variations of these results for Presburger-definable infinite systems.

We then turn to applications, and show how our metatheorems can be used to derive (in a unified manner) known asymptotic upper bounds for LTL model-checking over some classes of systems, or produce new (or improved) complexity bounds for LTL and its fragment over other classes. Our results are summarized in Table 1.

Finally, we study the degrees of undecidability for the model checking problem and the problem of checking a relevant semantic condition over all word/tree automatic transition systems. We point out their locations in the arithmetic and analytic hierarchies.

Organization. Definitions and notations are given in Section 2. Metatheorems are presented in Section 3. Applications are given in Section 4. Undecidability results are described in Section 5. Due to space limitations, most proofs are omitted and can be found in the full version.

Related Work. The study of logical structures generated by finite-state automata and transducers can be traced far back (e.g., [15]). Since then, various models of finite-state automata and transducers have been studied, e.g., rational transducers on words (cf. [5, 29]), synchronized rational transducers on words and trees (cf. [5, 7, 12, 34]), synchronized rational transducers on infinite words and infinite trees (cf. [5, 7, 8]), and length-preserving synchronized rational transducers on finite words (cf. [2]). See [5] for a detailed comparison of their expressive power. In this paper we are concerned only with synchronized rational transducers on finite words and trees. In the case of length-preserving rational transducers on finite words, it is easy to show that LTL model checking is decidable under condition **(C1)** and **(C2)** (cf. [2]). The difficulty of extending this result to (not necessarily length-preserving) synchronized rational transducers on finite words lies in the fact that one has to deal with *genuinely* infinite execution paths, which do not visit two states twice, in the transition systems. Such paths do not exist when the length-preserving restriction is imposed on the transducers.

It is natural to ask whether our results hold in the case of the more general class of rational transducers or synchronized rational transducers on ω-words (they are actually incomparable [5]). We do not know the answer to this question and leave this for future work. We also mention the paper [8], which offers a semi-algorithmic approach to handling LTL model checking over systems generated by deterministic-weak synchronized rational transducers on ω-words. Finally, we mention that even though the aforementioned notions of transducers are Turing-powerful, they cannot capture all transition systems generated by higher-order pushdown systems (cf. [5, 20]).

2 Preliminaries

Transition systems, reachability, and recurrent reachability. Let ACT be a finite set of *action labels*. A *transition system* over ACT is given as $\mathcal{S} = \langle S, (\to_a)_{a\in\mathrm{ACT}}\rangle$, where S is a set of *states* and each $\to_a$ is a binary relation on S, i.e., a subset of $S \times S$. The set S is not required to be finite. We write $\to$ for the union of all transition relations $\to_a$ ($a \in$ ACT) and $\to^+$ (resp. $\to^*$) to denote the transitive (resp. transitive-reflexive) closure of $\to$.

Given a transition system $\mathcal{S} = \langle S, (\to_a)_{a\in\mathrm{ACT}}\rangle$ and a set $X \subseteq S$, by $Reach^\infty(\mathcal{S}, X)$ we denote the set of states $s \in S$ from which there exists an infinite execution path in $\mathcal{S}$ visiting X infinitely often, i.e., there exists an infinite sequence $s \to^+ s_0 \to^+ s_1 \to^+ s_2 \to^+ \ldots$ so that $s_i \in X$ for all $i \geq 0$. We refer to these sets as *recurrent reachability* sets.

Automata and transducers. We assume basic familiarity with automata on finite and ω-words. Let Σ be a finite alphabet. Given an automaton $\mathcal{A} = (Q, \delta, q_0, F)$ with states Q, initial state q_0, final states F and transition function δ, a run of $\mathcal{A}$ on $w = a_1 \ldots a_n$ (with $n \leq \omega$) is a function $\rho : \{0, \ldots, n\} \to Q$ with $\rho(0) = q_0$ that obeys δ, i.e. $\rho(i+1) \in \delta(\rho(i), a_{i+1})$. The length $\|\rho\|$ of ρ is n. We use abbreviations NWA and NBWA for nondeterministic (Büchi) automata.

We use *synchronized rational (letter-to-letter) transducers* [7] to define relations P over Σ-words, i.e., $P \subseteq \Sigma^* \times \Sigma^*$. Such transducers are simply NWA R over $\Sigma_\perp \times \Sigma_\perp$, where $\Sigma_\perp := \Sigma \cup \{\perp\}$ and $\perp \notin \Sigma$ is a padding symbol (so that the NWA could take two input words of different length). More precisely, given two words $w = a_1 \ldots a_n$ and $w' = b_1 \ldots b_m$ over the alphabet Σ, we define a word $w \otimes w'$ of length $k = \max\{n, m\}$ over alphabet $\Sigma_\perp \times \Sigma_\perp$ so that the ith letter of $w \otimes w'$ is $\begin{bmatrix} a'_i \\ b'_i \end{bmatrix}$, where a'_i is a_i for $i \leq n$, and $\perp$ for $i > n$ (and likewise $b'_i = b_i$ for $i \leq m$ and $\perp$ for $i > m$). That is, the shorter word is padded with $\perp$'s, and the ith letter of $w \otimes w'$ is then the pair of the ith letters of padded w and w'. The binary relation "recognized" by the transducer R is the set $\{(w, w') \in \Sigma^* \times \Sigma^* : w \otimes w' \in L(R)\}$. Such a relation is also called *regular*. We refer to such an automaton as a *transducer* over Σ^*, since it can be alternatively viewed as mapping words $w \in \Sigma^*$ nondeterministically into words w' so that $w \otimes w'$ is accepted by R.

Likewise we define transducers over finite k-ary trees [7, 12, 34]. In the following, we recall the definition for $k = 2$. A binary tree $T = (D, \tau)$ consists of a tree domain (a finite prefix-closed subset of $\{0, 1\}^*$) and a node labeling function $\tau : D \to \Sigma$. The notation $T = T_1 \otimes T_2$ is used to refer to a tree over the labeling alphabet $\Sigma_\perp^2$ similarly to the definition of $w \otimes w'$. That is, the domain of T is $D_1 \cup D_2$, and the labeling $\tau : D_1 \cup D_2 \to \Sigma_\perp^2$ is defined as $\tau(u) = (a_1, a_2)$ so that $a_i = \tau_i(u)$ if $u \in D_i$ and $\perp$ otherwise, for $i = 1, 2$. With this definition, the notion of tree transducers is defined similarly to the notion of word transducers: as nondeterminsitic tree automata working on $T_1 \otimes T_2$. Binary relations over trees defined that can be recognized by such transducers are called *(tree) regular*. In the sequel, we use NTA (resp. NTT) for tree automata (resp. transducers).

We shall use the notations $L(\mathcal{A})$ (or $L(R)$) for the language (or relation) accepted by automaton (or transducer) $\mathcal{A}$ (or R).

Automatic presentations of transition systems. We deal with infinite transition systems that can be finitely presented by automata and transducers. A *word-automatic presentation* is $\vartheta = \langle \mathcal{A}; \{R_a\}_{a \in \mathrm{ACT}} \rangle$ where $\mathcal{A}$ is an automaton over some finite alphabet Σ, and each R_a is a transducer over Σ. This presentation generates an automatic transition system $\Lambda(\vartheta) = \langle S; \{\to_a\}_{a \in \mathrm{ACT}} \rangle$, where $S = L(\mathcal{A})$ and $\to_a := L(R_a) \cap S$ for each $a \in \mathrm{ACT}$. *Tree-automatic presentations* and transition systems generated by them are defined similarly except that $\mathcal{A}$ is a tree automaton and R_a's are tree transducers.

Given a transition system $\mathcal{S} = \langle S; \{\to_a\}_{a \in \mathrm{ACT}} \rangle$ generated by a word or a tree-automatic presentation, each first-order (FO) formula $\varphi(x)$ with one free variable (resp. $\varphi(x, y)$ with two free variables) can effectively be converted into a word or tree automaton defining $\{s \in S \mid \mathcal{S} \models \varphi(s)\}$ (resp. word or tree transducer defining $\{(s, s') \in S \times S : \mathcal{S} \models \psi(s, s')\}$. This could actually be generalized to k free variables [7].

We denote by $\mathrm{wAut_P}$ and $\mathrm{tAut_P}$ the classes of word-automatic and tree-automatic presentations, respectively. In the sequel, our metatheorems will talk about subclasses $\mathcal{C} \subseteq \mathrm{wAut_P}$ or $\mathcal{C} \subseteq \mathrm{tAut_P}$ satisfying certain conditions. The following several conditions will be tacitly assumed for such $\mathcal{C}$. First, it should be easy (i.e. in poly-time) to check membership in $\mathcal{C}$. This condition has a standard complexity-theoretic explanation: checking whether the input encoding of an instance to a problem is valid should be easily doable. Secondly, we do not require these classes $\mathcal{C}$ to be isomorphism-closed, i.e., there possibly exist two automatic presentations $\vartheta \in \mathcal{C}$ and $\vartheta' \in \overline{\mathcal{C}}$ generating two isomorphic transition systems $\Lambda(\vartheta)$ and $\Lambda(\vartheta')$. In fact, asserting closure under isomorphism is too strong as it is undecidable to check isomorphisms for automatic systems [7].

LTL The syntax of LTL over ACT is

$$\varphi, \varphi' := a \ (a \in \mathrm{ACT}) \mid \neg\varphi \ \mid \varphi \vee \varphi' \mid \varphi \wedge \varphi' \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi'.$$

We shall use the standard abbreviations: $\mathbf{F}\varphi$ for $true\mathbf{U}\varphi$, $\mathbf{G}\varphi$ for $\neg\mathbf{F}\neg\varphi$, and $\mathbf{F}_\mathrm{s}$ and $\mathbf{G}_\mathrm{s}$ for their strict versions: $\mathbf{F}_\mathrm{s}\varphi = \mathbf{X}\mathbf{F}\varphi$ and $\mathbf{G}_\mathrm{s}\varphi = \neg\mathbf{F}_\mathrm{s}\neg\varphi$.

Given an ω-word $w \in \mathrm{ACT}^\omega$, we define the satisfaction relation $w \models \varphi$ in the standard way. We write $[\![\varphi]\!]$ for the set of all $w \in \mathrm{ACT}^\omega$ such that $w \models \varphi$.

It is well-known [35] that there exists an exponential-time algorithm which, given an LTL formula φ, computes an NBWA $\mathcal{A}_\varphi$ satisfying $L(\mathcal{A}_\varphi) = [\![\varphi]\!]$.

Given a transition system $\mathcal{S} = \langle S, (\to_a)_{a \in \mathrm{ACT}} \rangle$ and a word $u = a_0 a_1 a_2 \ldots \in \mathrm{ACT}^\omega$, we say that $w_0 \in S$ *realizes* u if there is a sequence of $w_0, w_1, w_2, \ldots$ of elements of S so that $w_i \xrightarrow{a_i} w_{i+1}$ for all $i \geq 0$. We then define the semantics of LTL formulas in the standard way: $(\mathcal{S}, w_0) \models \varphi$ iff every ω-word $u \in \mathrm{ACT}^\omega$ realized by w_0 satisfies φ. We write $[\![\varphi]\!]_{\mathcal{S}}^{\forall}$ for the set of all $w_0 \in S$ such that $(\mathcal{S}, w_0) \models \varphi$ (where $\forall$ means that every path starting in w_0 satisfies φ). We also write $[\![\varphi]\!]_{\mathcal{S}}^{\exists}$ for the complement of the set $[\![\neg\varphi]\!]_{\mathcal{S}}^{\forall}$, i.e., for the set of w_0 that realizes at least one path satisfying φ.

3 Metatheorems for LTL and Its Fragments

Since LTL formulas are translated into Büchi automata, a starting point for us is a known metatheorem that gives a semantic condition which implies bounds (and structural properties) for recurrent reachability sets. We now define condition on a class $\mathcal{C}$ of presentations in $\mathrm{wAut_P}$ (resp. in $\mathrm{tAut_P}$):

(C1) *There exists an algorithm $A_{\mathcal{C}}$ which, given an input presentation $\vartheta = \langle \mathcal{A}; \{R_a\}_{a \in \mathrm{ACT}} \rangle \in \mathcal{C}$ of the automatic transition system $\Lambda(\vartheta) = \langle S; \{\to_a\}_{a \in \mathrm{ACT}} \rangle$, computes an NWT (resp. NTT) R^+ recognizing the transitive closure relation $\to^+ = \left(\bigcup_{a \in \mathrm{ACT}} \to_a\right)^+$.*

Intuitively, **(C1)** asserts that the transitive closure relations of systems $\Lambda(\vartheta)$ with $\vartheta \in \mathcal{C}$ are effectively regular. We denote the running time of $A_{\mathcal{C}}$ to be $t_{A_{\mathcal{C}}}$. The following results state that under **(C1)**, recurrent reachability sets can be computed with polynomial-time overhead.

Theorem 1 ([34]). *Fix any class $\mathcal{C} \subseteq \mathrm{wAut_P}$ (resp. $\mathcal{C} \subseteq \mathrm{tAut_P}$) satisfying (C1). Given an automatic presentation $\vartheta \in \mathcal{C}$ and an NWA (resp. NTA) $\mathcal{A}_0$, the set $\mathrm{Reach}^\infty(\Lambda(\vartheta), L(\mathcal{A}_0))$ is regular, for which an NWA (resp. NTA) is computable in time polynomial in $\|\vartheta\| + \|\mathcal{A}_0\| + t_{A_{\mathcal{C}}}(|\vartheta|)$. In particular, if $A_{\mathcal{C}}$ runs in poly-time, then an NWA (resp. NTA) for $\mathrm{Reach}^\infty(\Lambda(\vartheta), L(\mathcal{A}_0))$ is poly-time computable.*

3.1 A Metatheorem for LTL

We now adapt Theorem 1 to produce a metatheorem for LTL. Consider a finite system $\mathcal{F} = \langle Q = \{q_0, \ldots, q_n\}, \delta \rangle$, with $\delta : Q \times \mathrm{ACT} \to Q$. Given a presentation $\vartheta \in \mathrm{wAut_P}$ of the system $\Lambda(\vartheta) = \langle S \subseteq \Sigma^*, \{\to_a\}_{a \in \mathrm{ACT}} \rangle$, we define $\mathcal{F} \cdot \Lambda(\vartheta)$ to be the automatic transition system $\langle S'; \{\to'_a\}_{a \in \mathrm{ACT}} \rangle$ as follows:

– $S' := QS := \{qs : q \in Q, s \in S\}$; it is a regular language over $S \cup Q$.
– $qw \to'_a q'w'$ iff $q' \in \delta(q, a)$ and $w \to_a w'$.

It is easy to give an automatic presentation ϑ' of $\mathcal{F} \cdot \Lambda(\vartheta)$ and show that ϑ' is poly-time computable. For presentations $\vartheta \in \text{TAUT}_{\text{P}}$, we could define $\mathcal{F} \cdot \Lambda(\vartheta)$ in a similar way (e.g. by defining the domain to be $Q(S) = \{q(T) : q \in Q, T \in S\}$ where $q(T)$ is the tree obtained by attaching q to T as a root).

We now define another condition **(C2)** stating that the class $\mathcal{C}$ is closed under products with finite systems:

(C2) *if $\vartheta \in \mathcal{C}$ and $\mathcal{F}$ is a finite system then $\mathcal{F} \cdot \Lambda(\vartheta) \in \mathcal{C}$.*

The following theorem is now almost immediate from Theorem 1 and the standard translation from LTL into Büchi automata. Intuitively, it says that for automatic presentations satisfying both **(C1)** and **(C2)**, for LTL model-checking the overhead (compared to $t_{A_\mathcal{C}}$) is polynomial in the automatic presentation and exponential in the LTL formula. In particular, if $t_{A_\mathcal{C}}$ is polynomial itself, then LTL model-checking is polynomial in the size of the representation of the system and exponential in the size of the formula.

Theorem 2. *Fix any set $\mathcal{C} \subseteq \text{WAUT}_{\text{P}}$ (resp. $\mathcal{C} \subseteq \text{TAUT}_{\text{P}}$) satisfying both **(C1)** and **(C2)**. Given $\vartheta \in \mathcal{C}$ and an LTL formula φ, the set $[\![\neg\varphi]\!]^{\exists}_{\Lambda(\vartheta)}$ is regular, for which an automaton is computable in time polynomial in $\|\vartheta\| + \|\mathcal{A}\| + t_{A_\mathcal{C}}(2^{O(\|\varphi\|)} \times \|\vartheta\|)$. Thus, checking whether $(\Lambda(\vartheta), v_0) \models \varphi$ can be done in time polynomial in $\|\vartheta\| + \|v_0\| + t_{A_\mathcal{C}}(2^{O(\|\varphi\|)} \times \|\vartheta\|)$.*

There are many examples of classes of automatic structures of interest in verification that satisfy condition **(C1)** (see, e.g., [34] for a list). So it is natural to ask whether having condition **(C1)** for a class of automatic presentations $\mathcal{C}$ implies having it for products of structures in that class with finite systems. While we shall see some examples of classes where this happens (e.g., pushdown systems), in general such an extension is impossible even in very simple cases, e.g., for single structures, as the result below shows.

Proposition 3. *There exist an automatic presentation ϑ satisfying* **(C1)** *and a finite system $\mathcal{F}$ so that in $\mathcal{F} \cdot \Lambda(\vartheta)$ the reachability relation is not regular (in fact, not even recursive).*

So the applicability of Theorem 2 in full generality may be rather limited. We thus look at cases when conditions weaker than **(C2)** will allow us to conclude the decidability of model-checking. They will not apply to full LTL, but they will apply to some of its well-studied and important fragments. The distinguishing feature of these fragments is that formulas expressible in them can be translated into special types of Büchi automata, whose graph structures are rather nice (essentially, almost DAGs). We next look at such cases.

3.2 Metatheorems for LTL$_{\text{det}}$

We first recall the definition of the fragment LTL$_{\text{det}}$ of LTL [27].

$$\begin{aligned}\varphi, \varphi' \;:= p \mid \mathbf{X}\varphi \mid \varphi \wedge \varphi' \mid (p \wedge \varphi) \vee (\neg p \wedge \varphi') \mid \\ (p \wedge \varphi)\mathbf{U}(\neg p \wedge \varphi') \mid (p \wedge \varphi)\mathbf{W}(\neg p \wedge \varphi').\end{aligned}$$

Here p is a boolean combination of ACT, and $\varphi \mathbf{W} \varphi'$ is interpreted as the formula $\mathbf{G}\varphi \vee (\varphi \mathbf{U} \varphi')$, i.e., the "weak until" operator.

Formulae in this fragment can be translated into a special kind of automata called 1-weak NBWAs. Formally, a *1-weak NBWA* $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ is an NBWA with a partial order $\preceq$ on the set Q such that $q' \in \delta(q,a)$ implies $q \preceq q'$. Intuitively, the partial order ensures that once $\mathcal{A}$ leaves a state q, it will never be able to come back to q. In other words, graph-theoretically $\mathcal{A}$ looks like a dag possibly with self-loops.

It was shown in [27] that there exists a poly-time algorithm which, given an $\mathrm{LTL}_{\mathrm{det}}$ formula φ, computes a 1-weak NBWA $\mathcal{A}_{\neg\varphi}$ such that $L(\mathcal{A}_{\neg\varphi}) = [\![\neg\varphi]\!]$. (The running time was not explicitly mentioned in [27], but one can easily check that it is polynomial).

We now weaken the condition **(C2)** to the following:

(C2') *if $\vartheta \in \mathcal{C}$ and $\mathcal{F}$ is a finite system that is 1-weak then $\mathcal{F} \cdot \Lambda(\vartheta) \in \mathcal{C}$.*

Combining Theorem 1 with the translation of [27], we may proceed as in the proof of Theorem 2 and obtain the following theorem.

Theorem 4. *Fix any set $\mathcal{C} \subseteq$ wAut$_{\mathrm{P}}$ (resp. $\mathcal{C} \subseteq$ tAut$_{\mathrm{P}}$) satisfying **(C1)** and **(C2')**. Given $\vartheta \in \mathcal{C}$ and an LTL$_{det}$ formula φ, the set $[\![\neg\varphi]\!]^{\exists}_{\Lambda(\vartheta)}$ is regular, for which an automaton is computable in time polynomial in $\|\vartheta\| + \|\mathcal{A}\| + t_{A_\mathcal{C}}(\|\varphi\| \times \|\vartheta\|)$. Thus, checking whether $(\Lambda(\vartheta), v_0) \models \varphi$ can be done in time polynomial in $\|\vartheta\| + \|v_0\| + t_{A_\mathcal{C}}(\|\varphi\| \times \|\vartheta\|)$.*

We now show that decidability can still be obtained without assuming condition **(C2')** but by slightly strengthening condition **(C1)**. Namely, we use a condition stating that the transitive closure can be computed not only for $\to$ but also for all unions of $\to_a$'s:

(C1') *there exists an algorithm $A_\mathcal{C}$ which, given an input presentation $\vartheta = \langle \mathcal{A}_\delta; \{R_a\}_{a \in \mathrm{ACT}} \rangle \in \mathcal{C}$ of the automatic transition system $\Lambda(\vartheta) = \langle S; \{\to_a\}_{a \in \mathrm{ACT}} \rangle$ and each subset $\mathrm{ACT}' \subseteq \mathrm{ACT}$, computes an NWT (resp. NTT) $R^+_{\mathrm{ACT}'}$ recognizing the transitive closure relation $\left(\bigcup_{a \in \mathrm{ACT}'} \to_a\right)^+$.*

In practice, **(C1')** is not much stronger than **(C1)**; all our examples in the next section which satisfy **(C1)** also satisfy **(C1')**. In this case, $\mathrm{LTL}_{\mathrm{det}}$ model-checking can be done in PSPACE assuming an oracle for $t_{A_\mathcal{C}}$; its running time is only exponential in the the size of the formula.

Theorem 5. *Fix any set $\mathcal{C} \subseteq$ wAut$_{\mathrm{P}}$ (resp. $\mathcal{C} \subseteq$ tAut$_{\mathrm{P}}$) satisfying **(C1')**. Given a presentation $\vartheta \in \mathcal{C}$, and an LTL$_{det}$ formula φ, checking whether $(\Lambda(\vartheta), v_0) \models \varphi$ can be done in time polynomial in $\|\vartheta\|$, $\|v_0\|$, $t_{A_\mathcal{C}}(\|\vartheta\|)$, and exponential in $\|\varphi\|$. Whenever $\mathcal{C} \subseteq$ wAut$_{\mathrm{P}}$, the space consumed by the algorithm is polynomial in $\|\vartheta\|$, $\|v_0\|$, $t_{A_\mathcal{C}}(\|\vartheta\|)$, and $\|\varphi\|$.*

3.3 Metatheorems for $\mathrm{LTL}(\mathbf{F}_{\mathrm{s}}, \mathbf{G}_{\mathrm{s}})$

Recall that in $\mathrm{LTL}(\mathbf{F}_{\mathrm{s}}, \mathbf{G}_{\mathrm{s}})$ we use operators $\mathbf{F}_{\mathrm{s}}$ and $\mathbf{G}_{\mathrm{s}}$ rather than $\mathbf{U}$ and $\mathbf{X}$. It turns out that our conditions **(C1)** and **(C2)** imply bounds on $\mathrm{LTL}(\mathbf{F}_{\mathrm{s}}, \mathbf{G}_{\mathrm{s}})$ model-checking. We start with the following.

Theorem 6. *Fix any set $\mathcal{C} \subseteq \textsc{wAut}_{\textsc{p}}$ (resp. $\mathcal{C} \subseteq \textsc{tAut}_{\textsc{p}}$) satisfying **(C1)** and **(C2)**. Given a presentation $\vartheta \in \mathcal{C}$ and an LTL$(\mathbf{F}_s, \mathbf{G}_s)$ formula φ, checking whether $(\Lambda(\vartheta), v_0) \models \varphi$ can be done in coNP assuming an oracle for $t_{A_\mathcal{C}}$. More precisely, checking whether $(\Lambda(\vartheta), v_0) \not\models \varphi$ can be done in nondeterministic time polynomial in $\|\vartheta\| + \|v_0\| + t_{A_\mathcal{C}}(\|\varphi\| \times \|\vartheta\|)$.*

The proof of this result is based on a translation into 1-weak NBWAs extended with fairness constraints, which are conjunctions of formulas $\mathbf{G}_{\mathrm{s}}\mathbf{F}_{\mathrm{s}}p$, where p is a disjunction over action labels in ACT [31]. We have to extend translation results from [31] to obtain more precise information about the structure of the automata, and then use it to prove the result along the lines of the proof in the previous subsection. See full version for details.

Our second metatheorem for $\mathrm{LTL}(\mathbf{F}_{\mathrm{s}}, \mathbf{G}_{\mathrm{s}})$ uses only condition **(C1)** and produces slightly higher, but still acceptable, complexity bounds.

Theorem 7. *Fix any set $\mathcal{C} \subseteq \textsc{wAut}_{\textsc{p}}$ (resp. $\mathcal{C} \subseteq \textsc{tAut}_{\textsc{p}}$) satisfying **(C1)**. Given a presentation $\vartheta \in \mathcal{C}$, and an LTL$(\mathbf{F}_s, \mathbf{G}_s)$ formula φ, checking whether $(\Lambda(\vartheta), v_0) \models \varphi$ can be done in time polynomial in $\|\vartheta\|$, $\|v_0\|$, $t_{A_\mathcal{C}}(\|\vartheta\|)$, and exponential in $\|\varphi\|$.*

3.4 A Metatheorem for Presburger-Definable Systems

In this subsection, we will make an extra assumption that the input presentations can be given by *existential* Presburger formulas. More precisely, we consider presentations of the form $\vartheta = \langle \varphi(\overline{x}); \{\varphi_a(\overline{x}, \overline{y})\}_{a \in \mathrm{ACT}} \rangle$, where $\overline{x}$ and $\overline{y}$ are k-tuples of variables for some $k \in \mathbb{Z}_{>0}$ and φ's some existential Presburger formulas. Such a presentation gives rise to the system $\Lambda(\vartheta) = \langle S; \{\to_a\}_{a \in \mathrm{ACT}} \rangle$, where $S = \{\overline{a} \in \mathbb{N}^k : (\mathbb{N}, +) \models \varphi(\overline{a})\}$ and $\to_a = \{(\overline{a}, \overline{b}) \in \mathbb{N}^{2k} : (\mathbb{N}, +) \models \varphi_a(\overline{a}, \overline{b})\}$. Let $\textsc{presAut}_{\textsc{p}}$ denote the set of all such presentations. Automatic presentations for $\textsc{presAut}_{\textsc{p}}$ could be given (cf. [7]).

For sets $\mathcal{C} \subseteq \textsc{presAut}_{\textsc{p}}$ (which, as before, need not be isomorphism-closed), we define a new semantic condition, which is essentially an adaption of **(C1')** to the class of Presburger-definable systems:

(C3) *there exists an algorithm $A_\mathcal{C}$ which, given an input presentation $\vartheta = \langle \varphi; \{\varphi_a\}_{a \in \mathrm{ACT}} \rangle \in \mathcal{C}$ of the system $\Lambda(\vartheta) = \langle S; \{\to_a\}_{a \in \mathrm{ACT}} \rangle$ and a subset $\mathrm{ACT}' \subseteq \mathrm{ACT}$, computes an existential Presburger formula $R^+(\overline{x}, \overline{y})$ which defines the transitive closure relation $\left(\bigcup_{a \in \mathrm{ACT}'} \to_a\right)^+$.*

We denote by $t_{A_\mathcal{C}}$ the running time of $A_\mathcal{C}$ in **(C3)**. In addition, we require that the class $\mathcal{C}$ satisfy the following *monotonicity* condition: for every $\mathcal{S} \in \mathcal{C}$ every

$\overline{a}, \overline{b} \in \mathbb{N}^k$ satisfying $\overline{a} \preceq \overline{b}$ (i.e. inequality holds component-wise), if $\overline{a} \to_a \overline{a} + \overline{\delta}$ for some $\overline{\delta} \in \mathbb{Z}^k$ and $a \in \text{ACT}$, then $\overline{b} \to_a \overline{b} + \overline{\delta}$. This is a strong condition, but is still satisfied by any subclass of Petri nets.

Theorem 8. *Fix any monotone $\mathcal{C} \subseteq \text{PRESAUT}_{\text{P}}$ satisfying* **(C3)***. Given $\vartheta \in \mathcal{C}$, $v_0 \in \mathbb{N}^k$ represented in binary, and an LTL$(\mathbf{F}_s, \mathbf{G}_s)$ or LTL$_{det}$ formula ψ, checking whether $(\Lambda(\vartheta), v_0) \not\models \psi$ can be done in nondeterministic time polynomial in $\|\vartheta\|$, $\|v_0\|$, $t_{A_\mathcal{C}}(\|\vartheta\|)$ and $\|\varphi\|$*

4 Applications

In this section we apply our metatheorems from the previous section to obtain decidability and complexity results for LTL, LTL$(\mathbf{F}_\text{s}, \mathbf{G}_\text{s})$, and LTL$_\text{det}$ model checking over specific classes of infinite systems. In some cases, we re-derive known results with asymptotically the same complexity bounds; in other cases we obtain new results. Our results are summarized in Table 1.

Pushdown systems (PDS). A *pushdown system* [9] is a tuple $\mathcal{P} = (\text{ACT}, \Gamma, Q, \Delta)$ where Γ is a *stack alphabet*, Q a set of states, and Δ is a finite subset of $Q \times \Gamma \times \text{ACT} \times Q \times \Gamma^*$. Let Δ_a be the set of tuples (called "rules") in Δ of the form (q, σ, a, q', w). Let $\Lambda(\mathcal{P})$ be the transition system $\langle Q \times \Gamma^*; \{\to_a\}_{a \in \text{ACT}} \rangle$, where $\to_a := \{((q, w\sigma), (q', ww')) : (q, \sigma, a, q', w') \in \Delta_a\}$. It is straightforward (and in poly-time) to give a word-automatic presentation of $\mathcal{P}$ (cf. [34]), and show that the class PDS of such presentations satisfy **(C2)**. Furthermore, it is known [10, 34] that PDS satisfies **(C1)** with polynomial running time.

Combined with our results in the previous section, it follows that model checking LTL, LTL$(\mathbf{F}_\text{s}, \mathbf{G}_\text{s})$, and LTL$_\text{det}$ are respectively in EXPTIME, coNP, and PTIME. It also follows that all these problems are in PTIME when fixing the formula. It was known (cf. [9]) that the complexity of model checking LTL over pushdown systems is EXPTIME-complete, and is PTIME for a fixed formula. On the other hand, the results for LTL$(\mathbf{F}_\text{s}, \mathbf{G}_\text{s})$ and LTL$_\text{det}$ are new (in the case of LTL$(\mathbf{F}_\text{s}, \mathbf{G}_\text{s})$ coNP-hardness can be derived from [32]).

Prefix-recognizable systems (Pref-RS). A *prefix-recognizable system (with states)* $\mathcal{P}$ is a tuple $\langle \text{ACT}, \Gamma, Q, \Delta)$ where ACT, Γ, and Q are defined as in pushdown systems, whereas Δ is a set of rules of the form $((q, U, V), a, (q', V'))$, where $q, q' \in Q$; $a \in \text{ACT}$; and U,V, and V' are regular languages over Γ given by NWAs. Let $\Lambda(\mathcal{P})$ be the transition system $S = \langle Q \times \Gamma^*; \{\to_a\}_{a \in \text{ACT}} \rangle$, where $\to_a$ is the set of tuples $((q, uv), (q', uv')) \in S \times S$ such that, for some $((q, U, V), a, (q', V')) \in \Delta_a$, we have $u \in U$, $v \in V$, and $v' \in V'$. It is straightforward (and in poly-time) to give a word-automatic presentation for $\mathcal{P}$.

Using Theorem 2, we can also rederive the known EXPTIME upper bound [22] for LTL model checking over prefix-recognizable systems (details are in the full version). Furthermore, EXPTIME-hardness for model checking a fixed LTL$_\text{det}$ and LTL$(\mathbf{F}_\text{s}, \mathbf{G}_\text{s})$ formula is obtained by reducing from the unreachability problem for prefix-recognizable systems, which has recently been proven to be EXPTIME-complete [18].

Concurrent pushdown systems (CPDS). A concurrent pushdown system (cf. [30]) is a tuple $\mathcal{P} = (\mathrm{ACT}, \Gamma, Q, \Delta^0, \ldots, \Delta^N)$, where each $\mathcal{P}_i = (\mathrm{ACT}, \Gamma, Q, \Delta^i)$ is a pushdown system. Suppose that $\Lambda(\mathcal{P}_i) = \langle Q \times \Gamma^*; \{\rightarrow_{i,a}\}_{a \in \Gamma}\rangle$. Then the transition system $\Lambda(\mathcal{P})$ generated by $\mathcal{P}$ is $\langle Q \times (\Gamma^*)^N; \{\rightarrow_a\}_{a \in \Gamma}\rangle$, where $\rightarrow_a := \bigcup_{i=0}^{N} \rightarrow_{i,a}$. Although concurrent pushdown systems are well-known to be Turing-powerful (so checking safety and liveness is undecidable), [23, 30] have shown that reachability is NP-complete if we consider runs of $\mathcal{P}$ with a bounded number k of "context-switches" (k part of the input). Intuitively, a context of $\mathcal{P}$ is an uninterrupted sequence of actions performed by exactly one "thread" $\mathcal{P}_i$. A context-switch occurs when $\mathcal{P}$ interrupts the execution of a thread $\mathcal{P}_i$ and resumes by executing a (possibly different) thread $\mathcal{P}_j$. The context-bounded reachability for $\mathcal{P}$ is simply the problem of reachability restricted to executions of $\mathcal{P}$ with exactly k context-switches for any given input k. One could similarly define the context-bounded LTL model checking problem for concurrent pushdown systems by restricting the executions of $\mathcal{P}$ to those with exactly k context-switches for any given k.

Using the results of [23, 30] and our metatheorems, we can show that context-bounded model checking LTL, LTL($\mathbf{F}_\mathrm{s}, \mathbf{G}_\mathrm{s}$), and $\mathrm{LTL}_\mathrm{det}$ over concurrent pushdown systems are respectively EXPTIME-complete, coNP-complete, and coNP-complete. If the formula is fixed, they are all coNP-complete.

Discrete timed counter systems (RCM and d-RCM). Although verifying safety and liveness for general counter machines is undecidable, it is known that these problems are decidable (cf. [14, 21]) when all the counters but one are *reversal-bounded* (only executions with a fixed number of reversals are considered). We denote by RCM the class of such machines. The LTL model checking problem for RCM is also known to be decidable [14], but no complexity analysis was given for their algorithm. Furthermore, it was left as an open question whether the same result holds for such machines extended with discrete clocks (in the sense of [3]), for which reachability is known to be decidable [13]. We write d-RCM for the class of such machines.

We answer this open question positively and give upper bounds for the cases with and without discrete clocks. Using our metatheorems in combination with a slightly refined version of the algorithms for computing binary reachability relations in [13, 21], we can give an algorithm for model checking LTL (resp. $\mathrm{LTL}_\mathrm{det}$ and LTL($\mathbf{F}_\mathrm{s}, \mathbf{G}_\mathrm{s}$)) over RCM that runs in time exponential in the size of the machine and double exponential (resp. exponential) in the size of the LTL (resp. $\mathrm{LTL}_\mathrm{det}$ and LTL($\mathbf{F}_\mathrm{s}, \mathbf{G}_\mathrm{s}$)) formula. Details of the construction and the analysis are in the full version.

For d-RCM, we have exactly the same upper bound complexity except that the algorithms run double exponential in the number of clocks.

Communication-free Petri nets (BPP). *Communication-free nets* (a.k.a. BPPs) [16, 28] are Petri nets where each transition has exactly one incoming arc (and, hence, "communication-free"). The LTL model checking over BPPs is known to be EXPSPACE-complete when only infinite traces are considered (cf. [28]). When finite traces are also considered, the problem is still decidable but no primitive recursive upper bound is known [28], since reachability for Petri nets could be reduced to this problem.

In contrast, we could show that $\mathrm{LTL}_{\mathrm{det}}$ and $\mathrm{LTL}(\mathbf{F}_{\mathrm{s}}, \mathbf{G}_{\mathrm{s}})$ model checking for BPPs is coNP-complete even when finite traces are considered. In fact, it is known that the transitive closure relations for BPPs are semi-linear [16]. Furthermore, one can then adapt the proof of [36, Theorem 4] to show that there exists a poly-time algorithm computing an existential Presburger formula defining the transitive closure relation for a given BPP. Since any subclass of Petri nets is monotone, Theorem 8 (which also holds when finite traces are considered) implies that $\mathrm{LTL}(\mathbf{F}_{\mathrm{s}}, \mathbf{G}_{\mathrm{s}})$ and $\mathrm{LTL}_{\mathrm{det}}$ model checking over BPPs are in coNP.

Furthermore, a matching lower bound could be easily given for a fixed formula in $\mathrm{LTL}(\mathbf{F}_{\mathrm{s}}, \mathbf{G}_{\mathrm{s}})$ and $\mathrm{LTL}_{\mathrm{det}}$ by reducing from the non-reachability problem for BPPs, which is coNP-complete [16].

Weakly extended PA processes (wPA). PA (cf. [26, 28, 31]) is a well-known process algebra allowing sequential and parallel compositions, but no communication. It is a common generalization of BPP (with unary representation for numbers) and the class of pushdown systems with one state (a.k.a. BPA). It is known (cf. [28, 31]) that LTL model checking over PA is undecidable. It is also known that decidability could be retained when restricting to $\mathrm{LTL}(\mathbf{F}_{\mathrm{s}}, \mathbf{G}_{\mathrm{s}})$ and $\mathrm{LTL}_{\mathrm{det}}$ [31]. However, no upper bound to these problems are known.

We can use Theorem 5 and Theorem 7 in combination with the encoding of PA and their binary reachability relations as tree-automatic systems (cf. [26, 34]) to give an exponential time upper bound for these problems. They are coNP-hard, which can be shown by a reduction from non-reachability problem for BPP [16]. The upper bound also holds when we consider *weakly extended PA* (wPA) [31], which are simply PA extended with weak finite control (i.e. 1-weak NBWA).

Weakly extended ground-tree rewrite systems (wGTRS). A *ground tree rewrite system* (GTRS) (cf. [25, 34]) over Σ-labeled trees is a finite set Δ of "rules" of the form (t, a, t') where $t, t' \in \textsc{Tree}(\Sigma)$ and $a \in \mathrm{ACT}$. For a tree T and a node u in it, let T_u be the subtree of T rooted at u. For a given $t \in \textsc{Tree}(\Sigma)$, we write $T[t/u]$ for the tree obtained from T by replacing the subtree T_u by t. The GTRS Δ generates the transition system $\Lambda(\Delta) = \langle \textsc{Tree}(\Sigma); \{\rightarrow_a\}_{a \in \mathrm{ACT}} \rangle$ where $T \rightarrow_a T'$ iff there exists a node u in T and a rule $(t, a, t') \in \Delta$ such that $T_u = t$ and $T' = T[t'/u]$. One could easily conclude that LTL model checking over GTRS is undecidable, using results of [25, 31].

On the other hand, our results imply that decidability is retained when we restrict to $\mathrm{LTL}(\mathbf{F}_{\mathrm{s}}, \mathbf{G}_{\mathrm{s}})$ or $\mathrm{LTL}_{\mathrm{det}}$. This follows from the fact that **(C1')** is satisfied by the class of automatic presentations of GTRSs (cf. [12, 34]). Therefore, we obtain exponential-time algorithms for model checking $\mathrm{LTL}_{\mathrm{det}}$ and $\mathrm{LTL}(\mathbf{F}_{\mathrm{s}}, \mathbf{G}_{\mathrm{s}})$ over GTRS, whose complexity becomes polynomial when the formula is fixed. We could also show that these problems are coNP-hard for non-fixed formulas.

One can also extend these results to GTRSs with weak finite control, as we did for PA-processes. Details are in the full version.

Table 1. A summary of combined and data complexity that we obtain. Here, $\times$ (resp. ud) means that the result cannot be obtained using our metatheorems (resp. undecidable). Whenever written in bold, the results are new. Also, coNP-h means coNP-hard.

	LTL		LTL($\mathbf{F}_s, \mathbf{G}_s$)		LTL$_{det}$	
	Comb.	Data	Comb.	Data	Comb.	Data
PDS	EXP	in P	**coNP**	in P	**in P**	in P
Pref-RS	EXP	EXP	EXP	EXP	EXP	EXP
CPDS	**EXP**	**coNP**	**coNP**	**coNP**	**coNP**	**coNP**
BPP	$\times$	$\times$	**coNP**	**coNP**	**coNP**	**coNP**
(w)PA	$\times$ (ud)	$\times$ (ud)	**in EXP** coNP-h	**in EXP** coNP-h	**in EXP** coNP-h	**in EXP** coNP-h
GTRS	$\times$ (ud)	$\times$ (ud)	**in EXP** coNP-h	**in P**	**in EXP** coNP-h	**in P**
wGTRS	$\times$ (ud)	$\times$ (ud)	**in EXP coNP-h**	**in EXP coNP-h**	**in EXP coNP-h**	**in EXP coNP-h**
RCM	**in 2-EXP**	**in EXP**	**in EXP**	**in EXP**	**in EXP**	**in EXP**
d-RCM	**in 2-EXP**					

5 How Hard Are These Problems in General?

Relevant to condition **(C1)** is the problem of checking whether the transitive closure relation of a given automatic presentation is regular, and the problem of checking whether a given transducer R' represents the transitive closure relation of another one R (over the same domain). We shall point out the degrees of undecidability of such problems in the arithmetic hierarchy. We shall then point out the degrees of undecidability of the model checking problems in the general case (i.e. over all word/tree automatic presentations), and compare this with the length-preserving case. We start with the problems related to "computing" transitive closure relations.

Theorem 9.
- *Given two nondeterministic word transducers R and R', checking whether R' is the transitive closure of R is Π_2^0-complete.*
- *Given a nondeterministic word transducers R, checking whether its transitive closure is regular is in Σ_3^0 and Π_2^0-hard.*

We now address the degrees of undecidability for checking recurrent reachability and model checking LTL, LTL($\mathbf{F}_s, \mathbf{G}_s$), and LTL$_{det}$ over automatic transition systems. Unlike the problem of reachability which can be shown to be Σ_1^0-complete (cf. [7, 29]), checking liveness is highly undecidable:

Theorem 10. *Recurrent reachability for both word and tree automatic transition systems is Σ_1^1-complete, and model-checking LTL, LTL($\mathbf{F}_s, \mathbf{G}_s$), and LTL$_{det}$ for them are all Π_1^1-complete.*

In fact, Theorem 10 could be shown to also hold when the general class of rational transducers is used (instead of synchronized rational).

Finally, many examples in the "regular model checking" literature (cf. [2]) deal with the subcase of length-preserving synchronized rational transducers (i.e., $w \to_a w'$ would imply $|w| = |w'|$). In this case, the LTL (resp. recurrent reachability) model checking problem is usually defined with respect to a regular set $Init$ of initial states with either "existential" (resp. "universal") semantics in the following sense: there exists $w \in Init$ such that (resp. all $w \in Init$ satisfies) $(\Lambda(\vartheta), w) \models \varphi$. [Note: when $Init$ is finite, the model checking problems become decidable since then the set of reachable states from $Init$ is finite.] In contrast to Theorem 10, we have the following proposition.

Proposition 11. *For automatic transition systems with length-preserving transducers, global recurrent reachability and LTL model checking are all*

- Σ^0_1*-complete (when existential semantics is considered);*
- Π^0_1*-complete (when universal semantics is considered).*

This result confirms the intuition that checking liveness is much easier when considering length-preserving transducers.

Acknowledgements. We thank anonymous referees for their helpful comments. First author was supported by EPSRC grant E005039 and the ORS Award. Second author was supported by EPSRC grants E005039 and and G049165.

References

1. Abdulla, P.A., Jonsson, B.: Verifying programs with unreliable channels. Inf. Comput. 127(2), 91–101 (1996)
2. Abdulla, P.A., Jonsson, B., Nilsson, M., Saksena, M.: A survey of regular model checking. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 35–48. Springer, Heidelberg (2004)
3. Alur, R., Dill, D.: A theory of timed automata. TCS 126, 183–235 (1994)
4. Baier, C., Bertrand, N., Schnoebelen, P.: On Computing Fixpoints in Well-Structured Regular Model Checking, with Applications to Lossy Channel Systems. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS (LNAI), vol. 4246, pp. 347–361. Springer, Heidelberg (2006)
5. Barany, V.: Automatic Presentations of Infinite Structures. PhD Thesis, RWTH Aachen (2007)
6. Bardin, S., Finkel, A., Leroux, J., Petrucci, L.: FAST: acceleration from theory to practice. STTT 10(5), 401–424 (2008)
7. Blumensath, A., Grädel, E.: Finite presentations of infinite structures: automata and interpretations. Theory Comput. Syst. 37(6), 641–674 (2004)
8. Bouajjani, A., Legay, A., Wolper, P.: A Framework to Handle Linear Temporal Properties in (ω)Regular Model Checking CoRR abs/0901.4080 (2009)
9. Burkart, O., Caucal, D., Moller, F., Steffen, B.: Verification on infinite structures. In: Handbook of Process Algebra. Elsevier, Amsterdam (1999)
10. Caucal, D.: On the regular structure of prefix rewriting. TCS 106, 61–86 (1992)
11. Comon, H., Jurski, Y.: Multiple counters automata, safety analysis and Presburger arithmetic. In: Y. Vardi, M. (ed.) CAV 1998. LNCS, vol. 1427, pp. 268–279. Springer, Heidelberg (1998)

12 Comon, H., et al.: Tree Automata: Techniques and Applications (2007), http://www.grappa.univ-lille3.fr/tata
13 Dang, Z., Ibarra, O., Bultan, T., Kemmerer, R., Su, J.: Binary reachability analysis of discrete pushdown timed automata. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 69–84. Springer, Heidelberg (2000)
14 Dang, Z., Ibarra, O., Pietro, P.S.: Liveness verification of reversal-bounded multicounter machines with a free counter. In: Hariharan, R., Mukund, M., Vinay, V. (eds.) FSTTCS 2001. LNCS, vol. 2245, pp. 132–143. Springer, Heidelberg (2001)
15 Elgot, C., Mezei, J.: On relations defined by generalized finite automata. IBM J. Res. Develop. 9, 47–68 (1965)
16 Esparza, J.: Petri Nets, Commutative Context-Free Grammars, and Basic Parallel Processes. Fundam. Inform. 31(1), 13–25 (1997)
17 Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theor. Comput. Sci. 256(1-2), 63–92 (2001)
18 Göller, S.: Reachability on prefix-recognizable graphs. Inf. Process. Lett. 108(2), 71–74 (2008)
19 Grohe, M.: Logic, graphs, and algorithms. In: Logic and Automata - History and Perspectives, pp. 357–422. Amsterdam University Press (2007)
20 Hague, M., Ong, C.-H.L.: Symbolic Backwards-Reachability Analysis for Higher-Order Pushdown Systems. LMCS 4(4) (2008)
21 Ibarra, O., Su, J., Dang, Z., Bultan, T., Kemmerer, R.: Counter machines and verification problems. Theor. Comput. Sci. 289, 165–189 (2002)
22 Kupferman, O., Piterman, N., Vardi, M.: Model checking linear properties of prefix-recognizable systems. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 371–385. Springer, Heidelberg (2002)
23 Lal, A., Touili, T., Kidd, N., Reps, T.: Interprocedural Analysis of Concurrent Programs Under a Context Bound. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 282–298. Springer, Heidelberg (2008)
24 Leroux, J., Sutre, G.: Flat Counter Automata Almost Everywhere! In: Peled, D.A., Tsay, Y.-K. (eds.) ATVA 2005. LNCS, vol. 3707, pp. 489–503. Springer, Heidelberg (2005)
25 Löding, C.: Infinite Graphs Generated by Tree Rewriting, PhD thesis, RWTH Aachen (2003)
26 Lugiez, D., Schnoebelen, P.: Decidable first-order transition logics for PA-processes. Inf. Comput. 203(1), 75–113 (2005)
27 Maidl, M.: The Common Fragment of CTL and LTL. In: FOCS 2000, pp. 643–652 (2000)
28 Mayr, R.: Decidability and Complexity of Model Checking Problems for Infinite-State Systems. PhD thesis, TU-Munich (1998)
29 Morvan, C.: On rational graphs. In: FOSSACS 2000, pp. 252–266 (2000)
30 Qadeer, S., Rehof, J.: Context-Bounded Model Checking of Concurrent Software. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 242–254. Springer, Heidelberg (2005)
31 Rehak, V.: On Extensions of Process Rewrite Systems, PhD thesis, Masaryk University (2007)
32 Sistla, A.P., Clarke, E.M.: The Complexity of Propositional Linear Temporal Logics. J. ACM 32(3), 733–749 (1985)
33 Thomas, W.: Constructing infinite graphs with a decidable MSO-theory. In: Rovan, B., Vojtáš, P. (eds.) MFCS 2003. LNCS, vol. 2747, pp. 113–124. Springer, Heidelberg (2003)

34 To, A.W., Libkin, L.: Recurrent reachability analysis in regular model checking. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS (LNAI), vol. 5330, pp. 198–213. Springer, Heidelberg (2008)
35 Vardi, M.Y., Wolper, P.: Automata-theoretic techniques for modal logics of programs. JCSS 32(2), 183–221 (1986)
36 Verma, K.N., Seidl, H., Schwentick, T.: On the complexity of equational Horn clauses. In: Nieuwenhuis, R. (ed.) CADE 2005. LNCS (LNAI), vol. 3632, pp. 337–352. Springer, Heidelberg (2005)

Toward a Compositional Theory of Leftist Grammars and Transformations[⋆]

P. Chambart and Ph. Schnoebelen

LSV, ENS Cachan, CNRS
61, av. Pdt. Wilson, F-94230 Cachan, France

Abstract. Leftist grammars [Motwani *et al.*, STOC 2000] are special semi-Thue systems where symbols can only insert or erase to their left. We develop a theory of leftist grammars *seen as word transformers* as a tool toward rigorous analyses of their computational power. Our main contributions in this first paper are (1) constructions proving that leftist transformations are closed under compositions and transitive closures, and (2) a proof that bounded reachability is NP-complete even for leftist grammars with acyclic rules.

1 Introduction

Leftist grammars were introduced by Motwani *et al.* to study accessibility and safety in protection systems [7]. In this framework, leftist grammars are used to show that restricted accessibility grammars have decidable accessibility problems (unlike the more general access-matrix model).

Leftist grammars are both surprisingly simple and surprisingly complex. Simplicity comes from the fact that they only allow rules of the form "$a \rightarrow ba$" and "$cd \rightarrow d$" where a symbol inserts, resp. erases, another symbol to its left *while remaining unchanged*. But the combination of insertion and deletion rules makes leftist grammars go beyond context-sensitive grammars, and the decidability result comes with a high complexity-theoretical price [5]. Most of all, what is surprising is that apparently leftist grammars had not been identified as a relevant computational formalism until 2000.

The known facts on leftist grammars and their computational and expressive power are rather scarce. Motwani *et al.* show that it is decidable whether a given word can be derived (accessibility) and whether all derivable words belong to a given regular language (safety) [7]. Jurdziński and Loryś showed that leftist grammars can define languages that are not context-free [6] while leftist grammars restricted to acyclic rules are less expressive since they can only recognize regular languages. Then Jurdziński showed a PSPACE lower bound for accessibility in leftist grammars [4], before improving this to a nonprimitive-recursive lower bound [5].

Jurdziński's results rely on encoding classical computational structures (linear-bounded automata [4] and Ackermann's function [5]) in leftist grammars. Devising such encodings is difficult because leftist grammars are very hard to control. Thus, for computing Ackermann's function, devising the encoding is actually not the hardest part: the

⋆ Work supported by the Agence Nationale de la Recherche, grant ANR-06-SETIN-001.

harder task is to prove that the constructed leftist grammar cannot behave in unexpected ways. In this regard, the published proofs are necessarily incomplete, hard to follow, and hard to fully acknowledge. The final results and intermediary lemmas cannot easily be adapted or reused.

Our Contribution. We develop a compositional theory of leftist grammars and leftist transformations (i.e., operations on strings that are computed by leftist grammars) that provides fundamental tools for the analysis of their computational power. Our main contributions are effective constructions for the composition and the transitive closure of leftist transformations. The correctness proofs for these constructions are based on new definitions (e.g., for greedy derivations) and associated lemmas.

A first application of the compositional theory is given in Section 6 where we prove the NP-completeness of bounded reachability questions, even when restricted to acyclic leftist grammars.

A second application, and the main reason for this paper, is our forthcoming construction proving that leftist grammars can simulate lossy channel systems and "compute" all multiply-recursive transformations and nothing more (based on [3]), thus providing a precise measure of their computational power. Finally, after our introduction of Post's Embedding Problem [1,2], leftist grammars are another basic computational model that will have been shown to capture exactly the notion of multiply-recursive computation.

As further comparison with earlier work, we observe that, of course, the complex constructions in [4,5] are built modularly. However, the modularity is not made fully explicit in these works, the interfacing assumptions are incompletely stated, or are mixed with the details of the constructions, and correctness proofs cannot be given in full.

Outline of the Paper. Basic notations and definitions are recalled in Section 2. Section 3 defines leftist grammars and proves a generalized version of the completeness of greedy derivations. Sections 4 introduces leftist transformers and their sequential compositions. Section 5 specializes on the "simple" transformers that we use in Section 6 for our encoding of 3SAT. Finally Section 7 shows that so-called "anchored" transformers are closed under the transitive closure operation, this in an effective way. For lack of space, several proofs have been omitted in this extended abstract: they can be found in the long version of this paper, freely available at the `arXiv`.

2 Basic Definitions and Notations

Words. We use $x, y, u, v, w, \alpha, \beta, \ldots$ to denote words, i.e., finite strings of symbols taken from some alphabet. Concatenation is denoted multiplicatively with ε (the empty word) as neutral element, and the length of x is denoted $|x|$.

The congruence on words generated by the equivalences $a \approx aa$ (for all symbols a in the alphabet) is called the *stuttering equivalence* and is also denoted $\approx$: every word x has a minimal and canonical stuttering-equivalent x' obtained by repeatedly eliminating symbols in x that are adjacent to a copy of themselves.

We say that x is a *subword* of y, denoted $x \sqsubseteq y$, if x can be obtained by deleting some symbols (an arbitrary number, at arbitrary positions) from y. We further write $x \sqsubseteq_\Sigma y$

when all the symbols deleted from y belong to Σ (NB: we do not require $y \in \Sigma^*$), and let $\sqsupseteq$ denote the inverse relation $\sqsubseteq^{-1}$.

Relations and Relation Algebra. We see a relation R between two sets X and Y as a set of pairs, i.e., some $R \subseteq X \times Y$. We write $x\,R\,y$ rather than $(x,y) \in R$. Two relations R and R' can be composed, denoted multiplicatively with $R.R'$, and defined by $x\ (R.R')\ y \overset{\text{def}}{\Leftrightarrow} \exists z.\big(x\,R\,z \wedge z\,R'\,y\big)$.

The union $R+R'$, also denoted $R \cup R'$, is just the set-theoretic union. R^n is the n-th power $R.R\ldots R$ of R and R^{-1} is the inverse of R: $x\,R^{-1}\,y \overset{\text{def}}{\Leftrightarrow} y\,R\,x$. The transitive closure $\bigcup_{n=1,2,\ldots} R^n$ of R assumes $Y = X$ and is denoted R^+, while its reflexive-transitive closure is $R^+ \cup Id_X$, denoted R^*.

Below we often use notations from relation algebra to state simple equivalences. E.g., we write "$R = R'$" and "$R \subseteq S$" rather than "$x\,R\,y$ iff $x\,R'\,y$" and "$x\,R\,y$ implies $x\,S\,y$". Our proofs often rely on well-known basic laws from relation algebra, like $(R.R')^{-1} = R'^{-1}.R^{-1}$, or $(R+R').R'' = R.R'' + R'.R''$, without explicitly stating them.

3 Leftist Grammars

A *leftist grammar* (an LGr) is a triple $G = (\Sigma, P, g)$ where $\Sigma \cup \{g\} = \{a, b, \ldots\}$ is a finite *alphabet*, $g \notin \Sigma$ is a *final symbol* (also called "*axiom*"), and $P = \{r, \ldots\}$ is a set of production rules that may be *insertion rules* of the form $a \to ba$, and *deletion rules* of the form $cd \to d$. For simplicity, we forbid rules that insert or delete the axiom g (this is no loss of generality [6, Prop. 3]).

Leftist grammars are not context-free (deletions are contextual), or even context-sensitive (deletions are not length-preserving). For our purposes, we consider them as string rewrite systems, more precisely semi-Thue systems. Writing Σ_g for $\Sigma \cup \{g\}$, the rules of P define a 1-step rewrite relation in the standard way: for $u, u' \in \Sigma_g^*$, we write $u \Rightarrow^{r,p} u'$ whenever r is some rule $\alpha \to \beta$, u is some $u_1 \alpha u_2$ with $|u_1\alpha| = p$ and $u' = u_1 \beta u_2$. We often write shortly $u \Rightarrow^{r} u'$, or even $u \Rightarrow u'$, when the position or the rule involved in the step can be left implicit. On the other hand, we sometimes use a subscript, e.g., writing $u \Rightarrow_G v$, when the underlying grammar has to be made explicit.

A *derivation* is a sequence π of consecutive rewrite steps, i.e., is some $u_0 \Rightarrow^{r_1,p_1} u_1 \Rightarrow^{r_2,p_2} u_2 \cdots \Rightarrow^{r_n,p_n} u_n$, often abbreviated as $u_0 \Rightarrow^n u_n$, or even $u_0 \Rightarrow^* u_n$. A subsequence $(u_{i-1} \Rightarrow^{r_i,p_i} u_i)_{i=m,m+1,\ldots,l}$ of π is a *subderivation*. As with all semi-Thue systems, steps (and derivations) are closed under adjunction: if $u \Rightarrow u'$ then $vuw \Rightarrow vu'w$.

Two derivations $\pi_1 = (u \Rightarrow^* u')$ and $\pi_2 = (v \Rightarrow^* v')$ can be concatenated in the obvious way (denoted $\pi_1.\pi_2$) if $u' = v$. They are *equivalent*, denoted $\pi_1 \equiv \pi_2$, if they have same extremities, i.e., if $u = v$ and $u' = v'$.

We say that $u \in \Sigma^*$ is *accepted by* G if there is a derivation of the form $ug \Rightarrow^* g$ and we write $L(G)$ for the set of accepted words, i.e., the language recognized by G.

We say that $I \subseteq \Sigma^*$ is an *invariant* for an LGr $G = (\Sigma, P, g)$ if $u \in I$ and $ug \Rightarrow vg$ entail $v \in I$. Knowing that I is an invariant for G is used in two symmetric ways: (1) from $u \in I$ and $ug \Rightarrow^* vg$ one deduces $v \in I$, and (2) from $ug \Rightarrow^* vg$ and $v \notin I$ one deduces $u \notin I$.

insertion:
deletion:
Σ
g

Fig. 1. Universal type (schematically)

3.1 Graphs and Types for Leftist Grammars

When dealing with LGr's, it is convenient to write insertion rules under the simpler form "$a \rightarrow b$", and deletion rules as "$d \dashrightarrow c$", emphasizing the fact that a (resp. d) is not modified during the insertion of b (resp. the deletion of c) on its left. For $a \in \Sigma_g$, we let $\mathbf{ins}(a) \overset{\text{def}}{=} \{b \mid P \ni (a \rightarrow b)\}$ and $\mathbf{del}(a) \overset{\text{def}}{=} \{b \mid P \ni (a \dashrightarrow b)\}$ denote the set of symbols that can be inserted (respectively, deleted) by a. We write $\mathbf{ins}^+(a)$ for the smallest set that contains b and $\mathbf{ins}^+(b)$ for all $b \in \mathbf{ins}(a)$, while $\mathbf{del}^+(b)$ is defined similarly. We say that a is *inactive* in a LGr if $\mathbf{del}(a) \cup \mathbf{ins}(a) = \varnothing$.

It is often convenient to view LGr's in a graph-theoretical way. Formally, the *graph* of $G = (\Sigma, P, g)$ is the directed graph τ_G having the symbols from Σ_g as vertices and the rules from P as edges (coming in two kinds, insertions and deletions). Furthermore, we often decorate such graphs with extra bookkeeping annotations.

We say that G "*has type* τ" when τ_G is a sub-graph of τ. Thus a "type" is just a restriction on what are the allowed symbols and rules between them. Types are often given schematically, grouping symbols that play a similar role into a single vertex.

For example, Fig. 1 displays schematically the type (parametrized by the alphabet) observed by all LGr's.

3.2 Leftmost, Pure and Eager Derivations

We speak informally of a "letter", say a, when we really mean "an occurrence of the symbol a" (in some word). Furthermore, we follow letters along steps $u \Rightarrow v$, identifying the letters in u and the corresponding letters in v. Hence a "letter" is also a sequence of occurrences in consecutive words along a derivation.

A letter a is a n-th *descendant* of another letter b (in the context of a derivation) if a has been inserted by b (when $n = 1$), or by a $(n-1)$-th descendant of b.

Given a step $u \Rightarrow^{r,p} v$, we say that the p-th letter in u, written $u[p]$, is the *active letter*: the one that inserts, or deletes, a letter to its left. This is often emphasized by writing the step under the form $(u =) u_1 a u_2 \Rightarrow u_1' a u_2 (= v)$ (assuming $u[p] = a$).

A letter is *inert* in a derivation if it is not active *in any step* of the derivation. A set of letters is inert if it only contains inert letters. A derivation is *leftmost* if every step $u_1 a u_2 \Rightarrow u_1' a u_2$ in the derivation is such that u_1 is inert in the rest of the derivation.

A letter is *useful* in a derivation $\pi = (u \Rightarrow^* v)$ if it belongs to u or v, or if it inserts or deletes a useful letter along π. This recursive definition is well-founded: since letters only insert or delete to their left, the "inserts-or-deletes" relation between letters is acyclic. A derivation π is *pure* if all letters in π are useful. Observe that if π is not pure, it necessarily inserts at some step some letter a (called a *useless letter*) that stays inert and will eventually be deleted.

A derivation is *eager* if, informally, deletions occur as soon as possible. Formally, $\pi = (u_0 \Rightarrow^{r_1,p_1} u_1 \cdots \Rightarrow^{r_n,p_n} u_n)$ is not eager if there is some u_{i-1} of the form $w_1 baw_2$ where b is inert in the rest of π and is eventually deleted, where P contains the rule $a \dashrightarrow b$, and where r_i is not a deletion rule.[1]

A derivation is *greedy* if it is leftmost, pure and eager. Our definition generalizes [4, Def. 4], most notably because it also applies to derivations $ug \Rightarrow^* vg$ with nonempty v. Hence a subderivation π' of π is leftmost, eager, pure, or greedy, when π is.

The following proposition generalizes [4, Lemma 7].

Proposition 3.1 (Greedy derivations are sufficient). *Every derivation π has an equivalent greedy derivation π'.*

Proof. With a derivation π of the form $u_0 \Rightarrow^{r_1,p_1} u_1 \Rightarrow^{r_2,p_2} u_2 \cdots \Rightarrow^{r_n,p_n} u_n$, we associate its *measure* $\mu(\pi) \overset{\text{def}}{=} \langle n, p_1, \ldots, p_n \rangle$, a $(n+1)$-tuple of numbers. Measures are linearly ordered with the lexicographic ordering, giving rise to a quasi-ordering, denoted $\leq_\mu$, between derivations. A derivation is called *μ-minimal* if any equivalent derivation has greater or equal measure.

We can now prove Prop. 3.1 along the following lines: first prove that every derivation has a μ-minimal equivalent, then show that μ-minimal derivations are greedy. $\square$

Observe that $\leq_\mu$ is compatible with concatenation of derivations: if $\pi_1 \leq_\mu \pi_2$ then $\pi.\pi_1.\pi' \leq_\mu \pi.\pi_2.\pi'$ when these concatenations are defined. Thus any subderivation of a μ-minimal derivation is μ-minimal, hence also greedy.

μ-minimality is stronger than greediness, and is a powerful and convenient tool for proving Prop. 3.1. However, greediness is easier to reason with since it only involves local properties of derivations, while μ-minimality is "global". These intuitions are reflected by, and explain, the following complexity results.

Theorem 3.2. *1.* Greediness *(deciding whether a given derivation π in the context of a given LGr G is greedy) is in* L.
2. μ-Minimality *(deciding whether it is μ-minimal) is* coNP*-complete, even if we restrict to acyclic LGr's.*

Proof. 1. Being leftmost or eager is easily checked in logspace (i.e., is in L). Checking non-purity can be done by looking for a *last* inserted useless letter, hence is in L too.
2. μ-minimality is obviously in coNP. Hardness is proved as Coro. 6.9 below, as a byproduct of the reduction we use for the NP-hardness of Bounded Reachability. $\square$

4 Leftist Grammars as Transformers

Some leftist grammars are used as computing devices rather than recognizers of words. For this purpose, we require a strict separation between input and output symbols and speak of *leftist transformers*, or shortly LTr's.

[1] Eagerness does not require that r_i deletes b: other deletions are allowed, only insertions are forbidden.

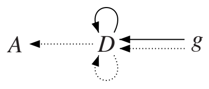


Fig. 2. Type of leftist transformers

4.1 Leftist Transformers

Formally, an LTr is a LGr $G = (\Sigma, P, g)$ where Σ is partitioned as $A \uplus B \uplus C$, and where symbols from A are inactive in P and are not inserted by P (see Fig. 2). This is denoted $G : A \vdash C$. Here A contains the *input symbols*, B the *temporary symbols*, and C the *output symbols*, and G is more conveniently written as $G = (A, B, C, P, g)$. When there is no need to distinguish between temporary and output symbols, we write G under the form $G = (A, D, P, g)$, where $D \overset{\text{def}}{=} B \cup C$ contains the *"working" symbols*,

A consequence of the restrictions imposed on LTr's is the following:

Fact 4.1 *A^*D^* is an invariant in any LTr $G = (A, D, P, g)$.*

With $G = (A, B, C, P, g)$, we associate a *transformation* (a relation between words) $R_G \subseteq A^* \times C^*$ defined by

$$u\, R_G\, v \overset{\text{def}}{\Leftrightarrow} ug \Rightarrow_G^* vg \wedge u \in A^* \wedge v \in C^*$$

and we say that G *realizes* R_G. Finally, a *leftist transformation* is any relation on words realized by some LTr. By necessity, a leftist transformation can only relate words written using disjoint alphabets (this is not contradicted by $\varepsilon\, R_G\, \varepsilon$).

Leftist transformations respect some structural constraints. In this paper we shall use the following properties:

Proposition 4.2 (Closure for leftist transformations). *If $G : A \vdash C$ is a leftist transformer, then $R_G = (\sqsupseteq_A . \approx .R_G. \approx)$.*

4.2 Composition

We say that two leftist transformations $R_1 \subseteq A_1^* \times C_1^*$ and $R_2 \subseteq A_2^* \times C_2^*$ are *chainable* if $C_1 = A_2$ and $A_1 \cap C_2 = \varnothing$. Two LTr's are chainable if they realize chainable transformations.

Theorem 4.3. *The composition $R_1.R_2$ of two chainable leftist transformations is a leftist transformation. Furthermore, one can build effectively a linear-sized LTr realizing $R_1.R_2$ from LTr's realizing R_1 and R_2.*

For a proof, assume $G_1 = (A_1, B_1, C_1, P_1, g)$ and $G_2 = (A_2, B_2, C_2, P_2, g)$ realize R_1 and R_2. Beyond chainability, we assume that $A_1 \cup B_1$ and $B_2 \cup C_2$ are disjoint, which can be ensured by renaming the intermediary symbols in B_1 and B_2. The composed LTr $G_1.G_2$ is given by

$$G_1.G_2 \overset{\text{def}}{=} (A_1, B_1 \cup C_1 \cup B_2, C_2, P_1 \cup P_2, g).$$

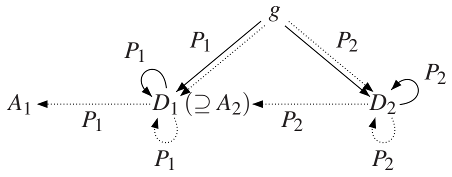
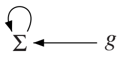


Fig. 3. The type of $G_1.G_2$



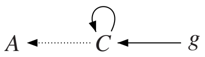


Fig. 4. Types of insertion grammars (left) and simple leftist transformers (right)

This is indeed a LTr from A_1 to C_2. See Fig. 3 for a schematics of its type. Since $G_1.G_2$ has all rules from G_1 and G_2 it is clear that $(\Rightarrow_{G_1} + \Rightarrow_{G_2}) \subseteq \Rightarrow_G$, from which we deduce $R_{G_1}.R_{G_2} \subseteq R_{G_1.G_2}$. Furthermore, the inclusion in the other direction also holds:

Lemma 4.4 (Composition Lemma). $R_{G_1.G_2} = R_{G_1}.R_{G_2}$.

Remark 4.5 (Associativity). The composition $(G_1.G_2).G_3$ is well-defined if and only if $G_1.(G_2.G_3)$ is. Furthermore, the two expressions denote exactly the same result. $\square$

5 Simple Leftist Transformations

As a tool for Sections 6 and 7, we now introduce and study restricted families of leftist grammars (and transformers) where deletion rules are forbidden (resp., only allowed on A).

An *insertion grammar* is a LGr $G = (\Sigma, P, g)$ where P only contain insertion rules. See Fig. 4 for a graphic definition. For an arbitrary leftist grammar G, we denote with $G^{\mathbf{ins}}$ the insertion grammar obtained from G by keeping only the insertion rules.

The *insertion relation* $I_G \subseteq \Sigma^* \times \Sigma^*$ associated with an insertion grammar $G = (\Sigma, P, g)$ is defined by $u\ I_G\ v \overset{\text{def}}{\Leftrightarrow} ug \Rightarrow_G^* vg$. Obviously, $I_G \subseteq \sqsubseteq_\Sigma$. Observe that I_G is not necessarily a leftist transformation since it does not require any separation between input and output symbols.

A *simple* leftist transformer is an LTr $G = (A, B, C, P, g)$ where $B = \varnothing$ and where no rule in P erases symbols from C.

See Fig. 4 for a graphic definition. We give, without proof, an immediate consequence of the definition:

Lemma 5.1. *Let $G = (A, \varnothing, C, P, g)$ be a simple LTr and assume $ug \Rightarrow_G^k vg$ for some $u \in A^*$ and $v \in C^*$. Then $k = |u| + |v|$.*

Given a simple LTr $G = (A, \varnothing, C, P, g)$ and two words $u = a_1 \cdots a_n \in A^*$ and $v = c_1 \cdots c_m \in C^*$, we say that a non-decreasing map $h : \{1, \ldots, n\} \to \{1, \ldots, m\}$ is a *G-witness* for u and v if P contains the rules $c_{h(i)} \dashrightarrow a_i$ and $c_{j+1} \to c_j$ (for all $i = 1, \ldots, n$ and $j = 1, \ldots, m$, with the convention that $c_{m+1} = g$). Finally, we write $u \nabla_G v$ when such a G-witness exists. Clearly, $\nabla_G \subseteq R_G$. Indeed, when G is a simple transformer, ∇_G can be used as a restricted version of R_G that is easier to control and reason about.

Lemma 5.2. *Let $G = (A, \varnothing, C, P, g)$ be a simple LTr. Then $R_G = \nabla_G . I_{G\mathbf{ins}}$.*

Combining Lemma 5.2 with $Id_{C^*} \subseteq I_{G\mathbf{ins}} \subseteq \sqsubseteq_C$, we obtain the following weaker but simpler statement.

Corollary 5.3. *Let $G = (A, \varnothing, C, P, g)$ be a simple LTr. Then $\nabla_G \subseteq R_G \subseteq \nabla_G . \sqsubseteq_C$.*

5.1 Union of Simple Leftist Transformers

We now consider the combination of two simple LTr's $G_1 = (A, \varnothing, C_1, P_1, g)$ and $G_2 = (A, \varnothing, C_2, P_2, g)$ that transform from a same A to disjoint output alphabets, i.e., with $C_1 \cap C_2 = \varnothing$. We define their *union* with $G_1 + G_2 \overset{\text{def}}{=} (A, \varnothing, C_1 \cup C_2, P_1 \cup P_2, g)$. This is clearly a simple LTr with $(R_{G_1} + R_{G_2}) \subseteq R_{G_1+G_2}$. It further satisfies:

Lemma 5.4. *If $u\, R_{G_1+G_2}\, v$ then $u\, (R_{G_1} + R_{G_2})\, v'$ for some $v' \sqsubseteq v$.*

Proof. Assume $u\, R_{G_1+G_2}\, v$. With Cor. 5.3, we obtain $u\, \nabla_{G_1+G_2}\, v'$ for some $v' = c_1 \cdots c_m \sqsubseteq v$. Hence $G_1 + G_2$ has insertion rules $c_{j+1} \rightarrow c_j$ for all $j = 1, \ldots, m$, and deletion rules of the form $c_{h(i)} \dashrightarrow u[i]$. Since C_1 and C_2 are disjoint, either all these rules are in G_1 (and $u\, \nabla_{G_1}\, v'$), or they are all in G_2 (and $u\, \nabla_{G_2}\, v'$). Hence $u\, (R_{G_1} + R_{G_2})\, v'$. □

6 Encoding 3SAT with Acyclic Leftist Transformers

This section proves the following result.

Theorem 6.1. Bounded Reachability *and* Exact Bounded Reachability *in leftist grammars are* NP*-complete, even when restricting to acyclic grammars.*

(Exact) Bounded Reachability is the question whether there exists a n-step derivation $u \Rightarrow^n v$ (respectively, a derivation $u \Rightarrow^{\leq n} v$ of non-exact length at most n) between given u and v. These questions are among the simplest reachability questions and, since we consider that the input n is given in unary,[2] they are obviously in NP for leftist grammars (and all semi-Thue systems).

Consequently, our contribution in this paper is the NP-hardness part. This is proved by encoding 3SAT instances in leftist grammars where reaching a given final v amounts to guessing a valuation that satisfies the formula. While the idea of the reduction is easy to grasp, the technicalities involved are heavy and it would be difficult to really prove the correctnessof the reduction without relying on a compositional framework like the one we develop in this paper. It is indeed very tempting to "prove" it by just running an example.

[2] It is natural to begin with this assumption when considering fundamental aspects of reachability since writing n more succinctly would blur the complexity-theoretical picture.

Rather than adopting this easy way, we shall describe the reduction as a composition of simple leftist transformers and use our composition theorems to break down the correctness proof in smaller, manageable parts. Once the ideas underlying the reduction are grasped, a good deal of the reasoning is of the type-checking kind: verifying that the conditions required for composing transformers are met.

Throughout this section we assume a generic 3SAT instance $\Phi = \bigwedge_{i=1}^{m} C_i$ with m 3-clauses on n Boolean variables in $X = \{x_1, \ldots, x_n\}$. Each clause has the form $C_i = \bigvee_{k=1}^{3} \varepsilon_{i,k} x_{i,k}$ for some polarity $\varepsilon_{i,k} \in \{+,-\}$ and $x_{i,k} \in X$. (There are two additional assumptions on Φ that we postpone until the proof of Coro. 6.5 for clarity.) We use standard model-theoretical notation like $\models \Phi$ (validity), or $\sigma \models \Phi$ (entailment) when σ is a Boolean formula or a Boolean valuation of some variables.

We write $\sigma[x \mapsto b]$ for the extension of a valuation σ with (x,b), assuming $x \notin Dom(\sigma)$. Finally, for a valuation $\theta : X \to \{\top, \bot\}$ and some $j = 0, \ldots, n$, we write θ_j to denote the restriction $\theta_{|\{x_1,\ldots,x_j\}}$ of θ on the first j variables.

6.1 Associating an LTr G_Φ with Φ

For the encoding, we use an alphabet $\Sigma = \{T_i^j, U_i^j, T'^j_i, U'^j_i \mid i = 1, \ldots, m \wedge j = 0, \ldots, n\}$, i.e., $4(n+1)$ symbols for each clause. The choice of the symbols is that a U means "*Undetermined*" and a T means "*True*", or determined to be valid.

For $j = 0, \ldots, n$, let $V_j \stackrel{\text{def}}{=} \{U_1^j, \ldots, U_m^j, T_1^j, \ldots, T_m^j\}$, $V'_j \stackrel{\text{def}}{=} \{U'^j_1, \ldots, U'^j_m, T'^j_1, \ldots, T'^j_m\}$, and $W_j \stackrel{\text{def}}{=} V_j \cup V'_j$, so that Σ is partitioned in levels with $\Sigma = \bigcup_{j=0}^{n} W_j$. With each $x_j \in X$ we associate two intermediary LTr's:

$$G_j^\top \stackrel{\text{def}}{=} (W_{j-1}, \varnothing, V_j, P_j, g), \qquad\qquad G_j^\bot \stackrel{\text{def}}{=} (W_{j-1}, \varnothing, V'_j, P'_j, g)$$

with sets of rules P_j and P'_j. The rules for $G_j^\top$ are given in Fig. 5: some deletion rules are conditional, depending on whether x_j appears in the clauses $C_1, \ldots, C_m$. The rules for $G_j^\bot$ are obtained by switching primed and unprimed symbols, and by having conditional rules based on whether $\neg x_j$ appears in the C_i's. One easily checks that $G_j^\top$ and $G_j^\bot$ are indeed simple transformers. They have same inputs and disjoint outputs so that the union $(G_j^\top + G_j^\bot) : W_{j-1} \vdash W_j$ is well-defined. Hence the following composition is well-formed:

$$G_\Phi \stackrel{\text{def}}{=} (G_1^\top + G_1^\bot).(G_2^\top + G_2^\bot) \cdots (G_n^\top + G_n^\bot).$$

We conclude the definition of G_Φ with an intuitive explanation of the idea behind the reduction. G_Φ operates on the word $u_0 = U_1^0 \cdots U_m^0$ where each U_i^0 stands for "*the validity of clause C_i is undetermined at step* 0 *(i.e., at the beginning)*". At step j, $G_j^\top + G_j^\bot$ picks a valuation for x_j: $G_j^\top$ picks "$x_j = \top$" while $G_j^\bot$ picks "$x_j = \bot$". This transforms U_i^{j-1} into U_i^j, and T_i^{j-1} into T_i^j, moving them to the next level. Furthermore, an undetermined U_i^{j-1} can be transformed into T_i^j if C_i is satisfied by x_j. In addition, and because $G_j^\top$ and $G_j^\bot$ must have disjoint output alphabets, the symbols in the V_j's come in two copies (hence the V'_j's) that behave identically when they are input in the transformer for the next step.

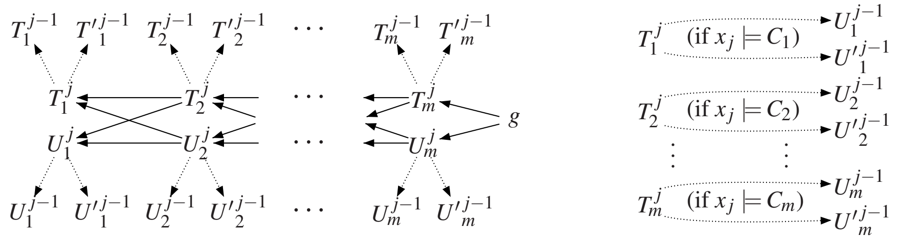


Fig. 5. P_j, the rules for $G_j^{\top}$: Fixed part on left, conditional part on right

The reduction is concluded with the following claim that we prove by combining Corollaries 6.5 and 6.8 below.

$$\begin{aligned}
\Phi \text{ is satisfiable iff } & U_1^0 U_2^0 \cdots U_m^0 g \Rightarrow_{G_\Phi}^{2mn} T_1^n T_2^n \cdots T_m^n g \\
\text{iff } & U_1^0 U_2^0 \cdots U_m^0 g \Rightarrow_{G_\Phi}^{\leq 2mn} T_1^n T_2^n \cdots T_m^n g \qquad \text{(Correctness)} \\
\text{iff } & U_1^0 U_2^0 \cdots U_m^0 g \Rightarrow_{G_\Phi}^{*} T_1^n T_2^n \cdots T_m^n g.
\end{aligned}$$

Observe finally that G_Φ is an acyclic grammar in the sense of [6], that is to say, its rules define an acyclic "*may-act-upon*" relation between symbols. Such grammars are much weaker than general LGr's since, e.g., languages recognized by LGr's with acyclic deletion rules (and arbitrary insertion rules) are regular [6].

Remark 6.2. The construction of G_Φ from Φ, mostly amounting to copying operations for the $G_j^{\top}$'s and $G_j^{\perp}$'s, to type-checking and sets-joining operations for the composition of the LTr's, can be carried out in logarithmic space. □

6.2 Correctness of the Reduction

We say that a word u is *j-clean* if it has exactly m symbols and if $u[i] \in \{T_i^j, T'^j_i, U_i^j, U'^j_i\}$ for all $i = 1, \ldots, m$. It is $\top$-*homogeneous* (resp. $\perp$-*homogeneous*) if it does not contain any (resp., only contains) primed symbols.

Let $0 \leq j \leq n$ and θ_j be a Boolean valuation of $x_1, \ldots, x_j$: we say that a j-clean u *respects* (Φ under) θ_j when, for all $i = 1, \ldots, m$, $\theta_j \models C_i$ when $u[i]$ is determined (i.e., $\in T_i^j + T'^j_i$). Finally u *codes* (Φ under) θ_j if additionally each $u[i]$ is determined when $\theta_j \models C_i$. Thus, a word u that codes some θ_j exactly lists (via determined symbols) the clauses of Φ made valid by θ_j, and the only flexibility in u is in using the primed or the unprimed copy of the symbols. Hence there is only one j-clean u coding θ_j that is $\top$-homogeneous, and only one that is $\perp$-homogeneous. If u respects θ_j instead of coding it, more latitude exists since symbols may be undetermined even if the corresponding clause is valid under θ_j.

Assume that, for some $j \in \{1, \ldots, n\}$, u_{j-1} codes θ_{j-1} and u_j codes θ_j. Write b for $\theta(x_j)$ (NB: $b \in \{\top, \perp\}$).

Lemma 6.3. *If u_j is b-homogeneous then $u_{j-1} \nabla_{G_j^b} u_j$.*

Proof. Let $h \overset{\text{def}}{=} Id_{\{1,\ldots,m\}}$. We claim that h is a G_j^b-witness for u_{j-1} and u_j, i.e., that G_j^b contains the required insertion and deletion rules.

Insertions. G_j^b has all insertion rules $g \longrightarrow u_j[m] \longrightarrow u_j[m-1] \longrightarrow \ldots \longrightarrow u_j[1]$ (leftmost rules in Fig. 5) since u_j is b-homogeneous.

Deletions. G_j^b has all deletion rules $u_j[i] \dashrightarrow u_{j-1}[i]$. Firstly, both undetermined symbols U_j^i and U'^i_j may delete their counterparts U_{j-1}^i and U'^i_{j-1}, and similarly for the determined symbols (the unconditional deletion rules in Fig. 5). This is used if C_i is not more valid under θ_j than under θ_{j-1}. Secondly, if C_i is valid under θ_j but not under θ_{j-1}, then $x_j \models C_i$ (or $\neg x_j \models C_i$, depending on b) and the conditional rules in Fig. 5 allow a determined T_i^j (or T'^j_i depending on b) to delete U_i^{j-1} or U'^{j-1}_i. □

Lemma 6.4. *If u_j is b-homogeneous, then $u_{j-1}g \Rightarrow^{2m}_{G_j^b} u_j g$.*

Proof. From $u_{j-1} \nabla_{G_j^b} u_j$ (Lemma 6.3) we deduce $u_{j-1} R_{G_j^b} u_j$, i.e., $u_{j-1}g \Rightarrow^*_{G_j^b} u_j g$, by Lemma 5.2, and then $u_{j-1}g \Rightarrow^{2m}_{G_j^b} u_j g$ by Lemma 5.1. □

Corollary 6.5. *If Φ is satisfiable, then $U_1^0 \cdots U_m^0 g \Rightarrow^{2mn}_{G_\Phi} T_1^n \cdots T_m^n g$.*

Proof. Since Φ is satisfiable, $\theta \models \Phi$ for some valuation θ. For $j = 1,\ldots,m$, we write b_j for $\theta(x_j)$ and let u_j be the only j-clean b_j-homogeneous word that codes for θ_j.

We now make two assumptions on Φ that are no loss of generality. First we require that no clause C_i contains both a literal and its negation, hence no C_i is tautologically valid. Then $u_0 \overset{\text{def}}{=} U_1^0 \cdots U_m^0$ codes the empty valuation θ_0. Second, we require that Φ is only satisfiable with $b_n = \top$ (which can be easily ensured by adding a few extra variables). Then necessarily $u_n = T_1^n \cdots T_m^n$.

Lemma 6.4 gives $u_0 g \Rightarrow^{2m}_{G_1^{b_1}} u_1 g \Rightarrow^{2m}_{G_2^{b_2}} u_2 g \cdots \Rightarrow^{2m}_{G_n^{b_n}} u_n g$. Since $\Rightarrow_{G_j^b} \subseteq \Rightarrow_{G_j} \subseteq \Rightarrow_{G_\Phi}$ for all b and j, we deduce $u_0 g \Rightarrow^{2mn}_{G_\Phi} u_n g$ as claimed. □

Fix some θ, some $j \in \{1,\ldots,n\}$ and let $b = \theta(x_j)$.

Lemma 6.6. *If u respects θ_{j-1} and $u \nabla_{G_j^b} v$, then v respects θ_j.*

Proof. Write l for $|v|$. From $u \nabla_{G_j^b} v$ (witnessed by some h) we deduce that G_j^b has insertion rules $g \longrightarrow v[l] \longrightarrow v[l-1] \longrightarrow \ldots \longrightarrow v[1]$. Inspecting Fig. 5, we conclude that necessarily $l \leq m$. Since deletion rules $v[h(i)] \dashrightarrow u[i]$ are required for all $i = 1,\ldots,m$, we further see from Fig. 5 that h is injective, so that $l \geq m$. Finally $l = m$, $h = Id_{\{1,\ldots,m\}}$, v is j-clean and b-homogeneous.

Now, knowing that G_j^b contains the rules $v[i] \dashrightarrow u[i]$, we show that v respects θ_j. Suppose, by way of contradiction, that it does not. Thus there is some $i \in \{1,\ldots,m\}$ with $v[i] = T_i^j$ (assuming $b = \top$ w.l.o.g.) while $\theta_j \not\models C_i$ (so that $\theta_{j-1} \not\models C_i$). From $\theta_j \not\models C_i$ we deduce that $x_j \not\models C_i$. Hence G_j^b does not have the conditional rules $T_i^j \dashrightarrow U_i^{j-1}$ and $T_i^j \dashrightarrow U'^{j-1}_i$. Thus $u[i] \notin \{U_i^{j-1}, U'^{j-1}_i\}$. But then u does not respect θ_{j-1}, contradicting our assumption. □

We immediately deduce:

Lemma 6.7. *If $x\ R_{G^b_j}\ y$ and there is some $u \sqsubseteq x$ that respects θ_{j-1}, then there is some $v \sqsubseteq y$ that respects θ_j.*

Proof. From the Closure Property 4.2, we get $u\ R_{G^b_j}\ y$. Then, from $R_{G^b_j} \subseteq \nabla_{G^b_j}.\sqsubseteq$ (Coro. 5.3) we deduce $u\ \nabla_{G^b_j}\ v$ for some $v \sqsubseteq y$. Now v respects θ_j thanks to Lemma 6.6. □

Corollary 6.8. *If $U^0_1 \cdots U^0_m g \Rightarrow^*_{G_\Phi} T^n_1 \cdots T^n_m g$, then Φ is satisfiable.*

Proof. Write u_0 for $U^0_1 \cdots U^0_m$ and u_n for $T^n_1 \cdots T^n_m$. From the definition of G_Φ and the Composition Lemma 4.4, we deduce that there exist some words $u_1, \ldots, u_{n-1}$ such that $u_{j-1}\ R_{G^\top_j + G^\perp_j}\ u_j$ for all $j = 1, \ldots, n$.

With Lemma 5.4, we further deduce that there exist some words $u'_1, \ldots, u'_n$ and Boolean values $b_1, \ldots, b_n$ such that $u'_j \sqsubseteq u_j$ and $u_{j-1}\ R_{G^{b_j}_j}\ u'_j$ for all $j = 1, \ldots, n$. Hence also $u'_{j-1}\ R_{G^{b_j}_j}\ u'_j$ by Prop. 4.2 (and letting $u'_0 = u_0$).

Write θ for $[x_1 \mapsto b_1, \ldots, x_n \mapsto b_n]$. With Lemma 6.7, induction on j, and since u'_0 respects θ_0, we further deduce that there exists some words $u''_1, \ldots, u''_n$ such that, for all $j = 1, \ldots, n$, $u''_j \sqsubseteq u'_j$ and u''_j respects θ_j. From $|u''_n| = m$ (it respects θ) and $u''_n \sqsubseteq u_n$, we deduce that $u''_n = u_n$. Finally, $\theta \models \Phi$ since u''_n respects θ and $u''_n = u_n = T^n_1 \cdots T^n_m$. □

Corollary 6.9. *μ-Minimality of a derivation is* coNP*-hard.*

Proof (Sketch). We define G'_Φ by taking G_Φ, adding k extra symbols $a_1, \ldots, a_k$, and adding the following two sets of rules:
(1) all $a_{i-1} \rightarrow a_i$ and $a_{i-1} \dashrightarrow a_i$ for $i = 1, \ldots, k$ (with the convention that a_0 is T^n_1);
(2) all $a_k \dashrightarrow U^0_i$ for $i = 1, \ldots, m$.

Observe that G'_Φ is acyclic. It has a derivation $\pi : U^0_1 \cdots U^0_m g \Rightarrow^{2m+2k} T^n_1 \cdots T^n_m g$ of the following form:

$$\begin{aligned} U^0_1 \cdots U^0_m g &\Rightarrow^m U^0_1 \cdots U^0_m T^n_1 \cdots T^n_m g \Rightarrow^k U^0_1 \cdots U^0_m a_k a_{k-1} \cdots a_1 T^n_1 \cdots T^n_m g \\ &\Rightarrow^m a_k a_{k-1} \cdots a_1 T^n_1 \cdots T^n_m g \Rightarrow^k T^n_1 \cdots T^n_m g. \end{aligned}$$

This derivation uses the extra symbols to bypass the normal behaviour of G_ϕ. If k is large enough, i.e., $k > m(n-1)$, π is μ-minimal if, and only if, Φ is not satisfiable. □

7 Anchored Leftist Transformers and Their Transitive Closure

When $b_1, b_2 \in B$ are two different working symbols, and (A, B, C, P, g) is a LTr, we call $G = (A, B, C, b_1, b_2, P, g)$ an *anchored* LTr, or shorly an ALTr. With an ALTr G we associate an *anchored tranformation* $S_G \subseteq A^* \times C^*$ defined by

$$u\ S_G\ v \overset{\text{def}}{\Leftrightarrow} b_1 u g \Rightarrow^*_G b_2 v g.$$

Here the *anchors* b_1, b_2 are used to control what happens at the left-hand end of transformed words. Mostly, they ensure that the derivation $b_1 ug \Rightarrow^* b_2 vg$ goes all the way to the left and erases b_1 rather than stopping earlier. One intuitive way of seeing S_G is that it is a variant of R_G *restricted to derivations that replace the anchors*.

A first difficulty for building the transitive closure of an anchored transformation $S_G \subseteq A^* \times C^*$ is that the input and output sets are disjoint (a requirement that allowed the developments of Sections 4 and 5). To circumvent this, we assume w.l.o.g. that A and C are two different copies of a same set, equipped with a bijective renaming $\bar{h} : C^* \to A^*$. Then, the closure $S_G.(\bar{h}.S_G)^*$ behaves like we would want S_G^+ to behave.

For the rest of this section, we assume h is a bijection between C and A. W.l.o.g., we write A and C under the forms $A = \{a_1, \dots, a_n\}$ and $C = \{c_1, \dots, c_n\}$ so that $h(c_i) = a_i$ for all $i = 1, \dots, n$. Then h is lifted as a (bijective) morphism $\bar{h} : C^* \to A^*$ that we sometimes see as a relation between words.

The exact statement we prove in this section is the following:

Theorem 7.1 (Transitive Closure). *Let $G : A \vdash C$ be an ALTr such that $S_G = S_G.\sqsubseteq_C$. Then there exists an ALTr $G^{(+)} : A \vdash C$ such that $S_{G^{(+)}} = S_G.(\bar{h}.S_G)^*$. Furthermore, it is possible to build $G^{(+)}$ from G using only logarithmic space.*

Let $b_1, b_2 \notin A \cup C$. The ALTr $\mathsf{R}_{b_2,b_1} \overset{\text{def}}{=} (C, b_2, b_1, A, P_{\mathsf{R}}, g)$ with

$$P_{\mathsf{R}} \overset{\text{def}}{=} \left\{ \begin{array}{c} g \longrightarrow a_i, a_i \longrightarrow a_j, a_i \longrightarrow b_1 \\ a_i \dashrightarrow c_i, b_1 \dashrightarrow b_2 \end{array} \middle| \text{ for all } i, j = 1, \dots, n \right\}$$

is called a *renamer (of C to A)*, and often shortly written R. Observe that $\mathsf{R} : C \vdash A$ is indeed an ALTr. It further satisfies $S_{\mathsf{R}} = \approx . \sqsubseteq . \bar{h}$.

We shall now glue an ALTr $G : A \vdash C$ with the renamer $\mathsf{R} : C \vdash A$ into some larger LGr H. But before this can be done we need to put some wrapping control on G (and on R) that will let us track what comes from G inside H's derivations.

Formally, given an ALTr $G = (A, B, C, b_1, b_2, P, g)$ and two new anchor symbols $\square_1, \square_2 \notin \Sigma_g$, we let $\Sigma_\square \overset{\text{def}}{=} \{\square_1, \square_2\}$ and define a new ALTr $F_{G,\square_1,\square_2}$ (or shortly just F_G) for "*wrapping G with $\square_1, \square_2$*", and given by $F_{G,\square_1,\square_2} \overset{\text{def}}{=} (\overline{A}, \overline{B}, \overline{C}, \square_1, \square_2, P', g)$ where

– $\overline{A} \overset{\text{def}}{=} A \cup A' \cup \{b_1, b'_1\}$, A', b'_1 being a copy of A, b_1,

– $\overline{B} \overset{\text{def}}{=} \{\square_1, \square_2\} \cup B \smallsetminus \{b_1\}$,

– $\overline{C} \overset{\text{def}}{=} C \cup \{b_2\} \cup C' \cup B' \smallsetminus \{b'_1\}$, B' and C' being copies of B and C.

Finally, let $D \overset{\text{def}}{=} C \cup B$ and $D' \overset{\text{def}}{=} C' \cup B'$. (The copies are denoted by priming the original symbols, and a primed set like $A' = \{a' \mid a \in A\}$ is just the set of corresponding primed symbols.) The rules in P' are derived from the rules of P in the following way.

kept: P' retains all rules of P that do not erase a letter in $A \cup \{b_1\}$,
replace: P' has a rule $d' \dashrightarrow a$ for each rule $d \dashrightarrow a$ in P that erases a letter in $A \cup \{b_1\}$,
mirror: P' has a rule $d \longrightarrow d'$ for each $d \in D$,
clean: P' has all rules $d' \dashrightarrow e'$ and $\square_2 \dashrightarrow a'$ for $d', e' \in D' \smallsetminus \{b'_1\}$ and $a' \in A' \cup \{b'_1\}$,
b-rules: P' has the rules $\square_2 \dashrightarrow \square_1$ and all rules $d' \longrightarrow \square_2$ for $d' \in D' \smallsetminus \{b'_1\}$.

We now relate the derivations in G and the derivations in F_G. For this, assume $u \in (A + b_1)^*$ and $v \in (C + b_2)^+$.

Lemma 7.2. *1. If $u.g \Rightarrow_G^+ v.g$ then for all words $\alpha \in (A' + b_1')^*$ there exists a symbol $\beta \in C' \cup \{b_2'\}$ such that $\square_1.\alpha.u.g \Rightarrow_{F_G}^+ \square_1.\alpha.\beta.v.g \Rightarrow_{F_G}^+ \square_2.\beta.v.g$.*
2. Reciprocally, for all $\alpha \in (A' + b_1')^*$, for all $\beta \in (C' + b_2')^+$ if $\square_1.\alpha.u.g \Rightarrow_{F_G}^+ \square_2.\beta.v.g$ then $u.g \Rightarrow_G^+ v.g$.

Thus we can relate anchored derivations in F_G with anchored derivations in G via:

Corollary 7.3. *Let $u \in (A + b_1)^*$ and $v \in (C + b_2)^+$. Then $b_1.u.g \Rightarrow_G^+ b_2.v.g$ if and only if there exists $\beta \in (C' \cup \{b_2'\})$ such that $\square_1.\alpha.b_1.u.g \Rightarrow_{F_G}^+ \square_2.\beta.b_2.v.g$. In other words, $u\ S_G\ v$ iff $\alpha.b_1.u\ S_{F_G}\ \beta.b_2.v$ for some $\beta \in (C' \cup \{b_2'\})$.*

We may now glue the wrapped versions of G and its associated R. Recall that $F_G = (\overline{A}, \overline{B}, \overline{C}, \square_1, \square_2, P', g)$. We denote the set of new symbols with $\Sigma \overset{\text{def}}{=} \overline{A} \cup \overline{B} \cup \overline{C}$ and observe that F_{R} (short for $F_{\mathsf{R}_{b_2,b_1}, \square_2, \square_1}$), being some $(C \cup C' \cup \{b_2, b_2'\}, \Sigma_\square, \overline{A}, \square_2, \square_1, P_{\mathsf{R}}', g)$, does not use more symbols. Let $H \overset{\text{def}}{=} (\Sigma, P_H, g)$ be the LGr such that and $P_H = P' \cup P_{\mathsf{R}}'$. Essentially, H is a union of the two wrapping ALTr's. Note that H is *not a LTr* since it does not respect any distinction between input, intermediary, and output symbols.

Lemma 7.4. *Let $\alpha, \beta \in A'^+$ and $u, v \in A^*$. If $\square_1.\alpha.u.g \Rightarrow_H^* \square_1.\beta.v.g$ and $S_G = (\sqsubseteq_A .S_G. \sqsubseteq_C)$ then $u \sqsubseteq_A .(S_G.\bar{h})^*\ v$.*

We now extend H to turn it into an ALTr $H' : \dot{A} \vdash A \cup A'$, introducing again new copies, denoted $\dot{a}$, ..., of previously used symbols and writing $\dot{u} = \dot{a}_1 \dot{a}_2 \dots \dot{a}_n$ for the dotted copy of some $u = a_1 a_2 \dots a_n$. Formally,

$$H' \overset{\text{def}}{=} (\dot{A}, B \cup B' \cup C \cup C' \cup \{\square_1, \square_2, \dot{\square}_1, \dot{\square}_2\}, A \cup A', \dot{\square}_1, \dot{\square}_2, P'', g)$$

where P'' extends P_H by the rules $\dot{\square}_2 \dashrightarrow \dot{\square}_1$, $\square_1 \rightarrow \dot{\square}_2$, and all $a \dashrightarrow \dot{a}$ for $a \in A$.

The anchored transformation $S_{H'}$ computed by H' is captured by the following:

Lemma 7.5. *Let $u, v \in A^*$. Then $\dot{u}\ S_{H'}\ \square_1.\beta.v$ for some $\beta \in A'^+$ iff $u\ [\bar{h}. \sqsubseteq_A .(S_G.\bar{h})^*]\ v$.*

We are nearly done. There only remains to compose H' with a LTr that checks for the presence of $\square_1.\beta$ (and then erases it). For this last step, we shall use further dotted copies $\ddot{\Sigma}$, $\dddot{\Sigma}$, ..., of the previously used symbols.

Formally, we define two new ALTr's T_1 and T_2: see full version. The rules of T_1 ensure that it satisfies

$$u\ S_{T_1}\ v \text{ iff } u = \square_1.\alpha.b_1.u' \text{ and } \ddot{u}\ I_{T_1}^{\text{ins}}\ v. \qquad (T_1\text{-spec})$$

Regarding T_2, let $u \in (\ddot{A} \cup \ddot{A}' \cup \{\ddot{b}_1, \ddot{b}_1'\})^*$ and $v \in \dddot{A}^*$. If $\ddot{u}'$ is the largest subword of u such that $u' \in A^*$, then

$$u\ S_{T_2}\ v \text{ iff } \dddot{u}' \sqsubseteq_{\dddot{A}}\ v. \qquad (T_2\text{-spec})$$

Combining (T_1-spec) and (T_2-spec) we obtain

$$u\ S_{T_1}.S_{T_2}\ v \text{ iff } u = \square_1.\alpha.b_1.u' \text{ and } \dddot{u}' \sqsubseteq_{\dddot{A}}\ v.$$

Composing these LTr's as $H'.T_1.T_2$ yields a resulting $G^{(+)} : \dot{A} \vdash \dddot{A}$, which, up to a bijective change of symbols, is what we need to build to prove Theorem 7.1.

8 Conclusion

In this paper we introduce a notion of transformations computed by leftist grammars and define constructions showing how these transformations are effectively closed under sequential composition and transitive closure.

These operations require that some "typing" assumptions are satisfied (e.g., we only know how to build a transitive closure on leftist transformers that are "*anchored*") which may be seen as a lack of elegance and generality of the theory, but which we see as an indication that leftist grammars are very hard to control and reason about.

Anyway, the restrictive assumptions are not a problem for our purposes: we intend to rely on the compositional foundations for building, in a modular way, complex leftist grammars that are able to simulate lossy channel systems. Here the modularity is essential not so much for *building* complex grammars. Rather, it is essential for proving their correctness by a divide-and-conquer approach, in the way we proved the correctness of our encoding of 3SAT instances in Section 6.

As another direction for future work, we would like to mention that the proof that accessibility is decidable for LGr's (see [7]) has to be fixed and completed.

Acknowledgements. Sylvain Schmitz helped tremendously with his numerous remarks and suggestions.

References

1. Chambart, P., Schnoebelen, P.: Post embedding problem is not primitive recursive, with applications to channel systems. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 265–276. Springer, Heidelberg (2007)
2. Chambart, P., Schnoebelen, P.: The ω-regular Post embedding problem. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 97–111. Springer, Heidelberg (2008)
3. Chambart, P., Schnoebelen, P.: The ordinal recursive complexity of lossy channel systems. In: Proc. LICS 2008, pp. 205–216. IEEE Comp. Soc. Press, Los Alamitos (2008)
4. Jurdziński, T.: On complexity of grammars related to the safety problem. Theoretical Computer Science 389(1-2), 56–72 (2007)
5. Jurdziński, T.: Leftist grammars are nonprimitive recursive. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 51–62. Springer, Heidelberg (2008)
6. Jurdziński, T., Loryś, K.: Leftist grammars and the Chomsky hierarchy. Mathematical Systems Theory 41(2), 233–256 (2007)
7. Motwani, R., Panigrahy, R., Saraswat, V.A., Venkatasubramanian, S.: On the decidability of accessibility. In: Proc. STOC 2000, pp. 306–315. ACM Press, New York (2000)

Degrees of Lookahead in Regular Infinite Games

Michael Holtmann[1], Łukasz Kaiser[2], and Wolfgang Thomas[1]

[1] RWTH Aachen, Lehrstuhl für Informatik 7, D-52056 Aachen
{holtmann,thomas}@automata.rwth-aachen.de

[2] RWTH Aachen, Mathematische Grundlagen der Informatik, D-52056 Aachen
kaiser@logic.rwth-aachen.de

Abstract. We study variants of regular infinite games where the strict alternation of moves between the two players is subject to modifications. The second player may postpone a move for a finite number of steps, or, in other words, exploit in his strategy some lookahead on the moves of the opponent. This captures situations in distributed systems, e.g. when buffers are present in communication or when signal transmission between components is deferred. We distinguish strategies with different degrees of lookahead, among them being the continuous and the bounded lookahead strategies. In the first case the lookahead is of finite possibly unbounded size, whereas in the second case it is of bounded size. We show that for regular infinite games the solvability by continuous strategies is decidable, and that a continuous strategy can always be reduced to one of bounded lookahead. Moreover, this lookahead is at most doubly exponential in the size of the parity automaton recognizing the winning condition. We also show that the result fails for non-regular games where the winning condition is given by a context-free ω-language.

1 Introduction

The algorithmic theory of infinite games is a powerful and flexible framework for the design of reactive systems (see e.g. [1]). It is well known that, for instance, the construction of a controller acting indefinitely within its environment amounts to the computation of a winning strategy in an infinite game. For the case of regular games, algorithmic solutions of this synthesis problem have been developed, providing methods for automatic construction of controllers. The basis of this approach is the Büchi-Landweber Theorem, which says that in a regular infinite game, i.e. a game over a finite arena with a winning condition given by an ω-regular language, a finite-state winning strategy for the winner can be constructed [2]. Much work in the past two decades has been devoted to generalizations of this fundamental result. The game-theoretic setting is built on two components, a *game arena* or game graph, representing the transition structure of a system, and a *winning condition*, usually given by a logic formula or an automata theoretic condition. Most generalizations address an extension of either of the two, or both. A rapidly growing literature is thus concerned with the case of infinite game graphs and non-regular winning conditions [3, 4, 5].

In the present paper we investigate a different kind of generalization of the basic setting, regarding the possibility to get a lookahead on the moves of the opponent. To explain this aspect it is convenient to refer to the simplest format of infinite games, also called Gale-Stewart games [6]. In such a game we abstract from arenas but just let the two players choose letters from a finite alphabet in turn. (For notational convenience let us only consider the typical case of the Boolean alphabet $\mathbb{B} := \{0, 1\}$.) A play is built up as a sequence $a_0 b_0 a_1 b_1 \cdots$ where a_i is chosen by one player and b_i by the other. A natural view is to consider the sequence $\alpha = a_0 a_1 \cdots$ as *input stream* and $\beta = b_0 b_1 \cdots$ as *output stream*. Accordingly, the players are called Player Input and Player Output, or short Player I and Player O. The play is won by Player O if the ω-word $\alpha^\wedge\beta := \binom{a_0}{b_0}\binom{a_1}{b_1}\binom{a_2}{b_2}\cdots \in (\mathbb{B}^2)^\omega$ satisfies the winning condition, i.e. if it belongs to a given ω-regular language L. In the classical setting, a strategy for Player O is a function f that maps a finite input prefix $a_0 \cdots a_i$ to the bit b_i that is to be chosen by Player O. Such a strategy induces an operator $\lambda : \mathbb{B}^\omega \to \mathbb{B}^\omega$ from input streams to output streams. In this work we study more generalized operators that correspond to strategies where the choice of b_i depends on $a_0 \cdots a_j$, for $j \neq i$. We show results on the existence of such strategies for different conditions on the relation between i and j.

There are two motivations for the study of such a generalization, a practical and a theoretical one. In many scenarios, the occurrence of delays (say between input and output) is realistic, either as a modeling assumption or as a feature of strategies. For example, the design of a controller may involve a buffer that allows to store a sequence of input bits of some fixed length d such that the bit b_i of the output sequence is to be delivered with lookahead d, i.e. on the basis of the input sequence $a_0 \cdots a_{i+d}$. Conversely, in the context of networked control (i.e. systems with components in different locations), there may be a delay d in the transmission of data, which means that the delivery of b_i is due at a point where only the input bits $a_0 \cdots a_{i-d}$ are available. It is clear that the occurrence of lookaheads and delays influences the existence of solutions. In the first case, we obtain for increasing d an increasing advantage for the output player, whereas in the second case we obtain an increasing disadvantage. Observe that the cases are symmetric in the two players, and thus are mutually reducible.

A more theoretical motivation is to explore more comprehensively and systematically the solution concepts for infinite games. The classical concept of a strategy gives a very special kind of operator, but there are natural options of higher generality, well-known already from background fields like descriptive set theory and topology [6]. Let us mention four fundamental levels of operators, corresponding to different levels of obligation for Player O to move. The most general ones are the continuous operators (see e.g. [7, 8]). An operator λ is continuous (in the Cantor space of infinite sequences over $\mathbb{B}$) if in the output sequence $\beta = \lambda(\alpha)$ the bit b_i is determined by a finite prefix of α. Referring only to the length of prefixes, we call an operator uniformly continuous if for some strictly monotone function $h : \mathbb{N} \to \mathbb{N}$ we have that b_i is determined by $a_0 \cdots a_{h(i)}$. For fixed h we then speak of h-delay operators. On a further level of specialization, we are dealing with operators of bounded delay. These are h-delay operators with $h(i) \leq i + d$, for some $d \in \mathbb{N}$. Analogously, if $h(i) = i + d$, then we speak of operators with constant delay d, and finally, the function $h(i) = i$ supplies the operators induced by standard strategies. All these levels of delay naturally correspond to different types of games; for example, a continuous strategy involves the moves "wait" or "output b" after each move of the opponent.

Our main result connects the different kinds of operators in the context of infinite games. We show that in a two-person game with regular winning condition, one can decide whether there is a continuous winning strategy for Player O, and in this case a strategy of constant delay can be constructed. Moreover, one can compute a suitable bound d for the delay. Thus, in the first mentioned application scenario, if a standard controller for satisfying a regular specification does not exist then one can decide whether some finite buffer will help, and determine the needed size of that buffer. We also show that the result fails when passing to non-regular specifications. However, which functions may be appropriate for uniformly continuous strategies in the non-regular case is left open. It seems that for infinite-state (or non-regular) games our result can serve as an entry into a much wider field of study.

As indicated above, the idea of generalized concepts of strategies is far from new. An early contribution is found in the (not well-known) paper of Hosch and Landweber [9]. It deals with constant delay strategies in regular games and exploits a result of Even and Meyer from Boolean circuit theory to establish a bound for delays [10]. We obtain this result here as a corollary of the main theorem. The extension of our result over [9] covers three dimensions: the connection with strategies of unbounded delay, a considerably simplified and transparent proof of the Hosch-Landweber-Theorem (the construction in [9] is highly complex), and finally better complexity bounds for suitable delays.

This paper is organized as follows. In the next section we introduce notation. In Section 3 we present several kinds of functions and the operators they induce. We also bridge from continuous operators to delay operators and introduce games with delay. In Sections 4–6 we prove our main result via a twostage reduction: In Section 4 we do the first step, switching over to block games. In Section 5 we deal with notions related to semigroups and define a semigroup game. This framework is finally used in Section 6 to establish the second step of the reduction, i.e. the connection between block games and the semigroup game. Sections 7 and 8 provide evidence that our results cannot be generalized to ω-context-free specifications and give an outlook on future investigations.

2 Preliminaries

Let Σ be a finite *alphabet*. By Σ^* and Σ^ω we denote the sets of finite and infinite *words* over Σ. Usually, finite words are denoted $u, v, \ldots$ whereas $\alpha, \beta, \ldots$ are infinite words. By $|u|$ we denote the *length* of u and $\Sigma^n := \{u \mid |u| = n\}$ is the set of words of length n. $\mathbb{N}$ is the set of natural numbers and $\mathbb{N}_+ := \mathbb{N} \setminus \{0\}$. Given $n_1, n_2 \in \mathbb{N}$ with $n_1 < n_2$ we write $\Sigma^{[n_1,n_2]}$ for $\bigcup_{n_1 \leq n \leq n_2} \Sigma^n$.

A *(deterministic) finite automaton*, DFA for short, over Σ is a tuple $\mathcal{A} = (Q, q_0, \delta, F)$ where Q is a (non-empty) finite set of *states*, $q_0 \in Q$ is the *initial state*, $\delta : Q \times \Sigma \to Q$ is a *transition function*, and $F \subseteq Q$ is a set of *final states*. The *run* ρ_u of $\mathcal{A}$ on $u := u_0 \cdots u_{n-1}$ is the finite sequence $\rho_u(0) \cdots \rho_u(n)$ with $\rho_u(0) = q_0$ and $\rho_u(i+1) = \delta(\rho_u(i), u_i)$ for $i = 0, \ldots, n-1$. We define $\mathcal{A}$ to *accept* u if and only if $\rho_u(n) \in F$. The set of all words accepted by $\mathcal{A}$ is called the $*$*-language* of $\mathcal{A}$ and denoted $L_*(\mathcal{A})$. Later in our work we need the following basic property of deterministic finite automata.

Lemma 1. *Let $\mathcal{A}$ be a DFA with n states and $|L_*(\mathcal{A})| = \infty$. Then, for all $i \in \mathbb{N}$, $\mathcal{A}$ accepts a word u_i of length $i \leq |u_i| \leq i + n$.*

A *(deterministic) parity automaton*, DPA for short, over Σ is similar to a DFA, but instead of the set F of final states it has a *coloring*, i.e. a function $c : Q \to \{0, \ldots, m\}$. A run of a DPA is the natural extension of a run of a DFA to infinite words. For $\alpha \in \Sigma^\omega$, the set $\mathrm{Inf}(\rho_\alpha)$ is the set of states visited infinitely often in run ρ_α. We define the parity automaton $\mathcal{A}$ to accept α if and only if $\max(\mathrm{Inf}(c(\rho_\alpha)))$ is even, i.e. the maximal color seen infinitely often in the run on α is even. Accordingly, the acceptance condition of $\mathcal{A}$ is called a *max-parity* acceptance condition. The set of all words accepted by $\mathcal{A}$ is called the ω-language of $\mathcal{A}$ and denoted $L_\omega(\mathcal{A})$.

In the sequel, we write $L(\mathcal{A})$ instead of $L_*(\mathcal{A})$ or $L_\omega(\mathcal{A})$, if it is clear from the context whether $\mathcal{A}$ is a DFA or DPA. It is well-known that the class of languages accepted by DPA is exactly the class of ω*-regular* languages (see e.g. [1]).

A *parity game* $\Gamma = (V, V_I, V_O, E, c)$ is played by two players, Player I and Player O, on a directed graph $G = (V, E)$:

- $V = V_I \uplus V_O$ is a partition of V into positions of Player I and Player O,
- $E \subseteq V \times V$ is the set of allowed *moves*, and
- $c : V \to \{0, \ldots, m\}$ is a coloring of V (w.l.o.g. $m \in 2\mathbb{N}$).

We assume that for each $v \in V$ there is a valid move from v, i.e. $vE := \{w \mid (v, w) \in E\} \neq \emptyset$. A *play* is an infinite path through G. A (standard) *strategy* for Player O is a function $f : V^*V_O \to V$ defining, for each position of Player O and each history $v_0 \cdots v_k$ of the play, her next move. Thus, for each $v_0 \cdots v_k$ (with $(v_i, v_{i+1}) \in E$ for all $i = 0, \ldots, k-1$) and $v_k \in V_O$, the function f is defined such that $(v_k, f(v_0 \cdots v_k)) \in E$. A play $v_0 v_1 \cdots$ is *consistent* with the strategy f if for each $v_i \in V_O$ the next position is given by f, i.e. $v_{i+1} = f(v_0 \cdots v_i)$.

The *parity winning condition* is again defined so that a play $v_0 v_1 \cdots$ is winning for Player O if and only if the maximal color occurring infinitely often in $\{c(v_i) \mid i \in \mathbb{N}\}$ is even. In the other case the play is winning for Player I. The function f is called a *winning strategy for Player O from* v_0 if each play starting in v_0 that is consistent with f is winning for Player O, and analogously for Player I. Parity games, even on infinite graphs, are *determined*, i.e. for each v either Player I or Player O has a winning strategy from v (see e.g. [1]).

For the rest of this paper, let us fix $\{0, 1\}$ as input and output alphabet, i.e. let $\Sigma_I = \Sigma_O := \mathbb{B}$. All the definitions and results are analogous for other finite alphabets of size at least two.

3 Operators and Games with Delay

In this section we introduce different kinds of functions and operators, and show how they induce games with different degrees of lookahead. In the sequel, we mostly use the term "delay" in place of "lookahead", following e.g. [9].

3.1 Delay Operators

Let λ denote a function from $\mathbb{B}^\omega$ to $\mathbb{B}^\omega$, also called an *operator*. We shall distinguish the following classes of operators, starting form the most general ones.

(1) *continuous operators*
(2) *uniformly continuous operators*
(3) *h-delay operators* for a fixed $h : \mathbb{N} \to \mathbb{N}$
(4) *bounded delay operators*
(5) *d-delay operators* for a fixed $d \in \mathbb{N}$

An operator λ is continuous if in the output sequence $\beta = \lambda(\alpha)$ each bit is determined by a finite prefix of α. This condition is equivalent to the standard topological definition, where λ is continuous if the preimage $\lambda^{-1}(U)$ of every open set $U \subseteq \mathbb{B}^\omega$ is open in $\mathbb{B}^\omega$. (Here, open sets in $\mathbb{B}^\omega$ are given by the standard Cantor topology, i.e. $U \subseteq \mathbb{B}^\omega$ is open if there exists $W \subseteq \mathbb{B}^*$ such that $U = \{w\mathbb{B}^\omega \mid w \in W\}$; for details see e.g. [8].) To formally capture the constraint that each output bit is determined by a finite prefix of the input, we define the continuity of λ using a map l that transforms each input bit into either 0 or 1 or $\triangleright$ (the latter meaning that the production of the next output bit is still deferred). The value $\lambda(\alpha)$ is then obtained from the sequence of l-values by deleting all entries $\triangleright$.

Definition 2. *An operator $\lambda : \mathbb{B}^\omega \to \mathbb{B}^\omega$ is* continuous *if there exists $l : \mathbb{B}^* \to \{0, 1, \triangleright\}$ such that for all $\alpha \in \mathbb{B}^\omega$ the word $l(\alpha) := l(\alpha_0)l(\alpha_0\alpha_1)l(\alpha_0\alpha_1\alpha_2)\cdots$ satisfies the following:*

(1) $l(\alpha)$ does not end with $\triangleright^\omega$, and
(2) $\lambda(\alpha) = strip(l(\alpha))$ where $strip(l(\alpha))$ is the word $l(\alpha)$ with all $\triangleright$ removed.

Let us now define *h-delay* and *uniformly continuous* operators. Let $h : \mathbb{N} \to \mathbb{N}$ be a strictly monotone function. We say that λ is an *h-delay* operator if, for each $\alpha \in \mathbb{B}^\omega$, the bit $(\lambda(\alpha))_i$ depends only on $\alpha_0 \cdots \alpha_{h(i)}$. An operator λ is uniformly continuous if there exists an h such that λ is an h-delay operator. Observe that each uniformly continuous operator is indeed continuous; the function h supplies the information how long the output $\triangleright$ should be produced.

For the space $\mathbb{B}^\omega$ it is known that the converse also holds. This is a consequence of König's Lemma, or equivalently of the fact that continuous functions on a closed bounded space are uniformly continuous.

Lemma 3. *For every continuous operator $\lambda : \mathbb{B}^\omega \to \mathbb{B}^\omega$ there exists a strictly monotone function $h : \mathbb{N} \to \mathbb{N}$ such that λ is an h-delay operator.*

By the above lemma, the classes of continuous operators $\mathbb{B}^\omega \to \mathbb{B}^\omega$ and uniformly continuous operators $\mathbb{B}^\omega \to \mathbb{B}^\omega$ are exactly the same. A space where this does not hold is e.g. $\mathcal{R} := \mathbb{B}^\omega \setminus \{0^\omega\}$. Consider $\lambda_1 : \mathcal{R} \to \mathcal{R}$ with

$$\lambda_1(\alpha) := \begin{cases} 01^\omega & \text{if } \alpha = 0^*10\beta \text{ for some } \beta \in \mathbb{B}^\omega \\ 1^\omega & \text{otherwise} \end{cases}$$

Note that $\lambda_1 : \mathcal{R} \to \mathcal{R}$ is continuous, but not uniformly continuous. Since our results rely in fact on uniform continuity, we adhere to the space $\mathbb{B}^\omega$.

Among the uniformly continuous operators, we distinguish an even more restricted class of bounded delay operators. A function $h : \mathbb{N} \to \mathbb{N}$ is said to be of *bounded delay* if there exist $i_0, d \in \mathbb{N}$ such that $h(i) = i + d$ for all $i \geq i_0$, and it is said to be a *d-delay* function (or a function of *constant* delay d) if $h(i) = i + d$ for all $i \in \mathbb{N}$. The induced operators are named accordingly.

In all definitions above, we assume the delay function h strictly monotone. For our purpose it is more convenient to consider the function $f_h : \mathbb{N} \to \mathbb{N}_+$, denoting the number of additional input bits until the next output bit:

$$f_h(i) := \begin{cases} h(0) + 1 & \text{if } i = 0 \\ h(i) - h(i-1) & \text{if } i > 0 \end{cases}$$

In the sequel, we work only with the functions f_h. Moreover, we use the special notation $\langle d \rangle$ for the function f_h with h of constant delay d: $\langle d \rangle(0) = d + 1$ and $\langle d \rangle(i) = 1$ for $i > 0$. From now on, we omit the subscript h in our notation.

3.2 Regular Games with Delay

In this section we introduce the regular infinite game $\Gamma_f(L)$. It is induced by an ω-language L (usually given by a DPA $\mathcal{A}$) over $\mathbb{B}^2$, and a function $f : \mathbb{N} \to \mathbb{N}_+$. (Since we focus on the impact of the function f, we omit L if it is clear from the context and write Γ_f.) The function f imposes a delay (or lookahead) on the moves of Player O. This means that in round i Player I has to choose $f(i)$ many bits, and Player O chooses one bit, afterwards. This way the players build up two infinite sequences; Player I builds up α and Player O builds up β, respectively. The corresponding play is winning for Player O if and only if the word $\alpha^\wedge\beta := \binom{a_0}{b_0}\binom{a_1}{b_1}\binom{a_2}{b_2}\cdots$ is accepted by $\mathcal{A}$. For a DPA $\mathcal{A}$, we say that $L(\mathcal{A})$ is *solvable with finite delay* if and only if there exists $f : \mathbb{N} \to \mathbb{N}_+$ such that Player O wins $\Gamma_f(L(\mathcal{A}))$ (analogously for restricted classes of functions).

Observe that the possible strategies for Player O in Γ_f correspond precisely to h-delay operators (for $f = h^\Delta$), since Player O must output her ith bit after receiving the next $f(i)$ bits of input. Thus, the question whether there exists an h-delay operator λ such that $\{\binom{\alpha}{\lambda(\alpha)} \mid \alpha \in \mathbb{B}^\omega\} \subseteq L(\mathcal{A})$ is equivalent to the question whether there exists a winning strategy for Player O in Γ_f.

A basic observation is that winning with delay is a monotone property. For two functions $f, g : \mathbb{N} \to \mathbb{N}_+$ we write $f \sqsubseteq g$ if and only if $f(i) \leq g(i)$ for all $i \in \mathbb{N}$.

Remark 4. If Player O wins Γ_{f_0} then she also wins Γ_f for each $f \sqsupseteq f_0$. Analogously, if Player I wins Γ_{g_0} then he also wins Γ_g, for each $g \sqsubseteq g_0$.

Example 5. Let $L \subseteq (\mathbb{B}^2)^\omega$ be given by the ω-regular expression

$$\binom{0\,a}{a\,*}\Sigma^\omega + \binom{1**b}{b***}\Sigma^\omega$$

where $a, b \in \mathbb{B}$ and $*$ denotes any bit. If Player I chooses 0 as his first bit then Player O needs to know a, so she needs delay one in this situation. Contrary, if Player I chooses 1 as his first bit then Player O needs delay three to obtain b. Thus, she wins the game with delay three, but neither with delay two nor one.

In the next sections we prove our main result (see Theorem 14): *Let $\mathcal{A}$ be a DPA with n states, m colors, and let $n' := 2^{(mn)^{2n}}$. Then, there is a continuous operator λ with $\binom{\alpha}{\lambda(\alpha)} \in L(\mathcal{A})$ (for all $\alpha \in \mathbb{B}^\omega$) if and only if there is a $(2n'-1)$-delay operator with the same property.* To obtain this result we show that $L(\mathcal{A})$ is solvable with finite delay if and only if $L(\mathcal{A})$ is solvable with delay $2n' - 1$.

4 The Block Game

In this section we make the first step in the proof of our main result, which is to relax the number of bits Player I can choose in each move. For this reason we introduce a new game Γ'_f, called the *block game*.

The game Γ'_f differs from Γ_f in two ways. Firstly, the lengths of the words to be chosen by the players are decided by Player I, within certain intervals determined by f. Secondly, Player I is one move ahead compared to Γ_f.

A play in Γ'_f is built up as follows: Player I chooses $u_0 \in \mathbb{B}^{[f(0),2f(0)]}$ and $u_1 \in \mathbb{B}^{[f(1),2f(1)]}$, then Player O chooses $v_0 \in \mathbb{B}^{|u_0|}$. In each round thereafter, i.e. for $i \geq 2$, Player I chooses $u_i \in \mathbb{B}^{[f(i),2f(i)]}$ and Player O responds by a word $v_{i-1} \in \mathbb{B}^{|u_{i-1}|}$. The winning condition is defined as before.

We show that Player I wins the game Γ_f for all functions f if and only if he wins the block game Γ'_f for all functions f. To this end, for $f : \mathbb{N} \to \mathbb{N}_+$, let f' be defined by $f'(0) := f(0) + f(1)$, and $f'(i) := f(i+1)$ for $i > 0$.

Proposition 6. *Let $f : \mathbb{N} \to \mathbb{N}_+$. If Player I wins $\Gamma_{f'}$ then he also wins Γ'_f.*

Proof. Assume Player I has a winning strategy in $\Gamma_{f'}$. For $i \in \mathbb{N}$, let u_i be the words chosen by Player I in $\Gamma_{f'}$ and u'_i the words chosen by Player I in Γ'_f, and analogously v_i, v'_i for Player O. The winning strategy yields $u_0 \in \mathbb{B}^{f'(0)}$ as Player I's first move. Since $f(0) + f(1) = f'(0)$ we can choose $u'_0 u'_1 = u_0$ as Player I's first move in Γ'_f. Player O answers by $v'_0 \in \mathbb{B}^{|u'_0|}$. We can use v'_0 to simulate the moves $v_0, \dots, v_{|v'_0|-1}$ of Player O in $\Gamma_{f'}$, each of which consists of one bit. Player I answers by $u_1, \dots, u_{|v'_0|}$ of lengths $f'(1), \dots, f'(|v'_0|)$. Since $|v'_0| \geq 1$, the sum $f'(1) + \cdots + f'(|v'_0|)$ is non-empty and at least $f'(1) = f(2)$. Accordingly, the word $u_1 \cdots u_{|v'_0|}$ is long enough to give u'_2 with $f(2) \leq |u'_2| \leq 2f(2)$. We choose u'_2 as the prefix of $u_1 \cdots u_{|v'_0|}$ of length $f(2)$. Player O answers in Γ'_f by v'_1 of length $|u'_1|$, and we can use it to simulate another $|v'_1|$ rounds in $\Gamma_{f'}$. Thereby, we obtain enough bits to give u'_3, and so on. This way, we build up the same plays in $\Gamma_{f'}$ and Γ'_f. Since Player I wins $\Gamma_{f'}$, he also wins Γ'_f. □

For $f : \mathbb{N} \to \mathbb{N}_+$, let f'' be inductively defined by $f''(0) := f(0)$ and

$$f''(i+1) := \sum_{j=0}^{2(f''(0)+\ldots+f''(i))} f(j).$$

Proposition 7. *Let $f : \mathbb{N} \to \mathbb{N}_+$. If Player I wins $\Gamma'_{f''}$ then he also wins Γ_f.*

Proof. Assume Player I has a winning strategy in $\Gamma'_{f''}$. For $i \in \mathbb{N}$, let u'_i be the words chosen by Player I in $\Gamma'_{f''}$ and u_i the words chosen by Player I in Γ_f, and analogously v'_i, v_i for Player O. Player I's winning strategy yields $u'_0 \in \mathbb{B}^{[f''(0),2f''(0)]}$ and $u'_1 \in \mathbb{B}^{[f''(1),2f''(1)]}$ as his first move in $\Gamma'_{f''}$. For $i \in \mathbb{N}$, let d'_i be the length of u'_i. Since

$$d'_0 + d'_1 \geq f''(0) + f''(1) = f(0) + \sum_{j=0}^{2f''(0)} f(j),$$

we can give the moves $u_0, \ldots, u_{d'_0}$ of Player I in Γ_f. This yields Player O's answers $v_0, \ldots, v_{d'_0-1}$, i.e. d'_0 bits. We can use them to simulate v'_0, i.e. Player O's first move in $\Gamma'_{f''}$. Player I's winning strategy yields u'_2 of length $f''(2) \leq d'_2 \leq 2f''(2)$. We need to give another d'_1 moves of Player I in Γ_f to obtain Player O's answers $v_{d'_0}, \ldots, v_{d'_0+d'_1-1}$. For that we need $f(d'_0+1) + \ldots + f(d'_0+d'_1)$ bits. With u'_2 in our hands we can give these moves, because

$$\begin{aligned} d'_2 \geq f''(2) &= f(0) + \ldots + f(2f''(0) + 2f''(1)) \\ &\geq f(0) + \ldots + f(d'_0 + d'_1) \\ &\geq f(d'_0 + 1) + \ldots + f(d'_0 + d'_1). \end{aligned}$$

Iterating this we obtain the same plays built up in $\Gamma'_{f''}$ and Γ_f. Since Player I wins $\Gamma'_{f''}$, he also wins Γ_f. □

The following corollary of Propositions 6 and 7 is the first step in our proof.

Corollary 8. *Let $\mathcal{A}$ be a DPA. Then the following are equivalent:*

(1) For all $f : \mathbb{N} \to \mathbb{N}_+$ Player I wins $\Gamma_f(L(\mathcal{A}))$.
(2) For all $f : \mathbb{N} \to \mathbb{N}_+$ Player I wins $\Gamma'_f(L(\mathcal{A}))$.

5 The Semigroup Game

In this section we introduce a game which is independent of particular delays. To define it, we extract from a DPA $\mathcal{A}$ two equivalence relations, one for each player, such that the moves of the players are equivalence classes of these relations. The first one (for Player O) is denoted $\sim$ and induces a finite semigroup on $(\mathbb{B}^2)^*$. The second one (for Player I) is denoted $\approx$ and ranges over $\mathbb{B}^*$. Roughly speaking, two (pairs of) words are equivalent if they effect the same behavior on $\mathcal{A}$.

Our approach to transform parity automata into finite semigroups is similar to the constructions presented in [11, 12]. Let $\mathcal{A} = (Q, q_0, \delta, c)$ be a DPA over $\mathbb{B}^2$. We use the semiring $\mathcal{S} := (\{\perp\} \cup c(Q), +, \cdot)$ in which addition is defined as maximum, i.e. $x + y := \max(x, y)$ with $\perp$ being the least element, and multiplication is defined as follows:

$$x \cdot y := \begin{cases} \max(x,y) & \text{if } x \neq \perp \text{ and } y \neq \perp \\ \perp & \text{otherwise} \end{cases}$$

Note that the set $L_{\text{eq}} := (\mathbb{B}^2)^*$, i.e. the set of pairs of words of equal length, is a regular language. With each pair $\binom{u}{v} \in L_{\text{eq}}$ we associate a matrix $\mu\binom{u}{v}$ of size $|Q|^2$ with entries in $\mathcal{S}$, i.e. $\mu\binom{u}{v} \in \mathcal{S}^{Q\times Q}$, defined as follows:

$$\mu\begin{pmatrix} u \\ v \end{pmatrix}_{p,q} := \begin{cases} \perp & \text{if } \delta^*\left(p, \begin{pmatrix} u \\ v \end{pmatrix}\right) \neq q \\ \max\{c(\pi)\} & \text{if } \delta^*\left(p, \begin{pmatrix} u \\ v \end{pmatrix}\right) = q \text{ and } \pi \text{ is the associated } \mathcal{A}\text{-path} \end{cases}$$

Observe that $\mathcal{S}^{Q\times Q}$ induces a finite semigroup and $\mu\binom{u}{v} \cdot \mu\binom{u'}{v'} = \mu\binom{uu'}{vv'}$. Let $\sim$ be the equivalence relation on L_{eq} defined by: $\binom{u}{v} \sim \binom{u'}{v'}$ if and only if $\mu\binom{u}{v} = \mu\binom{u'}{v'}$. For each $\binom{u}{v}$, the equivalence class $\left[\binom{u}{v}\right]$ is identified by a matrix $\mu \in \mathcal{S}^{Q\times Q}$. Since $\mathcal{S}$ and Q are finite, $\mathcal{S}^{Q\times Q}$ is finite as well, and so the relation $\sim$ has finite index, i.e. it has finitely many equivalence classes. We denote the index of $\sim$ by index($\sim$). Note that $L_{\text{eq}}/_{\sim}$ induces a finite semigroup, and μ is a semigroup morphism from $(L_{\text{eq}}/_{\sim}, \cdot)$ to $(\mathcal{S}^{Q\times Q}, \cdot)$.

Lemma 9. *Let* $\binom{u}{v} \in L_{\text{eq}}$*. Then, the set* $\left[\binom{u}{v}\right]$ *is a regular* $*$*-language over* $\mathbb{B}^2$*.*

Proof. We construct an automaton recognizing $\left[\binom{u}{v}\right]$ as follows: First, we construct for all $p, q \in Q, k \in c(Q)$ the automaton $\mathcal{A}_{p,q,k}$ recognizing the set of all words that induce a path from p to q in $\mathcal{A}$ where k is the highest color seen on that path. The idea for this construction is to simulate the behavior of $\mathcal{A}$ while memorizing the highest color seen. To this end, define $\mathcal{A}_{p,q,k} := (c(Q) \times Q, \mathbb{B}^2, (c(p), p), \delta', \{(k, q)\})$ where

$$\delta'\left((k', p'), \begin{pmatrix} x \\ y \end{pmatrix}\right) := \left(\max\left\{k', c\left(\delta\left(p', \begin{pmatrix} x \\ y \end{pmatrix}\right)\right)\right\}, \delta\left(p', \begin{pmatrix} x \\ y \end{pmatrix}\right)\right)$$

for all $k' \in c(Q), p' \in Q, x, y \in \mathbb{B}$. The automaton starts in the state $(c(p), p)$ and simulates the behavior of $\mathcal{A}$ on its input. If it stops in state (k, q) then it accepts. The automaton $\mathcal{A}_{[\binom{u}{v}]}$ is then obtained as the intersection of all $\mathcal{A}_{p,q,k}$ for p, q, k such that $\mu\binom{u}{v}_{p,q} = k$. $\square$

Since $\sim$ has finite index, we can find automata for all equivalence classes of $\sim$ in the following way: For $r \in \mathbb{N}$, let $\mathcal{A}_1, \ldots, \mathcal{A}_r$ be the automata already constructed. Then $\sim$ has index r if and only if $\bigcup_{i=1,\dots,r} L(\mathcal{A}_i) = L_{\text{eq}}$. This equality can be effectively checked, and if this test fails, then we repeat the construction with a word contained in $L_{\text{eq}} \setminus \bigcup_{i=1,\dots,r} L(\mathcal{A}_i)$.

Let $\approx$ be the equivalence relation on $\mathbb{B}^*$ defined by

$$u \approx u' :\iff \forall\Big[\binom{u_0}{v_0}\Big] : \Big(\exists v : \binom{u}{v} \in \Big[\binom{u_0}{v_0}\Big] \iff \exists v' : \binom{u'}{v'} \in \Big[\binom{u_0}{v_0}\Big]\Big).$$

For $u \in \mathbb{B}^*$, the $\approx$-equivalence class of u, denoted $[u]$, can be identified with a subset of the set of all $\sim$-classes. Since $\sim$ has finite index, we get that $\approx$ has finite index as well; more precisely it holds $\text{index}(\approx) \leq 2^{\text{index}(\sim)}$.

Lemma 10. *Let $u \in \mathbb{B}^*$. Then, the set $[u]$ is a regular $*$-language over $\mathbb{B}$.*

Proof. We construct an automaton recognizing the language $[u]$ as follows: First, we have to check for which $\sim$-classes $\big[\binom{u_0}{v_0}\big]$ there exists $v \in \mathbb{B}^{|u|}$ such that $\binom{u}{v} \in \big[\binom{u_0}{v_0}\big]$. Let $\mathcal{B}$ be a DFA recognizing $\big[\binom{u_0}{v_0}\big]$. We take the projection on the first component (deleting the second component from the transitions of $\mathcal{B}$) and test whether the resulting automaton, say $\mathcal{B}'$, accepts u. If we do the same for all $\sim$-classes, then we obtain r automata $\mathcal{B}'_1, \dots, \mathcal{B}'_r$ accepting u, and s automata $\mathcal{B}'_{r+1}, \dots, \mathcal{B}'_{r+s}$ not accepting u, where $r + s = \text{index}(\sim)$. From these automata we can effectively construct an automaton for $[u]$, because

$$[u] = \bigcap_{i=1,\dots,r} L(\mathcal{B}'_i) \cap \bigcap_{j=r+1,\dots,r+s} \overline{L(\mathcal{B}'_j)}.$$

□

We now define the game Γ^{SG} (induced by a DPA $\mathcal{A}$ over $\mathbb{B}^2$) where the moves of the players are classes from $\mathbb{B}^*/_\approx$ and $L_{\text{eq}}/_\sim$, respectively. Accordingly, we call Γ^{SG} the *semigroup game* of $\mathcal{A}$.

The game Γ^{SG} is defined similar to the block game Γ'. The difference is that the players do not choose concrete words but the respective classes from the relations $\sim$ and $\approx$. A play is built up as follows: Player I chooses infinite classes $[u_0], [u_1] \in \mathbb{B}^*/_\approx$, then Player O chooses a class $\big[\binom{u_0}{v_0}\big] \in L_{\text{eq}}/_\sim$. In each round thereafter, i.e. for $i \geq 2$, Player I chooses an infinite class $[u_i] \in \mathbb{B}^*/_\approx$ and Player O chooses a class $\big[\binom{u_{i-1}}{v_{i-1}}\big] \in L_{\text{eq}}/_\sim$. A play is winning for Player O if and only if $\binom{u_0}{v_0}\binom{u_1}{v_1}\binom{u_2}{v_2}\cdots$ is accepted by $\mathcal{A}$.

Note that $\mathbb{B}^*/_\approx$ contains at least one infinite class and that for each class $[u]$ there exists at least one class in $L_{\text{eq}}/_\sim$ associated with $[u]$ (by the definition of $\approx$). Hence, both players can always move. Furthermore, the winning condition of Γ^{SG} is well-defined because acceptance of $\mathcal{A}$ is independent of representatives: If $\big[\binom{u_i}{v_i}\big] = \big[\binom{u'_i}{v'_i}\big]$ for all $i \in \mathbb{N}$, then $\binom{u_0}{v_0}\binom{u_1}{v_1}\cdots \in L(\mathcal{A}) \iff \binom{u'_0}{v'_0}\binom{u'_1}{v'_1}\cdots \in L(\mathcal{A})$.

Γ^{SG} can be modeled by a parity game on a graph of size $O(2^{2(mn)^n} mn)$. (Thus, its winner is computable [1].) In the vertices we keep track of the $\approx$-classes recently chosen by Player I, a color depending on the course of the play and the current state q of $\mathcal{A}$. The vertex reached by a move $\big[\binom{u}{v}\big]$ of Player O is colored by $\mu\binom{u}{v}_{q,q'}$, where q' is the state reached in $\mathcal{A}$ from q when reading $\binom{u}{v}$.

6 Connecting the Block Game and the Semigroup Game

In this section we show that Player I wins the block game Γ'_f for all functions $f : \mathbb{N} \to \mathbb{N}_+$ if and only if he wins the semigroup game Γ^{SG}. This completes the reduction and also yields the proof of our main result.

The basic idea of the proof of Theorem 12 (see below) is, for arbitrary f, to simulate the moves of the players in Γ'_f by the corresponding equivalence classes of the relations $\sim$ and $\approx$, respectively, and vice versa. For the last-mentioned direction, one has the problem whether a class $[u_i]$ contains an appropriate representative, i.e. one of length between $f(i)$ and $2f(i)$. We use Lemma 1 to show that there exists a particular f such that each function g with $g \sqsupseteq f$ indeed has this property. Then, the following lemma completes the proof.

Lemma 11. *Player I wins Γ'_f for all functions $f : \mathbb{N} \to \mathbb{N}_+$ if and only if there exists a function $g_0 : \mathbb{N} \to \mathbb{N}_+$ such that Player I wins Γ'_g for all $g \sqsupseteq g_0$.*

Proof. The direction from left to right is immediate. Conversely, assume there exists f_0 such that Player I does not win Γ'_{f_0}. Determinacy yields that Player O wins Γ'_{f_0}. By Proposition 6 Player O wins $\Gamma_{f'_0}$, and from Remark 4 it follows that she also wins Γ_f for all $f \sqsupseteq f'_0$. Proposition 7 yields that Player O wins $\Gamma'_{f''}$, for all $f \sqsupseteq f'_0$. Towards a contradiction, let g_0 be a function such that Player I wins Γ'_g for all $g \sqsupseteq g_0$, and let f_* be the maximum of g_0 and f'_0, i.e. for all $i \in \mathbb{N}$

$$f_*(i) := \max\{g_0(i), f'_0(i)\}.$$

Since $f_* \sqsupseteq f'_0$ it holds that Player O wins $\Gamma'_{f''_*}$. However, since $f''_* \sqsupseteq f_* \sqsupseteq g_0$ Player I must win $\Gamma'_{f''_*}$, by assumption. This yields a contradiction which means that g_0 cannot exist. □

Lemma 11 and the next theorem establish the second step of our reduction.

Theorem 12. *Player I wins Γ^{SG} if and only if there is a function $f : \mathbb{N} \to \mathbb{N}_+$ such that Player I wins Γ'_g for all $g \sqsupseteq f$.*

Proof. We start with the direction from right to left. Let $f : \mathbb{N} \to \mathbb{N}_+$ be a function such that Player I wins Γ'_g for all $g \sqsupseteq f$. We define a function g_0 such that $g_0 \sqsupseteq f$ and each word of length $g_0(i)$ is contained in an infinite $\approx$-class, for all $i \in \mathbb{N}$. To this end, let d' be the length of a longest word in all finite $\approx$-classes[1] and define, for all $i \in \mathbb{N}$, $g_0(i) := \max\{f(i), d' + 1\}$.

Since $g_0 \sqsupseteq f$, Player I wins Γ'_{g_0} by assumption, and a winning strategy yields his first two moves u_0, u_1. Both $[u_0]$ and $[u_1]$ are infinite, and so he can choose them in Γ^{SG}. We simulate Player O's answer $\left[\binom{u_0}{v_0}\right]$ by choosing v_0 in Γ'_{g_0}, and Player I's winning strategy yields u_2 with $[u_2]$ being infinite. Choosing $[u_2]$ in Γ^{SG} we obtain Player O's next move $\left[\binom{u_1}{v_1}\right]$, and so on.

We argue that the plays built up have the same maximal color occurring infinitely often. It suffices to show that in both plays a move of Player O leads

[1] If $\approx$ has no finite equivalence class, then we define $d' := 0$.

$\mathcal{A}$ to the same state, via paths with equal maximal color. Then, the rest follows by induction. Let q_i be the current state of $\mathcal{A}$ and u_i, u_{i+1} be the words chosen by Player I. If Player O chooses $\left[\binom{u_i}{v_i}\right]$ in Γ^{SG}, then we reach the state $q_{i+1} := \delta^*\left(q_i, \binom{u_i}{v_i}\right)$ via the maximal color $\mu\binom{u_i}{v_i}_{q_i,q_{i+1}}$. The state q_{i+1} is well-defined because from q_i every $\binom{u_i'}{v_i'} \in \left[\binom{u_i}{v_i}\right]$ leads $\mathcal{A}$ to the same state, though via different paths, but with the same maximal color. In Γ'_{g_0} Player O chooses v_i. As in Γ^{SG}, we reach the state q_{i+1} via the maximal color $\mu\binom{u_i}{v_i}_{q_i,q_{i+1}}$.

Conversely, assume that Player I wins Γ^{SG}. Let $\mathcal{A}_1, \ldots, \mathcal{A}_r$ be automata recognizing all the $\approx$-classes, and n' the maximal number of states among these automata, i.e. $n' := \max\{n_1, \ldots, n_r\}$, where n_j is the number of states of $\mathcal{A}_j$ $(j = 1, \ldots, r)$. Let f be the constant function with $f(i) := n'$ for all $i \in \mathbb{N}$. We first show that Player I wins Γ'_f: Player I's winning strategy in Γ^{SG} yields $[u_0], [u_1]$. Since $[u_0], [u_1]$ are infinite, we can apply Lemma 1. Accordingly, each $\mathcal{A}_j$ accepts a word of length between f and $f + n_j$ and thus between f and $2f$, because $n_j \leq f$.[2] Hence, we can assume w.l.o.g. that $f \leq |u_0|, |u_1| \leq 2f$. Player I chooses u_0, u_1 in Γ'_f and Player O answers by a word v_0 with $|v_0| = |u_0|$. We simulate this move by $\left[\binom{u_0}{v_0}\right]$ in Γ^{SG} and obtain Player I's answer $[u_2]$, so the next move of Player I in Γ'_f is u_2 (for appropriate u_2). Player O chooses v_1 with $|v_1| = |u_1|$, and so on.

The plays built up this way have the same maximal color occurring infinitely often, using the same inductive argument as above. Starting at q_i, Player O's move v_i in Γ'_f has the same effect as the corresponding move $\left[\binom{u_i}{v_i}\right]$ in Γ^{SG}, i.e. we reach the state $q_{i+1} := \delta^*\left(q_i, \binom{u_i}{v_i}\right)$ via the maximal color $\mu\binom{u_i}{v_i}_{q_i,q_{i+1}}$.

We complete the proof by showing that Player I wins Γ'_g for all $g \sqsupseteq f$. Let $|[a,b]| := b - a$ be the size of the interval $[a,b]$. If $g \sqsupseteq f$, then (since $|[f, 2f]| = n'$) it holds $|[g(i), 2g(i)]| \geq n'$, for all $i \in \mathbb{N}$. Hence, to win Γ'_g Player I simply chooses longer representatives of the $\approx$-classes than in Γ'_f. $\square$

A thorough analysis of the constructions of the $\sim$-classes and $\approx$-classes, respectively, yields an upper bound for n'. Let n be the number of states of $\mathcal{A}$ and m the number of colors. Let $u, v \in \mathbb{B}^*$ with $|u| = |v|$. Since $\mathcal{A}$ is deterministic, there is exactly one entry distinct from $\perp$ in each of the n rows of $\mu\binom{u}{v}$, and $\mathcal{A}_{p,q,k}$ has at most mn states. Hence, each $\mathcal{A}_{[\binom{u}{v}]}$ has at most $(mn)^n$ states, i.e. as many as the product of n (deterministic) automata of size mn. To obtain an automaton for a class $[u]$ we have to intersect index($\sim$) languages (cf. page 261). By the same argument as above, there are at most $(mn)^n$ possible matrices identifying all the $\sim$-classes. Since our construction includes determinization, we obtain each $\mathcal{A}_{[u]}$ having at most k states, where

$$k \leq (2^{(mn)^n})^{(mn)^n} = 2^{(mn)^{2n}}.$$

Next, we obtain our main result showing that in regular games constant delay is sufficient for Player O to win, if she can win with delay at all.

[2] To simplify matters we write f instead of $f(i)$.

Lemma 13. *Let n' be as in the proof of Theorem 12. Then, Player O wins Γ^{SG} if and only if Player O wins $\Gamma_{\langle 2n'-1 \rangle}$.*

Proof. Define $f(i) := n'$ for all $i \in \mathbb{N}$ and let w of length d' be a longest word in all finite $\approx$-classes. Moreover, let $L(\mathcal{A}') = [w]$, where $\mathcal{A}'$ has n states. Then we have $d' < n$. Otherwise, the run of $\mathcal{A}'$ on w had a loop, which is a contradiction to the finiteness of $L(\mathcal{A}')$. Since $n \leq n'$ we get $d' < n'$ and so $d' + 1 \leq n'$. Thus, each $\approx$-class containing a word of length at least f is infinite.

Assume that Player O wins Γ^{SG}. We first show that Player O wins Γ'_f. Let u_0, u_1 with $n' \leq |u_0|, |u_1| \leq 2n'$ be the first move of Player I in Γ'_f. By the above remarks $[u_0], [u_1]$ are infinite, and we can simulate $[u_0], [u_1]$ in Γ^{SG}. Player O's winning strategy in Γ^{SG} yields $\left[\binom{u_0}{v_0}\right]$ for some suitable v_0. Let him choose v_0 in Γ'_f. Then Player I chooses u_2 and we simulate $[u_2]$ in Γ^{SG}, and so on.

As in the proof of Theorem 12, we obtain plays with the same maximal color occurring infinitely often, and so Player O wins Γ'_f. Simulating a winning strategy for Γ'_f she also wins $\Gamma_{\langle 2n'-1 \rangle}$. The factor 2 comes from the fact that we need at least $2n'$ bits when simulating Player I's first move in Γ'_f.

Conversely, let Player O win $\Gamma_{\langle 2n'-1 \rangle}$ and $g(i) := 2n'$, for all $i \in \mathbb{N}$. Since $g \sqsupseteq \langle 2n'-1 \rangle$, Player O wins Γ_g. Then, by Proposition 7, she also wins $\Gamma'_{g''}$. Given a winning strategy for Player O in $\Gamma'_{g''}$ we can specify one for her in Γ^{SG} as follows: A move $[u_i]$ of Player I is simulated by u_i in $\Gamma'_{g''}$, for $g''(i) \leq |u_i| \leq 2g''(i)$. (By Lemma 1, an appropriate representative u_i must exist because $g'' \sqsupseteq g$, and so $|[g''(i), 2g''(i)]| \geq n'$ for all $i \in \mathbb{N}$.) We use Player O's answer v_{i-1} to choose $\left[\binom{u_{i-1}}{v_{i-1}}\right]$ in Γ^{SG}. This yields a play winning for Player O in Γ^{SG}. $\square$

With Corollary 8, Lemma 11 and Theorem 12 we have shown that the problem whether $L(\mathcal{A})$ is solvable with finite delay is reducible to the question whether Player O wins Γ^{SG}. Finally, Lemma 13 shows that $L(\mathcal{A})$ is solvable with finite delay if and only if it is solvable with constant delay.

Theorem 14. *Let $\mathcal{A}$ be a DPA over $\mathbb{B}^2$. Then, $L(\mathcal{A})$ is solvable with finite delay if and only if $L(\mathcal{A})$ is solvable with delay $2n' - 1$. There is a continuous operator λ such that $\{\binom{\alpha}{\lambda(\alpha)} \mid \alpha \in \mathbb{B}^\omega\} \subseteq L(\mathcal{A})$ if and only if there is a $(2n'-1)$-delay operator with the same property.*

We have estimated the size of Γ^{SG} to be $O(2^{2(mn)^n} mn)$. Since it requires only m colors, its winner can be computed in time $O((2^{2(mn)^n} mn)^m)$.

Corollary 15. *Let $\mathcal{A}$ be a DPA over $\mathbb{B}^2$. The problem whether $L(\mathcal{A})$ is solvable with finite delay and the problem whether there is a continuous operator λ with $\{\binom{\alpha}{\lambda(\alpha)} \mid \alpha \in \mathbb{B}^\omega\} \subseteq L(\mathcal{A})$ are in* 2EXPTIME.

7 Lookahead in Non-Regular Games

In this section we show that the above results do not hold for context-free ω-languages (CFL$_\omega$, for an introduction see e.g. [13]). Let us first recall that it is undecidable whether a context-free ω-language $L \subseteq \mathbb{B}^\omega$ is universal, i.e. whether $L = \mathbb{B}^\omega$ holds.

Theorem 16 (see also [14]). *Let $L \subseteq (\mathbb{B}^2)^\omega$ be a context-free ω-language. Then, it is undecidable whether there exists f such that Player O wins $\Gamma_f(L)$.*

Proof. We make a reduction from the universality problem for context-free ω-languages. Let $L_I \in \mathrm{CFL}_\omega$ and $L := \left\{\binom{\alpha}{\beta} \mid \alpha \in L_I, \beta \in \mathbb{B}^\omega\right\}$. If L_I is universal then L is universal as well, and Player O wins with any f. Conversely, if L_I is not universal, then Player I wins by playing a word $\alpha \notin L_I$. There is no response β such that $\binom{\alpha}{\beta} \in L$, therefore Player O looses with each f. Altogether, L_I is universal if and only if there exists f such that Player O wins $\Gamma_f(L)$. □

The situation is different for *deterministic* ω-context-free specifications: in this case at least the winner of the standard game $\Gamma_{\langle 0\rangle}$ is decidable [3].

In addition to undecidability for the general case, we show that there exist context-free specifications which are solvable with finite delay, but not with constant delay.

Example 17. Let $L \subseteq (\mathbb{B}^2)^\omega$ be defined such that if Player I chooses an ω-word of the form $\alpha = 1^{2m_0}0^{n_0}1^{2m_1}0^{n_1}\cdots$, for $m_i, n_i \in \mathbb{N}_+$, then Player O wins if and only if he answers by $\beta = 1^{m_0}0^{m_0+n_0}1^{m_1}0^{m_1+n_1}\cdots$. This means Player O's ith block of 1s must have exactly half the length of Player I's ith block of 1s, and both blocks must start at the same position. If α is not of the above form, then Player O wins as well.

The language L is recognized by a deterministic ω-pushdown automaton. As long as the input is $\binom{1}{1}$, we push a symbol on the stack. If we read the first $\binom{1}{0}$ after $\binom{1}{1}$, we start to pop symbols from the stack. If we reach the initial stack symbol at the same time as we read the first $\binom{0}{0}$ after $\binom{1}{0}$ then we are satisfied and visit a final state.

Observe that Player O wins $\Gamma_f(L)$, if $f(i) := 2$ for all $i \in \mathbb{N}$. When she has to give her ith bit β_i she already knows Player I's $(2i)$th bit α_{2i}, and that is enough to decide whether to play 0 or 1.

Let us show that L is not solvable with constant delay. Towards a contradiction, assume Player O wins $\Gamma_{\langle d\rangle}$ for some $d \in \mathbb{N}$. We construct a winning strategy for Player I in $\Gamma_{\langle d\rangle}$ as follows: Player I chooses 1^{d+1} as initial move and 1 as each of his d subsequent moves. Player O must answer each of these $d+1$ moves by choosing 1. Otherwise, she loses immediately. Afterwards, Player I chooses another 1 to complete his block of 1s to even length. (After this move, Player I has chosen exactly twice as many 1s as Player O.) Whatever Player O answers, say b, Player I wins by choosing $1-b$ next. This is due to the fact that the block of 1s chosen by Player O gets either too short or too long.

8 Conclusion

In this paper we introduced and compared strategies with different kinds of lookahead in regular infinite games. We showed that continuous strategies can be reduced to uniformly continuous strategies of a special form, namely strategies with constant lookahead. This result is a first step into a wider – and it seems rather unexplored – topic. Let us mention some aspects. First, it is straightforward to present the results in a set-up that is symmetric in the two players. We also skipped here a lower bound proof for the double exponential size in Theorem 14. It is also possible to think of "infinite lookahead" where, for instance, the second player may use information about the first player's sequence up to a partition of the space of sequences into regular sets.

We showed that bounded lookahead is not enough for continuous strategies in non-regular games. It is open which functions may be appropriate for uniformly continuous operators in such games, in particular for context-free games. Also, it is open whether solvability with continuous or uniformly continuous strategies is decidable for these games.

References

[1] Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games. LNCS, vol. 2500. Springer, Heidelberg (2002)
[2] Büchi, J.R., Landweber, L.H.: Solving sequential conditions by finite-state strategies. Transactions of the AMS 138, 295–311 (1969)
[3] Walukiewicz, I.: Pushdown processes: Games and model checking. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 62–74. Springer, Heidelberg (1996)
[4] Cachat, T.: Higher order pushdown automata, the caucal hierarchy of graphs and parity games. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 556–569. Springer, Heidelberg (2003)
[5] Bouquet, A.J., Serre, O., Walukiewicz, I.: Pushdown games with unboundedness and regular conditions. In: Pandya, P.K., Radhakrishnan, J. (eds.) FSTTCS 2003. LNCS, vol. 2914, pp. 88–99. Springer, Heidelberg (2003)
[6] Moschovakis, Y.N.: Descriptive Set Theory. Studies in Logic and the Foundations of Mathematics, vol. 100. North-Holland Publishing Company, Amsterdam (1980)
[7] Trakhtenbrot, B.A., Barzdin, Y.M.: Finite Automata, Behavior and Synthesis. North Holland, Amsterdam (1973)
[8] Thomas, W., Lescow, H.: Logical specifications of infinite computations. In: de Bakker, J.W., de Roever, W.-P., Rozenberg, G. (eds.) REX 1993. LNCS, vol. 803, pp. 583–621. Springer, Heidelberg (1994)
[9] Hosch, F.A., Landweber, L.H.: Finite delay solutions for sequential conditions. In: Nivat, M. (ed.) Automata, Languages and Programming, Paris, France, pp. 45–60. North-Holland, Amsterdam (1972)
[10] Even, S., Meyer, A.: Sequential boolean equations. IEEE Transactions on Computers C-18, 230–240 (1969)
[11] Perrin, D., Pin, J.: Semigroups and automata on infinite words. In: Fountain, J. (ed.) NATO Advanced Study Institute Semigroups, Formal Language and Groups, pp. 49–72. Kluwer Academic Publishers, Dordrecht (1995)
[12] Pin, J.: Finite semigroups and recognizable languages: An introduction (1995)
[13] Cohen, R.S., Gold, A.Y.: Omega-computations on deterministic pushdown machines. Journal of Computer and System Sciences 16, 275–300 (1978)
[14] Finkel, O.: Topological properties of omega context-free languages. Theoretical Computer Science 262, 669–697 (2001)

Reachability Analysis of Communicating Pushdown Systems

Alexander Heußner[⋆], Jérôme Leroux, Anca Muscholl, and Grégoire Sutre

LaBRI, Université Bordeaux, CNRS – France

Abstract. The reachability analysis of recursive programs that communicate asynchronously over reliable FIFO channels calls for restrictions to ensure decidability. We extend here a model proposed by La Torre, Madhusudan and Parlato [16], based on communicating pushdown systems that can dequeue with empty stack only. Our extension adds the dual modality, which allows to dequeue with non-empty stack, and thus models interrupts for working threads. We study (possibly cyclic) network architectures under a semantic assumption on communication that ensures the decidability of reachability for finite state systems. Subsequently, we determine precisely how pushdowns can be added to this setting while preserving the decidability; in the positive case we obtain exponential time as the exact complexity bound of reachability. A second result is a generalization of the doubly exponential time algorithm of [16] for bounded context analysis to our symmetric queueing policy. We provide here a direct and simpler algorithm.

Introduction

The verification of safety properties for distributed programs, e.g., client/server environments, peer-to-peer networks, or Grid applications, relies on the decidability of the reachability problem. In this paper we reconsider *recursive queueing concurrent processes* (RQCP), one possible model for such systems which was studied recently by La Torre, Madhusudan, and Parlato [16]. It is a natural idea to combine peer-to-peer asynchronous communication (via point-to-point, unbounded, reliable FIFO channels) with some automaton-based model for individual peers (e.g., pushdown automata or Petri nets). We call such combined models *queueing concurrent processes* (QCP). Since communicating finite-state automata are the most elementary instantiation of QCP, reachability is in general undecidable [6]. Furthermore, adding recursion (i.e., replacing finite-state by pushdown automata) yields an additional source of undecidability. One of the main motivations in this paper is to separate these two sources of undecidability: we consider behavioral restrictions for which reachability for communicating finite-state machines is decidable, and then look under which conditions we can add recursion to the model. The challenging task is to derive conditions that conserve the simplicity and expressiveness of the model.

⋆ Work supported by the ANR project AVERISS.

In general, there are three main directions to cope with the undecidability of communicating finite-state machines: restricting the communication architecture, assuming that channels are lossy, or adding semantic restrictions, e.g., that sending/receiving of messages can be scheduled in such a way that runs can be executed with channels of bounded size. In this paper we stick to the latter approach, as described in more detail below. Our point of departure is the work of La Torre et al. [16], which introduced RQCP together with a behavioral restriction on the combined use of channels and pushdowns. Informally, a RQCP is *well-queueing* if pushdown processes can only dequeue (read) messages when the stack is empty (they can enqueue messages without restriction). Well-queueing expresses an event-based programming paradigm: tasks are executed by threads without interrupt, i.e., a thread accepts the next task only after it finished the current one. One of the results of [16] is that RQCP have a decidable reachability problem if and only if their communication architecture is a directed forest; in the decidable case, the latter paper provides a doubly exponential upper bound by a reduction to bounded-phase multi-stack pushdown systems [15].

Our contribution. We extend the work of La Torre et al. [16] in several directions. First, we add a dual notion to well-queueing: a pushdown process can enqueue (send) messages only with empty stack (but can dequeue messages without restriction). *Oriented* communication architectures, as presented here, combine these two notions, by fixing the behaviour of the two endpoints of each channel. This dual notion to well-queueing arises naturally if one wants to model interrupts: a server might need to accept tasks from high priority clients independently of the status of the running task.

Second, we exhibit a precise characterization of those oriented architectures for which the RQCP model has a decidable reachability problem over so-called *eager* runs. Informally, a run is eager if the sending of a message is immediately followed by its reception, a notion closely related to existentially 1-bounded communication [14]. Eagerly communicating finite-state machines are a well-studied model, enjoying good expressiveness and decidability properties [10]. Here, we use eager runs in order to rule out undecidability stemming from unbounded channels. We show that reachability of RQCP over eager runs is EXPTIME-complete in the decidable case. This result generalizes and improves the doubly exponential time decision procedure of [16], which holds for architectures without undirected cycles (*polyforest* architectures).

Eagerness is a relatively strong requirement, hence, we show how it arises rather naturally, by imposing a semantic restriction on the communication flow: the *mutex* restriction demands that in every reachable configuration there is no more than one non-empty channel per cycle. In particular, QCP over polyforest architectures are mutex. Actually mutex can be seen as a generalization of the half-duplex restriction studied in [7].

La Torre et al. [16] propose a second approach to solve the reachability problem for RQCP, inspired by recent work on reachability with bounded contexts in the verification of concurrent programs [19]. They show that bounded-context reachability for well-queueing RQCP is decidable in time doubly exponential in the number of contexts. Again, this result is obtained by a reduction to bounded-phase multi-stack pushdown systems [15]. Our second main contribution is to extend the bounded-context result to RQCP that allow for the two dual notions of well-queueing. Moreover, our algorithm is direct and simpler than the one involving bounded-phase multi-stack pushdown systems.

A long version of this paper that includes all proofs omitted due to space limitations can be found at `http://hal.archives-ouvertes.fr/hal-00443529/`.

Related work. In the context of thread programming, other notions of synchronization between pushdowns arise naturally. Earlier publications considered synchronization via shared memory, such as local/global memory in [4,5] or bags in [20,12]. The paper [4] showed that bounded-context reachability can be solved in exponential time, whereas [20] provided an exponential space lower bound for reachability (without context bounds). More recently, synchronization in the form of state observation was considered in [2]. The latter model was shown to be decidable only for acyclic architectures, and is strongly related to lossy systems [3,9].

1 Queueing Concurrent Processes

We write $X \dot\cup X'$ for the *disjoint* union of X and X'. Let Σ denote an *alphabet* (i.e., a finite set of *letters*). We write Σ^* for the set of all *finite words* (*words* for short) over Σ, and we let ε denote the *empty word*. Moreover, we use standard complexity classes such as polynomial space (PSPACE), deterministic exponential time (EXPTIME), and doubly exponential time (2EXPTIME). For more detailed definitions the reader is referred to textbooks like [18].

A *communication architecture* $\mathcal{T}$ (or architecture for short) is a pair $\langle \mathcal{P}, \mathrm{Ch} \rangle$ with a finite non-empty set $\mathcal{P}$ of processes and a finite set of point-to-point channels $\mathrm{Ch} \subseteq (\mathcal{P} \times \mathcal{P}) \setminus id_{\mathcal{P}}$.

Remark 1. Our definition of architecture forbids self-loops, as well as two distinct channels in the same direction between a pair of processes (without further restriction, these settings are immediately Turing equivalent).

Definition 1. *A system of* queueing concurrent processes (QCP) *over a given architecture $\mathcal{T} = \langle \mathcal{P}, Ch \rangle$ is a tuple $\mathcal{A} = \langle (S_p)_{p\in\mathcal{P}}, (\Sigma_p)_{p\in\mathcal{P}}, (\Delta_p)_{p\in\mathcal{P}}, (s_p^0)_{p\in\mathcal{P}}, M \rangle$ with M a finite message alphabet. For each process $p \in \mathcal{P}$, the tuple $\langle S_p, \Sigma_p, \Delta_p, s_p^0 \rangle$ describes a (local) transition system on the state set S_p with actions from $\Sigma_p = \Sigma_p^{loc} \dot\cup \Sigma_p^{com}$, which are either local (i.e., in Σ_p^{loc}) or communication actions in $\Sigma_p^{com} = \{p!q(m) \mid (p,q) \in Ch \text{ and } m \in M\} \cup \{p?q(m) \mid (q,p) \in Ch \text{ and } m \in M\}$. Local transitions are given by the rules in $\Delta_p \subseteq S_p \times \Sigma_p \times S_p$, and the initial state of process p is s_p^0.*

The global state space is $S = \prod_{p\in\mathcal{P}} S_p$ and the global initial state is $s^0 = (s_p^0)_{p\in\mathcal{P}} \in S$. By $\Sigma = \bigcup_{p\in\mathcal{P}} \Sigma_p$ we denote the set of all possible actions in $\mathcal{A}$. The size *of $\mathcal{A}$ is $\sum_{p\in\mathcal{P}} |\Delta_p|$.*

As usual, $p!q(m)$ denotes the send of message m from process p to process q, whereas $q?p(m)$ denotes the matching receive on process q.

Note also that the S_p and hence S need not necessarily be finite. If S is finite, we will call $\mathcal{A}$ a *finite* QCP (or communicating finite-state machine, CFM). The local transition systems given by $\langle S_p, \Sigma_p, \Delta_p, s_p^0 \rangle$ could be, for example, finite automata, counter automata (including Petri nets), pushdown automata. As usual, we define the semantics of $\mathcal{A}$ as labeled infinite-state transition system:

Definition 2. *A* QCP $\mathcal{A}$ *represents an* LTS $[\![\mathcal{A}]\!] = \langle C, \Sigma, \rightarrow, c^0 \rangle$ *with configurations* $C = S \times (M^*)^{Ch}$ *and the initial configuration* $c^0 = \big(s^0, (\varepsilon, \ldots, \varepsilon)\big)$*, i.e., all channels are initially empty. We write a configuration as* $c = \langle s, w \rangle$ *where* $s = (s_p)_{p \in \mathcal{P}}$ *is the global state and* $w = (w_{p,q})_{(p,q) \in Ch}$ *are the channel contents. Further, for any* $p \in \mathcal{P}$ *and* $a \in \Sigma_p$, $\langle s, w \rangle \xrightarrow{a} \langle s', w' \rangle$ *is a transition in* $C \times \Sigma \times C$ *with* $s' = (s'_p)_{p \in \mathcal{P}}$, $w' = (w'_{p,q})_{(p,q) \in Ch}$*, if* $(s_p, a, s'_p) \in \Delta_p$ *and the following holds:*
(i) $s_q = s'_q$ *for all* $q \neq p$*,*
(ii) if $a \in \Sigma_p^{loc}$ *then* $w = w'$*,*
(iii) if $a \in \Sigma_p^{com}$ *with* $a = p!q(m)$ *then* $w'_{p,q} = w_{p,q}m$ *and* $w'_{s,t} = w_{s,t}$ *for* $(s,t) \in Ch \backslash \{(p,q)\}$*,*
(iv) if $a \in \Sigma_p^{com}$ *with* $a = p?q(m)$ *then* $w_{q,p} = mw'_{q,p}$ *and* $w'_{s,t} = w_{s,t}$ *for* $(s,t) \in Ch \backslash \{(q,p)\}$*.*

A *finite run* ρ in the labeled transition system $[\![\mathcal{A}]\!]$ from a configuration c_0 to $c_n \in C$ is a sequence $\langle c_0, a_1, c_1, a_2, c_2, \cdots, a_n, c_n \rangle$ where $c_{i-1} \xrightarrow{a_i} c_i$ for $1 \leq i \leq n$. The *length* of ρ is n, and a run of length 0 is defined as $\rho = \langle c_0 \rangle$.

A configuration $c \in \mathcal{C}$ is *reachable* in the QCP $\mathcal{A}$ if there exists a finite run $\rho = \langle c_0, a_1, c_1, \cdots, c_n \rangle$ starting in the initial configuration $c_0 = c^0$ and ending in $c_n = c$. We define the *reachability set* as $Reach_{\mathcal{A}} = \{c \in \mathcal{C} \mid c \text{ is reachable in } \mathcal{A}\}$. The *reachability question* asks for a given $\mathcal{A}$ and a configuration $c \in \mathcal{C}$ whether $c \in Reach_{\mathcal{A}}$. Given a state $s \in S$, the *control state reachability question* asks whether one can reach a configuration with (control) state component s regardless of the channel content, i.e., whether $\{s\} \times (M^*)^{\mathrm{Ch}} \cap Reach_{\mathcal{A}} \neq \emptyset$. Both questions are undecidable for finite QCP with at least two processes that are connected by two channels [6].

The *trace* of a run $\rho = \langle c_0, a_1, c_1, a_2, c_2, \cdots, a_n, c_n \rangle$ is the sequence of actions $tr(\rho) = a_1 \cdots a_n \in \Sigma^*$. Since channels are FIFO, we can speak about *matching* send/receive pairs: a_i, a_j form such a pair if (1) $a_i = p!q(m)$, $a_j = q?p(m)$, and (2) $|\{\ell \mid \ell \leq i, a_\ell = p!q(n), n \in M\}| = |\{\ell \mid \ell \leq j, a_\ell = q?p(n), n \in M\}|$. We call two runs ρ, ρ' *order-equivalent* if they can be transformed one into the other by iteratively commuting adjacent transitions labeled by a and b, resp., such that (i) a, b do *not* belong to the same process, and (ii) a, b are *not* a matching send/receive pair.

Lemma 1. *If* ρ, ρ' *are order-equivalent runs of* $\mathcal{A}$ *starting in the same configuration, then* ρ, ρ' *end in configurations with the same control state.*

Definition 3. *A run* ρ *with trace* $tr(\rho) = a_1 \cdots a_n$ *is* eager *if the following holds: if* $a_i = p!q(m)$ *for some* $0 \leq i < n$ *then either* $a_{i+1} = q?p(m)$ *or no action* a_j *with* $j > i$ *is a receive on* (p,q)*.*

A channel that does not permit further receives is in its "growing phase".

A QCP $\mathcal{A}$ is *eager* if each $c \in \mathit{Reach}_{\mathcal{A}}$ is reachable by some eager run. Eager runs, modulo the fact that Definition 3 allows for runs which end in a sequence of (unmatched) send actions, are closely related to globally 1-bounded runs, whereas eager QCP are close to existentially globally 1-bounded CFM [14,11]. The *(control-state) reachability question on eager runs* asks whether one can reach a given (control state) configuration by some eager run.

Recursive QCPs and oriented communication architectures. In the following we introduce RQCP together with a symmetric version of the "well-queueing" property from [16]. Informally speaking, RQCP are QCP where all basic processes are pushdown automata.

A *recursive* QCP (RQCP) is a QCP given by $\langle (S_p)_{p\in\mathcal{P}}, \Sigma, (\Delta_p)_{p\in\mathcal{P}}, s^0, M, \Gamma\rangle$ where each process p is a pushdown process over the local states $S_p \subseteq Z_p \times \Gamma^*$ with a finite set Z_p of control states and the content of the pushdown stack represented by a word over the stack alphabet Γ; further, the local actions Σ_p^{loc} contain $push(\gamma)$ and $pop(\gamma)$ for each $\gamma \in \Gamma$, and we assume that in the initial state all stacks are empty, i.e., $s^0 \in \prod_{p\in\mathcal{P}}(Z_p \times \{\varepsilon\})$.

A well-queueing RQCP in [16] is one where a process can only receive when its stack is empty. Here, we dualize this concept by also allowing channels where the sender (but not the receiver) must have an empty stack.

Definition 4. *An architecture $\mathcal{T} = \langle \mathcal{P}, Ch\rangle$ together with a labeling of the channels $Ch = Ch_{\mathtt{s}} \cup Ch_{\mathtt{r}}$ as "send restricted" ($Ch_{\mathtt{s}}$) and/or "receive restricted" ($Ch_{\mathtt{r}}$), is called* oriented.

For pushdown networks the previous definition translates, informally speaking, as follows: a process p can send on a channel $(p,q) \in \mathrm{Ch}_s$ only with empty stack. Symmetrically, p can receive on a channel $(q,p) \in \mathrm{Ch}_r$ only with empty stack. By definition, channels are restricted at least at one end.

The semantics of an RQCP $\mathcal{R}$ is given by a labeled transition system $[\![\mathcal{A}]\!] = \langle C, \Sigma, \rightarrow, c^0\rangle$ analogously to Definition 2 except for transitions $(s,w)\xrightarrow{a}(s',w')$ that correspond to a (local) pushdown transition $(s_p, a, s'_p) \in \Delta_p$:

(i)' if $a \in \Sigma_p^{loc}$,
then $w = w'$ and further *push* and *pop* behave as local pushdown actions;
(ii)' for $a = p!q(m) \in \Sigma_p^{com}$,
we demand additionally to (ii) that if $(p,q) \in \mathrm{Ch}_{\mathtt{s}}$ then $s_p \in Z_p \times \{\varepsilon\}$;
(iii)' for $a = p?q(m) \in \Sigma_p^{com}$,
we demand additionally to (iii) that if $(q,p) \in \mathrm{Ch}_{\mathtt{r}}$ then $s_p \in Z_p \times \{\varepsilon\}$.

Given an oriented architecture $\mathcal{T} = \langle \mathcal{P}, \mathrm{Ch}\rangle$ we will use the following notation, that forgets about the direction of the channels and focuses on the (un)limited use of pushdowns: for two processes $p, q \in \mathcal{P}$ we write $p \bullet\!\!-\!\!\circ\, q$ if $(p,q) \in \mathrm{Ch} \setminus \mathrm{Ch}_{\mathtt{s}}$ or $(q,p) \in \mathrm{Ch} \setminus \mathrm{Ch}_{\mathtt{r}}$. Moreover, we write $p \circ\!\!-\!\!\circ\, q$ if $(p,q) \in \mathrm{Ch}_{\mathtt{s}} \cap \mathrm{Ch}_{\mathtt{r}}$ or $(q,p) \in \mathrm{Ch}_{\mathtt{s}} \cap \mathrm{Ch}_{\mathtt{r}}$.

Informally, $p \bullet\!\!-\!\!\circ\, q$ means that for at least one channel between p and q, process p can use its stack without restriction. Similarly,

$p \circ\!\!-\!\!\circ q$ means that neither p nor q can use their stacks when communicating. Finally, $p \circ\!\!-\!\!\bullet q$ is equivalent to $q \bullet\!\!-\!\!\circ p$.

As this notation refers implicitly to a given channel between p and q, we might have both $p \bullet\!\!-\!\!\circ q$ as well as $p \circ\!\!-\!\!\bullet q$ (or $p \circ\!\!-\!\!\circ q$) — since both channels (p, q) and (q, p) may exist.

Remark 2. A channel can be both send and receive restricted, but we exclude — per definition — channels that are unrestricted at both ends, as this leads immediately to undecidability: one can reduce right away the intersection of two context-free languages to the reachability question on a topology with a single channel that is unrestricted at both ends (and eager runs suffice).

2 Decidable Oriented Architectures

Several factors lead to the undecidability of the (control-state) reachability question for Rqcp. Particularly, the model is already undecidable even without pushdowns. Our actual motivation in this section is therefore to separate the undecidability which stems from unbounded Fifo queues from the undecidability originating from an unrestricted usage of pushdowns. Hence, we consider a restricted version of the control state reachability question, namely the one on eager runs. In the next section, we show how eager Qcp naturally arise from some natural (and decidable) restrictions on cyclic communication. The most simple example of an eager Qcp is an Rqcp on a polyforest architecture.

Definition 5. *An oriented architecture $\mathcal{T} = \langle \mathcal{P}, Ch \rangle$ is called* confluent *if there exist distinct processes $p = r_0, r_1, \ldots, r_k, r_{k+1} = q$ $(k \geq 1)$ in $\mathcal{T}$, satisfying the following conditions: (1) $(r_i, r_{i+1}) \in Ch \cup Ch^{-1}$ for all $0 \leq i \leq k$, and (2) $p \bullet\!\!-\!\!\circ r_1$ and $r_k \circ\!\!-\!\!\bullet q$.*

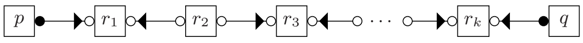


Fig. 1. Example of a confluent architecture (mixing $\bullet\!\!-\!\!\circ$ and $\rightarrow =$ Ch notation)

Theorem 1. *An oriented architecture $\mathcal{T} = \langle \mathcal{P}, Ch \rangle$ admits a decidable* Rqcp *control-state reachability problem on eager runs if and only if it is non confluent. Moreover, the problem is* ExpTime*-complete in the latter case.*

The only-if direction of the theorem above is not difficult to show. The if-direction is based on two main ingredients: first, we show how to reorder runs such that we can identify subruns on subarchitectures that start and end with empty stacks; second, we use induction on subarchitectures.

The subsequent lemma is the core of the remaining proof of Theorem 1.

Lemma 2. *Let $\mathcal{R}$ be an* Rqcp *over a non-confluent architecture $\mathcal{T}$, and consider an eager run ρ of $\mathcal{R}$ starting with all stacks empty. Further assume that $\rho = \rho_1 a \, \rho_2 \, b \, \rho_3$ with $a, b \in \Sigma$ such that for some process $p \in \mathcal{P}$:*

(i) the stack of p *is continuously empty during both subruns* ρ_1 *and* ρ_3*, and continuously non-empty during* ρ_2*, respectively;*

(ii) there is no *eager run* ρ' *that is order-equivalent to* ρ *and has the form* $\rho' = \rho_1 c\, \rho'_2\, d\, \rho_3$ *where* $c, d \in \Sigma$ *with* $c \neq a$ *or* $d \neq b$*.*

Then, the stacks of all processes occurring in ρ_2 *are empty after both* ρ_1 *and* ρ_2*.*

Proof. By (ii), each process q occurring in ρ_2 has a “proof” for its presence in ρ_2. This proof consists of a simple, unoriented path in $\mathcal{T}$ between p and q.

We explain this more formally: notice first that a is necessarily a push action and b its matching pop action on p. Consider now the first action of some q occurring in ρ_2: this is a communication either with p, or with some different process r. Thus, we can assume inductively that there is a simple unoriented path between p and r, which can be extended to a path between p and q.

Since p's stack is continuously nonempty during ρ_2, we know that the communication between p and the first node p' on the path above is on a channel of type $p \bullet\!\!-\!\!\circ\, p'$. Due to the non-confluent property, we cannot have $r \circ\!\!-\!\!\bullet\, q$; hence, q's stack must be empty at the beginning of ρ_2. A symmetric argument applies at the end of ρ_2. □

Proposition 1. *The control-state reachability problem for* RQCP *on eager runs over non-confluent architectures is* EXPTIME *complete.*

Proof. For the upper bound we show how to compute inductively in EXPTIME all pairs of global states $(s, s') \in S^2$ such that there is an eager run over $\mathcal{T} = \langle \mathcal{P}, \mathrm{Ch} \rangle$, starting in s with stacks and queues empty, as well as ending in s' with possibly empty stacks and queues. Actually, we need to compute more, namely for which sets $P \subseteq \mathcal{P}$ of connected processes we can reach s' from s (with empty stacks and queues). W.l.o.g. we assume that $\mathcal{T}$ is connected.

We (arbitrarily) order the processes in $\mathcal{P}$ and choose the first process $p \in \mathcal{P}$ that has at least one edge (channel) of type $p \bullet\!\!-\!\!\circ *$ in $\mathcal{T}$ *and* at least one of type either $p \circ\!\!-\!\!\bullet *$ or $p \circ\!\!-\!\!\circ *$ (with $*$ denoting an arbitrary process in $\mathcal{P} \setminus \{p\}$). Since the architecture is non confluent, only two cases can occur if such a process does *not* exist: either (i) $\mathcal{T}$ contains no $\bullet\!\!-\!\!\circ$ edge at all, or (ii) $\mathcal{T} = \langle \mathcal{P} = \langle r, r_1, \ldots, r_k \rangle, \mathrm{Ch} \rangle$, where:

- a channel of type $\bullet\!\!-\!\!\circ$ is one of (r, r_i), for some i,
- a channel of type $\circ\!\!-\!\!\circ$ is one of (r_i, r_j), for some i, j.

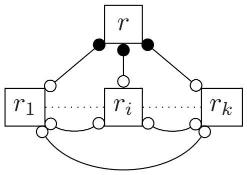


Fig. 2. Case (ii)

Let us first consider case (i) where $\mathcal{T}$ has no channel of type $\bullet\!\!-\!\!\circ$, i.e., all channels in $\mathcal{T}$ are of type $\circ\!\!-\!\!\circ$. In this case we can reorder any eager run into an order-equivalent eager run where messages alternate with local pushdown runs, each of them starting and ending with empty stack. This amounts to solving the control-state reachability problem for *one* pushdown automaton of size exponential in $|\mathcal{P}|$, which is possible in EXPTIME.

We consider now case (ii). Here, we may assume w.l.o.g. that $r \bullet\!-\!\circ\, r_i$ for every $1 \leq i \leq k$ (just add channels of type $r \bullet\!-\!\circ\, r_i$ if there is no channel between r and r_i). Assume that there exists an eager run from s to s', starting with all stacks empty. For simplicity we also assume that the run ends with all channels empty (the special case of unmatched sends can be handled similarly). This run can be reordered into an order-equivalent eager run ρ of the following form:

$$\rho = \sigma_0\alpha_0\, m_1\sigma_1\alpha_1\, m_2\sigma_2\alpha_2 \cdots m_n\sigma_n\alpha_n \tag{1}$$

where

- process r does not occur in $\sigma_0 \cdots \sigma_n$,
- each m_i is a message (i.e., send/receive pair) between r and some r_j,
- each α_i is a sequence of local (pushdown) actions of r,
- each σ_i $(i < n)$ starts and ends with all stacks of processes r_j empty.

We obtain the previous reordering by scheduling the messages between r and the r_j as late as possible. That is, the run starts first with actions not involving r (subrun σ_0), plus some pushdown actions of r (subrun α_0). Then the first message m_1 between r and some r_j follows. All remaining actions are in the future of this message, and σ_1 is a run over the r_j (which may synchronize among themselves by communication), etc.

We check that the stacks of the r_j are always empty at the beginning/end of each σ_i by considering the subruns $\sigma_0\alpha_0 m_1\sigma_1$, $\sigma_0\alpha_0 m_1\sigma_1 m_2\alpha_2\sigma_2$, etc.: consider the first occurrence of some process q occurring e.g. in σ_1. Either this occurrence is the message m_1 between q and r (so $r \bullet\!-\!\circ\, q$), or it is a synchronization with some r_i, so $r_i \circ\!-\!\circ\, q$. In particular, q's stack must be empty at the end of σ_0.

The existence of an eager run as above can be checked by first pre-computing the control-state reachability for $\mathcal{T} \setminus r$, which corresponds to the first case previously considered in this proof (recall that we compute summaries, i.e., all pairs of global states that can be reached starting/ending with empty stacks and queues). Then the question for $\mathcal{T}$ reduces to the control-state reachability of a pushdown automaton (that of process p) of size exponential in $|\mathcal{P}|$, thus showing the claim.

We now get back to the situation where $\mathcal{T}$ contains some process $p \in \mathcal{P}$ with at least one channel of type $p \bullet\!-\!\circ *$ in $\mathcal{T}$, and at least one channel of type either $p \circ\!-\!\bullet *$, or $p \circ\!-\!\circ *$ in $\mathcal{T}$.

Consider an (eager) run over $\mathcal{T}$, starting (and possibly ending) with all stacks empty. Again we assume, for convenience, that all channels are empty at the end. The run can be reordered such that we obtain an order-equivalent run of the form $\rho_0\sigma_0\rho_1 \cdots \sigma_{n-1}\rho_n$, where:

- every subrun σ_i $(i < n)$ starts and ends with empty stacks for all processes $q \neq p$ occurring in σ_i,
- p's stack is continuously empty during ρ_i, and continuously non-empty during σ_i, for each i.

We need to explain why we may assume that the subruns σ_i start / end with empty stacks for each $q \neq p$. The reason is that we schedule the internal push / pop actions of p that start / end a phase with non-empty p-stack such that push actions have lowest priority, and pop ones have the highest one. Such push / pop pairs on p delimit the subruns σ_i and Lemma 2 can be applied to $\rho_0\sigma_0\rho_1$, $(\rho_0\sigma_0\rho_1)\sigma_1\rho_2$, etc.

The existence of suitable subruns σ_i can be checked inductively: notice that channels of type $p \circ\!\!-\!\!\bullet *$ and $p \circ\!\!-\!\!\circ *$ can be removed from $\mathcal{T}$, since p's pushdown is non-empty during every σ_i, hence such channels are not used. More formally, we check for each (connected) subset $P \subseteq \mathcal{P}$ with $p \in P$, and each pair of starting/ending states $s = (s_q)_{q \in P}$, $s' = (s'_q)_{q \in P}$ whether there is a run of processes from P from s to s' in the modified architecture, starting/ending with empty stacks and queues.

Consider now a set P corresponding to processes occurring in some run σ_i. Notice first that every process in P can be reached from p via messages involving only P; symmetrically, every process in P can reach p via messages involving only P. For each such set P we can introduce new synchronization messages between processes in P such that we replace σ_i by a sequence S_i of new messages with the following properties: the first message in S_i is sent by p, the last message is received by p, and every process in P occurs among the receivers in S_i. Such a sequence S_i can be used to enforce that processes in P_i go (in a sort of meta-transition) from state $s = (s_q)_{q \in P}$ to state $s' = (s'_q)_{q \in P}$, thus replacing σ_i. We can enforce that the new messages occur only in form of sequences S_i, by encoding S_i with message contents. In order to avoid an exponential blow-up in the size of the RQCP we record the possible sequences S_i separately. Notice that using these sequences in the base step of the induction does not affect the ExpTime upper bound.

We can now apply induction on the modified RQCP in order to check whether there is some run of the form $\rho' = \rho_0 S_0 \rho_1 S_1 \dots S_{n-1} \rho_n$. The induction is possible since we can transform the channels of type $p \bullet\!\!-\!\!\circ *$ into type $\circ\!\!-\!\!\circ$ (since p does not use its pushdown in ρ').

To summarize, the induction is done on two parameters: either we decrease the overall number of channels, or we change at least one channel of type $\bullet\!\!-\!\!\circ$ into type $\circ\!\!-\!\!\circ$. We first check the existence of runs σ_i inductively in exponential time, by computing reachability for every pair of global states and subset of processes (of which there are exponentially many). Then we modify the RQCP according to the previous calls and check inductively the existence of a *single* run ρ' (again, this is done for each pair of global states and set of processes).

Finally, let us comment on the lower bound: It is known (and probably folklore) that the following problem is ExpTime-complete: checking the emptiness of the intersection of a pushdown with n finite automata. The hardness follows easily by a reduction from linearly bounded alternating Turing machines. Actually, a closely related problem is shown to be ExpTime-hard in [8], namely the reachability problem for pushdowns with checkpoints. Clearly, the intersection between a pushdown and n finite automata can be simulated on a topology $\mathcal{T} = \langle \mathcal{P}, \mathrm{Ch} \rangle$ with $\mathcal{P} = \{r, r_1, \dots, r_n\}$ and $r \bullet\!\!-\!\!\circ r_i$ for each i. $\square$

3 From Mutex QCP to Eager QCP

The previous section showed how to decide the control state reachability for RQCP (and therewith finite QCP) on eager runs. Nevertheless, restricting the communication to eager runs seems rather strong at first glance. In the following, we will show how eagerness arises naturally on two practically relevant communication architectures: polytrees and cyclic architectures with mutex restriction. Further, we discuss the reduction of the control state reachability problem for (possibly infinite) mutex QCP to their underlying local transition systems, like e.g., Petri nets.

Any communication architecture $\mathcal{T}$ can be regarded as directed graph $\langle \mathcal{P}, \mathrm{Ch} \rangle$; let $\mathit{UCycle}(\mathcal{T})$ be the set of its undirected simple cycles. A cycle is *undirected* if we ignore the direction of the channels, and *simple* if it has no subcycle of smaller length.

Definition 6. *A configuration c of a* QCP $\mathcal{A}$ *is* mutex *with respect to a given architecture $\mathcal{T}$ if for every cycle α of $\mathit{UCycle}(\mathcal{T})$ at most one of the channels occurring in α is non-empty in c. A* QCP $\mathcal{A}$ *is called* mutex *with respect to a given architecture $\mathcal{T}$ if every $c \in \mathit{Reach}_{\mathcal{A}}$ is mutex.*

Before discussing mutex QCP in detail, we first recall two known results that are subsumed by our definition of mutex:

Remark 3. A special case of mutex QCP was considered in [16]: polyforest architectures over finite QCP, as well as (well-queueing) root-to-leaf directed forests for RQCP. Their decidability proof relied on the idea that, on any tree architecture, we can reorder runs such that first all actions of the root process are scheduled, and then, in breadth-first order, the actions of all others. Consequently, each run could be partitioned into a bounded number of contexts (bounded by $|\mathcal{P}|$) where in each context only one process executes all its actions by reading on one unique incoming channel from its tree parent (and — in the case of RQCP— solely when its local stack is empty). Hence, the decidability problem reduced to the control state reachability for a bounded-phase multi-stack pushdown system, which is known to be decidable in doubly exponential time [15].

We will show in the following that mutex QCP are eager, and, consequently, apply the results of the previous section to obtain the decidability of control-state reachability via a direct proof. Moreover, recall that the complexity of the algorithm of the previous section is EXPTIME, so one exponential less than the positive results of [16].

Remark 4. Runs over an architecture of two finite processes connected by two channels where each reachable configuration is mutex are known as "half-duplex communication". For these, it is known how to decide the (general) reachability question by computing a recognizable description of the channel contents [7]. Quasi-stable systems are a semantic ad-hoc extension of this idea to larger, cyclic architectures of finite QCP [7], which is subsumed by our mutex condition.

Proposition 2. *Given a* QCP $\mathcal{A}$ *that is mutex with respect to a given architecture* $\mathcal{T}$*, each of its runs has an order-equivalent eager run.*

Proof. In the following, we will differentiate the occurrences of one action by referring to them as events. For each process $p \in \mathcal{P}$ occurring in ρ there is a first, initial event w.r.t. ρ which will be abbreviated f_p; further, each receive event has a preceding matching send in ρ. All events that belong to the same process are totally ordered. Consequently, we define the partial order "before" (denoted by $<$) between events as the transitive closure of the previous two cases. In the following we will focus only on matched communication events, by considering send actions on channels that already entered their growing phase as internal actions.

We will inductively define a reordering for a run ρ whose first configuration $c_0 = \langle s, w \rangle$ fulfills $w_{p,q} = \varepsilon$ if there exists a receive event $q?p$ in ρ.

Assume we have a run ρ from c to c' and c fulfills the previous property. First, we pick an initial send event f_p on a channel from process p to q which either (i) has no matching receive in ρ (i.e., it is the first send in a growing phase), or (ii) its matching receive on process q is also initial, hence, equal to f_q. In case (i) we schedule f_p first and reorder inductively the remaining run starting from c'' with $c \xrightarrow{f_p} c''$ towards c'. For case (ii) we first schedule f_p and then f_q before we inductively reorder the remaining run starting from c'' with $c \xrightarrow{f_p} c''' \xrightarrow{f_q} c''$. Note than in both cases c'' satisfies our requirement.

Next we have to show that it is always possible to apply cases (i) or (ii) above, to any run ρ of our mutex QCP. The general idea is as follows. Suppose that we pick an initial send event f_{p_0} on process p_0 that has a matching receive e_{p_0} on process p_1, but e_{p_0} is not initial. Then we can restart our search for an initial event from p_1 on. If f_{p_1} is a send, then we proceed as for p_0; else, if f_{p_1} is a receive, we continue with its matching send on process p_2. As we only have finitely many processes, an unsuccessful, repeated search leads to a cycle in $\mathit{UCycle}(\mathcal{T})$: $\langle p_0, p_1, p_2, \ldots, p_i, p_{i+1}, \ldots, p_k, p_{k+1} \rangle$ with $p_{k+1} = p_i$ and all p_i $(i \leq k)$ pairwise different. Moreover, we show the existence of at least two non-empty channels on this cycle.

In the following, we slightly abuse notation by writing f_i and instead of f_{p_i} for the initial event of process p_i. We focus on the initial events $f_i, \ldots, f_k$ and their matching events $e_i, \ldots, e_k$ on processes $p_{i+1}, \ldots, p_k, p_{k+1} = p_i$. Obviously, $f_{j+1} < e_j$ for all $i \leq j \leq k$, since both f_{j+1}, e_j occur on process p_{j+1} and f_{j+1} is initial. We distinguish the following cases:

(a) all initial events $f_i, \ldots, f_k$ are receives (cf. Fig. 3(a)), then $e_{j+1} < f_{j+1} < e_j$ for all $i \leq j \leq k$; hence, we arrive at the contradiction $e_i < e_i$;

(b) there are at least two sends among the initial actions, for example f_j and f_l with $i \leq j < l \leq k$; consequently, $c \xrightarrow{f_j} c'' \xrightarrow{f_l} c'''$ leads to a configuration which is not mutex (cf. Fig. 3(b)) and, hence another contradiction;

(c) there is only one send event among the initial events of the cycle, say f_i. Then, $f_i = f_{k+1}$ is before e_k, and e_k is a send event, too (since all f_j with $i < j \leq k$ are receives). It is easy to see that $e_k < f_k < e_{k-1} < \cdots < f_{i+1} < e_i$. In particular, all events on each of $p_{i+1}, \ldots, p_k$ are after e_k.

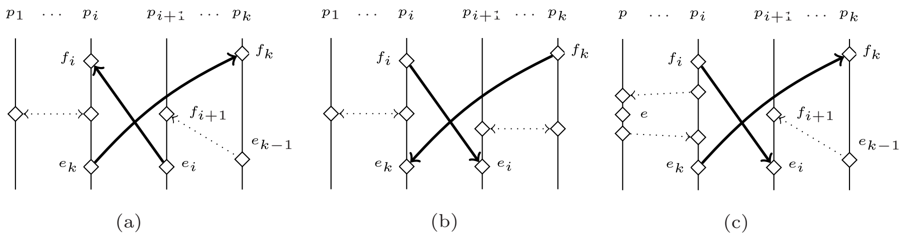


Fig. 3. Cycles in the proof of Lemma 2 ($\diamond$: events, arrows : messages)

Consider now an event e with $f_i < e < e_k$. Notice that e cannot belong to any of $p_{i+1}, \dots, p_k$, as all events on these processes must take place after e_k (cf. Fig. 3(c) for e on a process p that does not participate in the cycle); consequently, the configuration obtained after executing all events before e_k is not mutex, as the channels (p_i, p_{i+1}) and (p_i, p_k) are both non-empty. $\square$

Corollary 1. *If a* QCP $\mathcal{A}$ *is mutex with respect to a given architecture* $\mathcal{T}$ *then* $\mathcal{A}$ *is eager.*

Proposition 3. *The control state reachability for finite* QCP *that are mutex with respect to the given architecture is* PSPACE*-complete.*

Remark 5. Control-state reachability is decidable for particular *infinite-state* mutex QCP. For example, if all local transition systems are Petri nets, then the control state reachability question reduces to a Petri net reachability question which is known to be decidable [17,13].

The mutex property can be checked effectively for QCP.

Proposition 4. *It is* PSPACE*-complete to check whether a finite* QCP *is mutex with respect to a given architecture.*

4 Bounded Phase Reachability

Besides their proven practical relevance in the verification of concurrent programs [19], bounded-context reachability allows to attack the (control-state) reachability problem on QCP from a different angle. In this section, we neither restrict the communication architecture, nor constrain the runs to be eager (or mutex). The price we pay is a (strong) restriction on the form of the possible runs, by fixing the number of contexts. We present in this section a construction that subsumes the 2EXPTIME algorithm for bounded-context reachability for well-queueing RQCP described in [16]. Recall that the latter algorithm is based on a reduction to bounded-phase reachability for multi-stack systems. In contrast, our construction below is direct and simpler.

We define a *phase* in an RQCP run ρ over an oriented architecture as a (contiguous) subrun of ρ consisting of actions on a unique process, say p, that are subject to one of the two restrictions below (defining M-phases and N-phases, resp.):

(M) Receives are from a unique process, say q, with $(q, p) \in \mathrm{Ch}$ of type $p \circ\!\!-\!\!\bullet\, q$ or $p \circ\!\!-\!\!\circ\, q$. Sends go to (arbitrarily many) processes r with $(p, r) \in \mathrm{Ch}$ of type $p \bullet\!\!-\!\!\circ\, r$.

(N) Sends go to a unique process, say q, with $(p, q) \in \mathrm{Ch}$ of type $p \circ\!\!-\!\!\bullet\, q$ or $p \circ\!\!-\!\!\circ\, q$. Receives come from (arbitrarily many) processes r with $(r, p) \in \mathrm{Ch}$ of type $p \bullet\!\!-\!\!\circ\, r$.

Notice that M-phases are precisely the phases (contexts) used in [16], whereas N-phases represent the dual notion. We will refer below to the channel (q, p) (M-phase) resp. (p, q) (N-phase) as the *special* channel of the phase. A run ρ of an RQCP is *K-bounded*, if we can write $\rho = \rho_1 \cdots \rho_K$, with each ρ_i a phase as above.

Theorem 2. *Given an* RQCP $\mathcal{A}$ *and an integer* K*, the* K*-bounded control state reachability problem for* $\mathcal{A}$ *can be solved in time doubly exponential in the number* K *of phases (but polynomial in the size of* $\mathcal{A}$*).*

Sketch of proof. The basic idea is to decrease the number of phases in a particular order: M-phases are deleted for right to left (a sort of pre-computation), whereas N-phases are deleted from left to right (post-computation). Deleting a phase i belonging to some process p amounts to synchronizing a finite automaton obtained from $\mathcal{A}_p$ and phase i with the current automaton $\mathcal{A}_q$ of the process communicating with p on the special channel of phase i. We obtain this finite automaton by exploiting the fact that p's stack is empty while communicating in phase i on the special channel. In addition we must ensure that the phase i that we delete starts and ends with empty p-stack. Finally, for a single phase we need to solve a reachability problem for a single pushdown with doubly exponentially many states. The details can be found in the long version of this paper.

Remark 6. Adapting proof ideas from [15,1], we can show that the complexity bound in Theorem 2 is tight.

5 Conclusion

Applications. QCP combine an automata-based local process model with point-to-point communication, which results in an intuitive and simple framework.

Since we subsume well-queueing RQCP, we also inherit their application domains, e.g., event-based programs. The dual restriction to well-queueing (i.e., that sending on a channel is only possible if the stack is empty) covers e.g. "interrupt based" programming models, i.e., threads that can receive messages *while* still in recursion, as well as extended sensor networks where peers can collect and send data *while* using their pushdown for computations.

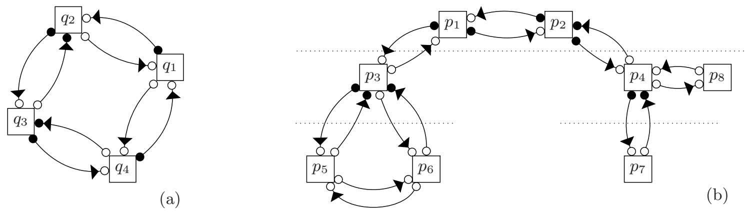


Fig. 4. Non-confluent architectures: (a) ring, and (b) hierarchical master-worker setting — tree-like architecture with •–∘-channels between master and workers (distribute tasks and collect results while in computation, send result to own master when computation finished, i.e., stack empty) as well as ∘–∘-channels between workers of the same master; note the •–∘-cycle on the top level

Fig. 4 (b) shows an example for non-confluent architectures that are on the rise with the current focus on Grid computing. The topology depicts a hierarchical overlay network as implemented, for example, in a master-worker protocols. Here, mutual communication is restricted with respect to the hierarchy (in general: •–∘ top-down and ∘–∘ between siblings). Notice also the use of the dual notion to well-queueing, when sending information from lower to higher levels.

Proposition 2 allows for further applications, since it does not assume that the QCP is finite: we can combine locally decidable models for multi-threaded programs (with or without local data), as well as local event-based programs together with eager (or mutex) communication architectures; natural candidates for local models would be Petri Nets, WSTS, or multi-set pushdown systems [20].

Outlook. We discussed in detail the class of eager RQCP (as well as mutex QCP) which both generalize the current lineup of decidable models for asynchronously communicating pushdown systems. Further, we presented an optimal decision procedure for eager RQCP over non-confluent architectures in EXPTIME, as well as a direct and simpler construction for bounded phase reachability for RQCP.

This paper dealt with the most basic form of verification, namely control-state reachability. More general reachability questions (w.r.t. configurations) may be interesting to consider. Further decision problems for QCP, like boundedness or liveness, will be investigated in future work.

References

1. Atig, M.F., Bouajjani, A., Habermehl, P.: Emptiness of multi-pushdown automata is 2ETIME-complete. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 121–133. Springer, Heidelberg (2008)
2. Atig, M.F., Bouajjani, A., Touili, T.: On the reachability analysis of acyclic networks of pushdown systems. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 356–371. Springer, Heidelberg (2008)
3. Abdulla, P.A., Jonsson, B.: Verifying programs with unreliable channels. Inf. Comput. 127(2), 91–101 (1996)

4. Bouajjani, A., Esparza, J., Schwoon, S., Strejcek, J.: Reachability analysis of multithreaded software with asynchronous communication. In: Sarukkai, S., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 348–359. Springer, Heidelberg (2005)
5. Bouajjani, A., Müller-Olm, M., Touili, T.: Regular symbolic analysis of dynamic networks of pushdown systems. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 473–487. Springer, Heidelberg (2005)
6. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. of the ACM 30(2), 323–342 (1983)
7. Cécé, G., Finkel, A.: Verification of programs with half-duplex communication. Inf. Comput. 202(2), 166–190 (2005)
8. Esparza, J., Kucera, A., Schwoon, S.: Model checking LTL with regular valuations for pushdown systems. Inf. Comput. 186(2), 355–376 (2003)
9. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theoretical Computer Science 256(1-2), 63–92 (2001)
10. Genest, B., Kuske, D., Muscholl, A.: A Kleene theorem and model checking algorithms for existentially bounded communicating automata. Inf. Comput. 204(6), 920–956 (2006)
11. Genest, B., Kuske, D., Muscholl, A.: On communicating automata with bounded channels. Fundamenta Informaticae 80, 147–167 (2007)
12. Jhala, R., Majumdar, R.: Interprocedural analysis of asynchronous programs. In: POPL 2007, pp. 339–350. ACM, New York (2007)
13. Rao Kosaraju, S.: Decidability of reachability in vector addition systems. In: STOC 1982, pp. 267–281. ACM, New York (1982)
14. Lohrey, M., Muscholl, A.: Bounded MSC communication. Inf. Comput. 189(2), 160–181 (2004)
15. La Torre, S., Madhusudan, P., Parlato, G.: A robust class of context-sensitive languages. In: LICS 2007, pp. 161–170. IEEE, Los Alamitos (2007)
16. 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)
17. Mayr, E.W.: An algorithm for the general Petri net reachability problem. SIAM J. Comput. 13(3), 441–460 (1984)
18. Papadimitriou, C.: Computational Complexity. Addison Wesley, Reading (1994)
19. 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)
20. Sen, K., Viswanathan, M.: Model checking multithreaded programs with synchronous atomic methods. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 300–314. Springer, Heidelberg (2006)

The Complexity of Synchronous Notions of Information Flow Security

Franck Cassez[1,⋆], Ron van der Meyden[2,⋆⋆], and Chenyi Zhang[3]

[1] National ICT Australia & CNRS, Sydney, Australia
[2] University of New South Wales, Sydney, Australia
[3] University of Luxembourg, Luxembourg

Abstract. The paper considers the complexity of verifying that a finite state system satisfies a number of definitions of information flow security. The systems model considered is one in which agents operate synchronously with awareness of the global clock. This enables timing based attacks to be captured, whereas previous work on this topic has dealt primarily with asynchronous systems. Versions of the notions of nondeducibility on inputs, nondeducibility on strategies, and an unwinding based notion are formulated for this model. All three notions are shown to be decidable, and their computational complexity is characterised.

1 Introduction

Information flow security is concerned with the ability for agents in a system to deduce information about the activity and secrets of other agents. An information flow security policy prohibits some agents from knowing information about other agents. In an insecure system, an agent may nevertheless be able to make inferences from its observations, that enable it to deduce facts that it is not permitted to know. In particular, a class of system design flaws, referred to as covert channels, provide unintended ways for information to flow between agents, rendering a system insecure.

Defining what it is for a system to satisfy an information flow security policy has proved to be a subtle matter. A substantial literature has developed that provides a range of formal systems models and a range of definitions of security. In particular, in nondeterministic systems it has been found necessary to clarify the attack model, and distinguish between a passive attacker, which merely aims to deduce secret information from observations it is able to make from its position outside the security domain to be protected, and a more active attacker, that may have planted a Trojan Horse in the domain to be protected, and which seeks to use covert channels to pass information out of this domain. While this distinction turns out not to matter in asynchronous systems, in synchronous settings, it leads to two different definitions of security, known as Nondeducibility

⋆ Author supported by a Marie Curie International Outgoing Fellowship within the 7th European Community Framework Programme.

⋆⋆ Work supported by an Australian Research Council Discovery grant.

on Inputs (NDI), and Nondeducibility on Strategies (NDS). (The term strategies in the latter refers to the strategies that a Trojan Horse may employ to pass information out of the security domain.) Considerations of proof methods for security, and compositionality of these methods, has lead to the introduction of further definitions of security, such as *unwinding relations* and the associated definition of *restrictiveness* (RES).

One of the dimensions along which it makes sense to evaluate a definition of security is the practicality of verification techniques it enables. The early literature on the topic dealt primarily with theorem proving verification methods, but in recent years the feasibility of automated verification techniques has begun to be investigated. This recent work on automated verification of security has dealt primarily with asynchronous systems models. In this paper we investigate the complexity of automated verification for a range of definitions of information flow in a *synchronous* systems model, in which agents are aware of a global clock and may use timing information in their deductions. This model is significant in that a number of timing-based attacks have been demonstrated that, e.g., enable cryptographic keys to be deduced just from the amount of time taken to perform cryptographic operations [Koc96]. It is therefore desirable that systems designs are free of timing-based covert channels; the asynchronous definitions of security that have been the focus of much of the literature fail to ensure this.

We study three definitions of security in this paper: synchronous versions of Nondeducibility on Inputs (NDI), Nondeducibility on Strategies (NDS) and an unwinding based definition (RES). We consider just a two-agent setting, with agents L for a low security domain and H for a high security domain, and the (classical) security policy that permits H to know about L's activity, but prohibits L from knowing about the activity of H. We show that all three definitions are decidable in finite state systems, and with complexities of PSPACE-complete for NDI, EXPSPACE-complete for NDS, and polynomial time for RES.

The structure of the paper is as follows. Section 2 introduces our systems model, the definitions of security that we study, and states the main results of the paper. The following sections discuss the proofs of these results. Section 3 deals with Nondeducibility on Inputs, Section 4 deals with Nondeducibility on Strategies, and Section 5 deals with the unwinding-based definition. Related literature is discussed in Section 6, and Section 7 makes some concluding remarks.[1]

2 Semantic Model and Definitions

We work with a synchronous, nondeterministic state machine model for two agents, H and L. At each step of the computation, the agents (simultaneously) perform an action, which is resolved nondeterministically into a state transition. Both agents make (possibly incomplete) observations of the state of the system, and do so with awareness of the time.

[1] Proof details excluded due to space limitations will appear in the full version.

A *synchronous machine* M is a tuple of the form $\langle S, A, s_0, \rightarrow, O, obs\rangle$ where

- S is the set of states,
- $A = A_H \times A_L$ is a set of joint actions (or joint inputs), each composed of an action of H from the set A_H and an action of L from the set A_L,
- s_0 is the initial state,
- $\rightarrow \subseteq S \times A \times S$ defines state transitions resulting from the joint actions,
- O is a set of observations,
- $obs : S \times \{H, L\} \rightarrow O$ represents the observations made by each agent in each state.

We write obs_u for the mapping $obs(\cdot, u) : S \rightarrow O$, and $s \xrightarrow{a} s'$ for $\langle s, a, s'\rangle \in \rightarrow$. We assume that machines are *input-enabled*, by requiring that for all $s \in S$ and $a \in A$, there exists $s' \in S$ such that $s \xrightarrow{a} s'$. We write $\mathbb{M}^s$ for the set of synchronous machines.

A *run* r of M is a finite sequence $r = s_0 a_1 s_1 \ldots a_n s_n$ with: $a_i \in A$ and $s_i \xrightarrow{a_{i+1}} s_{i+1}$ for all $i = 0 \ldots n-1$. We write $\mathcal{R}(M)$ for the set of all runs of M. We denote the sequence of joint actions $a_1 \ldots a_n$ in the run r by $Act(r)$. For each agent $u \in \{H, L\}$ we define $p_u : A \rightarrow A_u$ to be the projection of joint actions onto agent u's actions. We write $Act_u(r)$ for the sequence of agent u's actions in $Act(r)$, e.g., if $Act(r) = a_1 \ldots a_n$ then $Act_u(r) = p_u(a_1) \ldots p_u(a_n)$.

For a sequence w, and $1 \leq i \leq |w|$, we write w_i for the i-th element of w, and $w[i]$ for the prefix of w up to the i-th element. We assume agents have a *synchronous* view of the machine, making an observation at each moment of time and being aware of each of their own actions (but not the actions of the other agent, which are given simultaneously and independently). Given a synchronous machine M, and $u \in \{H, L\}$, we define the mappings $\mathtt{view}_u : \mathcal{R}(M) \rightarrow O(A_u O)^*$ by:

$$\mathtt{view}_u(s_0 a_1 s_1 a_2 \cdots a_n s_n) = obs_u(s_0)\, p_u(a_1)\, obs_u(s_2)\, p_u(a_2) \cdots p_u(a_n)\, obs_u(s_n).$$

Intuitively, this says that an agent's view of a run is the history of all its state observations as well as its own actions in the run. We say that a sequence v of observations and actions is a *possible* u *view in a system* M if there exists a run r of M such that $v = \mathtt{view}_u(r)$. The mapping $\mathtt{view}_u$ extends straightforwardly to sets of runs $R \subseteq \mathcal{R}(M)$, by $\mathtt{view}_u(R) = \{\mathtt{view}_u(r) \mid r \in R\}$. We define the length $|v|$ of a view v to be the number of actions it contains.

We remark that the model is sufficiently expressive to represent an alternate model in which agents act in turn under the control of a scheduler. We say that a machine is *scheduled* if for each state $s \in S$ either

- for all actions $a \in A_H$ and $b, b' \in A_L$, and states $t \in S$, $s \xrightarrow{(a,b)} t$ iff $s \xrightarrow{(a,b')} t$, or
- for all actions $a, a' \in A_H$ and $b \in A_L$, and states $t \in S$, $s \xrightarrow{(a,b)} t$ iff $s \xrightarrow{(a',b)} t$.

This definition says that state transitions in a scheduled machine are determined by the actions of *at most one* of the agents (the agent scheduled at that state);

the other agent has no control over the transition. The model involving machines under the control of a scheduler of [vdMZ08], in which at most one agent acts at each step of the computation, can be shown to be interpretable as scheduled synchronous machines.

We consider a number of different notions of information flow security. Each definition provides an interpretation for the security policy $L \rightarrow H$, which states that information is permitted to flow from L to H, but not from H to L. Our definitions are intended for synchronous systems, in which the agents share a clock and are able to make deductions based on the time. (Much of the prior literature has concentrated on asynchronous systems, in which an agent may not know how many actions another agent has performed.) The first definition we consider states that L should not be able to infer H actions from its view.

Definition 1. *A synchronous machine M satisfies Non-Deducibility on Inputs ($M \in \mathtt{NDI}$) if for every possible L view v in M and every sequence of H actions $\alpha \in A_H^*$ with $|\alpha| = |v|$, there exists a run $r \in \mathcal{R}(M)$ such that $Act_H(r) = \alpha$ and $\mathtt{view}_L(r) = v$.*

Intuitively, in a synchronous system, L always knows how many actions H has performed, since this is always identical to the number of actions that L has itself performed. In particular, if L has made view v, then L knows that H has performed $|v|$ actions. The definition says that the system is secure if this is *all* that L can learn about what sequence of actions H has performed. Whatever L observes is consistent with any sequence of actions by H of this length. More precisely, define $K_L(v)$ for an L view v to be the set of H action sequences $Act_H(r)$ for r a run with $v = \mathtt{view}_L(r)$; this represents what L knows about H's actions in the run. Then $M \in \mathtt{NDI}$ iff for all possible L views v we have $K_L(v) = A_H^{|v|}$.

The definition of NDI takes the viewpoint that a system is secure if it is not possible for L to make any nontrivial deductions about H behaviour, provided that H does not actively seek to communicate information to L. This is an appropriate definition when H is trusted not to deliberately act so as to communicate information to L, and the context is one where H is equally likely to engage in any of its possible behaviours. In some circumstances, however, NDI proves to be too weak a notion of security. In particular, this is the case if the attack model against which the system must be secure includes the possibility of Trojan Horses at the H end of the system, which must be prevented from communicating H secrets to L. The following example, due in essence to Wittbold and Johnson [WJ90] shows that it is possible for a system to satisfy NDI, but still allow for L to deduce H information.

Example 1. We present a synchronous machine that satisfies NDI in Fig. 1. We use the convention in such figures that the observations are shown on a state s in the form of $obs_H(s)/obs_L(s)$. Edges are labelled with joint actions (a, b) where $a \in A_H$ and $b \in A_L$. When a is x this means that there is such an edge for all $a \in A_H$. In this example the action sets are $A_H = \{0, 1\}$, $A_L = \{0\}$. Note that in state s_1 and s_2, L's observation in the next state is determined as the *exclusive-or* of H's current observation and H's action. The system is in

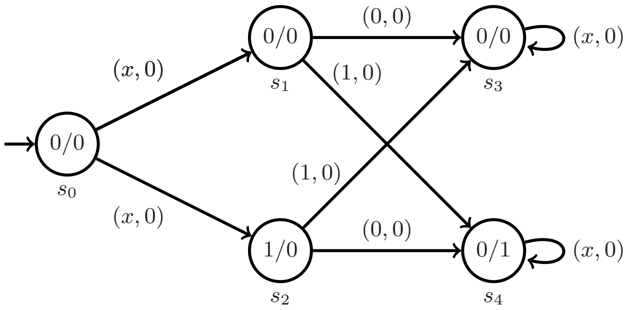


Fig. 1. A synchronous machine in NDI, but not in NDS, where $x \in \{0, 1\}$

NDI since every H action sequence is compatible with every L view of the same length. For example, the L view 00000 is consistent with H action sequence $x0$ (path $s_0s_1s_3$) and with H action sequence $x1$ (path $s_0s_2s_3$). Nevertheless, H can communicate a bit b of information to L, as follows. Note that H is able to distinguish between state s_1 and s_2 by means of the observation it makes on these states (at time 1). Suppose $b = 1$, then H chooses action 1 at s_1 and action 0 at s_2; in either case the next state is s_4, and L observes 1. Alternately, if $b = 0$, then H chooses action 0 at s_1 and action 1 at s_2; in either case the next state is s_3, and L observes 0. Whatever the value of b, H has guaranteed that L observes b at time 2, so this bit has been communicated. Intuitively, this means that the system is unable to block Trojan Horses at H from communicating with L, even though it satisfies NDI. (The structure can be repeated so that H can communicate a message of any length to L in plaintext.) □

The essence of this example is that L is able to deduce H secrets based not just on its knowledge of the system, but also its knowledge that H is following a particular strategy for communication of information to L. In response to this example, Wittbold and Johnson proposed the following stronger definition of security. First, define an H *strategy* in a system M to be a function π mapping each possible view of H in M to an H action. Intuitively, H's behaviour must depend on what H has been able to observe in the system. Say that a run $r = s_0a_1s_1 \dots a_ns_n$ is *consistent with* an H strategy π if for all $i = 0 \dots n-1$, we have $p_H(a_{i+1}) = \pi(\mathtt{view}_H(s_0a_1s_1 \dots a_is_i))$. We write $\mathcal{R}(M, \pi)$ for the set of runs of M that are consistent with the H strategy π.

Definition 2. *A synchronous system M satisfies* Nondeducibility on Strategies *($M \in$ NDS), if for all H strategies π_1, π_2 in M, we have* $\mathtt{view}_L(\mathcal{R}(M, \pi_1)) = \mathtt{view}_L(\mathcal{R}(M, \pi_2))$.

Intuitively, this definition says that the system is secure if L is not able to distinguish between different H strategies by means of its views. In Example 1, H uses a strategy when $b = 1$ that produces the L view 00001 that is not produced when H uses the strategy for $b = 0$. Thus, the sets of L views differ for these two strategies, so the system is not in NDS.

An alternate formulation of the definition can be obtained by noting that for every possible L view v, there is an H strategy π such that $v \in \mathtt{view}_L(\mathcal{R}(M,\pi))$, viz., if $v = \mathtt{view}_L(r)$, we take π to be a strategy that always performs the same action at each time $i < |r|$ as H performs at time i in r. Thus, we can state the definition as follows:

Proposition 1. *$M \in \mathtt{NDS}$ iff for all H strategies π in M, we have $\mathtt{view}_L(\mathcal{R}(M,\pi)) = \mathtt{view}_L(\mathcal{R}(M))$.*

This formulation makes clear that H cannot communicate any information to L by means of its strategies. It is also apparent that allowing H strategies to be nondeterministic (i.e., functions from H views to a *set* of H actions) would not lead to a different definition of NDS, since the more choices H has in a strategy the more L-views are compatible with that strategy. We remark that in asynchronous systems (in which we use an asynchronous notion of view), similarly defined notions of non-deducibility on inputs and non-deducibility on strategies turn out to be equivalent [FG95, vdMZ06]. The example above shows that this is not the case in synchronous machines, where the two notions are distinct.

Nondeducibility-based definitions of security are quite intuitive, but they turn out to have some disadvantages as a basis for secure systems development. One is that they are not compositional: combining two systems, each secure according to such a definition, can produce a compound system that is not secure [McC88]. For this reason, some stronger, but less intuitive definitions have been advocated in the literature.

One of these, McCullough's notion of restrictiveness [McC88], is closely related to an approach to formal proof of systems security based on what are known as "unwinding relations." A variety of definitions of unwinding relations have been proposed in the literature [GM84, Rus92, Man00b, BFPR03], in the context of a number of different underlying systems models and associated definitions of security for which they are intended to provide a proof technique. We propose here a variant of such definitions that is appropriate to the machine model we consider in this paper, drawing on definitions proposed by van der Meyden and Zhang [vdMZ08] for machines acting under the control of a scheduler.

A *synchronous unwinding relation* on a system M is a symmetric relation $\sim \subseteq S \times S$ satisfying the following:

- $s_0 \sim s_0$,
- $s \sim t$ implies $obs_L(s) = obs_L(t)$.
- $s \sim t$ implies for all $a_1, a_2 \in A_H$ and $a_3 \in A_L$, if $s \xrightarrow{(a_1,a_3)} s'$ then there exists a state t' such that $t \xrightarrow{(a_2,a_3)} t'$, and $s' \sim t'$.

Intuitively, an unwinding relation is a bisimulation-like relation over S that shows L observations are locally uncorrelated with H actions.

Definition 3. *A synchronous machine M satisfies restrictiveness ($M \in \mathtt{RES}$), if there exists a synchronous unwinding relation on M.*

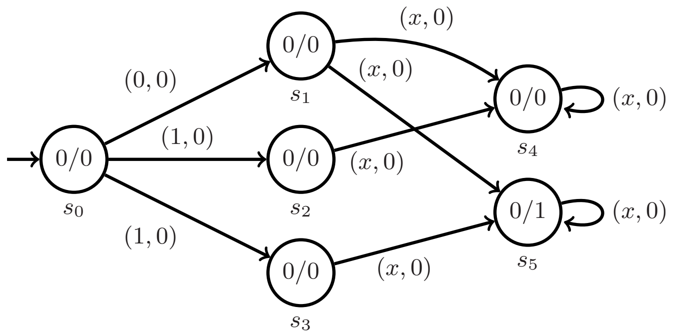


Fig. 2. A synchronous machine M in NDS, but not in RES, where $x \in \{0,1\}$. Every state s is labelled with a pair $obs_H(s)/obs_L(s)$.

Part of the significance of RES is that it provides a proof technique for our notions of nondeducibility, as shown by the following result, which relates the three notions of security we have introduced:

Theorem 1. *The following containments hold and are strict:* RES $\subset$ NDS $\subset$ NDI.

Example 2. We present a machine in fig. 2 that satisfies NDS but does not satisfy RES. In this system we let $A_H = \{0,1\}$, $A_L = \{0\}$. We use the conventions from Example 1. One may easily observe that L's view is in the pattern of $000((00)^* + (01)^*)$ all of which are compatible with every possible H strategy. However, there does not exist a synchronous unwinding relation to relate s_0 to s_0. Suppose there were such a relation $\sim$ such that $s_0 \sim s_0$, then for joint actions $(0,0)$ and $(1,0)$, we have $s_0 \xrightarrow{(0,0)} s_1$, $s_0 \xrightarrow{(1,0)} s_2$ and $s_0 \xrightarrow{(1,0)} s_3$, and we would require s_1 to be related to either s_2 or s_3. However, neither s_2 nor s_3 can be related to s_1: from s_2 user L can only observe $(00)^*$ in the future, and from s_3 only $(01)^*$ can be observed by L. Note from s_1 both $(00)^*$ and $(01)^*$ are possible for L. □

Our main contribution in this paper is to characterise the complexity of checking the three notions of security we have introduced above. The results are given in the following theorem:

Theorem 2. *Restricted to finite state synchronous machines, and with respect to PTIME reductions,*

1. NDI *is PSPACE-complete,*
2. NDS *is EXPSPACE-complete, and*
3. RES *is in PTIME.*

We remark that the lower bound results for NDI and NDS require only scheduled machines, so these problems are already PSPACE-hard and EXPSPACE-hard, respectively, on this subclass. We describe the proof of these three claims in the following three sections, in turn.

3 Synchronous Nondeducibility on Inputs

In this section we establish the complexity of `NDI`, Theorem 2(1).

3.1 `NDI`: Upper Bound

Stating the definition in the negative, a system is not in `NDI` if there exists an L view v and a sequence of H actions α with $|\alpha| = |v|$ such that there exists no run r with $Act_H(r) = \alpha$ and $\mathtt{view}_L(r) = v$. We show that `NDI` is decidable by a procedure that searches for such an L view v and H action sequence α. The key element of the proof is to show that we need to maintain only a limited amount of information during this search, so that we can bound the length of the witness (v, α), and the amount of space needed to show that such a witness exists.

To show this, given a sequence $\alpha \in A_H^*$ and a possible L view v, with $|\alpha| = |v|$, we define the set $K(\alpha, v)$ to be the set of all final states of runs r such that $Act_H(r) = \alpha$ and $\mathtt{view}_L(r) = v$. For the system M define the labelled transition system $L(M) = (Q, q_0, \Rightarrow)$ as follows:

1. $Q = S \times \mathcal{P}(S)$,
2. $q_0 = (s_0, \{s_0\})$,
3. $\Rightarrow$ is the labelled transition relation on Q with labels in $A_H \times A_L \times A_H$, defined by $(s, T) \Rightarrow^{(a,b,a')} (s', T')$ if $a \in A_H$, $b \in A_L$, $a' \in A_H$ such that $s \xrightarrow{(a,b)} s'$ and $T' = \{t' \in S \mid \text{for some } t \in T \text{ we have } t \xrightarrow{(a',b)} t' \text{ and } obs_L(t') = obs_L(s')\}$.

Intuitively, the component s in a state $(s, T) \in Q$ is used to ensure that we generate an L view v that is in fact possible. The components a, b in a transition $(s, T) \Rightarrow^{(a,b,a')} (s', T')$ represent the actions used to generate the run underlying v, and the component a' is used to generate a sequence α. The set T represents $K(\alpha, v)$. More precisely, we have the following result:

Lemma 1. *If $q_0 \Rightarrow^{(a_1,b_1,a_1')} (s_1, T_1) \Rightarrow \cdots \Rightarrow^{(a_n,b_n,a_n')} (s_n, T_n)$, then the sequence $v = obs_L(s_0) b_1 obs_L(s_1) \ldots b_n obs_L(s_n)$ is a possible L view, and $\alpha = a_1' \ldots a_n'$ is a sequence of H actions such that $|v| = |\alpha|$ and $K(\alpha, v) = T_n$.*

Conversely, for every possible L view v with $|v| = n$, and sequence of H actions $\alpha = a_1' \ldots a_n'$, there exists a path $q_0 \Rightarrow^{(a_1,b_1,a_1')} (s_1, T_1) \Rightarrow \cdots \Rightarrow^{(a_n,b_n,a_n')} (s_n, T_n)$ such that $v = obs_L(s_0) b_1 obs_L(s_1) \ldots b_n obs_L(s_n)$ and $K(\alpha, v) = T_n$.

We now note that for an H action sequence α and a possible L view v, with $|v| = |\alpha|$, there exists no run r such that $Act_H(r) = \alpha$ and $\mathtt{view}_L(r) = v$ iff $K(\alpha, v) = \emptyset$. The existence of such a pair (α, v), is therefore equivalent, by Lemma 1, to the existence of a path in $L(M)$ from q_0 to a state (s, T) with $T = \emptyset$. This can be decided in $NSPACE(O(|M|)) = DSPACE(O(|M|^2)) \subseteq PSPACE$. This proves the following theorem.

Theorem 3. *$M \in$ `NDI` is decidable in $PSPACE$.*

We note, moreover, that since there are at most $|S| \times 2^{|S|}$ states in Q, if there exists a pair (α, v) witnessing that $M \notin$ `NDI` there exists such a pair with $|\alpha| \leq |S| \times 2^{|S|}$.

3.2 NDI: Lower Bound

We show that NDI is PSPACE-hard already in the special case of scheduled machines. The proof is by a polynomial time reduction from the problem of deciding if the language $L(\mathcal{A})$ accepted by a nondeterministic finite state automaton $\mathcal{A}$ on alphabet Σ is equal to $\Sigma^* \setminus \{\varepsilon\}$. This is easily seen to be a PSPACE-hard problem, since testing $L(\mathcal{A}) = \Sigma^*$ is known to be PSPACE-hard [SM73].

Let $\mathcal{A} = \langle Q, Q_0, \Sigma, \delta, F\rangle$ be a nondeterministic finite state automaton (without ϵ-transitions), with states Q, initial states $Q_0 \subseteq Q$, alphabet Σ, transition function $\delta : Q \times \Sigma \to \mathcal{P}(Q)$, and final states F. Without loss of generality, we assume $Q_0 \cap F = \emptyset$.[2] We define $M(\mathcal{A}) = \langle S, A, s_0, \to, obs, O\rangle$ to be a scheduled machine, and use a function $sched : S \to \{H, L\}$ to indicate the agent (if any) whose actions determine transitions. In view of this, when $sched(s) = u$ and $a \in A_u$, we may write $s \xrightarrow{a} t$ to represent that $s \xrightarrow{b} t$ for all joint actions b with $p_u(b) = a$. The components of $M(A)$ are defined as follows.

- $S = Q \cup \{s_0, s_1, s_2, s_3\}$, where $Q \cap \{s_0, s_1, s_2, s_3\} = \emptyset$,
- $sched(s_0) = H$ and $sched(s) = L$ for all $s \in S \setminus \{s_0\}$,
- $A = A_H \cup A_L$ where $A_L = \Sigma$ and $A_H = \{h, h'\}$,
- $O = \{0, 1\}$,
- $obs : \{H, L\} \times S \to O$ with $obs_H(s) = 0$ for all $s \in S$ and $obs_L(s) = 0$ for all $s \in S \setminus \{s_2\}$, and $obs_L(s_2) = 1$.
- $\longrightarrow \subseteq S \times A \times S$ is defined as consisting of the following transitions (using the convention noted above)
  - $s_0 \xrightarrow{h} q$ for all $q \in Q_0$, and $s_0 \xrightarrow{h'} s_1$.
  - $s_1 \xrightarrow{a} s_1$ and $s_1 \xrightarrow{a} s_2$ for all $a \in \Sigma$,
  - $s_2 \xrightarrow{a} s_2$ for all $a \in \Sigma$,
  - $s_3 \xrightarrow{a} s_3$ for all $a \in \Sigma$,
  - for $q, q' \in Q$ and $a \in A_L = \Sigma$ we have $q \xrightarrow{a} q'$ for all $q' \in \delta(q, a)$
  - for $q \in Q$ and $a \in A_L = \Sigma$ such that $\delta(q, a) \cap F \neq \emptyset$, we have $q \xrightarrow{a} s_2$,
  - for $q \in Q$ and $a \in A_L = \Sigma$ such that $\delta(q, a) = \emptyset$, we have $q \xrightarrow{a} s_3$.

The construction of $M(\mathcal{A})$ from $\mathcal{A}$ can be done in polynomial time.

Proposition 2. *$L(\mathcal{A}) = \Sigma^* \setminus \{\varepsilon\}$ iff $M(\mathcal{A}) \in$ NDI.*

Proof. Intuitively, the runs of $M(\mathcal{A})$ produce two sets of L views. Runs in which H does h' in the first step produce all L views in $0\Sigma 0(\Sigma 0)^*(\Sigma 1)^*$. Runs in which H does h in the first step correspond to simulations of $\mathcal{A}$ and produce L views in $0\Sigma 0(\Sigma 0)^*$ or of the form $0\Sigma 0 a_1 0 \ldots a_{n-1} 0 a_n 1(\Sigma 1)^*$ with $a_1 \ldots a_n \in L(\mathcal{A})$. Note that L may obtain any view in $0\Sigma 0(\Sigma 0)^*$ by means of a run that stays in Q for as long as possible, and moves to s_3 whenever an action is not enabled. Views in $0\Sigma 0 a_1 0 \ldots a_{n-1} 0 a_n 1(\Sigma 1)^*$ come from runs that pass through Q and then jump to s_2. Note that since H is not scheduled after the first step, replacing

[2] This is just to let $\varepsilon \notin L(\mathcal{A})$. If not, we apply unfolding on the initial states, and study the resulting automaton which accepts the language $L(\mathcal{A}) \setminus \{\varepsilon\}$.

any action by H after the first step in a run by any other action of H results in another run, with no change to the L view. Thus the only thing that needs to be checked to determine whether $M(\mathcal{A}) \in \texttt{NDI}$ is whether the same can be said for the first step.

- For the 'only if' part, suppose $L(\mathcal{A}) = \Sigma^* \setminus \{\varepsilon\}$. We show that $M(\mathcal{A}) \in \texttt{NDI}$. Let $r = s_0(b_1, a_1)t_1 \ldots (b_n, a_n)t_n$ be a run of $M(\mathcal{A})$, with the $b_i \in A_H$ and the $a_i \in A_L$. Let $b'_1 \ldots b'_n$ be any sequence of actions in A_H. If $b'_1 = h'$, then it is clear that we can find a run $r = s_0(b'_1, a_1)t'_1 \ldots (b'_n, a_n)t'_n$ with $\texttt{view}_L(r) = \texttt{view}_L(r')$. On the other hand, if $b'_1 = h$ then the same is true, since $L(\mathcal{A}) = \Sigma^* \setminus \{\varepsilon\}$. Thus, the run required by $M(\mathcal{A}) \in \texttt{NDI}$ has been shown to exist.
- For the 'if' part, suppose there is a word $w = a_1a_2 \ldots a_n \notin L(\mathcal{A})$. Then for an arbitrary $a_0 \in \Sigma$, the L view $0a_00a_10a_2 \ldots a_n1$ cannot be obtained from runs in which the first H action is h, because otherwise w would be accepted by $\mathcal{A}$. However this view is obtained from a run in which the first action is h'. Therefore $M(\mathcal{A}) \notin \texttt{NDI}$. □

4 Nondeducibility on Strategies

In this section we establish the complexity of NDS, Theorem 2(2).

4.1 NDS: Upper Bound

For the proof that NDS is decidable in EXPSPACE, we show that the problem is in DSPACE($2^{O(n)}$).

We use characterization of NDS given in Proposition 1. Let π be an H strategy, let α be an H view, and let β be an L view, with $|\alpha| \leq |\beta|$. Say that π *excludes* β if there does not exist a run r consistent with π such that $\beta = \texttt{view}_L(r)$. Since always $\mathcal{R}(M, \pi) \subseteq \mathcal{R}(M)$, by Proposition 1, a system M satisfies NDS if it is not the case that there exists a possible L view β in M and a strategy π such that π *excludes* β. We first give a lemma that enables strategies excluding a particular L view to be put into a uniform structure.

Given an H view $\alpha \in \texttt{view}_H(\mathcal{R}(M, \pi))$, define $K(\alpha, \pi, \beta)$ to be the set of all final states of runs r consistent with π such that $\texttt{view}_H(r) = \alpha$ and $\texttt{view}_L(r)$ is a prefix of β.

Lemma 2. *If there exists a strategy π that excludes β, then there exists a strategy π' that also excludes β, and has the property that $K(\alpha, \pi', \beta) = K(\alpha', \pi', \beta)$ and $|\alpha| = |\alpha'|$ implies $\pi'(\alpha) = \pi'(\alpha')$ for all H views α and α'.*

Proof. Suppose that π excludes β. For purposes of the proof, note that we can assume without loss generality that β is infinite — this helps to avoid mention of views longer than β as a separate case.[3] It is convenient to consider K

[3] Note that it is equivalent to say that π excludes some prefix of β.

and strategies π to be defined over the larger set $P = O(A_H O)^*$ rather than $\texttt{view}_H(\mathcal{R}(M,\pi))$. In case of K, we take $K(\alpha,\pi,\beta) = \emptyset$ when there is no run r consistent with π such that $\texttt{view}_H(r) = \alpha$ and $\texttt{view}_L(r)$ is a prefix of β.

Let f be any mapping from P to P such that for all $\alpha, \alpha' \in P$ we have (1) $|f(\alpha)| = |\alpha|$ and (2) $K(\alpha,\pi,\beta) = K(f(\alpha),\pi,\beta)$, and (3) if $|\alpha| = |\alpha'|$ and $K(\alpha,\pi,\beta) = K(\alpha',\pi,\beta)$, then $f(\alpha) = f(\alpha')$. Such a mapping always exists; intuitively, it merely picks, at each length, a representative $f(\alpha) \in [\alpha]_\sim$ of the equivalence classes of the equivalence relation defined by $\alpha \sim \alpha'$ if $K(\alpha,\pi,\beta) = K(\alpha',\pi,\beta)$.

Now define the mapping g on P as follows. Let $\alpha_0 = O_H(s_0)$ be the only possible H view of length 0. For $\alpha \in P$ of length 0, we define $g(\alpha) = \alpha_0$ if $\alpha = \alpha_0$ and $g(\alpha) = \alpha$ otherwise. For longer α, we define $g(\alpha a o) = f(g(\alpha))\pi(f(g(\alpha))o$. Also, define the strategy π' by $\pi'(\alpha) = \pi(f(g(\alpha)))$.

We claim that for all $\alpha \in P$ we have $K(\alpha,\pi',\beta) = K(g(\alpha),\pi,\beta)$. The proof is by induction on the length of α. The base case is straightforward, since α_0 is consistent with all strategies, so $K(\alpha_0,\pi',\beta) = \{s_0\} = K(\alpha_0,\pi,\beta)$, and $K(\alpha,\pi',\beta) = \emptyset = K(\alpha,\pi,\beta)$ for $\alpha \neq \alpha_0$. Suppose the claim holds for $\alpha \in P$ of length i. Let $\alpha a o \in P$. By induction and (2), $K(\alpha,\pi',\beta) = K(g(\alpha),\pi,\beta) = K(f(g(\alpha)),\pi,\beta)$. Since action $a = \pi'(\alpha) = \pi(f(g(\alpha))$, $K(\alpha a o,\pi',\beta)$ is equal to $K(f(g(\alpha))ao,\pi,\beta) = K(g(\alpha a o),\pi,\beta)$, as required.

To see that π' has the required property, if $K(\alpha,\pi',\beta) = K(\alpha',\pi',\beta)$ with $|\alpha| = |\alpha'|$, then we have $K(g(\alpha),\pi,\beta) = K(g(\alpha'),\pi,\beta)$. By (3) we have $f(g(\alpha)) = f(g(\alpha'))$. Therefore $\pi'(\alpha) = \pi(f(g(\alpha))) = \pi(f(g(\alpha'))) = \pi'(\alpha')$ by definition.

Since π excludes β, there exists a length n such that for all $\alpha \in P$ with $|\alpha| = n$, we have $K(\alpha,\pi,\beta) = \emptyset$. Thus, we also have for all α of length n that $K(\alpha,\pi',\beta) = K(g(\alpha),\pi,\beta) = \emptyset$. This means that π' also excludes β. $\square$

Based on Lemma 2, we construct a transition system $(Q, q_0 \Rightarrow)$ that simultaneously searches for the strategy π and an L view β that is omitted by π. The states Q are sets of sets $k \subseteq S$. The initial state q_0 is $\{\{s_0\}\}$. We use an "update" function $\delta_{a_L,o_L,a_H,o_H} : \mathcal{P}(S) \to \mathcal{P}(S)$, for each $a_L \in A_L$, $a_H \in A_H$, and $o_L, o_H \in O$, defined by

$$\delta_{a_L,o_L,a_H,o_H}(k) = \{t \in S \mid \text{there exists } s \in k \text{ with } s \xrightarrow{(a_L,a_H)} t \text{ and } obs_H(t) = o_H \text{ and } obs_L(t) = o_L\}.$$

The transitions are defined as follows: $q \Rightarrow^{(\rho,a_L,o_L)} q'$ if $\rho : q \to A_H$, $a_L \in A_L$ and $o_L \in O$, and $q' = \{\delta_{a_L,o_L,a_H,o_H}(k) \mid k \in q \text{ and } a_H = \rho(k) \text{ and } o_H \in O\}$. Intuitively, each state q represents a collection of all possible knowledge sets that H can be in at a certain point of time, while attempting to omit some sequence β. More specifically, each set k in $q \in Q$ corresponds to an H view α such that $k = K(\alpha,\pi,\beta)$. In a transition, we both determine the next phase of π, by extending π so that $\pi(\alpha) = \rho(K(\alpha,\pi,\beta))$, and extend β to $\beta a_L o_L$.

The following results justify the correspondence between this transition system and NDS.

Lemma 3. *There exists a strategy π and a Low view β such that π excludes β, iff there exists a path $q_0 \Rightarrow^* q_n = \{\emptyset\}$.*

We obtain the claimed complexity bound from Lemma 3, simply by noting that it reduces `NDS` to a reachability problem in the transition system. Since the states of the system can be represented in space $|S| \cdot 2^{|S|} = 2^{O(|S|)}$, we obtain from Savitch's theorem that we can do the search in DSPACE($2^{O(|S|)}$).

4.2 `NDS`: Lower Bound

To show that `NDS` is EXPSPACE-hard, we show how to encode the game BLIND-PEEK of Reif [Rei84]. We need only scheduled machines for the encoding, so the problem is EXPSPACE-hard already for this subclass.

BLIND-PEEK is a variant of the two-player game PEEK introduced by Stockmeyer and Chandra [SC79]. A PEEK game consists of a box with two open sides and containing horizontally stacked plates; the players sit at opposite sides of the box. Each plate has two positions, 'in' and 'out', and contains a knob at one side of the box, so that this plate can be controlled by one of the players. At each step, one of the two players may grasp a knob from his side and push it 'in' or 'out'. The player may also pass. Both the top of the box and the plates have holes in various positions, and each hole is associated to a player. If, just after a move of player $a \in \{1, 2\}$, the plates are positioned so that for one of the player's holes, each plate has a hole positioned directly underneath that hole, so that the player can peek through a sequence of holes from the top of the box to the bottom, then player a wins. In PEEK, both players can observe the position of all plates at all times. BLIND-PEEK [Rei84] is a modification of PEEK in which player 1's side of the box is partially covered, so that it is not possible for player 1 to see the positions of the plates controlled by player 2. Deciding whether there exists a winning strategy for player 1 in a PEEK game is EXPTIME-hard, and it is EXPSPACE-hard in the case of BLIND-PEEK. We make a reduction from a BLIND-PEEK game to achieve the lower bound result for `NDS`.

Due to space limitations, we just give a brief sketch of the construction. Given an instance G of BLIND-PEEK, we construct a synchronous system $M(G)$, with the following property: player 1 has a winning strategy in G iff there exists an L view v_L and an H strategy π that excludes v_L in $M(G)$. Note that since player 1 plays blindfold in G, a player 1 strategy can be represented as simply a sequence of player 1 moves, rather than a function from player 1 views to player 1 moves. This sequence of player 1 moves will be encoded in the L view v_L. Nondeterminism will be used to represent the universal behaviour of player 2, and also to guess certain aspects of game state transitions. The role of the H strategy π in the encoding will be to perform certain checking operations. We make use of the sets $K(v_H, \pi, v_L)$ to represent states of G. H will observe all actions of H and L in the game, so H is always aware of the game state. However, there remains some uncertainty in H's knowledge of the state of the system $M(G)$ — we use this to represent the positions of the n plates in the game as a set of n states of the system $M(G)$.

5 Synchronous Bisimulation-Based Notions

In this section we establish the complexity of RES, Theorem 2(3). We first note:

Lemma 4. *1. The largest synchronous unwinding relation is transitive.*
2. If all states in M are reachable then the largest synchronous unwinding relation $\sim$ is an equivalence relation.
3. A system satisfies RES *iff its restriction to its reachable states satisfies* RES.

It is not hard to show that finding the largest synchronous unwinding relation (if any exists) on the reachable fragment of a machine $\langle S, A, s_0, \rightarrow, O, obs\rangle$ can be done in polynomial time. The following algorithm, which works in $\mathcal{O}(|S|^3 \times |\rightarrow|)$ time, resembles the algorithm for calculating the relational coarsest partition by Kanellakis and Smolka [KS83].

Algorithm 1 *Let $S_o = \{s \in S \mid obs_L(s) = o\}$, and the initial partition $P_0 = \{S_o \mid o \in O\} \cup p_{bad}$ with $p_{bad} = \emptyset$. We repeat the following for all $i \geq 0$ until termination. Try every $p_1, p_2 \in P_i$ in the following transformation rules:*

1. *If there exist $s \in p_1$, $a_1, a_2 \in A_H$ and $a_3 \in A_L$ such that (1) there exist $s \xrightarrow{(a_1,a_3)} t_1$ and $t_1 \in p_2$, and (2) $s \xrightarrow{(a_2,a_3)} t_2$ implies $t_2 \notin p_2$, then $P_{i+1} = P_i \setminus \{p_1, p_{bad}\} \cup \{p_1 \setminus \{s\}, p_{bad} \cup \{s\}\}$.*
2. *If there exist $a_1 \in A_H$ and $a_2 \in A_L$, and we can split p_1 into nonempty sets p_{11} and p_{12} such that (1) for all $s_1 \in p_{11}$, $\{t' \in S \mid s_1 \xrightarrow{(a_1,a_2)} t'\} \cap p_2 \neq \emptyset$ and (2) for all $s_2 \in p_{12}$, $\{t' \in S \mid s_2 \xrightarrow{(a_1,a_2)} t'\} \cap p_2 = \emptyset$, then we let $P_{i+1} = P_i \setminus \{p_1\} \cup \{p_{11}, p_{12}\}$.*

When neither transformation rule applies, $P_{i+1} = P_i$, and we return P_i.

The above algorithm produces the coarsest partition over S according to the definition of synchronous unwinding, which yields a relation that is not necessarily reflexive. If (s_0, s_0) is within that relation then the system is in RES. In practice, whether or not (s_0, s_0) is still within a 'good' partition can be checked on-the-fly: note that once (s_0, s_0) is moved into p_{bad}, the algorithm can immediately return false, indicating that the system is not in RES.

6 Related Work

In asynchronous machines the verification complexities of NDI and NDS are both PSPACE-complete, and RES (based on asynchronous unwinding) is in polynomial time [FG95, FG96, vdMZ07]. Interestingly, PSPACE is also the complexity result for verifying Mantel's BSP conditions [Man00a] on asynchronous finite state systems. For (asynchronous) push-down systems, the verification problem is undecidable [DHK$^+$08].

A number of works have defined notions of security for synchronous or timed systems, but fewer complexity results are known. Köpf and Basin [KB06] define a notion similar to RES and show it is PTIME decidable. Similar definitions are also used in the literature on language-based security [Aga00, VS97].

Focardi et al [FGM00] define a spectrum of definitions related to ours in a timed process algebraic setting, and state a decidability result for one of them, close to our notion tNDS. However, this result concerns an approximation to the notion tBNDC that is their real target, and they do not give a complexity result. Beauquier and Lanotte defined *covert channels* in timed systems with *tick* transitions by using strategies [BL06]. They prove that the problem of the existence of a covert channel in such systems is decidable. However, their definition of covert channel requires that H and L have strategies to enforce a system into sets of runs with projections into disjoint sets of L views. Intuitively, the induced definition on *free of covert channels* turns out to be a weaker notion than `NDS`.

7 Conclusion

We remarked above that nondeducibility-based notions of security may have the disadvantage that they do not readily support a compositional approach to secure systems development, motivating the introduction of unwinding-based definitions of security. The complexity results of the present paper can be interpreted as lending further support to the value of unwinding-based definitions. We have found that the two nondeducibility notions we have considered, while both decidable, are intractable. On the other hand, the unwinding-based notion of synchronous restrictiveness has tractable complexity. This makes this definition a more appropriate basis for automated verification of security. Even if the desired security property is nondeducibility on inputs or nondeducibility on strategies, it is sufficient to verify that a system satisfies synchronous restrictiveness, since this is a stronger notion of security. It remains to be seen whether there is a significant number of practical systems that are secure according to the nondeducibility-based notions, but for which there does not exist a synchronous unwinding. If so, then an alternate methodology needs to be applied for the verification of security for such systems.

References

[Aga00] Agat, J.: Transforming out timing leaks. In: Proc. ACM Symp. on Principles of Programming Languages, pp. 40–53 (2000)

[BFPR03] Bossi, A., Focardi, R., Piazza, C., Rossi, S.: Bisimulation and unwinding for verifying possibilistic security properties. In: Proc. Int. Conf. on Verication, Model Checking, and Abstract Interpretation, pp. 223–237 (2003)

[BL06] Beauquier, D., Lanotte, R.: Hiding information in multi level security systems. In: Dimitrakos, T., Martinelli, F., Ryan, P.Y.A., Schneider, S. (eds.) FAST 2006. LNCS, vol. 4691, pp. 250–269. Springer, Heidelberg (2007)

[DHK$^+$08] D'Souza, D., Holla, R., Kulkarni, J., Ramesh, R.K., Sprick, B.: On the decidability of model-checking information flow properties. In: Proc. Int. Conf. on Information Systems Security, pp. 26–40 (2008)

[FG95] Focardi, R., Gorrieri, R.: A classification of security properties for process algebras. Journal of Computer Security, 5–33 (1995)

[FG96] Focardi, R., Gorrieri, R.: The compositional security checker: A tool for the verification of information flow security properties. Technical Report UBLCS-96-14, Università di Bologna (August 1996)

[FGM00] Focardi, R., Gorrieri, R., Martinelli, F.: Information flow analysis in a discrete-time process algebra. In: Proc. Computer Security Foundation Workshop, pp. 170–184 (2000)

[GM84] Goguen, J.A., Meseguer, J.: Unwinding and inference control. In: Proc. IEEE Symp. on Security and Privacy, p. 75 (1984)

[KB06] Köpf, B., Basin, D.A.: Timing-sensitive information flow analysis for synchronous systems. In: Gollmann, D., Meier, J., Sabelfeld, A. (eds.) ESORICS 2006. LNCS, vol. 4189, pp. 243–262. Springer, Heidelberg (2006)

[Koc96] Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)

[KS83] Kanellakis, P.C., Smolka, S.A.: CCS expressions, finite state processes, and three problems of equivalence. In: Proc. 2nd Annual ACM Symp. on Principles of Distributed Computing, New York, NY, pp. 228–240 (1983)

[Man00a] Mantel, H.: Possiblistic definitions of security – an assembly kit. In: Proc. Computer Security Foundations Workshop, pp. 185–199 (2000)

[Man00b] Mantel, H.: Unwinding security properties. In: Cuppens, F., Deswarte, Y., Gollmann, D., Waidner, M. (eds.) ESORICS 2000. LNCS, vol. 1895, pp. 238–254. Springer, Heidelberg (2000)

[McC88] McCullough, D.: Noninterference and the composability of security properties. In: Proc. IEEE Symp. on Security and Privacy, pp. 177–186 (1988)

[Rei84] Reif, J.H.: The complexity of two-player games of incomplete information. Journal of Computer and System Science 29(2), 274–301 (1984)

[Rus92] Rushby, J.: Noninterference, transitivity, and channel-control security policies. Technical report, SRI international (December 1992)

[SC79] Stockmeyer, L.J., Chandra, A.K.: Provably difficult combinatorial games. SIAM Journal of Computing 8(2), 151–174 (1979)

[SM73] Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time (preliminary report). In: Proc. ACM Symp. on Theory of computing, pp. 1–9 (1973)

[vdMZ06] van der Meyden, R., Zhang, C.: A comparison of semantic models for noninterference. In: Dimitrakos, T., Martinelli, F., Ryan, P.Y.A., Schneider, S. (eds.) FAST 2006. LNCS, vol. 4691, pp. 235–249. Springer, Heidelberg (2007)

[vdMZ07] van der Meyden, R., Zhang, C.: Algorithmic verification on noninterference properties. ENTCS 168, 61–75 (2007)

[vdMZ08] van der Meyden, R., Zhang, C.: Information flow in systems with schedulers. In: Proc. Computer Security Foundation Symp., June 2008, pp. 301–312 (2008)

[VS97] Volpano, D.M., Smith, G.: A type-based approach to program security. In: Bidoit, M., Dauchet, M. (eds.) CAAP 1997, FASE 1997, and TAPSOFT 1997. LNCS, vol. 1214, pp. 607–621. Springer, Heidelberg (1997)

[WJ90] Wittbold, J.T., Johnson, D.M.: Information flow in nondeterministic systems. In: Proc. IEEE Symp. on Security and Privacy, pp. 144–161 (1990)

Monads Need Not Be Endofunctors

Thorsten Altenkirch[1], James Chapman[2], and Tarmo Uustalu[2]

[1] School of Computer Science, University of Nottingham
[2] Institute of Cybernetics, Tallinn University of Technology
txa@cs.nott.ac.uk, {james,tarmo}@cs.ioc.ee

Abstract. We introduce a generalisation of monads, called relative monads, allowing for underlying functors between different categories. Examples include finite-dimensional vector spaces, untyped and typed λ-calculus syntax and indexed containers. We show that the Kleisli and Eilenberg-Moore constructions carry over to relative monads and are related to relative adjunctions. Under reasonable assumptions, relative monads are monoids in the functor category concerned and extend to monads, giving rise to a coreflection between monads and relative monads. Arrows are also an instance of relative monads.

1 Introduction

Monads are the most successful programming pattern arising in functional programming. Apart from their use to model a generic notion of effect they also serve as a convenient interface to generalized notions of substitution. Research in the area on the border between category theory and functional programming focusses on unveiling new programming and reasoning constructions similar to monads, such as comonads [18], arrows [9] and idioms (closed functors) [13]. Indeed, especially when working in an expressive and total language with dependent types, such as Agda [3], we can exploit monads not only as a way to structure our programs but also their verification.

The present paper is concerned with a generalisation of monads which arises naturally in dependently typed programming, namely monad-like entities that are not endofunctors. Consider the following example, which arose when implementing notions related to quantum programming, namely finite-dimensional vector spaces [19,4]. (See also Piponi [15] for this and other interesting uses of vector spaces in functional programming.)

Example 1. In quantum computing, we consider complex vector spaces, but for the present development any semiring $(R, +, 0, \times, 1)$ is sufficient. Finite-dimensional vector spaces can be given by:

$$\begin{aligned}
&\mathsf{Vec} \in |\mathbf{Fin}| \to |\mathbf{Set}| \\
&\mathsf{Vec}\, m =_{\mathrm{df}} J_{\mathrm{f}}\, m \to R \\
&\eta \in \Pi_{m \in |\mathbf{Fin}|} J_{\mathrm{f}}\, m \to \mathsf{Vec}\, m \\
&\eta_m\, (i \in \underline{m}) =_{\mathrm{df}} \lambda j \in \underline{m}.\ \text{if}\ i = j\ \text{then}\ 1\ \text{else}\ 0 \\
&(-)^* \in \Pi_{m,n \in |\mathbf{Fin}|} (J_{\mathrm{f}}\, m \to \mathsf{Vec}\, n) \to (\mathsf{Vec}\, m \to \mathsf{Vec}\, n) \\
&A^*\, x =_{\mathrm{df}} \lambda j \in \underline{n}.\ \textstyle\sum_{i \in \underline{m}} x\, i \times A\, i\, j
\end{aligned}$$

Here **Fin** is the category of finite cardinals (the skeletal version of finite sets). The objects are natural numbers $m \in \mathbb{N}$ and the maps between m and n are functions between $\underline{m}$ and $\underline{n}$ where $\underline{m} =_{\mathrm{df}} \{0, 1, \ldots, m-1\}$. By $J_{\mathrm{f}} \in \mathbf{Fin} \to \mathbf{Set}$ we mean the natural embedding $J_{\mathrm{f}}\ m =_{\mathrm{df}} \underline{m}$. The finite summation $\sum$ is just the finite iteration of $+$ over 0. Indeed η_m is just the unit $m \times m$-matrix (alternatively, a function assigning to every coordinate $i \in \underline{m}$ the corresponding unit vector) and $A^*\, x$ corresponds to the product of the matrix A with the vector x, where both matrices and vectors are described as functions.

By the types of its data, the structure $(\mathsf{Vec}, \eta, (-)^*)$ looks suspiciously like a monad, except that **Fin** is not **Set** and in the types for η and $(-)^*$ we have used the embedding J_{f} to repair the mismatch. It is easy to verify that the structure also satisfies the standard monad laws, modulo the same discrepancy.

The category of finite-dimensional vector spaces arises as a kind of Kleisli category. Its objects are $m \in \mathbb{N}$, understood as finite coordinate systems (describing vector spaces), and its morphisms are functions $J_{\mathrm{f}} m \to \mathsf{Vec}\, n$, i.e., matrices (describing linear transformations).

The structure cannot generally be pushed to a monad on **Set**. $(-)^*$ requires that we can sum over a set. Summation over general index sets is not available, if R is just a semiring. Also, in a constructive setting, η requires that the set has a decidable equality, which is not the case for general sets.

We view Vec as a *relative monad* on the embedding $J_{\mathrm{f}} \in \mathbf{Fin} \to \mathbf{Set}$. Other examples of relative monads include untyped and simply typed λ-terms and the notions of indexed functors and indexed containers as developed in [14].

Overview of the paper. In Sect. 2 we develop the notion of relative monads on a functor $J \in \mathbb{J} \to \mathbb{C}$, showing that they arise from relative adjunctions, and generalize Kleisli and Eilenberg-Moore constructions to relative monads.

Since monads on $\mathbb{C}$ correspond to monoids in the endofunctor category $[\mathbb{C}, \mathbb{C}]$, a natural question is whether a relative monad on $\mathbb{J}$ gives rise to a monoid in the category $[\mathbb{J}, \mathbb{C}]$. If $\mathbb{J}$ is small and $\mathbb{C}$ is cocomplete (e.g., **Set**), the left Kan extension along J exists and give rise to a lax monoidal structure where the unit is J and the tensor is given by $F \cdot^J G =_{\mathrm{df}} \mathrm{Lan}_J\, F \cdot G$. Indeed, relative monads give rise to lax monoids in this lax setting (Sect. 3).

Going further, we identify conditions on the functor J, under which the lax monoidal structure induced by Lan_J is properly monoidal. In this case, we obtain a proper monoid in the category of functors. Moreover, relative monads extend to monads via Lan_J and we get a coreflection between monads and relative monads (Sect. 4). In the example of vector spaces, $\mathrm{Lan}_J\, \mathsf{Vec}$ is the monad whose Kleisli category is that of vector spaces over general sets of coordinates where a vector over an infinite set of coordinates may only have finitely many non-zero components. However, it is worthwhile not to ignore the non-endofunctor case, because frequently this is the structure we actually want to use. E.g., in quantum computing we are interested in dagger compact closed categories [2].

Finally, we show that arrows are relative monads (Sect. 5) on the Yoneda embedding. This leads to the, maybe surprising, outcome that while arrows generalize ordinary monads, they are actually a special case of relative monads.

Related work The untyped λ-calculus syntax as has been identified as a monoid in $[\mathbf{Fin}, \mathbf{Set}]$ by Fiore et al. [7]. Heunen and Jacobs [8] have shown that arrows on $\mathbb{C}$ are actually monoids in the category $[\mathbb{C}^{\mathrm{op}} \times \mathbb{C}, \mathbf{Set}]$ of endoprofunctors; Jacobs et al. have proved the Freyd construction of [16] is, in a good sense, the Kleisli construction for arrows. Spivey [17] has studied a generalization of monads, which differs from ours, but is similar in spirit and related (see Conclusion). That the monoid nature of monads is important in developing applications of monads, was recently shown by Jaskelioff in his work on modular monad transformers [11].

Notation. We will be using a mixture of categorical and type-theoretic notation. In particular we will be using λ-calculus notation for defining functions (maps in **Set** or subcategories). Customarily for both category theory and type theory, we often hide some arguments of patterns and function applications (normally subscripted arguments, e.g., an object a natural transformation is applied to).

We write $|\mathbb{C}|$ for the objects of $\mathbb{C}$ and $\mathbb{C}\,(X, Y)$ for the homsets. Given categories $\mathbb{C}$,$\mathbb{D}$ we write the functor category as $[\mathbb{C}, \mathbb{D}]$. We write id, $\circ$ for the identities and composition of maps and I, $\cdot$ for the identities and composition of functors.

2 Relative Monads and Relative Adjunctions

We start by defining relative monads. Then we give some examples and show how the theory of ordinary monads carries over to the relative case.

2.1 Relative Monads

Rather than being defined for a category $\mathbb{C}$ like a monad, a relative monad is defined for a functor J between two categories $\mathbb{J}$ and $\mathbb{C}$.

Definition 1. *A (Manes-style [12])* relative monad *on a functor $J : \mathbb{J} \to \mathbb{C}$ is given by*

- *an object mapping $T \in |\mathbb{J}| \to |\mathbb{C}|$,*
- *for any $X \in |\mathbb{J}|$, a map $\eta_X \in \mathbb{C}\,(J\,X, T\,X)$ (the* unit*),*
- *for any $X, Y \in |\mathbb{J}|$ and $k \in \mathbb{C}\,(J\,X, T\,Y)$, a map $k^* \in \mathbb{C}\,(T\,X, T\,Y)$ (the* Kleisli extension*)*

satisfying the conditions

- *for any $X, Y \in |\mathbb{J}|$, $k \in \mathbb{C}\,(J\,X, T\,Y)$, $k = k^* \circ \eta$,*
- *for any $X \in |\mathbb{J}|$, $\eta^*_X = \mathrm{id}_{TX} \in \mathbb{C}\,(T\,X, T\,X)$,*
- *for any $X, Y, Z \in |\mathbb{J}|$, $k \in \mathbb{C}\,(J\,X, T\,Y)$, $\ell \in \mathbb{C}\,(J\,Y, T\,Z)$, $(\ell^* \circ k)^* = \ell^* \circ k^*$.*

The data and laws of a relative monad are exactly as those of a monad, except that $\mathbb{C}$ has become $\mathbb{J}$ in some places and, to ensure type-compatibility, some occurrences of J have been inserted.

Although this is not stated in the axioms, they imply that T is functorial: $T \in \mathbb{J} \to \mathbb{C}$. Indeed, for $X, Y \in |\mathbb{J}|$, $f \in \mathbb{J}\,(X, Y)$, we can define a map

$T\,f \in \mathbb{C}\,(T X, T Y)$ by $T\;f =_{\mathrm{df}} (\eta \circ J f)^*$ and this satisfies the functor laws. Also, η and $(-)^*$ are natural.

A definition of relative monads based on a multiplication μ rather than a Kleisli extension $(-)^*$ is not immediately available: the simple functor composition $T \cdot T$ is not well-typed. In the next section, we will show that a suitable notion of functor composition is available under a condition.

Clearly, monads are a special case of relative monads via $\mathbb{J} =_{\mathrm{df}} \mathbb{C}$, $J =_{\mathrm{df}} I_{\mathbb{C}}$.

For general $\mathbb{J}$, $\mathbb{C}$ and J, we always have that $T\,X =_{\mathrm{df}} J\,X$ is a relative monad with $\eta_X =_{\mathrm{df}} \mathrm{id}_{JX}$ and $k^* =_{\mathrm{df}} k$. A whole class of examples of relative monads on J is given by restricting monads on $\mathbb{C}$ (the relative monad J arises from restricting the monad $I_{\mathbb{C}}$).

Theorem 1. *For any $J \in \mathbb{J} \to \mathbb{C}$, a monad $(T, \eta, (-)^*)$ on $\mathbb{C}$ restricts to a relative monad $(T^\flat, \eta^\flat, (-)^{(*^\flat)})$ on J, defined by $T^\flat\,X =_{\mathrm{df}} T\,(J\,X)$, $\eta^\flat_X =_{\mathrm{df}} \eta_{J\,X}$, $k^{(*^\flat)} =_{\mathrm{df}} k^*$.*

As a first truly non-trivial example, we saw the relative monad of finite-dimensional vector spaces in the introduction. Here are some further examples.

Example 2. The syntax of untyped (but well-scoped) λ-calculus is a relative monad on $J_{\mathrm{f}} \in \mathbf{Fin} \to \mathbf{Set}$, as the finite-dimensional vector spaces relative monad, i.e., we have $\mathbb{J} =_{\mathrm{df}} \mathbf{Fin}$, $\mathbb{C} =_{\mathrm{df}} \mathbf{Set}$, $J =_{\mathrm{df}} J_{\mathrm{f}}$. We view $\mathbf{Fin}$ as the category of nameless untyped contexts. The set of untyped λ-terms $\mathsf{Lam}\,\Gamma$ over a context Γ satisfies the isomorphism

$$\mathsf{Lam}\,\Gamma \cong J_{\mathrm{f}}\,\Gamma + \mathsf{Lam}\,\Gamma \times \mathsf{Lam}\,\Gamma + \mathsf{Lam}\,(1 + \Gamma)$$

The summands correspond to variables from the context (seen as terms), applications, and abstractions (their bodies are terms over an extended context). The functor $\mathsf{Lam} \in \mathbf{Fin} \to \mathbf{Set}$ is defined as the carrier of the initial algebra of the functor $F \in [\mathbf{Fin}, \mathbf{Set}] \to [\mathbf{Fin}, \mathbf{Set}]$ defined by

$$F\,G\,\Gamma =_{\mathrm{df}} J_{\mathrm{f}}\,\Gamma + G\,\Gamma \times G\,\Gamma + G\,(1 + \Gamma)$$

Lam is a relative monad. The unit $\eta \in J_{\mathrm{f}}\,\Gamma \to \mathsf{Lam}\,\Gamma$ is given by variables-as-terms and the Kleisli extension takes a finite substitution rule $k \in J_{\mathrm{f}}\,\Gamma \to \mathsf{Lam}\,\Delta$ to the corresponding substitution function $k^* \in \mathsf{Lam}\,\Gamma \to \mathsf{Lam}\,\Delta$.

This example was described as a relative monad (under the name Kleisli structure) by Altenkirch and Reus [5]. Fiore et al. [7] described it as a monoid in a monoidal structure on $[\mathbf{Fin}, \mathbf{Set}]$. Their account of this example is an instance of our general description of relative monads as monoids from Section 4.

Example 3. Typed λ-terms form a relative monad in a similar fashion. Let Ty be the set of types (over some base types), which we see as a discrete category. We take $\mathbb{J}$ to be $\mathbf{Fin} \downarrow \mathsf{Ty}$, which is the category whose objects are pairs (Γ, ρ) where $\Gamma \in |\mathbf{Fin}|$ and $\rho \in \Gamma \to \mathsf{Ty}$ (typed contexts) and maps from (Γ, ρ) to (Γ', ρ') are maps $f \in \mathbf{Fin}\,(\Gamma, \Gamma')$ such that $\rho = \rho' \circ f$ (typed context maps).

We further take $\mathbb{C}$ to be the functor category $[\mathsf{Ty}, \mathbf{Set}]$ and let $J \in \mathbf{Fin} \downarrow \mathsf{Ty} \to [\mathsf{Ty}, \mathbf{Set}]$ be the natural embedding defined by $J\,(\Gamma, \rho)\;\sigma =_{\mathrm{df}} \{x \in \Gamma \mid \rho\,x = \sigma\}$.

Now, for $(\Gamma, \rho) \in |\mathbf{Fin} \downarrow \mathsf{Ty}|$ and $\sigma \in \mathsf{Ty}$, the set of typed λ-terms $\mathsf{TyLam}\,(\Gamma, \rho)\,\sigma$ has to satisfy the isomorphism

$$\begin{aligned}\mathsf{TyLam}\,(\Gamma, \rho)\,\sigma \cong{} & J\,(\Gamma, \rho)\,\sigma \\ & + \Sigma_{\tau \in \mathsf{Ty}} \mathsf{TyLam}\,(\Gamma, \rho)\,(\tau \Rightarrow \sigma) \times \mathsf{TyLam}\,(\Gamma, \rho)\,\tau \\ & + \text{if } \sigma \text{ is of the form } \tau \Rightarrow \tau' \text{ then } \mathsf{TyLam}\,(1 + \Gamma, \begin{bmatrix} \mathrm{inl}\,{*} \mapsto \tau \\ \mathrm{inr}\,x \mapsto \rho\,x \end{bmatrix})\,\tau'\end{aligned}$$

The functor $\mathsf{TyLam} \in \mathbf{Fin} \downarrow \mathsf{Ty} \to [\mathsf{Ty}, \mathbf{Set}]$ is given by an initial algebra. It is a monad on J, with the unit and Kleisli extension given by variables-as-terms and substitution, like in the case of Lam. Fiore et al. [6] studied TyLam as a monoid in $[\mathbf{Fin} \downarrow \mathsf{Ty}, [\mathsf{Ty}, \mathbf{Set}]]$.

Note that choosing $\mathbb{J}$ to be $[\mathsf{Ty}, \mathbf{Fin}]$ rather than $\mathbf{Fin} \downarrow \mathsf{Ty}$ would have given contexts possibly supported by infinitely many types: in every type there are finitely many variables, but the total number of variables can be infinite.

Example 4. Morris and Altenkirch [14] investigated generalization of the notion of containers [1] to a dependently typed setting and used it to show that strictly positive families can be reduced to W-types. Relative monads played a central role in this development.

Let $\mathsf{U} \in \mathbf{Set}$ together with $\mathsf{El} \in \mathsf{U} \to \mathbf{Set}$ be a universe of small sets. This induces a category $\mathbf{U}$ with $|\mathbf{U}| =_{\mathrm{df}} \mathsf{U}$ and $\mathbf{U}\,(a, b) =_{\mathrm{df}} \mathsf{El}\,a \to \mathsf{El}\,b$. The functor $J_{\mathbf{U}} \in \mathbf{U} \to \mathbf{Cat}$ is given by $J_{\mathbf{U}}\,a =_{\mathrm{df}} \mathsf{El}\,a$ on objects and the identity on maps (viewing $\mathsf{El}\,a$ as a discrete category). We assume that $\mathbf{U}$ is locally cartesian closed, i.e., the universe is closed under dependent product and function types as well as equality types.

As ordinary containers represent endofunctors on $\mathbf{U}$ (or any other locally cartesian closed category), indexed containers represent functors from a slice over a given $a \in U$, we define the category $\mathsf{IF}\,a$ of indexed functors over a by $\mathsf{IF}\,a =_{\mathrm{df}} [[\mathsf{El}\,a, \mathbf{U}], \mathbf{U}]$. The functor $\mathsf{IF} \in \mathbf{U} \to \mathbf{Cat}$ is a relative monad on $J_{\mathbf{U}}$. The unit $\eta_a \in J_{\mathbf{U}}\,a \to \mathsf{IF}\,a$ is defined by $\eta_a\,x =_{\mathrm{df}} \lambda f.\, f\,x$ and the Kleisli extension $k^* \in \mathsf{IF}\,a \to \mathsf{IF}\,b$ of $k \in J_{\mathbf{U}}\,a \to \mathsf{IF}\,b$ is defined by $k^*\,G\,f = G\,(\lambda x. k\,x\,f)$. The definitions clearly resemble the continuation monad apart from the size issue.

The main result of [14] was that strictly positive families (SPF) can be interpreted as indexed functors by via indexed containers (IC). Just as IF, both SPF and IC are relative monads on $J_{\mathbf{U}}$ and the interpretations preserve this structure, i.e., are relative monad maps.

2.2 Relative Adjunctions

As ordinary monads are intimately related to adjunctions, relative monads are related to a corresponding generalization of adjunctions.

Definition 2. *A* relative adjunction *between $J \in \mathbb{J} \to \mathbb{C}$ and $\mathbb{D}$ is given by two functors $L \in \mathbb{J} \to \mathbb{D}$ and $R \in \mathbb{D} \to \mathbb{C}$, and a natural isomorphism $\phi \in \mathbb{C}\,(J\,X, R\,Y) \cong \mathbb{D}\,(L\,X, Y)$.*

As expected, ordinary adjunctions are a special case of relative adjunctions with $\mathbb{J} =_{\mathrm{df}} \mathbb{C}$, $J =_{\mathrm{df}} I$. Just like any adjunction defines a monad, relative adjunctions define relative monads.

Theorem 2. *Any relative adjunction* (L, R, ϕ) *between a functor* $J \in \mathbb{J} \to \mathbb{C}$ *and category* $\mathbb{D}$ *gives rise to a relative monad, defined by* $T\,X =_{\mathrm{df}} R\,(L\,X)$, $\eta_X =_{\mathrm{df}} \phi^{-1}\,(\mathrm{id}_{L\,X})$, $k^* =_{\mathrm{df}} R\,(\phi\,k)$.

$$\begin{tikzcd} & \mathbb{D} \arrow[dr, "R"] & \\ \mathbb{J} \arrow[ur, "L"] \arrow[rr, "J"] \arrow[rr, bend left=90, "T"] & & \mathbb{C} \end{tikzcd}$$

If a relative monad T on J is related to a relative adjunction (L, R, ϕ) between J and some category $\mathbb{D}$ in the above way, we call the relative adjunction a *splitting* of the relative monad via $\mathbb{D}$.

2.3 Kleisli and Eilenberg-Moore Constructions

For monads we know that they split into an adjunction in two canonical ways: the Kleisli and Eilenberg-Moore constructions. Moreover, the splittings form a category where the Kleisli and EM splittings are the initial and terminal objects. We shall now establish that the same holds in the relative situation.

The Kleisli category $\mathbf{Kl}(T)$ of a relative monad T has as objects the objects of $\mathbb{J}$ and as maps between X, Y the maps between $J\,X$, $T\,Y$ of $\mathbb{C}$: $|\mathbf{Kl}(T)| =_{\mathrm{df}} |\mathbb{J}|$ and $\mathbf{Kl}(T)\,(X, Y) =_{\mathrm{df}} \mathbb{C}\,(J\,X, T\,Y)$. The identity and composition (we denote them by id^T, $\circ^T$) are defined by $\mathrm{id}^T_X =_{\mathrm{df}} \eta_X$ and $\ell \circ^T k =_{\mathrm{df}} \ell^* \circ k$.

The Kleisli relative adjunction between J and $\mathbf{Kl}(T)$ is defined by $L\,X =_{\mathrm{df}} X$, $L\,f =_{\mathrm{df}} \eta \circ J f$ (note that L is identity-on-objects), $R\,X =_{\mathrm{df}} T\,X$, $R\,k =_{\mathrm{df}} k^*$ and ϕ is identity. This relative adjunction is a splitting: it is immediate that $R\,(L\,X) = T\,X$, $\eta_X = \phi^{-1}\,(\mathrm{id}^T_{L\,X})$, $k^* = R\,(\phi\,k)$.

The Eilenberg-Moore (EM) category $\mathbf{EM}(T)$ is given by EM-algebras and EM-algebra maps of the relative monad T. Since the usual definition of an EM-algebra refers to μ, which is not immediately available, we generalize a version based on $(-)^*$. For ordinary monads this is equivalent to the standard definition.

Definition 3. *An EM-*algebra *of a relative monad* T *on* $J \in \mathbb{J} \to \mathbb{C}$ *is given by an object* $X \in |\mathbb{C}|$ *and, for any* $Z \in |\mathbb{J}|$, *a map function* $\chi \in \mathbb{C}\,(J\,Z, X) \to \mathbb{C}\,(T\,Z, X)$, *satisfying the conditions*

– *for any* $Z \in |\mathbb{J}|$, $f \in \mathbb{C}\,(J\,Z, X)$, $f = \chi\,f \circ \eta$,
– *for any* $Z, W \in |\mathbb{J}|$, $k \in \mathbb{C}\,(J\,Z, T\,W)$, $f \in \mathbb{C}\,(J\,W, X)$, $\chi\,(\chi\,f \circ k) = \chi\,f \circ k^*$.

These conditions ensure, among other things, that χ *is natural.*

*An EM-*algebra map *from* (X, χ) *to* (Y, υ) *is a map* $h \in \mathbb{C}\,(X, Y)$ *satisfying*

– *for any* $Z \in |\mathbb{J}|$, $f \in \mathbb{C}\,(J\,Z, X)$, $h \circ \chi\,f = \upsilon\,(h \circ f)$.

The identity and composition of $\mathbf{EM}(T)$ are inherited from $\mathbb{C}$.

The Eilenberg-Moore relative adjunction between J and $\mathbf{EM}(T)$ is defined by $L\,Y =_{\mathrm{df}} (T\,Y, (-)^*)$, $L\,f =_{\mathrm{df}} T\,f$, $R\,(X,\chi) =_{\mathrm{df}} X$, $R\,h =_{\mathrm{df}} h$ (so R is identity-on-maps), $\phi_{X,(Y,\upsilon)}\,f =_{\mathrm{df}} \upsilon\,f$ and $\phi^{-1}_{X,(Y,\upsilon)}\,h =_{\mathrm{df}} h \circ \eta_X$. This is also a splitting.

Theorem 3. *The splittings of a relative monad T on $J \in \mathbb{J} \to \mathbb{C}$ form a category. An object is given by a category $\mathbb{D}$ and an adjunction (L, R, ϕ) splitting T via $\mathbb{D}$. A splitting morphism between $(\mathbb{D}, L, R, \phi)$ and (D', L', R', ϕ') is a functor $V \in \mathbb{D} \to \mathbb{D}'$ such that $V \cdot L = L'$, $R = R' \cdot V$, and $V\,\phi_{X,Y} = \phi'_{X,V\,Y}$. The Kleisli construction is the initial and the Eilenberg-Moore construction the terminal splitting.*

Example 5. The Kleisli category of Vec has as objects the objects of **Fin** understood as finite coordinate systems (describing vector spaces). The maps are maps $J_{\mathrm{f}}\,m \to \mathsf{Vec}\,n$, i.e., $m \times n$-matrixes (describing linear transformations). The identities are the unit $m \times m$-matrices, the composition is multiplication of matrices.

Example 6. The Kleisli category of Lam has as objects the objects of **Fin** understood as untyped contexts. The maps are maps $J_{\mathrm{f}}\,\mathit{\Gamma} \to \mathsf{Lam}\,\mathit{\Delta}$, i.e., substitution rules (assignments of terms over $\mathit{\Delta}$ to the variables in $\mathit{\Gamma}$). The identities are the trivial substitution rules. The composition is composition of substitution rules.

3 Relative Monads as Lax Monoids

A monad on $\mathbb{C}$ is the same as a monoid in the endofunctor category $[\mathbb{C}, \mathbb{C}]$. It has a monoidal structure given by the identity functor I and composition of functors $\cdot$, which are strictly unital and associative. A monad can be specified by an object $T \in |[\mathbb{C}, \mathbb{C}]|$ and maps $\eta \in [\mathbb{C}, \mathbb{C}]\,(I, T)$ and $\mu \in [\mathbb{C}, \mathbb{C}]\,(T \cdot T, T)$ satisfying the laws of a monoid in the strict monoidal category $([\mathbb{C}, \mathbb{C}], I, \cdot)$.

Can we similarly define a relative monad on $J \in \mathbb{J} \to \mathbb{C}$ as a monoid in the functor category $[\mathbb{J}, \mathbb{C}]$? This requires a monoidal structure on $[\mathbb{J}, \mathbb{C}]$, ideally similar to that on $[\mathbb{C}, \mathbb{C}]$. The functor J is a good candidate for the unit, but the tensor is problematic, as functors $\mathbb{J} \to \mathbb{C}$ cannot be composed by simple functor composition. We shall use a left Kan extension to overcome the difficulty and obtain a lax monoidal structure where relative monads are lax monoids.

3.1 Left Kan Extensions

Left Kan extensions are one of the two canonical constructions for extending functors. The left Kan extension along $J \in \mathbb{J} \to \mathbb{C}$ extends functors $\mathbb{J} \to \mathbb{D}$ to functors $\mathbb{C} \to \mathbb{D}$.

$$\begin{tikzcd} & \mathbb{D} & \\ \mathbb{J} \arrow[ur, bend left, "F"] \arrow[rr, "J"] & & \mathbb{C} \arrow[ul, bend right, "\mathrm{Lan}_J F"'] \end{tikzcd}$$

It is defined as the left adjoint (if it exists) of the restriction functor $- \cdot J \in [\mathbb{C}, \mathbb{D}] \to [\mathbb{J}, \mathbb{D}]$. By definition, it is given by a functor $\mathrm{Lan}_J \in [\mathbb{J}, \mathbb{D}] \to [\mathbb{C}, \mathbb{D}]$ and a natural isomorphism

$$[\mathbb{J}, \mathbb{D}]\,(F, G \cdot J) \cong [\mathbb{C}, \mathbb{D}]\,(\mathrm{Lan}_J\, F, G)$$

While it is possible to work directly with this definition of left Kan extension, we use an alternative definition, based on the coend formula

$$\mathrm{Lan}_J\, F\, X \cong \int^{Y \in |\mathbb{J}|} \mathbb{C}\,(J\, Y, X) \bullet F\, Y$$

Accordingly, we take that a left Kan extension of a functor $F \in \mathbb{J} \to \mathbb{D}$ along $J \in \mathbb{J} \to \mathbb{C}$ to be given by

- an object function $\mathrm{Lan}_J\, F \in |\mathbb{C}| \to |\mathbb{D}|$,
- for any $X \in |\mathbb{C}|$, a natural transformation $\iota_{F,X} \in [\mathbb{J}^{\mathrm{op}}, \mathbf{Set}]\,(\mathbb{C}\,(J\, -, X), \mathbb{D}\,(F\, -, \mathrm{Lan}_J\, F\, X))$,
- for any $X \in |\mathbb{C}|$, $Y \in |\mathbb{D}|$ and $\theta \in [\mathbb{J}^{\mathrm{op}}, \mathbf{Set}]\,(\mathbb{C}\,(J\, -, X), \mathbb{D}\,(F\, -, Y))$, a map $[\theta] \in \mathbb{D}\,(\mathrm{Lan}_J\, F\, X, Y)$.

satisfying the conditions $[\theta] \circ \iota\, g = \theta\, g$, $[\iota] = \mathrm{id}$ and $f \circ [\theta] = [\lambda g. f \circ \theta\, g]$.

Left Kan extensions $\mathrm{Lan}_J\, F\, X$ are functorial in both arguments F and X, i.e., $\mathrm{Lan}_J \in [\mathbb{J}, \mathbb{D}] \to [\mathbb{C}, \mathbb{D}]$. For any $F \in |[\mathbb{J}, \mathbb{D}]|$, $X, Y \in |\mathbb{C}|$, $f \in \mathbb{C}\,(X, Y)$,

$$\begin{array}{l} \mathrm{Lan}_J\, F\, f \in \mathbb{D}\,(\mathrm{Lan}_J\, F\, X, \mathrm{Lan}_J\, F\, Y) \\ \mathrm{Lan}_J\, F\, f =_{\mathrm{df}} [\lambda g.\, \iota\, (f \circ g)] \end{array}$$

And for any $F, G \in |[\mathbb{J}, \mathbb{D}]|$, $\tau \in [\mathbb{J}, \mathbb{D}]\,(F, G)$, $X \in |\mathbb{C}|$, we have

$$\begin{array}{l} \mathrm{Lan}_J\, \tau\, X \in \mathbb{D}\,(\mathrm{Lan}_J\, F\, X, \mathrm{Lan}_J\, G\, X) \\ \mathrm{Lan}_J\, \tau\, X =_{\mathrm{df}} [\lambda g.\, \iota\, g \circ \tau] \end{array}$$

In general $\mathrm{Lan}_J \in [\mathbb{J}, \mathbb{D}] \to [\mathbb{C}, \mathbb{D}]$ exists, if $\mathbb{J}$ is small and $\mathbb{D}$ is cocomplete.

3.2 $[\mathbb{J}, \mathbb{C}]$ Is Lax Monoidal

If $\mathrm{Lan}_J \in [\mathbb{J}, \mathbb{C}] \to [\mathbb{C}, \mathbb{C}]$ exists, we can turn any functor $F \in |[\mathbb{J}, \mathbb{C}]|$ to one in $|[\mathbb{C}, \mathbb{C}]|$. Hence we can define a composition-like operation

$$\begin{array}{l} (\cdot^J) \in |[\mathbb{J}, \mathbb{C}]| \times |[\mathbb{J}, \mathbb{C}]| \to |[\mathbb{J}, \mathbb{C}]| \\ F \cdot^J G =_{\mathrm{df}} \mathrm{Lan}_J\, F\ \cdot\ G \end{array}$$

This is our candidate for the tensor on $[\mathbb{J}, \mathbb{C}]$. We also need the unital and associative laws. We define several families of maps indexed by $X \in |\mathbb{C}|$:

$$\begin{array}{l} \overline{\lambda}_X \in \mathbb{C}\,(\mathrm{Lan}_J\, J\, X, X) \\ \overline{\lambda}_X =_{\mathrm{df}} [\lambda g.\, g] \\ \overline{\overline{\alpha}}_{F,G,X} \in \mathbb{C}\,(\mathrm{Lan}_J\,(F \cdot G)\, X, F\,(\mathrm{Lan}_J\, G\, X)) \\ \overline{\overline{\alpha}}_{F,G,X} =_{\mathrm{df}} [\lambda g.\, F\,(\iota\, g)] \\ \overline{\alpha}_{F,G,X} \in \mathbb{C}\,(\mathrm{Lan}_J\,(\mathrm{Lan}_J\, F \cdot G)\, X, \mathrm{Lan}_J\, F\,(\mathrm{Lan}_J\, G\, X)) \\ \overline{\alpha}_{F,G,X} =_{\mathrm{df}} \overline{\overline{\alpha}}_{\mathrm{Lan}_J F, G} = [\lambda g.\, [\lambda g'.\, \iota\,(\iota\, g \circ g')]] \end{array}$$

All these families are natural in X, hence maps in $|[\mathbb{C}, \mathbb{C}]|$.

From these we further define our candidate unital and associative laws.

$$\begin{array}{l}
\rho_F \in [\mathbb{J},\mathbb{C}]\,(F, F \cdot^J J) \\
\rho_F =_{\mathrm{df}} \iota\,\mathrm{id} \\
\lambda_F \in [\mathbb{J},\mathbb{C}]\,(J \cdot^J F, F) \\
\lambda_F =_{\mathrm{df}} \overline{\lambda} \cdot F \\
\alpha_{F,G,H} \in [\mathbb{J},\mathbb{C}]\,((F \cdot^J G) \cdot^J H, F \cdot^J (G \cdot^J H)) \\
\alpha_{F,G,H} =_{\mathrm{df}} \overline{\alpha}_{F,G} \cdot H
\end{array}$$

It turns out that the data so defined provide a structure that is almost monoidal, but not quite. It is lax monoidal: λ, ρ, α are generally not isomorphisms. In the next section we will identify conditions on J that enable us to construct the inverses, turning the lax monoidal structure into properly monoidal.

Theorem 4. *If* $\mathrm{Lan}_J \in [\mathbb{J},\mathbb{C}] \to [\mathbb{C},\mathbb{C}]$ *exists, then* $([\mathbb{J},\mathbb{C}], J, \cdot^J, \lambda, \rho, \alpha)$ *is a lax monoidal category, i.e.,* $\cdot^J$ *is functorial,* λ*,* ρ*,* α *are natural and the following diagrams commute:*

$$\begin{tikzcd}
& J \cdot^J J \arrow[dr, "\lambda_J"] & \\
J \arrow[ur, "\rho_J"] \arrow[rr, equal] & & J
\end{tikzcd}
\qquad
\begin{tikzcd}
(F \cdot^J J) \cdot^J G \arrow[r, "\alpha_{F,J,G}"] & F \cdot^J (J \cdot^J G) \arrow[d, "F \cdot^J \lambda_G"] \\
F \cdot^J G \arrow[u, "\rho_F \cdot^J G"] \arrow[r, equal] & F \cdot^J G
\end{tikzcd}$$

$$\begin{tikzcd}
(J \cdot^J F) \cdot^J G \arrow[rr, "\alpha_{J,F,G}"] \arrow[dr, "\lambda_F \cdot^J G"'] & & J \cdot^J (F \cdot^J G) \arrow[dl, "\lambda_{F \cdot^J G}"] \\
& F \cdot^J G &
\end{tikzcd}
\qquad
\begin{tikzcd}
(F \cdot^J G) \cdot^J J \arrow[rr, "\alpha_{F,G,J}"] & & F \cdot^J (G \cdot^J J) \\
& F \cdot^J G \arrow[ul, "\rho_{F \cdot^J G}"] \arrow[ur, "F \cdot^J \rho_G"'] &
\end{tikzcd}$$

$$\begin{tikzcd}
(F \cdot^J (G \cdot^J H)) \cdot^J K \arrow[rr, "\alpha_{F,G \cdot^J H,K}"] & & F \cdot^J ((G \cdot^J H) \cdot^J K) \arrow[d, "F \cdot^J \alpha_{G,H,K}"] \\
((F \cdot^J G) \cdot^J H) \cdot^J K \arrow[u, "\alpha_{F,G,H} \cdot^J K"] \arrow[r, "\alpha_{F \cdot^J G,H,K}"] & (F \cdot^J G) \cdot^J (H \cdot^J K) \arrow[r, "\alpha_{F,G,H \cdot^J K}"] & F \cdot^J (G \cdot^J (H \cdot^J K))
\end{tikzcd}$$

3.3 Relative Monads Are the Same as Lax Monoids in $[\mathbb{J},\mathbb{C}]$

With a lax monoidal structure present on the functor category $[\mathbb{J},\mathbb{C}]$, we should expect that relative monads on J are the same thing as lax monoids in this structure, generalizing the case of ordinary monads on $\mathbb{C}$ and the strict monoidal structure on the endofunctor category $[\mathbb{C},\mathbb{C}]$. This is indeed the case.

Theorem 5. *Assume that* $\mathrm{Lan}_J \in [\mathbb{J},\mathbb{C}] \to [\mathbb{C},\mathbb{C}]$ *exists.*

1. *Given a relative monad* $(T, \eta, (-)^*)$ *on* J*, define, for any* $X \in |\mathbb{J}|$*, a map* $\mu_X \in \mathbb{C}\,(\mathrm{Lan}_J T\,(T\,X), T\,X)$ *by* $\mu_X =_{\mathrm{df}} [(-)^*]$*. This is well-defined, since* $(-)^*$ *is natural:* $(-)^* \in [\mathbb{J}^{\mathrm{op}}, \mathbf{Set}]\,(\mathbb{C}\,(J\,-, T\,X), \mathbb{C}\,(T\,-, T\,X))$*.*
   Then (T, η, μ) *is a lax monoid in the lax monoidal category* $([\mathbb{J},\mathbb{C}], J, \cdot^J, \lambda, \rho, \alpha)$*: we have that* $T \in |[\mathbb{J},\mathbb{C}]|$*,* $\eta \in [\mathbb{J},\mathbb{C}]\,(J, T)$ *and* $\mu \in [\mathbb{J},\mathbb{C}]\,(T \cdot^J T, T)$*, and the following diagrams commute in* $[\mathbb{J},\mathbb{C}]$*:*

$$\begin{tikzcd}
T \cdot^J J \arrow[r, "T \cdot^J \eta"] & T \cdot^J T \arrow[d, "\mu"] \\
T \arrow[u, "\rho_T"] \arrow[r, equal] & T
\end{tikzcd}
\qquad
\begin{tikzcd}
J \cdot^J T \arrow[r, "\lambda_T"] \arrow[d, "\eta \cdot^J T"'] & T \arrow[d, equal] \\
T \cdot^J T \arrow[r, "\mu"] & T
\end{tikzcd}
\qquad
\begin{tikzcd}
& T \cdot^J (T \cdot^J T) \arrow[r, "T \cdot^J \mu"] & T \cdot^J T \arrow[d, "\mu"] \\
(T \cdot^J T) \cdot^J T \arrow[ur, "\alpha_{T,T,T}"] \arrow[d, "\mu \cdot^J T"'] & & \\
T \cdot^J T \arrow[rr, "\mu"] & & T
\end{tikzcd}$$

2. *Given a lax monoid (T, η, μ) in $([\mathbb{J}, \mathbb{C}], J, \cdot^J, \lambda, \rho, \alpha)$, define, for any $X, Y \in |\mathbb{J}|$, a function $(-)^* \in \mathbb{C}\,(J\,X, T\,Y) \to \mathbb{C}\,(T\,X, T\,Y)$ by $k^* =_{\mathrm{df}} \mu_Y \circ \iota\, k$. Then $(T, \eta, (-)^*)$ is a relative monad on J.*
3. *The above correspondence is bijective.*

The bijective correspondence between relative monads on J and lax monoids in $[\mathbb{J}, \mathbb{C}]$ extends to an equivalence of categories, but we must omit the details here (we have defined neither relative monad maps nor lax monoid maps).

Moreover, just as the availability of $\mathrm{Lan}_J \in [\mathbb{J}, \mathbb{C}] \to [\mathbb{C}, \mathbb{C}]$ allows us to define relative monads based on μ rather than $(-)^*$, it also facilitates a more traditional-style definition of EM-algebras; we must omit the details.

4 Well-Behaved Relative Monads

It is somewhat unsatisfactory to obtain that $[\mathbb{J}, \mathbb{C}]$ is just lax monoidal, rather than properly monoidal. This begs the question: would some conditions on J ensure a properly monoidal structure? The answer is affirmative. Mild conditions turn the lax monoidal structure of $[\mathbb{J}, \mathbb{C}]$ into properly monoidal. What is more, the same conditions also make relative monads on J extend to monads on $\mathbb{C}$.

4.1 Well-Behavedness Conditions

We define three well-behavedness conditions on J. They are additional to the existence of $\mathrm{Lan}_J \in [\mathbb{J}, \mathbb{C}] \to [\mathbb{C}, \mathbb{C}]$ and require the constituent maps of three canonical families, which are actually natural, to be isomorphisms. For our purposes, these conditions are mild.

Definition 4. *$J \in \mathbb{J} \to \mathbb{C}$ is* well-behaved, *if not only does $\mathrm{Lan}_J \in [\mathbb{J}, \mathbb{C}] \to [\mathbb{C}, \mathbb{C}]$ exist, but also the following three conditions hold:*

1. *J is* fully faithful, *i.e., for any $X, Y \in |\mathbb{J}|$, there is an inverse to the map*

$$\begin{aligned} &J_{X,Y} \in \mathbb{J}\,(X, Y) \to \mathbb{C}\,(J\,X, J\,Y) \\ &J_{X,Y}\, f =_{\mathrm{df}} J\, f \end{aligned}$$

2. *J is* dense, *i.e., for any $X, Y \in |\mathbb{C}|$, there is an inverse to the map*

$$\begin{aligned} &K_{X,Y} \in \mathbb{C}\,(X, Y) \to [\mathbb{J}^{\mathrm{op}}, \mathbf{Set}]\,(\mathbb{C}\,(J\,-, X), \mathbb{C}\,(J\,-, Y)) \\ &K_{X,Y}\, f =_{\mathrm{df}} \lambda g.\, f \circ g \end{aligned}$$

i.e. $K \in \mathbb{C} \to [\mathbb{J}^{\mathrm{op}}, \mathbf{Set}]$, with $K\,X =_{\mathrm{df}} \mathbb{C}\,(J\,-, X)$, is fully faithful.

3. *For any $F \in \mathbb{J} \to \mathbb{C}$, $X \in |\mathbb{J}|$, $Y \in |\mathbb{C}|$, there is an inverse to the map*

$$\begin{array}{l} L^F_{X,Y} \in \mathrm{Lan}_J\,(\mathbb{C}\,(J\,X, F-))\,Y \to \mathbb{C}\,(J\,X, \mathrm{Lan}_J\,F\,Y) \\ L^F_{X,Y} =_{\mathrm{df}} [\lambda g.\,\lambda g'.\,\iota\,g \circ g'] \end{array}$$

Example 7. The functor $J_{\mathrm{f}} \in \mathbf{Fin} \to \mathbf{Set}$ is well-behaved. The functor $J_{\mathbf{U}} \in \mathbf{U} \to \mathbf{Cat}$ of Example 4 is well-behaved, if the type-theoretic universe $\mathsf{U} \in \mathbf{Set}, \mathsf{El} \in \mathsf{U} \to \mathbf{Set}$ is is closed under dependent products (categorically this corresponds to the induced category $\mathbf{U}$ being cartesian).

From the well-behavedness of J_{f}, it follows that $[\mathbf{Fin}, \mathbf{Set}]$ is monoidal and Lam is a monoid. These facts were proved by Fiore et al. [7].

4.2 $[\mathbb{J}, \mathbb{C}]$ Is Monoidal

Our well-behavedness conditions suffice to ensure that the unital and associativity laws of the lax monoidal structure on $[\mathbb{J}, \mathbb{C}]$ are isomorphisms. Specifically, the existence of inverses of J, K, L ensures that $\rho, \overline{\lambda}, \overline{\alpha}$ (and consequently also λ, α) have inverses too.

Theorem 6. *If $J \in \mathbb{J} \to \mathbb{C}$ is well-behaved, then*

1. *for any $F \in \mathbb{J} \to \mathbb{C}$, $X \in |\mathbb{J}|$, the map $\rho^{-1}_{F,X} \in \mathbb{C}\,(\mathrm{Lan}_J\,F\,(J\,X), F\,X)$ defined by $\rho^{-1}_{F,X} =_{\mathrm{df}} [\lambda g.\,F\,(J^{-1}\,g)]$ is an inverse of $\rho_{F,X}$;*
2. *for any $X \in |\mathbb{J}|$, the map $\overline{\lambda}^{-1}_X \in \mathbb{C}\,(X, \mathrm{Lan}_J\,J\,X)$ defined by $\overline{\lambda}^{-1}_X =_{\mathrm{df}} K^{-1}\,\iota_{J,X}$ is an inverse of $\overline{\lambda}_X$;*
3. *for any $F, G \in \mathbb{J} \to \mathbb{C}$, $X \in |\mathbb{J}|$, the map $\bar{\alpha}^{-1}_{F,G,X} \in \mathbb{C}\,(\mathrm{Lan}_J\,F\,(\mathrm{Lan}_J\,G\,X), \mathrm{Lan}_J\,(\mathrm{Lan}_J\,F \cdot G)\,X)$ defined by $\bar{\alpha}^{-1}_{F,G,X} =_{\mathrm{df}} [\lambda g.\,[\lambda g.\,\lambda g'.\,\iota\,g \circ \iota\,g']\,(L^{-1}\,g)]$ is an inverse of $\overline{\alpha}_{F,G,X}$.*

Hence, the category $([\mathbb{J}, \mathbb{C}], J, \cdot^J, \lambda, \rho, \alpha)$ is monoidal.

As an immediate corollary, we get that, in the well-behaved case, relative monads are proper monoids in a properly monoidal structure.

Corollary 1. *If $\mathbb{J} \to \mathbb{C}$ is well-behaved, then a relative monad $(T, \eta, (-)^*)$ is the same as a monoid (T, η, μ) in the monoidal category $([\mathbb{J}, \mathbb{C}], J, \cdot^J, \lambda, \rho, \alpha)$.*

4.3 Relative Monads Extend to Monads

As a pleasant bonus, the well-behavedness conditions also ensure that a relative monad extends to an ordinary monad. Crucial here is that, if J is well-behaved, we have that $\overline{\lambda}$ and $\overline{\alpha}$ are isomorphisms.

Theorem 7. *If $J \in \mathbb{J} \to \mathbb{C}$ is well-behaved, then a monoid (T, η, μ) in $[\mathbb{J}, \mathbb{C}]$ (equivalently, a relative monad on J) extends to a monoid $(T^\#, \eta^\#, \mu^\#)$ in $[\mathbb{C}, \mathbb{C}]$ (equivalently, a monad on $\mathbb{C}$), defined by*

$$\begin{array}{rl} T^\# &=_{\mathrm{df}} \mathrm{Lan}_J\,T \\ \eta^\# &=_{\mathrm{df}}\ I \xrightarrow{\bar{\lambda}^{-1}} \mathrm{Lan}_J\,J \xrightarrow{\mathrm{Lan}_J\,\eta} \mathrm{Lan}_J\,T \\ \mu^\# &=_{\mathrm{df}}\ \mathrm{Lan}_J\,T \cdot \mathrm{Lan}_J\,T \xrightarrow{\bar{\alpha}^{-1}_{T,T}} \mathrm{Lan}_J\,(\mathrm{Lan}_J\,T \cdot T) \xrightarrow{\mathrm{Lan}_J\,\mu} \mathrm{Lan}_J\,T \end{array}$$

We see that, in the well-behaved case, we can not only restrict monads to relative monads but also extend relative monads to monads. Thanks to ρ being an isomorphism, this correspondence is an embedding-projection pair.

Theorem 8. *If $J \in \mathbb{J} \to \mathbb{C}$ is well-behaved, then the correspondence between monoids in $[\mathbb{J}, \mathbb{C}]$ (equivalently, relative monads) and monoids in $[\mathbb{C}, \mathbb{C}]$ (equivalently, monads) given in Theorems 7 and 1 is an embedding-projection pair up to the natural isomorphism ρ: If (T, η, μ) is a monoid in $[\mathbb{J}, \mathbb{C}]$, then ρ_T is a monoid isomorphism between (T, η, μ) and $(T^{\#\flat}, \eta^{\#\flat}, \mu^{\#\flat})$, i.e., $\rho_T \in [\mathbb{J}, \mathbb{C}]\,(T, T^{\#\flat})$ is an isomorphism and the following diagrams in $[\mathbb{J}, \mathbb{C}]$ commute:*

$$\begin{tikzcd} J \arrow[r, "\eta"] \arrow[dr, "\eta^{\#\flat}"'] & T \arrow[d, "\rho_T"] \\ & T^{\#\flat} \end{tikzcd} \qquad \begin{tikzcd} T \cdot^J T \arrow[r, "\mu"] \arrow[d, "\rho_T \cdot^J \rho_T"'] & T \arrow[d, "\rho_T"] \\ T^{\#\flat} \cdot^J T^{\#\flat} \arrow[r, "\mu^{\#\flat}"] & T^{\#\flat} \end{tikzcd}$$

In fact, $(-)^\flat$ extends to a functor from the category of monads on $\mathbb{C}$ to relative monads on J; $(-)^\#$ is its left adjoint. The relative monad maps $\rho_T \in [\mathbb{J}, \mathbb{C}]\,(T, \mathrm{Lan}_J\, T \cdot J)$ give the unit of the adjunction; the fact that it is a natural isomorphism strengthens the adjunction into a coreflection. Remarkably, this adjunction is a lifting from functors to relative monads of the adjunction $\mathrm{Lan}_J \dashv - \cdot J$ between $[\mathbb{C}, \mathbb{C}]$ and $[\mathbb{J}, \mathbb{C}]$, the defining adjunction of Lan_J.

The counit of the adjunction is $(T \cdot \overline{\lambda}) \circ \overline{\overline{\alpha}}_{T,J} \in [\mathbb{C}, \mathbb{C}]\,(\mathrm{Lan}_J\,(T \cdot J), T)$. Unlike the unit, it is generally not an isomorphism, so the adjunction is not also a reflection. For example, for $\mathbb{C} =_{\mathrm{df}} \mathbf{Set}$, $\mathbb{J} =_{\mathrm{df}} \mathbf{Fin}$, $J =_{\mathrm{df}} J_{\mathrm{f}}$, the counit is an isomorphism if and only if the monad T is finitary. This is important for us: the categories of monads on $\mathbb{C}$ and relative monads on J are generally not equivalent.

Example 8. For the powerset monad $\mathcal{P}$ on $\mathbf{Set}$, we have that $\mathcal{P}\,X$ is the powerset of a set X, $\mathcal{P}^\flat\,X {=_{\mathrm{df}}} \mathcal{P}\,(J_{\mathrm{f}}\,X)$ is the powerset of a finite cardinal X, and $\mathcal{P}^{\flat\#}\,X {=_{\mathrm{df}}} \mathrm{Lan}_{J_{\mathrm{f}}}\,\mathcal{P}^{\#}\,X$ is the finitary powerset (the set of finite subsets) of a (possibly infinite) set X. The difference between $\mathcal{P}$ and $\mathcal{P}^{\flat\#}$ arises because $\mathcal{P}$ is not finitary.

Example 9. For the relative monad Vec on J_{f}, $\mathsf{Vec}^{\#}\,X$ is the space of vectors over a possibly infinite coordinate system X that may only have finitely many non-zero components.

Example 10. For the relative monad Lam on J_{f}, we have that $\mathsf{Lam}\,X$ is the set of λ-terms over a finite, nameless context X and $\mathsf{Lam}^{\#}\,X$ is given by the set of λ-terms over a possibly infinite, name-carrying context X. The functor $\mathsf{Lam}^{\#}$ is the carrier of the initial algebra of the functor $F \in [\mathbf{Set}, \mathbf{Set}] \to [\mathbf{Set}, \mathbf{Set}]$ defined by $F\,G\,X =_{\mathrm{df}} X + G\,X \times G\,X + G\,(1 + X)$.

For the relative monad Lam^∞ the picture is different. $\mathsf{Lam}^\infty\,X$ is the set of non-wellfounded λ-terms over a finite, nameless context, but $\mathsf{Lam}^{\infty\#}\,X$ is the set of non-wellfounded λ-terms using a finite number of variables from a possibly infinite, name-carrying context. This differs from the non-finitary carrier of the final coalgebra of F, capturing general non-wellfounded λ-terms that may use infinitely many variables.

5 Arrows as a Special Case of Relative Monads

We now turn to a whole class of examples, Hughes's arrows [9]. As we shall see, arrows are relative monads on the Yoneda embedding. Arrow are commonly perceived as a generalization of monads. With relative monads, this relationship is turned upside down!

The rigorous definition of arrows by Heunen and Jacobs [8] is as follows:[1]

Definition 5. *A* (**Set**-*valued)* arrow *on a category* $\mathbb{J}$ *is given by*

- *a function* $R \in |\mathbb{J}| \times |\mathbb{J}| \to |\mathbf{Set}|$,
- *for any* $X, Y \in |\mathbb{J}|$, *a function* $\mathsf{pure} \in \mathbb{J}\,(X, Y) \to R\,(X, Y)$,
- *for any* $X, Y, Z \in |\mathbb{J}|$, *a function* $(\lll) \in R\,(Y, Z) \times R\,(X, Y) \to R\,(X, Z)$,

satisfying the conditions

- $\mathsf{pure}\,(g \circ f) = \mathsf{pure}\, g \circ \mathsf{pure}\, f$,
- $s \lll \mathsf{pure}\,\mathrm{id} = s$,
- $\mathsf{pure}\,\mathrm{id} \lll r = r$,
- $t \lll (s \lll r) = (t \lll s) \lll r$.

It follows from the conditions that R is functorial (contravariantly in the first argument), i.e., $R : \mathbb{J}^{\mathrm{op}} \times \mathbb{J} \to \mathbf{Set}$, which is the same as to say that R is an endoprofunctor on $\mathbb{J}$, and pure and $\lll$ are natural/dinatural.

A monad $(T, \eta, (-)^*)$ on $\mathbb{J}$ defines an arrow $(R, \mathsf{pure}, \lll)$ on $\mathbb{J}$ by $R\,(X, Y) =_{\mathrm{df}} \mathbf{Kl}(T)\,(X, Y)$, $\mathsf{pure}\, f =_{\mathrm{df}} L\, f$ and $\ell \lll k =_{\mathrm{df}} \ell \circ^T k$ where L is the left adjoint in the Kleisli adjunction and $\circ^T$ is the Kleisli composition.

We show now that an arrow on $\mathbb{J}$ is the same thing as a relative monad on the Yoneda embedding $\mathbf{Y} \in \mathbb{J} \to [\mathbb{J}^{\mathrm{op}}, \mathbf{Set}]$ defined by $\mathbf{Y}\, X\, Y =_{\mathrm{df}} \mathbb{J}\,(Y, X)$.

Theorem 9. *1. An arrow* $(R, \mathsf{pure}, \lll)$ *on* $\mathbb{J}$ *gives rise to a relative monad* $(T, \eta, (-)^*)$ *on* $\mathbf{Y}$ *defined by* $T\, X\, Y =_{\mathrm{df}} R\,(Y, X)$, $T\, \_\, f\, r =_{\mathrm{df}} r \lll f$, $\eta\, f =_{\mathrm{df}} \mathsf{pure}\, f$, $k^*\, r =_{\mathrm{df}} k\,\mathrm{id} \lll r$.

2. A relative monad $(T, \eta, (-)^*)$ *on* $\mathbf{Y}$ *gives rise to an arrow* $(R, \mathsf{pure}, \lll)$ *on* $\mathbb{J}$ *defined by* $R\,(X, Y) =_{\mathrm{df}} T\, Y\, X$, $\mathsf{pure}\, f =_{\mathrm{df}} \eta\, f$, $s \lll r =_{\mathrm{df}} (\lambda f.\, T\, \_\, f\;\, s)^* r$. *(The last item is well-defined, as* $\lambda f.\, T\, \_\, f\;\, s$ *is natural.)*

3. The above is a bijective correspondence.

The arrows on $\mathbb{J}$ and relative monads on $\mathbf{Y}$ form categories and the bijection between them extends to an equivalence of their categories.

It is easy to verify that the Freyd category of an arrow is the Kleisli category of the corresponding relative monad. Jacobs et al. [10] have previously proved that "Freyd is Kleisli for arrows" taking "Kleisli for arrows" to mean a construction that is Kleisli-like under a 2-categorical view of the Kleisli construction for monads. We can take it to mean "Kleisli for arrows as relative monads".

[1] Since we compare arrows to monads, not strong monads, we mean "weak" arrows here: $\mathbb{J}$ does not have to be symmetric monoidal and no first operation is required.

The Yoneda embedding is well-behaved. We reconstruct the result of Heunen and Jacobs [8] about arrows being monoids as an instance of a generality.

Theorem 10. *If $\mathbb{J}$ is small, then $\mathbf{Y}$ is well-behaved, hence the category $[\mathbb{J}, [\mathbb{J}^{\mathrm{op}}, \mathbf{Set}]]$ is monoidal. An arrow on $\mathbb{J}$ is a monoid in this category.*

Jacobs and Heunen considered the special case of arrows and showed an arrow to be a monoid in $[\mathbb{J}^{\mathrm{op}} \times \mathbb{J}, \mathbf{Set}]$ (the category of endoprofunctors on $\mathbb{J}$) as a monoidal category, which is an equivalent statement.

6 Conclusions and Further Work

We have introduced a generalisation of monads, relative monads, which is motivated by examples and subsumes arrows, a well-known generalisation of monads. Indeed, when moving to a more precise type discipline, the illusion that everything takes place in only one ambient category (say, **Set**) can no longer be maintained and as a consequence we have to revisit the categorically inspired concepts of functional programming. We believe that our examples demonstrate that monad-like entities which are not endofunctors are natural; fortunately, they are precisely monoids in the functor category. We also suggest that our presentation of relative monads given in Sect. 2.1 is accessible for functional programmers, indeed it does not differ substantially from ordinary monads.

Our development is only the first step. Due to lack of space, we have not written about monad maps; we did not comment on the relationship between relative adjunctions and adjunctions etc.; strong monads (esp. versus strong arrows) are a further additional topic. We will elsewhere comment on the relation of our relative monads to the the recent generalization of monads by Spivey [17] that was also motivated by programming examples: he fixes a functor $K \in \mathbb{C} \to \mathbb{J}$ (notice the direction) to then look for monad-like structures with an underlying functor $\mathbb{J} \to \mathbb{C}$. With Paul Levy we have checked that a fair amount of monad theory transfers to his generalized monads, but they are not monoids in $[\mathbb{J}, \mathbb{C}]$ unless K has a left adjoint, in which case they are equivalent to relative monads.

It seems clear that many of the concepts known from ordinary monads carry over to the relative setting. We have already mentioned Jaskelioff's work on monad transformers which is expressed in a general monoidal setting and hence carries over to relative monads. We hope that this generalisation of the monadic approach leads to new programming structures supporting a greater reusability of concepts and programs.

Acknowledgements. We are grateful to Paul Levy for valuable comments and hints. T. Altenkirch was supported by the Engineering and Physical Sciences Research Council (EPSRC) grant no. EP/G034109/1. J. Chapman and T. Uustalu were supported by the Estonian Centre of Excellence in Computer Science, EXCS, financed by the European Regional Development Fund. T. Uustalu was also supported by the Estonian Science Foundation grant no. 6940.

References

1. Abbott, M., Altenkirch, T., Ghani, N.: Containers—constructing strictly positive types. Theor. Comput. Sci. 342(1), 3–27 (2005)
2. Abramsky, S., Coecke, B.: A categorical semantics of quantum protocols. In: Proc. of 19th Ann. IEEE Symp. on Logic in Computer Science, LICS 2004, pp. 415–425. IEEE CS Press, Los Alamitos (2004)
3. Agda team: Agda (2009), `http://appserv.cs.chalmers.se/users/ulfn/wiki/agda.php`
4. Altenkirch, T., Green, A.: The Quantum IO Monad. In: Gay, S., McKie, I. (eds.) Semantic Techniques in Quantum Computation, pp. 173–205. Cambridge Univ. Press, Cambridge (2009)
5. Altenkirch, T., Reus, B.: Monadic presentations of lambda terms using generalized inductive types. In: Flum, J., Rodríguez-Artalejo, M. (eds.) CSL 1999. LNCS, vol. 1683, pp. 453–468. Springer, Heidelberg (1999)
6. Fiore, M.: Semantic analysis of normalisation by evaluation for typed lambda calculus. In: Proc. of 4th ACM SIGPLAN Conf. on Principles and Practice of Declarative Programming, PPDP 2002, pp. 26–37. ACM Press, New York (2002)
7. Fiore, M., Plotkin, G., Turi, D.: Abstract syntax and variable binding. In: Proc. of 14th Ann. IEEE Symp. on Logic in Computer Science, LICS 1999, pp. 193–202. IEEE CS Press, Los Alamitos (1999)
8. Heunen, C., Jacobs, B.: Arrows, like monads, are monoids. In: Brookes, S., Mislove, M. (eds.) Proc. of 22nd Ann. Conf. on Mathematical Foundations of Programming Semantics, MFPS XXII. Electron. Notes in Theor. Comput. Sci, vol. 158, pp. 219–236. Elsevier, Amsterdam (2006)
9. Hughes, J.: Generalising monads to arrows. Sci. of Comput. Program. 37(1-3), 67–111 (2000)
10. Jacobs, B., Heunen, C., Hasuo, I.: Categorical semantics for arrows. J. of Funct. Program. 19(3-4), 403–438 (2009)
11. Jaskelioff, M.: Lifting of Operations in Modular Monadic Semantics. PhD thesis, University of Nottingham (2009)
12. Manes, E.G.: Algebraic Theories. Springer, Heidelberg (1976)
13. McBride, C., Paterson, R.: Applicative programming with effects. J. of Funct. Program. 18(1), 1–13 (2008)
14. Morris, P., Altenkirch, T.: Indexed containers. In: Proc. of 24th Ann. IEEE Symp. on Logic in Computer Science, LICS 2009, pp. 277–285. IEEE CS Press, Los Alamitos (2009)
15. Piponi, D.: Commutative monads, diagrams and knots. In: Proc. of 14th Int. Conf. on Functinal Programming, ICFP 2009, p. 231. ACM Press, New York (2009) (see the video)
16. Power, J., Robinson, E.: Premonoidal categories and notions of computation. Math. Struct. in Comput. Sci. 7(5), 453–468 (1997)
17. Spivey, J.M.: Algebras for combinatorial search. J. of Funct. Program. 19(3-4), 469–487 (2009)
18. Uustalu, T., Vene, V.: Comonadic notions of computation. In: Adamék, J., Kupke, C. (eds.) Proc. of 9th Int. Wksh. on Coalgebraic Methods in Computer Science, CMCS 2008. Electron. Notes in Theor. Comput. Sci., vol. 203(5), pp. 263–284. Elsevier, Amsterdam (2008)
19. Vizzotto, J.K., Altenkirch, T., Sabry, A.: Structuring quantum effects: Superoperators as arrows. Math. Struct. in Comput. Sci. 16(3), 453–468 (2006)

CIA Structures and the Semantics of Recursion[⋆]

Stefan Milius[1,⋆⋆], Lawrence S. Moss[2], and Daniel Schwencke[1]

[1] Institut für Theoretische Informatik, Technische Universität Braunschweig, Germany
mail@stefan-milius.eu, schwencke@iti.cs.tu-bs.de
[2] Department of Mathematics, Indiana University, Bloomington, IN, USA
lsm@cs.indiana.edu

Abstract. Final coalgebras for a functor serve as semantic domains for state based systems of various types. For example, formal languages, streams, non-well-founded sets and behaviors of CCS processes form final coalgebras. We present a uniform account of the semantics of recursive definitions in final coalgebras by combining two ideas: (1) final coalgebras are also initial *completely iterative algebras* (cia); (2) additional algebraic operations on final coalgebras may be presented in terms of a *distributive law* λ. We first show that a distributive law leads to new extended cia structures on the final coalgebra. Then we formalize recursive function definitions involving operations given by λ as recursive program schemes for λ, and we prove that unique solutions exist in the extended cias. We illustrate our results by the four concrete final coalgebras mentioned above, e. g., a finite stream circuit defines a unique stream function and we show how to define new process combinators from given ones by sos rules involving recursion.

Keywords: recursion, semantics, completely iterative algebra, coalgebra, distributive law.

1 Introduction

Recursive definitions are a useful tool to specify infinite system behavior. For example, Milner [21] proved that in his calculus CCS, one may specify a process uniquely by the equation $P = a.(P|c) + b$. More generally, such recursive equations have unique solutions whenever each recursion variable is in the scope of some action prefix. Another example is the shuffle product on streams of real numbers uniquely defined by $r.\sigma \otimes s.\tau = rs.(r.\sigma \otimes \tau + \sigma \otimes s.\tau)$. And as a third example consider non-well-founded sets [2, 10], a framework originating as a semantic basis for circular definitions. Here we can solve recursive function definitions such as $g(x) = \{g(\mathcal{P}(x)) \times x, x\}$ uniquely. It is the aim of this paper to develop abstract tools and results that explain why there exist unique solutions to all the aforementioned equations.

The key observation is that streams, non-well-founded sets and process behaviors constitute *final coalgebras* for certain functors on appropriate categories. Furthermore, the structure $c : C \to HC$ of a final coalgebra is an isomorphism [14], and the H-algebra (C, c^{-1}) is the initial *completely iterative algebra* (cia) for H [18]; cias are

⋆ In this extended abstract all proofs are omitted. They can be found in the full version of our paper at www.stefan-milius.eu.

⋆⋆ Supported by the German Research Foundation (DFG) under the project MI 717/2–1.

algebras in which recursive (function) definitions involving the operations given by c^{-1} can be solved uniquely. However, cia structures for H are not sufficient to yield the existence and uniqueness of solutions in our motivating examples; these involve additional algebraic operations not captured by H. For example, $|, +$ in CCS, the stream addition $+$, and powerset and cartesian product $\mathcal{P}, \times$ in the example from non-well-founded set theory.

Additional algebraic operations are often presented by *distributive laws* in various guises. In process algebra one defines operations such as $|$ or $+$ by structural operational semantics (sos) [1]. Plotkin and Turi [22] showed how to capture sos rules as a distributive law of the functor (or monad) M describing the desired algebraic operations over the "behavior" functor H. This distributive law then induces an algebraic structure for M on the final H-coalgebra C. Other instances of distributive laws are behavioral differential equations in stream calculus, see [24], and definitions of operations on non-well-founded sets.

Bartels systematically studies definition formats giving rise to distributive laws in his thesis [9] (see also [8]) and shows how to solve parameter-free first order recursive equations involving operations presented by a distributive law.

After recalling his results in Section 2 we extend them in Section 3 by combining them with our previous work in [3, 18, 19]. We first prove (Theorems 3.2 and 3.3) that the final H-coalgebra carries the structure of a cia for HM and for MHM. These results show how to construct new structures of cias on C out of the initial one using a distributive law. This improves Bartels' result in the sense that first order recursive definitions may employ constant parameters in the final coalgebra. This also explains why recursively defined operations may be used in subsequent recursive definitions.

In Section 4 we obtain new ways to provide the semantics of recursive definitions by applying the existing solution theorems with the new cia structures, and in Section 5 we turn our attention to functional recursive definitions like the above shuffle product or the above function g on non-well-founded sets. We introduce for a distributive law λ the notion of a recursive program scheme (λ-rps, for short). Our main result is that any λ-rps has a unique solution in the final coalgebra C. Moreover, we show that these solutions extend the cia structure of C, which means that they can be used in subsequent recursive definitions. This compositionality of taking solutions of recursive equations does not appear in any previous work in this generality. In fact, we believe that our result is the first one that allows to obtain recursively defined operations directly as the unique solutions of their specifications.

Finally, in Section 6 we demonstrate the value of our results by instantiating them in four different concrete applications: (1) CCS-processes—we explain how Milner's solution theorem from [21] arises as a special case of Theorem 3.3, and we also show how to define new process combinators recursively from given ones; (2) streams of real numbers—here we prove that every finite stream circuit defines a unique stream function; (3) non-well-founded sets—we prove that operations on non-well-founded sets are uniquely determined as solutions of λ-rps's; (4) formal languages—here we show how operations on formal languages like union, concatenation, complement, etc. arise step-by-step using the compositionality of unique solutions of λ-rps's.

Related Work. The work in [22] was taken further by Lenisa, Power and Watanabe in [15, 16]. Jacobs [13] shows how to apply Bartels' result to obtain the (first order) solution theorems from [3, 18]. Capretta et al. [11] work in a dual setting and generalize the results of [8] beyond terminal coalgebras and they also obtain the (dual of) the solution theorem from [3, 18] by an application of their general results. Our Theorems 3.2 and 3.3 are similar to results in [11] but extend the ones of [8] in a different direction by considering parameters in recursive definitions. So our results in the present paper go beyond what can be accomplished with previous work. For example, while [16] gives an abstract explanation of adding operations to a process calculus, it gives no account of the kind of compositionality we have in our results.

2 Distributive Laws and Bialgebras

We shall assume some familiarity with basic notions from category theory such as functors, (initial) algebras and (final) coalgebras, monads, see e. g. [17, 23, 5].

Suppose we are given an endofunctor H on some category $\mathcal{A}$ describing the behavior type of a class of systems. In our work we shall be interested in additional algebraic operations on the final coalgebra C for H. The type of these algebraic operations is given by an endofunctor M on $\mathcal{A}$. Specification of algebraic operations by sos rules is abstractly captured by giving a distributive law of M over H, see e. g. [22, 8, 13]. Often M comes with the extra structure of a pointed endofunctor or a monad. Our goal is to provide a setting in which recursive equations involving the algebraic operations described by M can be uniquely solved.

Assumption 2.1. Throughout this section we assume that $H : \mathcal{A} \to \mathcal{A}$ is a functor, and that $c : C \to HC$ is a final coalgebra. In addition, we assume that M is a *pointed endofunctor* on $\mathcal{A}$, i. e., M comes equipped with a natural transformation $\eta : \mathrm{Id} \to M$.

Definition 2.2. *(1) An* algebra for (M, η) *is a pair* (A, a) *where* A *is an object of* $\mathcal{A}$ *and* $a : MA \to A$ *is a morphism satisfying the* unit law $a \cdot \eta_A = \mathrm{id}_A$.

(2) A distributive law *of* M *over* H *is a natural transformation* $\lambda : MH \to HM$ *such that we have*

$$H\eta = (\, H \xrightarrow{\eta H} MH \xrightarrow{\lambda} HM \,)\,.$$

Remark 2.3. (1) In most concrete examples, M in Definition 2.2 is part of a monad (M, η, μ). Then any distributive law for the monad M over H is obviously also a distributive law for the pointed endofunctor M over H: in fact, recall that a distributive law of the monad M over H is a distributive law as in Definition 2.2(2) above additionally satisfying one obvious law concerning compatibility of λ and μ.

(2) Suppose that M is a free monad $\widehat{K}$ on an endofunctor $K : \mathcal{A} \to \mathcal{A}$ given objectwise by free K-algebras $\widehat{K}X$. Bartels [9] (Lemma 3.4.24) shows that distributive laws $\lambda : \widehat{K}H \to H\widehat{K}$ (of the monad $\widehat{K}$ over H) correspond to natural transformations $\ell : KH \to H\widehat{K}$. Indeed, given λ we get $\ell = \lambda \cdot \kappa H$, where $\kappa : K \to \widehat{K}$ is the universal natural transformation of the free monad. Conversely, let η and μ be the unit

and multiplication of the free monad $\widehat{K}$. Given ℓ, we see that $H\mu_X \cdot \ell_{\widehat{K}X} : KH\widehat{K}X \to H\widehat{K}\widehat{K}X \to H\widehat{K}X$ is an algebra for K. Thus, by the freeness of the algebra $\widehat{K}HX$, there exists a unique K-algebra homomorphism $\lambda_X : \widehat{K}HX \to H\widehat{K}X$ extending $H\eta_X$, i. e., such that $\lambda_X \cdot \eta_{HX} = H\eta_X$. One readily shows that λ is a distributive law of $\widehat{K}$ over H.

Construction 2.4. [8, 9] Let $\lambda : MH \to HM$ be a distributive law of the pointed endofunctor M over H. By using coinduction (i. e., the universal property of the final coalgebra) we define an M-algebra structure on C as the unique coalgebra homomorphism b from the coalgebra $\lambda_C \cdot Mc : MC \to HMC$ to the final coalgebra, i. e., $b : MC \to C$ is such that

$$c \cdot b = Hb \cdot \lambda_C \cdot Mc\,. \tag{2.1}$$

Bartels [8] showed that (C, b) is indeed an algebra for M. Moreover, if M is a monad and λ a distributive law of M over H, then (C, b) is an Eilenberg-Moore algebra for the monad M.

Definition 2.5. *For every distributive law λ we call $b : MC \to C$ from Construction 2.4 the* λ-interpretation *in C. In the case described in Remark 2.3(2), where λ is induced by ℓ, we shall also say that b is the* ℓ-interpretation *in C.*

Examples 2.6. We review a couple of examples of interest in this paper where $\mathcal{A} = \mathsf{Set}$. We shall elaborate these examples in Section 6.

(1) Formal languages. Consider the endofunctor $HX = X^A \times 2$, where $2 = \{0, 1\}$. Coalgebras for H are precisely the (possibly infinite) deterministic automata over the set A (as an alphabet). The final coalgebra $c : C \to HC$ consists of all formal languages with $c(L) = (\lambda a.D_a(L), i)$ with $i = 1$ iff the empty word ε is in L and where $D_a(L) = \{ w \mid aw \in L \}$.

To specify e. g. the intersection of formal languages via a distributive law, let $KX = X \times X$ and let $\ell : KH \to HK$ be given by $\ell_X((f, i), (g, j)) = (\langle f, g\rangle, i \wedge j)$ where $\wedge$ denotes the "and"-operation on $\{0, 1\}$. Take $M = K + \mathrm{Id}$ and $\lambda = \mathsf{can} \cdot (\ell + \mathrm{id}_H) : MH \to HM$, where $\mathsf{can} = [H\mathsf{inl}, H\mathsf{inr}] : HK + H \to H(K + \mathrm{Id})$. Then the λ-interpretation $b : C \times C + C \to C$ has its left-hand component given by intersection of formal languages.

(2) Streams have been studied in a coalgebraic setting by Rutten [24]. Here we take the functor $HX = \mathbb{R} \times X$ whose final coalgebra (C, c) is carried by the set $\mathbb{R}^\omega$ of all streams over $\mathbb{R}$ and $c = \langle \mathsf{hd}, \mathsf{tl}\rangle : \mathbb{R}^\omega \to \mathbb{R} \times \mathbb{R}^\omega$ is given by the usual head and tail functions on streams.

Consider the functor $KX = \mathbb{R} \times X + X \times X$ corresponding to the signature Σ with a unary operation symbol $r.(-)$ for every real number r and with a binary operation symbol zip. Now zip is defined by the behavioral differential equation $\mathsf{zip}(r.x, s.y) = (r, \mathsf{zip}(s.y, x))$. This can conveniently be expressed as a natural transformation $\ell : KH \to H\widehat{K}$, where $\widehat{K}$ is the free monad on K (recall that $\widehat{K}X$ is the set of all Σ-terms on the set X of variables). Indeed, define ℓ for the two coproduct components of $KHX = \mathbb{R} \times HX + HX \times HX$ by $\ell_X(r, (s, x)) = (r, s.x)$ and $\ell_X((r, x), (s, y)) = (r, \mathsf{zip}(s.y, x))$, and note that the right-hand sides are pairs in $\mathbb{R} \times \widehat{K}X$. The ℓ-interpretation $b : \widehat{K}C \to C$ interprets Σ-terms built from the operation symbols as expected.

(3) Processes. We shall be interested in Milner's CCS [21]. Let κ be a regular cardinal and $\mathcal{P}_\kappa$ be the functor assigning to the set X the set of all subsets Y with $|Y| < \kappa$. Here we consider the functor $HX = \mathcal{P}_\kappa(A \times X)$ where A is some fixed alphabet of actions. Following Milner [21], we assume that for every $a \in A$ we also have a complement $\bar{a} \in A$ (so that $\bar{\bar{a}} = a$) and a special silent action $\tau \in A$.

Recall that the final coalgebra for the finite power set functor $\mathcal{P}_{\mathsf{fin}}$ was described by Worrell [27]: it is carried by the set of all strongly extensional finitely branching trees, where an unordered tree t is called *strongly extensional* if two subtrees rooted at distinct children of some node of t are never bisimilar. Similarly, the final coalgebra for the countable power set functor $\mathcal{P}_{\mathsf{c}}$ is carried by the set of all strongly extensional countably branching trees, see [25]. The technique by which this result is obtained in loc. cit. generalizes to the functor $\mathcal{P}_\kappa(A \times X)$ from above: its final coalgebra C turns out to consist of all strongly extensional κ-branching trees with edges labelled in A; strongly extensional has the analogous meaning as above: two subtrees rooted at distinct children of some node are never bisimilar if both edges to the children carry the same label. The elements of C can be considered as (denotations of) CCS-agents modulo strong bisimilarity.

Notice that the inverse $c^{-1} : \mathcal{P}_\kappa(A \times C) \to C$ assigns to a set $\{(a_i, E_i) \mid i < \kappa\}$ of pairs of actions and agents the agent $\sum_{i<\kappa} a_i.E_i$. The usual process combinators "$a.-$" (prefixing), "$|$" (composition), "$\sum_{i<\kappa}$" (sum), $-[f]$ (relabelling) and "$-\backslash L$" (restriction) are given by sos rules. Let E, E', F, F' be agents and $a \in A$ some action, then these rules are:

$$\frac{E\xrightarrow{a}E'}{E|F\xrightarrow{a}E'|F} \qquad \frac{F\xrightarrow{a}F'}{E|F\xrightarrow{a}E|F'} \qquad \frac{E\xrightarrow{a}E' \quad F\xrightarrow{\bar{a}}F'}{E|F\xrightarrow{\tau}E'|F'}\ (a\neq\tau)$$

$$\frac{}{a.E\xrightarrow{a}E} \qquad \frac{E_j\xrightarrow{a}E'_j}{(\sum_{i<\kappa}E_i)\xrightarrow{a}E'_j}\ (j<\kappa) \qquad \frac{E\xrightarrow{a}E'}{E[f]\xrightarrow{f(a)}E'[f]} \qquad \frac{E\xrightarrow{a}E'}{E\backslash L\xrightarrow{a}E'\backslash L}\ (a,\bar{a}\notin L)$$

Now let K be the polynomial functor for the signature given by taking the combinators as operation symbols. Then the above rules are easily seen to give a natural transformation $\ell : KH \to H\widehat{K}$, and the ℓ-interpretation $b : \widehat{K}C \to C$ in C evaluates all terms built from the considered combinators in C. Further details are presented in Section 6.1.

Our first result (Theorem 3.2) improves a result from [8, 9] that we now recall. We are interested in λ-*bialgebras* for a distributive law $\lambda : MH \to HM$. We shall not recall the formal definition here since it is not needed for this paper. We only note that for the λ-interpretation $b : MC \to C$ the triple (C, b, c) is a λ-bialgebra; in fact, it is the final one, see [9], Corollary 3.4.12).

Definition 2.7. *A* λ-equation *is an HM-coalgebra; that is, a morphism of the form $e : X \to HMX$. A* solution *of e in the λ-bialgebra (C, b, c) is a morphism $e^\dagger : X \to C$ such that the following equation holds: $c \cdot e^\dagger = Hb \cdot HMe^\dagger \cdot e$.*

Theorem 2.8. [8, 9] *Let $\mathcal{A}$ be a cocomplete category, and let λ be a distributive law of the pointed endofunctor M over H. Then every λ-equation has a unique solution in (C, b, c).*

Bartels proved a version of this theorem in the setting where the functor M is not necessarily pointed. So this result is just a slight variation on Bartels' Theorem 4.2.2.

It is worthwhile to note that Bartels' work also contains a similar result for the case where M is a monad and where λ is a distributive law of M over H, see [9], Corollary 4.3.6. In that case the assumption that $\mathcal{A}$ is cocomplete is not necessary. This last result is the dual of a result obtained independently and at the same time by Uustalu, Vene and Pardo (see [26], Theorem 1), and in [11] Capretta, Uustalu and Vene generalize this work further obtaining Theorem 2.8 as a special case.

3 Completely Iterative Algebras

In addition to our assumptions in 2.1 we assume from now on that our base category $\mathcal{A}$ is cocomplete.

It is our aim in this section to extend Theorem 2.8 so as to obtain several new structures of completely iterative algebras (for functors other than H) on C. We briefly recall the basic definitions and one example; more details and examples can be found in [18, 7, 19].

Definition 3.1. *[18] A* flat equation morphism *in an object A (of parameters) is a morphism $e : X \to HX + A$. An H-algebra $a : HA \to A$ is called* completely iterative *(or a* cia*, for short) if every flat equation morphism in A has a unique solution, i. e., for every $e : X \to HX + A$ there exists a unique morphism $e^\dagger : X \to A$ such that*

$$e^\dagger = (X \xrightarrow{e} HX + A \xrightarrow{He^\dagger + A} HA + A \xrightarrow{[a,A]} A).$$

Recall that the inverse of the structure $c : C \to HC$ of the final coalgebra is, equivalently, an initial cia for H, see [18]. The following show that the distributive law λ induces further structures of completely iterative algebras on C.

Theorem 3.2. *Consider the algebra $k = (HMC \xrightarrow{Hb} HC \xrightarrow{c^{-1}} C)$, where $b : MC \to C$ is the λ-interpretation in C. Then (C, k) is a cia for the composite functor HM.*

Theorem 3.3. (Sandwich Theorem) *Consider the algebra $k' = MHMC \xrightarrow{Mk} MC \xrightarrow{b} C$, where $b : MC \to C$ is the λ-interpretation in C and $k = c^{-1} \cdot Hb$ as in Theorem 3.2. Then (C, k') is a cia for the composite functor MHM.*

These two results extend Theorem 2.8 in two important ways. Firstly, the structure of a cia allows one to reuse solutions of given recursive specification by using constants in C on the right-hand sides of recursive equations, i. e., in a cia we can solve open recursive equations not just closed ones. This gives a clear explanation of why it is possible to use recursively defined objects (streams, processes, etc.) in subsequent recursive definitions. This kind of compositionality of the unique solutions is a useful and desired property often employed in specifications.

Secondly, Theorem 3.3 permits the right-hand sides of recursive specifications to be from a wider class. For example, Milner's solution theorem for CCS (see [21], Chapter 4, Proposition 14) allows recursion over process terms E in which the recursion variables occur within the scope of some prefixing combinator $a.-$. This combinator can occur *anywhere within* E, not necessarily at the head of that term. Hence, Theorem 3.3 allows us to obtain Milner's result as a special case, directly. This will be explained in detail in Section 6.1.

4 Solution Theorems for Free

Using the new cia structures obtained from Theorems 3.2 and 3.3, the existing body of results on the semantics of recursion in cias [3, 18, 19] now gives us further theorems.

We begin with a terse review of some terminology from the area [3, 18, 19]. We assume that, in addition to the final H-coalgebra C, for every object X the final coalgebra $T^H X$ for $H(-)+X$ exists, i. e., in the terminology of loc. cit., H is *iteratable*. Our examples in 2.6 are all iteratable endofunctors of Set.

The inverse of the final coalgebra structure $T^H X \to HT^H X + X$ gives morphisms $\eta^H_X : X \to T^H X$ and $\tau^H_X : HT^H X \to T^H X$. It is proved in [18] that $(T^H X, \tau^H_X)$ is a free cia on X with the universal arrow η^H_X. From this it easily follows that T^H is the object assignment of a monad and that η^H and τ^H are natural transformations. Denote by κ^H the natural transformation $\tau^H \cdot H\eta^H : H \to T^H$.

Let (A,a) be a cia for H. Then there is a unique homomorphism $\widetilde{a} : T^H A \to A$ of H-algebras such that $\widetilde{a} \cdot \eta^H_A = \mathrm{id}_A$. We call $\widetilde{a}$ the *evaluation morphism* associated to A. Notice that $\widetilde{a} \cdot \kappa^H_A = a$.

In our previous work we have shown how to obtain unique solutions of more general (first-order) recursive equations than the flat ones appearing in the definition of a cia:

Definition 4.1. [3, 18] *An* equation morphism *is a morphism of the form $e : X \to T^H(X+A)$. It is called* guarded *if there exists a factorization $f : X \to HT^H(X+A)+A$ such that $e = [\tau^H_{X+A}, \eta^H_{X+A} \cdot \mathsf{inr}] \cdot f$.*

A solution *of an equation morphism e in a cia (A,a) is a morphism $e^\dagger : X \to A$ such that the following equation holds: $e^\dagger = \widetilde{a} \cdot T^H([e^\dagger, \mathrm{id}_A]) \cdot e$.*

Theorem 4.2. [18] *Let (A,a) be a cia for H. Then every guarded equation morphism has a unique solution in A.*

An even more general property of cias was proved in [19]; one can solve recursive function definitions uniquely in a cia. We recall the respective result.

Definition 4.3. *Let V be an endofunctor such that $H+V$ is iteratable. A* recursive program scheme *(rps, for short) is a natural transformation $e : V \to T^{H+V}$. It is called* guarded *if there exists a natural transformation $f : V \to HT^{H+V}$ such that $e = \tau^{H+V} \cdot (\mathsf{inl}T^{H+V}) \cdot f$, where $\mathsf{inl} : H \to H+V$.*

Now let (A,a) be a cia for H. An interpreted solution *of e in A is a V-algebra structure $e^\ddagger_A : VA \to A$ giving rise to an Eilenberg-Moore algebra structure $\beta : T^{H+V}A \to A$ (i. e., $\beta \cdot \kappa^{H+V}_A = [a, e^\ddagger_A]$) such that we have $e^\ddagger_A = \beta \cdot e_A$.*

Theorem 4.4. [19] *In a cia, every guarded rps has a unique interpreted solution.*

We are now able to prove more:

Theorem 4.5. *Let $e : V \to T^{H+V}$ be a guarded rps, and let $a : HA \to A$ be a cia. Then the interpreted solution $e^\ddagger_A : VA \to A$ extends the cia structure on A; more precisely, the algebra $[a, e^\ddagger_A] : (H+V)A \to A$ is a cia for $H+V$.*

The last result implies that operations obtained as solutions of recursive program schemes can be used in subsequent recursive function definitions, which will still have unique solutions. For the special case of interpreted rps solutions in cias this strengthens the results in [20].

Now assume that the composite HM is iteratable. By applying the above two Theorems 4.2 and 4.4 to the cia $k : HMC \to C$ from Theorem 3.2 we get two more solutions theorems for free, and two similar theorems hold for the cia $k' : MHMC \to C$ obtained from Theorem 3.3:

Corollary 4.6. *Every guarded equation morphism* $e : X \to T^{HM}(X + C)$ *has a unique solution in the cia* (C, k).

Corollary 4.7. *Every guarded rps* $e : V \to T^{HM+V}$ *has a unique interpreted solution in the cia* (C, k).

5 Recursive Function Definitions over the Behavior

All results we have seen so far do not allow us to obtain functions such as the shuffle product on streams (see introduction) as a unique solution since its definition refers to the behavior of the arguments of the function. In this section we introduce a special form of recursive program schemes called λ-*rps's* which accommodate such examples. We prove that every λ-rps has a unique solution in the final coalgebra C, and this solution extends the cia structure for HM on C given by Theorem 3.2—this is a compositionality result similar to the one given in Theorem 4.5 for ordinary rps.

We show that every λ-rps easily gives rise to a distributive law, and so the results in this section are essentially an application of the work in [8] and our results in Section 3. However, to prove the uniqueness of solutions we need that free algebras can always be constructed as colimits of transfinite chains. Hence, the following

Assumption 5.1. We assume that our base category $\mathcal{A}$ is Set, and in this section we only consider endofunctors on Set that are *accessible*, i. e., they preserve α-filtered colimits for some regular cardinal α. This implies that they are iteratable.

We further assume that the functor $M = \widehat{K}$ is the free monad on some endofunctor K and that $\lambda : \widehat{K}H \to H\widehat{K}$ arises from some natural transformation $\ell : KH \to H\widehat{K}$ as explained in Remark 2.3(2).

Notation 5.2. (1) Recall that for an accessible endofunctor $F : \mathsf{Set} \to \mathsf{Set}$ the free monad $\widehat{F}$ does indeed exist and is given objectwise by the free F-algebras on X, see [4]. We denote by $\varphi_X : F\widehat{F}X \to \widehat{F}X$ the structure of this algebra and by $u_X : X \to \widehat{F}X$ the universal morphism. We abuse notation and write φ_X and u_X for different functors F.
(2) For any F-algebra $a : FA \to A$ we denote the corresponding Eilenberg-Moore algebra for $\widehat{F}$ by $\widehat{a} : \widehat{F}A \to A$. Observe that $\widehat{a}$ is the unique F-algebra homomorphism with $\widehat{a} \cdot u_A = \mathrm{id}_A$.
(3) Let $b : \widehat{K}C \to C$ be the λ-interpretation in C. We denote by b_0 the corresponding K-algebra structure $b_0 = b \cdot \varphi_C \cdot Ku_C$. Then clearly we have $b = \widehat{b_0}$.

Definition 5.3. *A recursive program scheme w. r. t. λ (shortly, λ-*rps*) is a natural transformation $e : VH \to H\widehat{K+V}$.*

An interpreted solution *of e in C is a V-algebra structure $s : VC \to C$ such that*

$$s \cdot Vc^{-1} = c^{-1} \cdot H\widehat{[b_0, s]} \cdot e_C : VHC \to C.$$

Theorem 5.4. *For every λ-rps there exists a unique interpreted solution s in C. In addition, s extends the cia structure on C, i. e., the following is the structure of a cia for $H\widehat{K+V}$ on C:*

$$H\widehat{K+V}(C) \xrightarrow{H\widehat{[b_0,s]}} HC \xrightarrow{c^{-1}} C\,. \tag{5.1}$$

Notice again that the fact that the unique solution s of a λ-rps extends the cia structure on C means that the operations on C defined in this way may be part of recursive definitions according to the Corollaries 4.6, 4.7 (where $M = \widehat{K+V}$) and also Theorem 5.4 (where K is now replaced by $K+V$). We will make use of this feature in our applications below.

Theorem 5.5. *Let $e_i : V_iH \to H\widehat{K+V_i}$, $i = 1, 2$, be two λ-rps's. Then the cia structure on C extended by the unique solutions $s_i : V_iC \to C$ of the e_i is independent of the order of extension.*

More precisely, we may first take $s_1 : V_1C \to C$ to obtain an extended cia structure as in (5.1), and then take the solution of s_2 in the new cia, or vice versa. Either way, the resulting extended cia structure is $c^{-1} \cdot H\widehat{[b_0, s_1, s_2]} : H(\widehat{K+V_1+V_2})(C) \to C$.

6 Applications

6.1 Process Algebras

Recall Example 2.6(3) where $HX = \mathcal{P}_\kappa(A \times X)$. We shall first explain more in detail how the natural transformation ℓ is obtained. Recall that K is the polynomial functor corresponding to the types of the CCS combinators, i. e., KX is a coproduct of the following components: $A \times X$ for agent expressions $a.x$, $\coprod_{n<\kappa} X^n$ for agent expressions $\sum_{i=1}^n x_i$, $X \times X$ for $x_1|x_2$, $\coprod_f X$, where f ranges over functions on the action set $A \setminus \{\tau\}$ with $\overline{f(a)} = f(\bar{a})$, for renaming $x[f]$, and $\coprod_{L\subseteq A\setminus\{\tau\}} X$, for restriction $x\backslash L$. The natural transformation $\ell : KH \to H\widehat{K}$ is given by the sos rules in 2.6(3) in terms of the components of the coproduct KH, i. e., for each combinator separately. The first component $\ell_X(a, S)$, for $S \subseteq A \times X$, is $(a, \sum_{(a_i,x_i)\in S}(a_i.x_i))$, the second one is for every $n < \kappa$ and every family $(S_i)_{i<n}$ with $S_i \subseteq A \times X$ given by $\ell_X((S_i)_{i<\kappa}) = \bigcup_{i<\kappa} S_i$. The third one $\ell_X(S_1, S_2)$, where $S_1, S_2 \in HX$, is the union of three sets: (i) all $(a, x|(\sum_{(a_i,x_i)\in S_2}(a_i.x_i)))$, where (a, x) ranges through S_1, (ii) all $(a, (\sum_{(a_i,x_i)\in S_1}(a_i.x_i))|x)$, where (a, x) ranges through S_2, and (iii) all $(\tau, x|y)$ where for some $a \in A \setminus \{\tau\}$ we have $(a, x) \in S_1$ and $(\bar{a}, y) \in S_2$. The remaining two components are $\ell_X(S) = \{(f(a), x[f]) \mid (a, x) \in S\}$ (here we mean $f(\tau) = \tau$, of course) and $\ell_X(S) = \{(a, x\backslash L) \mid (a, x) \in S, a, \bar{a} \notin L\}$. The form of these definitions is very similar to the ones given by Aczel [2] in the setting of non-well-founded set theory. We already mentioned the ℓ-interpretation $b : \widehat{K}C \to C$ giving the desired operations on agents, and this gives the two new cia structures for $H\widehat{K}$ and $\widehat{K}H\widehat{K}$ as in Theorems 3.2 and 3.3.

Remark 6.1. If we replaced the second component $\coprod_{n<\kappa} X^n$ of K by $\mathcal{P}_\kappa X$ we still have a distributive law. Furthermore, in both cases the induced (binary) operation of summation is automatically commutative, associative and idempotent: these three laws that have to be proved in process theory come "for free" by encoding them in the distributive law using the union operation.

Now let us recall Milner's solution theorem for CCS agents from [21]. Suppose that E_i, $i \in I$, are agent expressions with the free variables x_i, $i \in I$. Suppose further that each variable x_j in each E_i, $i, j \in I$ is *weakly guarded*, i. e., it only occurs within the scope of some prefix combinator $a.-$. Then there is a unique solution of the recursive system $x_i = E_i$ of equations. More precisely, let $\sim$ denote strong bisimilarity, and let $E_i[\boldsymbol{P}/\boldsymbol{x}]$ denote simultaneous substitution of P_j for x_j for every j. Then we have

Theorem 6.2. [21] *There exist unique CCS agents P_i such that $P_i \sim E_i[\boldsymbol{P}/\boldsymbol{x}]$ holds for each $i \in I$.*

It is easy to see that this theorem is a consequence of our Theorem 3.3; to give a system $x_i = E_i$ where each variable is weakly guarded is the same as to give a map $X \to \widehat{K}H\widehat{K}X$, where $X = \{x_i \mid i \in I\}$. This map can be extended to a flat equation morphism $X \to \widehat{K}H\widehat{K}X + C$, which has a unique solution in C. Actually, the extra summand C allows us to use constant agents in recursive specifications. So, for example, we can obtain the agent P as the unique solution of $x = a.(x|c) + b$ in the introduction and then use it in a system like $x = b.(x + y), y = P$ which has a unique solution by Theorem 3.3.

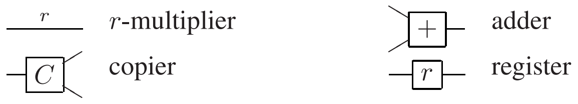
Finally, suppose we want to define the new combinators op_1 and op_2 by the rule

$$\frac{E \xrightarrow{a} F}{\mathsf{op}_1(E) \xrightarrow{a} F|\mathsf{op}_2(F+E) \quad \mathsf{op}_2(E) \xrightarrow{a} F + \mathsf{op}_1(F|E)}.$$

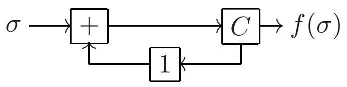
Then Theorem 5.4 tells us that this rule uniquely determines the two combinators. Indeed, we translate the rule into a λ-rps: let $V = \mathrm{Id} + \mathrm{Id}$ (two unary combinators are defined) and let $e : VH \to H\widehat{K+V}$ be given by $e(S) = \{(a, x|\mathsf{op}_2(x + \sum_{(a,x)\in S} a.x)) \mid (a,x) \in S\}$ on the first component and $e(S) = \{(a, x + \mathsf{op}_1(x|\sum_{(a,x)\in S} a.x)) \mid (a,x) \in S\}$ on the second one. The unique solution of e gives us two new unary combinators on C extending its cia structure. This means that Theorem 6.2 remains true for the extended calculus, without further work.

6.2 Streams

Recall from Example 2.6(2) that here we take $HX = \mathbb{R} \times X$ and we have $C = \mathbb{R}^\omega$ with the structure given by $\langle \mathsf{hd}, \mathsf{tl} \rangle : C \to \mathbb{R} \times C$. Recursive function definitions in this realm are often given in terms of stream circuits, and we show how this arises as a special case of our results. Stream circuits are usually defined as pictorial compositions of the following basic stream circuits



The r-multiplier multiplies all elements in a stream by $r \in \mathbb{R}$, the adder performs componentwise addition, the copier yields two copies of a stream, and the register prepends $r \in \mathbb{R}$ to a stream σ to yield $r.\sigma$. The stream circuits are then built from the basic circuits by plugging wires together, and there may also be feedback (loops). For example the following picture shows a simple stream circuit:



(6.1)

Now let K be the signature functor associated to the signature Σ given by r-multiplication, the adder and the register operations (copying will be implicit via variable sharing). In symbols, $KX = \mathbb{R} \times X + X \times X + \mathbb{R} \times X$. These operations are defined by so-called *behavioral differential equations* [24] with $\sigma_0 = \mathsf{hd}(\sigma)$ and $\sigma' = \mathsf{tl}(\sigma)$:

$$\begin{aligned} (r\sigma)_0 &= r\sigma_0 & (r\sigma)' &= r\sigma' \\ (\sigma+\tau)_0 &= \sigma_0 + \tau_0 & (\sigma+\tau)' &= \sigma' + \tau' \\ (r.\sigma)_0 &= r & (r.\sigma)' &= \sigma \end{aligned}$$

These definitions are easily seen to give rise to a natural transformation $\ell : KH \to H\widehat{K}$, for example the middle component $(\mathbb{R} \times X)^2 \to \mathbb{R} \times \widehat{K}X$ is given by $\ell_X((r,x),(s,y)) = (r+s, x+y)$ where $r+s \in \mathbb{R}$ and $x+y$ is a Σ-term. We then get the ℓ-interpretation in C and the corresponding extended cia structures by Theorems 3.2 and 3.3. Recall that a stream circuit is called *valid* if every loop passes through at least one register. It is well-known that every *closed* valid stream circuit defines a unique stream at every output wire, see [24]. Our Theorem 5.4 now also implies that open circuits define unique stream functions. To our knowledge, this is a new result in coalgebraic stream calculus.

Theorem 6.3. *Every finite valid stream circuit defines a unique stream function.*

Moreover, the fact that the unique solution of a λ-rps extends the cia structure on C explains why stream circuits can be used as building blocks as if they were basic operations in subsequent stream circuits. And Theorem 6.3 remains valid for the extended circuits.

The proof of 6.3 essentially gives a translation of an arbitrary valid stream circuit into a λ-rps. Instead of giving the full proof here, we demonstrate this on the circuit given in (6.1) above. First we introduce for the output a function symbol f and for the register output the function symbol g. To determine their arity we count the number of input wires which have a (directed) path to the register and the output, respectively. In both cases the arity is one. Now we must give a definition of $f(r.x)$ and $g(r.x)$ for an abstract input stream with head $r \in \mathbb{R}$. These definitions are each given by a pair (s,t)

where $s \in \mathbb{R}$ and t is a term in the one variable x over operations corresponding to the basic circuits and f, g. We define

$$g(r.x) = (1, r.x + g(r.x)) \qquad f(r.x) = (r + 1, x + (r.x + g(r.x))\,.$$

For $g(r.x)$ we take the value 1 of the register as first component, and the right-hand term is obtained as follows: we follow all paths from the register backwards until we find an input or a register. Since the given circuit is valid, all such paths are finite and there are only finitely many of them. So we get a finite tree or, equivalently, the desired term. For $f(r.x)$ we first follow all paths to inputs and registers backwards to get the term $t' = x_I + x_R$, where x_I represents the input and x_R the register. For the first component of $f(r.x)$ we evaluate t' with the head r of the input and the initial value 1 of the register, and for the second component we replace in t' the input by x and the register by the second component of its function $g(r.x)$. The two equations above are easily seen to yield a λ-rps $e : VH \to H\widehat{K + V}$, where $V = \mathrm{Id} + \mathrm{Id}$ is the polynomial functor for the signature with two unary symbols f and g. The unique solution of e gives two unary operations (for f and g) on C, and the one for f is precisely the function computed by the circuit (6.1). Since these new unary operations on C extend the cia structure, we can use f (and also g) as "black-boxes" in subsequent recursive definitions or stream circuits.

6.3 Non-Well-Founded Sets

For background on non-well-founded sets, the antifoundation axiom (AFA), and classes, please see the books [2, 10]. We work here on the category $\mathcal{C}$ of classes. Observe first that even though we are working in a different category than Set, the results of Section 5 hold true for $\mathcal{C}$. This is because the construction of free algebras for a functor as colimits of transfinite chains works in $\mathcal{C}$, see [6].

Consider $\mathcal{P} : \mathcal{C} \to \mathcal{C}$ taking a class X to the class $\mathcal{P}X$ of sub*sets* of X. AFA is equivalent to the assertion that (V, c) is a final coalgebra, where V is the class of all sets, and $c : V \to \mathcal{P}V$ takes a set and considers it a set of sets. (That is, $c(s) = s$ for all s.) Let us note some natural transformations:

$$\begin{array}{lll} p : \mathcal{P} \to \mathcal{P}\mathcal{P} & op : \mathrm{Id} \times \mathrm{Id} \to \mathcal{P}\mathcal{P} & cp : \mathcal{P} \times \mathcal{P} \to \mathcal{P}(\mathrm{Id} \times \mathrm{Id}) \\ p_X(x) = \mathcal{P}(x) & op_X(x, y) = \{\{x\}, \{x, y\}\} & cp_X(x, y) = x \times y \end{array}$$

Also note that c^{-1} is the operation on V taking a family $x \subseteq V$ of sets to the set $\{\, y \mid y \in x \,\}$.

We will now define three additional operations on V: the powerset operation $b_1 : x \mapsto \{\, y \mid y \subseteq x \,\}$, the Kuratowski pair $b_2 : (x, y) \mapsto \{\{\, x \,\}, \{\, x, y \}\}$ and the cartesian product $b_3 : (x, y) \mapsto x \times y$. So let K be the functor $\mathrm{Id} + (\mathrm{Id} \times \mathrm{Id}) + (\mathrm{Id} \times \mathrm{Id}) + \mathcal{P} + \mathcal{P}^2$; its first three components represent (the type of) our three desired operations, the fourth component $\mathcal{P}$ represents c^{-1} and the fifth one represents $c^{-1} \cdot \mathcal{P}c^{-1}$—the latter two are needed for the definition of the former three. We write the coproduct injections of K as $\mathsf{inj}_1, \ldots, \mathsf{inj}_5$. We define a natural transformation $\ell : K\mathcal{P} \to \mathcal{P}K$ componentwise, using $\mathcal{P}\mathsf{inj}_4 \cdot p : \mathcal{P} \to \mathcal{P}K$, $\mathcal{P}\mathsf{inj}_5 \cdot op\mathcal{P} : \mathcal{P} \times \mathcal{P} \to \mathcal{P}K$, $\mathcal{P}\mathsf{inj}_2 \cdot cp : \mathcal{P} \times \mathcal{P} \to \mathcal{P}K$, $\mathcal{P}\mathsf{inj}_4 : \mathcal{P}\mathcal{P} \to \mathcal{P}K$ and $\mathcal{P}\mathsf{inj}_5 : \mathcal{P}\mathcal{P}\mathcal{P} \to \mathcal{P}K$. Then ℓ lifts to a distributive law $\lambda = \mathsf{can} \cdot (\ell + \mathrm{id}_{\mathcal{P}})$ of the free pointed endofunctor $M = K + \mathrm{Id}$ over $\mathcal{P}$. Let

$b : MV \to V$ be the λ-interpretation in V. Let us write $b_1, \dots, b_5$ for the components of b corresponding to the left-hand component K of M, so $b_i = b \cdot \mathsf{inl} \cdot (\mathsf{inj}_i)_V$. To obtain explicit formulas for these, we use equation (2.1) and the above definitions to write:

$$\begin{aligned} c \cdot b_1 &= \mathcal{P}b_4 \cdot p_V \cdot c & c \cdot b_4 &= \mathcal{P}b_4 \cdot \mathcal{P}c \\ c \cdot b_2 &= \mathcal{P}b_5 \cdot op_{\mathcal{P}V} \cdot (c \times c) & c \cdot b_5 &= \mathcal{P}b_5 \cdot \mathcal{P}^2 c \\ c \cdot b_3 &= \mathcal{P}b_2 \cdot cp_V \cdot (c \times c) \end{aligned}$$

We check easily that $b_4 = c^{-1}$ and $b_5 = c^{-1} \cdot \mathcal{P}c^{-1}$ satisfy the last two equations. From these we see that $b_1 = c^{-1} \cdot \mathcal{P}c^{-1} \cdot p_V \cdot c$, $b_2 = c^{-1} \cdot \mathcal{P}b_5 \cdot op_{\mathcal{P}V} \cdot (c \times c)$, and $b_3 = c^{-1} \cdot \mathcal{P}b_2 \cdot cp_V \cdot (c \times c)$. In words, b_4 and b_5 are the identity, and b_1, b_2 and b_3 are as desired.

By Theorem 3.2, we have a cia structure $(V, c^{-1} \cdot \mathcal{P}b)$ for the composite $\mathcal{P}M$.

Remark 6.4. We could have obtained the various operations on V in a step by step fashion starting with b_4 and b_5 and then defining b_1, b_2, b_3 by successive applications of Theorem 5.4. We decided against this, to keep the presentation short. But in the next section on formal languages we follow this approach.

Continuing our discussion of non-well-founded sets, we may solve systems of equations which go beyond what one finds in the standard literature on non-well-founded sets [2, 10]. For example, one may solve the system $x = \{\mathcal{P}(y)\}$, $y = \{y \times y, \emptyset\}$. Further, one may uniquely solve recursive function definitions such as $g(x) = \{g(\mathcal{P}(x)) \times x, x\}$ from the introduction. Indeed, for $W = \mathrm{Id}$ this equation yields a λ-rps $e : W\mathcal{P} \to \mathcal{P}\widehat{K + W}$ whose unique solution given by Theorem 5.4 is a function $g_V : V \to V$ behaving as specified.

6.4 Formal Languages

Recall Example 2.6(1); here we have $HX = X^A \times 2$ on Set. A coalgebra $x : X \to X^A \times 2$ for H is precisely a deterministic automaton with the (possibly infinite) state set X. Here $C = \mathcal{P}(A^*)$, and the unique homomorphism $h : (X, x) \to (C, c)$ assigns to each state the language it accepts. We shall now show how various operations on formal languages can be defined in a compositional way using Theorem 5.4. It is well-known that such operations can be defined as interpretations of one distributive law in C, see e. g. [12]. However, the previous bialgebraic account does not explain why one may define these operations in a step-by-step fashion by subsequent recursive definitions. This is the added value of Theorem 5.4.

We start with the functor $K_0 = C_\emptyset$ (that means, we start from scratch with no given operations) and with $\ell_0 : C_\emptyset H \to H\widehat{C_\emptyset} = H$ given by the empty maps. So the corresponding distributive law λ_0 is the identity on H, and its interpretation is the identity on C. Thus, the cia structure for $H\widehat{K_0}$ on C given by Theorem 3.2 is simply the initial cia (C, c^{-1}) for H. At each subsequent step we are given a functor K_i with $\ell_i : K_i H \to H\widehat{K_i}$ with its interpretation $b_i : \widehat{K_i}C \to C$. We then give a λ_i-rps $e_i : V_i H \to H\widehat{K_i + V_i}$ and its unique solution $s_i : V_i C \to C$ extends the cia structure as follows: let $K_{i+1} = K_i + V_i$ and let $\ell_{i+1} = [H\widehat{\mathsf{inl}} \cdot \ell_i, e_i] : K_{i+1}H \to H\widehat{K_{i+1}}$, where $\widehat{\mathsf{inl}} : \widehat{K_i} \to \widehat{K_{i+1}}$ is the monad morphism induced by $\mathsf{inl} : K_i \to K_{i+1}$. By induction it is easy to see that the ℓ_{i+1}-interpretation is $b_{i+1} = \widehat{[s_j]_{j=0,\dots,i}} : \widehat{K_{i+1}}C \to C$. And this gives an extended cia $c^{-1} \cdot Hb_{i+1} : H\widehat{K_{i+1}}(C) \to C$ by Theorem 3.2.

As a first step we define constants in C for $\emptyset$, $\{\varepsilon\}$, and $\{a\}$ for each $a \in A$, and the operation $I = c^{-1} : C^A \times 2 \to C$ as solutions of a λ_0-rps. (Note that $I((L_a)_{a\in A}, j)$ is the language $L = \bigcup_{a\in A}\{a\}L_a$ if $j = 0$ and $L \cup \{\varepsilon\}$ otherwise.) We express this as a λ_0-rps as follows: take the functor $V_0X = 1 + 1 + A + X^A \times 2$ expressing the above constants and the operation I. We define $e_0 : V_0H \to H\widehat{K_0 + V_0} = H\widehat{V_0}$ componentwise. We write for every set X, $\emptyset$ for $\mathsf{inj}_1(*) \in V_0X$, ε for $\mathsf{inj}_2(*) \in V_0X$. Then $e_0(\emptyset) = ((\emptyset)_{a\in A}, 0)$, $e_0(\varepsilon) = ((\emptyset)_{a\in A}, 1)$, $e_0(a) = ((t_b)_{b\in A}, 0)$ with $t_b = \varepsilon$ for $b = a$ and $t_b = \emptyset$ otherwise, and finally, $e_0((p_a)_{a\in A}, j) = ((Ip_a)_{a\in A}, j)$ where each $p_a \in X^A \times 2$. It is now straightforward to check that the unique solution s_0 of e_0 yields the desired operations on C extending the cia structure.

Next we add the operations of union, intersection and language complement to the cia structure. Let $K_1 = K_0 + V_0$ and let ℓ_1 as above with interpretation $b_1 = \widehat{s}_0$. Let $V_1X = X \times X + X \times X + X$ be the polynomial functor corresponding to two binary symbols $\cup$ and $\cap$ and one unary one $\overline{(-)}$. We give the λ_1-rps $e_1 : V_1H \to H\widehat{K_1 + V_1}$ componentwise in the form of the three assignments (where a ranges over A):

$$((x_a), j) \cup ((y_a), k) \mapsto ((x_a \cup y_a), j \vee k) \qquad \overline{((x_a), j)} \mapsto ((\overline{x_a}, \neg j)$$
$$((x_a), j) \cap ((y_a), k) \mapsto ((x_a \cap y_a), j \wedge k)$$

where $\vee$, $\wedge$ and $\neg$ are the evident operations on $2 = \{0, 1\}$. The corresponding unique solution $s_1 : V_1C \to C$ is easily checked to provide the desired operations extending the cia structure on C.

The next step adds concatenation to the cia structure on C. For this let $V_2X = X \times X$ and e_2 is given by the assignment $((x_a), j) \cdot ((y_a), k) \mapsto ((t_a), j \wedge k)$ where $t_a = (x_a \cdot I((y_a), k)) \cup y_a$ if $j = 1$ and $t_a = x_a \cdot I((y_a), k)$ otherwise. Its unique solution $s_2 : C \times C \to C$ is the concatenation operation.

As the final step we add the Kleene star operation by taking $V_3X = X$ and e_3 given by $e_3((x_a), j) = ((x_a \cdot (I((x_a), j)^*), 1)$. Notice that this definition makes use of concatenation which was a solution at the previous stage and concatenation makes use of union which was a solution at stage 1.

7 Conclusions

In many areas of theoretical computer science, one is interested in recursive definitions of functions on final coalgebras C for various functors H. This paper provides a more comprehensive foundation for recursive definitions than had been presented up until now. The overall idea is to present operations in terms of a distributive law λ of a pointed endofunctor M over H. We proved that λ induces new completely iterative algebra structures for HM and MHM on C. As a result, we are able to define operations with useful algebraic properties such as commutativity or associativity "for free". We also introduced the notion of a λ-rps and showed how to uniquely solve recursive function definitions in C which are given by a λ-rps. Our results explain why taking unique solutions of such equations is a compositional process. And we have seen that our results can be applied to provide the semantics of recursive specifications in a number of different areas of theoretical computer science.

References

[1] Aceto, L., Fokking, W., Verhoef, C.: Structural Operational Semantics. In: Handbook of Process Algebra. Elsevier, Amsterdam (2001)

[2] Aczel, P.: Non-Well-Founded Sets. CLSI Lecture Notes, vol. 14. CLSI Publications, Stanford (1988)

[3] Aczel, P., Adámek, J., Milius, S., Velebil, J.: Infinite trees and completely iterative theories: A coalgebraic view. Theoret. Comput. Sci. 300, 1–45 (2003)

[4] Adámek, J.: Free algebras and automata realizations in the language of categories. Comment. Math. Univ. Carolin. 15, 589–602 (1974)

[5] Adámek, J.: Introduction to coalgebra. Theory Appl. Categ. 14, 157–199 (2005)

[6] Adámek, J., Milius, S., Velebil, J.: On coalgebras based on classes. Theoret. Comput. Sci. 316, 3–23 (2004)

[7] Adámek, J., Milius, S., Velebil, J.: Elgot algebras. Log. Methods Comput. Sci. 2(5:4), 31 (2006)

[8] Bartels, F.: Generalized coinduction. Math. Structures Comput. Sci. 13(2), 321–348 (2003)

[9] Bartels, F.: On generalized coinduction and probabilistic specification formats. PhD thesis, Vrije Universiteit Amsterdam (2004)

[10] Barwise, J., Moss, L.S.: Vicious circles. CLSI Publications, Stanford (1996)

[11] Capretta, V., Uustalu, T., Vene, V.: Recursive coalgebras from comonads. Inform. and Comput. 204, 437–468 (2006)

[12] Jacobs, B.: A bialgebraic review of deterministic automata, regular expressions and languages. In: Futatsugi, K., Jouannaud, J.-P., Meseguer, J. (eds.) Algebra, Meaning, and Computation. LNCS, vol. 4060, pp. 375–404. Springer, Heidelberg (2006)

[13] Jacobs, B.: Distributive laws for the coinductive solution of recursive equations. Inform. and Comput. 204(4), 561–587 (2006)

[14] Lambek, J.: A fixpoint theorem for complete categories. Math. Z. 103, 151–161 (1968)

[15] Lenisa, M., Power, A.J., Watanabe, H.: Distributivity for endofunctors, pointed and copointed endofunctors, monads and comonads. In: Reichel, H. (ed.) Proc. Coalgebraic Methods in Computer Science. Electron. Notes Theor. Comput. Sci., vol. 33. Elsevier, Amsterdam (2000)

[16] Lenisa, M., Power, A.J., Watanabe, H.: Category theory for operational semantics. Theoret. Comput. Sci. 327, 135–154 (2004)

[17] MacLane, S.: Categories for the working mathematician, 2nd edn. Springer, Heidelberg (1998)

[18] Milius, S.: Completely iterative algebras and completely iterative monads. Inform. and Comput. 196, 1–41 (2005)

[19] Milius, S., Moss, L.S.: The category theoretic solution of recursive program schemes. Theoret. Comput. Sci. 366, 3–59 (2006) (fundamental study)

[20] Milius, S., Moss, L.S.: Equational properties of recursive program scheme solutions. Cah. Topol. Gèom. Diffèr. Catèg. 50, 23–66 (2009)

[21] Milner, R.: Communication and Concurrency. International Series in Computer Science. Prentice Hall, Englewood Cliffs (1989)

[22] Plotkin, G.D., Turi, D.: Towards a mathematical operational semantics. In: Proc. Logic in Computer Science, LICS (1997)

[23] Rutten, J.J.M.M.: Universal coalgebra: a theory of systems. Theoret. Comput. Sci. 249(1), 3–80 (2000)
[24] Rutten, J.J.M.M.: A coinductive calculus of streams. Math. Structures Comput. Sci. 15(1), 93–147 (2005)
[25] Schwencke, D.: Coequational logic for accessible functors. Accepted for publication in Inform. and Comput. (2009)
[26] Uustalu, T., Vene, V., Pardo, A.: Recursion schemes from comonads. Nordic J. Comput. 8(3), 366–390 (2001)
[27] Worrell, J.: On the final sequence of a finitary set functor. Theoret. Comput. Sci. 338, 184–199 (2005)

Coalgebraic Correspondence Theory

Lutz Schröder[1,⋆] and Dirk Pattinson[2,⋆⋆]

[1] DFKI Bremen and Department of Computer Science, Universität Bremen
[2] Department of Computing, Imperial College London

Abstract. We lay the foundations of a first-order correspondence theory for coalgebraic logics that makes the transition structure explicit in the first-order modelling. In particular, we prove a coalgebraic version of the van Benthem/Rosen theorem stating that both over arbitrary structures and over finite structures, coalgebraic modal logic is precisely the bisimulation invariant fragment of first-order logic.

Introduction

Viewing modal logic as a sub-language of first-order logic via the standard translation is the starting point of modal correspondence theory. One of the core results of this area, van Benthem's theorem [22], states that modal logic is precisely the bisimulation invariant fragment of first-order logic over relational structures. This result has been extended to finite structures by Rosen [18], and special frame classes have been considered in [5]. Results of this kind characterize the expressive power of modal logic – slightly reworded, they state that modal logic can express the same bisimulation-invariant properties as first-order logic.

Here, we extend these results to coalgebraic modal logic [16], thus making initial forays into *coalgebraic correspondence theory*. Coalgebraic modal logic is a generic framework for modal logics that captures a wide range of modal logics from the literature, e.g. the modal logic of neighbourhood frames (called classical modal logic in [3]), normal modal logics [2], graded and probabilistic modal logics [7,14,12], and various conditional logics [3]. The parameters of the framework are a *type functor*, whose coalgebras serve as models, and a choice of *predicate liftings* defining the modal operators. The predicate liftings act as an interface to the type functor, and as such form an integral part of the semantics.

Our correspondence language is a multi-sorted first-order logic, inspired by the correspondence language for neighbourhood frames of [11]. It includes a dedicated sort to represent the type functor and thus provides a full model of coalgebras. Moreover, the language explicitly incorporates predicate liftings, following the semantic principles outlined above. We adapt the method of Rosen (and a related proof by Otto [15]) to prove that, under suitable assumptions, coalgebraic modal logic is, both over finite and over arbitrary structures, precisely the fragment of the coalgebraic correspondence language characterized

⋆ Work performed as part of the DFG project *Generic Algorithms and Complexity Bounds in Coalgebraic Modal Logic* (SCHR 1118/5-1).

⋆⋆ Partially supported by EPSRC grant EP/F031173/1.

by *invariance under behavioural equivalence*. As Rosen's method avoids compactness and saturation, which feature prominently in the original proof of van Benthem's theorem, we can deal also with classes of coalgebras that fail to be first-order axiomatizable, which is a fairly typical phenomenon.

To show that a first-order formula that is invariant under behavioural equivalence can be characterized by a *finitary* formula, we have to assume that the underling signature functor preserves finite sets. This covers Kripke and neighbourhood semantics, as well as the selection function semantics of conditional logic and a bounded version of graded modal logic, but excludes e.g. graded and probabilistic modal logic. For the general case, we do provide a characterization result in terms of bounded-rank modal formulas with infinitary conjunction, and a counter-example showing that equivalence to a finitary modal formula fails in general. This result applies to essentially all logics of interest, and indeed covers most of the way to the finitary result in terms of its proof, as the latter is an easy corollary to it in the case of finite signatures. As an application, we obtain e.g. that every formula in a natural first order logic with counting quantifiers over multigraphs that is invariant under behavioural equivalence over finite structures is equivalent to a possibly infinitary formula *of bounded depth* in graded modal logic. Although similar results have previously been obtained over the class of *all* structures [6], our result seems to be the first Rosen-type result for graded modal logic over *finite* structures.

We note that the design of the correspondence language used as the setting for our results is a delicate affair: the translation of coalgebraic semantics, like already the translation of neighbourhood semantics used in [11], needs to include a sort of neighbourhoods, i.e. subsets of the state space. On the other hand, we need to avoid the full expressive power of monadic second order logic, in which the van Benthem/Rosen theorem fails to hold, as it contains the μ-calculus (which, in the standard relational case, is in fact its bisimulation-invariant fragment [13]). This forces us to adopt a relaxed interpretation of the neighbourhood sort by suitable subsets of the full powerset. In our setup, the key to a suitable notion of model in this sense is the inclusion of explicit distinguished supports in the language; a particularly pleasant effect of this language extension is that it simultaneously acts as the key to enabling the use of Gaifman locality.

1 Coalgebra and Modal Logic

Throughout the paper, we fix a modal similarity type Λ consisting of modal operators with associated arities. As we will be considering models rather than frames, we express propositional variables as nullary modal operators. The set $\mathcal{F}(\Lambda)$ of Λ-*formulas* is then given by the grammar

$$\mathcal{F}(\Lambda) \ni \phi, \psi ::= \bot \mid \phi \wedge \psi \mid \neg\phi \mid \heartsuit(\phi_1, \dots, \phi_n)$$

where $\heartsuit \in \Lambda$ is n-ary. We denote the language that admits arbitrary conjunctions of sets of formulas rather than just binary conjunctions by $\mathcal{F}_\infty(\Lambda)$. We write $\mathrm{rank}(\phi)$ for the maximal nesting depth of modal operators in the formula ϕ, defined formally as $\mathrm{rank}(\bot) = 0$, $\mathrm{rank}(\bigwedge \Phi) = \sup_{\phi \in \Phi} \mathrm{rank}(\phi)$, $\mathrm{rank}(\neg\phi) = \mathrm{rank}(\phi)$ and $\mathrm{rank}(\heartsuit(\phi_1, \dots, \phi_n)) = 1 + \max\{\mathrm{rank}(\phi_1), \dots, \mathrm{rank}(\phi_n)\}$. Thus, the rank of a formula in $\mathcal{F}_\infty(\Lambda)$ may be infinite.

Formulas over Λ are interpreted over coalgebras with respect to a Λ-*structure* that consists of an endofunctor $T : \mathsf{Set} \to \mathsf{Set}$ on the category of sets, together with an assignment

$$[\![\heartsuit]\!] : \mathcal{Q}^n \to \mathcal{Q} \circ T$$

of natural transformations (the *predicate liftings*) where $\mathcal{Q} : \mathsf{Set}^{op} \to \mathsf{Set}$ is the contravariant powerset functor. The functor T is called the *underlying functor* of the structure, and we usually refer to the structure just in terms of its underlying functor, leaving the assignments of predicate liftings implicit.

Assumption 1. We assume w.l.o.g. that T preserves injective maps [1]. For ease of notation, we will in fact sometimes assume that subset inclusions $X \hookrightarrow Y$ are mapped to subset inclusions $TX \hookrightarrow TY$. Moreover, we assume w.l.o.g. that T is non-trivial, i.e. $TX = \emptyset \implies X = \emptyset$ (otherwise, $TX = \emptyset$ for all X).

Using the above assumption, we can give a simple definition of *support*, which will play a role in our correspondence language:

Definition 2. A set $A \subseteq X$ is a *support* of $t \in TX$ if $t \in TA$.

Support has played a role in various coalgebraic model constructions, see e.g. [21]. We keep the notion of support as broad as possible, and in particular do not insist on minimality, as set of supports of $t \in TX$ does not necessarily have a smallest element with respect to subset inclusion [9].

Given a Λ-structure T, a T-*coalgebra* is a pair (C, γ) where C is a set (of states) and $\gamma : C \to TC$ is a (transition) function. We identify T-coalgebras (C, γ) with their carrier set C in case the transition function is clear from the context. The *semantics* $[\![\phi]\!]_C \subseteq C$ of a Λ-formula ϕ with respect to a T-coalgebra (C, γ) is given inductively by

$$[\![\heartsuit(\phi_1, \dots, \phi_n)]\!]_C = \gamma^{-1} \circ [\![\heartsuit]\!]_C([\![\phi_1]\!]_C, \dots, [\![\phi_n]\!]_C)$$

where $\heartsuit \in \Lambda$ is n-ary, together with the usual clauses for the propositional connectives. We write $(C, c) \models \phi$ if $c \in [\![\phi]\!]_C$.

Example 3. The following logics are covered by the coalgebraic approach.

1. Kripke models over a set P of propositional variables are triples (W, R, σ) where W is a set, $R \subseteq W \times W$ is a binary relation, and $\sigma : \mathsf{P} \to \mathcal{P}(W)$ is a valuation of propositional variables. It is easy to see that Kripke models are in 1-1 correspondence with T-coalgebras for $TX = \mathcal{P}(X) \times \mathcal{P}(\mathsf{P})$. The syntax of the modal logic K comes about via the similarity type $\Lambda = \{\Diamond\} \cup \mathsf{P}$ where $\Diamond$ is unary and each $p \in \mathsf{P}$ doubles as a nullary modality. The language $\mathcal{F}(\Lambda)$ is interpreted over T-coalgebras by virtue of the structure

$$[\![\Diamond]\!]_X(A) = \{(B, C) \in TX \mid B \cap A \neq \emptyset\} \qquad [\![p]\!]_X = \{(B, C) \in TX \mid p \in C\}.$$

Clearly this semantics coincides with the standard textbook semantics of K [2].

2. The modal logic of neighbourhood frames (classical modal logic in [3]) arises via the same similarity type, but is interpreted over neighbourhood models, i.e. coalgebras for the functor $TX = \mathcal{Q}(\mathcal{Q}(X)) \times \mathcal{P}(\mathsf{P})$ where again P is a set of propositional variables and $\mathcal{Q}$ denotes contravariant powerset. For a T-coalgebra (C, γ), we say that $A \subseteq C$ *is a neighbourhood of* $c \in C$ if $\gamma(c) = (N, B)$ where $A \in N$. The interpretation of propositional constants (nullary modalities) is as above and the semantics of classical modal logic arises via the lifting

$$[\![\Box]\!]_X(A) = \{(N, B) \in TX \mid A \in N\}$$

which again gives rise to the standard semantics.

3. Monotone modal logic has the same syntax as classical modal logic, but is interpreted over monotone neighbourhood models, i.e. coalgebras for the functor

$$TX = \{A \in \mathcal{P}\mathcal{P}(X) \mid A \text{ upwards closed}\} \times \mathcal{P}(\mathsf{P})$$

where upwards closure refers to subset inclusion.

4. Conditional logic [3] has a binary modal operator $\Rightarrow$ that we write in infix notation. Conditional models over a set P of propositional variables come about as coalgebras for the functor

$$TX = \{f : \mathcal{P}(X) \to \mathcal{P}(X) \mid \ f \text{ a function}\} \times \mathcal{P}(\mathsf{P})$$

(again, the powerset on the left of the function space is contravariant) where propositional constants are interpreted as above and the lifting

$$[\![\Rightarrow]\!]_X(A, B) = \{(f, D) \in TX \mid f(A) \subseteq B\}$$

induces the standard semantics of conditional logic.

5. The similarity type of graded modal logic features, apart from propositional constants, an indexed collection of operators $\Diamond_k$ for $k \in \omega$. The intuitive reading of $\Diamond_k \phi$ is that ϕ holds in more than k successors. To retain naturality of predicate liftings, we slightly deviate from the traditional semantics [7] and interpret graded modal logic over coalgebras of the functor T (left) where we use the liftings $[\![\Diamond_k]\!]$ (right)

$$TX = \{f : X \to \omega\} \times \mathcal{P}(\mathsf{P}) \qquad [\![\Diamond_k]\!]_X(A) = \{(f, D) \in TX \mid \textstyle\sum_{x \in A} f(x) > k\}$$

to interpret modal operators. In other words, we interpret graded modal logic over multigraphs [4] where the graded modalities refer to the weighted sum of successors. This semantics is equivalent to the standard Kripke semantics w.r.t. satisfiability of formulas, as multigraphs can be converted to Kripke frames by inserting the appropriate number of copies for each successor [19].
A variation of graded modal logic arises by limiting the overall (weighted) sum of successor states. If we consider the sub-functor

$$T_k X = \{f \in TX \mid \textstyle\sum_{x \in X} f(x) \leq k\} \times \mathcal{P}(\mathsf{P})$$

for some $k \geq 0$, we may describe k-bounded multigraphs as T-coalgebras, and interpret the sub-language that only features the modalities $\Diamond_i$ for $i < k$.

6. For probabilistic logics, we prefer to work with subprobabilities for technical reasons, where a subprobability distribution P on a set X is a discrete measure on X with $P(X) \leq 1$. The similarity type of the *modal logic of subprobabilities*, a variant of probabilistic modal logic [12], contains, apart from propositional variables, the modal operators M_p for rational $p \in [0,1] \cap \mathbb{Q}$. This language is interpreted over T-coalgebras where TX is the set of finitely supported subprobability distributions over X, that is,

$$TX = \{\mu : X \to [0,1] \mid \textstyle\sum_{x \in X} \mu(x) \leq 1\} \times \mathcal{P}(\mathsf{P})$$

where the modalities M_p, read as "with probability of more than p", are interpreted via the liftings $[\![M_p]\!]_X(A) = \{(\mu, D) \in TX \mid \sum_{x \in A} \mu(x) > p\}$ which induces, up to the move to subprobabilities, the standard semantics.

We note that all similarity types except that of (unbounded) graded modal logic and the modal logic of subprobabilities are finite, provided that we only have finitely many propositional variables.

The aim of this work is to characterize the expressive power of $\mathcal{F}(\Lambda)$ as the fragment of first-order logic that is invariant under behavioural equivalence. The latter is best described in terms of coalgebra homomorphisms. A *morphism* between T-coalgebras (C, γ) and (D, δ) is a function $f : C \to D$ such that $\delta \circ f = Tf \circ \gamma$. Given T-coalgebras (C, γ) and (D, δ), two states $(c, d) \in C \times D$ are called *behaviourally equivalent*, written $C, c \approx D, d$, if they can be identified by a morphism of T-coalgebras, i.e. there are morphisms $f : (C, \gamma) \to (E, \epsilon)$ and $g : (D, \delta) \to (E, \epsilon)$ into a T-coalgebra (E, ϵ) such that $f(c) = g(d)$. Formulas of $\mathcal{F}(\Lambda)$ are invariant under behavioural equivalence:

Lemma 4. *Let C, D be T-coalgebras and let $(c, d) \in C \times D$ be behaviourally equivalent. Then $c \models \phi$ iff $d \models \phi$ for all $\phi \in \mathcal{F}(\Lambda)$.*

In other words, modal formulas are invariant under behavioural equivalence. Our main theorem extends [22] to a coalgebraic setting and establishes that all first order formulas in a suitable correspondence language with this property are in fact equivalent to modal formulas. The proof follows Rosen [18] and Otto [15], and in particular makes use of the stratification of behavioural equivalence that explicitly accounts for the number of transition steps. From a coalgebraic perspective, this comes about by considering the projections of (states of) coalgebras into the so-called *terminal sequence* of the underlying functor (see [17] for a detailed exposition from the logical viewpoint). The objects of the terminal sequence are given by $T_0 = 1$ for an arbitrary one-element set and $T_n = T(T_{n-1})$, and are connected by functions $p_n : T_{n+1} \to T_n$ where $p_0 : T_1 \to 1$ is uniquely determined and $p_n = Tp_{n-1}$. Every T-coalgebra (C, γ) defines a cone over the terminal sequence by $\gamma_0 : C \to 1$ and $\gamma_n = T\gamma_{n-1} \circ \gamma : C \to T_n$. Given two T-coalgebras C and D, we can now call a pair $(c, d) \in C \times D$ *n-step equivalent*, in symbols $C, c \approx_n D, d$, if $\gamma_n(c) = \delta_n(d)$. The following lemma relates n-step equivalence and behavioural equivalence:

Lemma 5. *Let C, D be T-coalgebras, and let $n \geq k \in \omega$. For $(c, d) \in C \times D$, $c \approx d$ implies that $c \approx_n d$, and $c \approx_n d$ implies that $c \approx_k d$.*

The converse is true if modal operators distinguish "enough" successor states.

Definition 6. The Λ-structure T is *separating* if, for all sets X, every $t \in TX$ is uniquely determined by $\{(\heartsuit, A) \mid \heartsuit \in \Lambda \; n\text{-ary}, A \in \mathcal{P}(X)^n, t \in [\![\heartsuit]\!]_X(A)\}$.

Separation is sufficient to establish the Hennessy-Milner property for coalgebraic modal logics [17,20], and all the structures in Example 3 are indeed separating. In particular, we obtain characteristic formulas for n-step equivalence.

Lemma 7. *If the Λ-structure T is separating and (C, γ) is a T-coalgebra, every $\approx_k$-equivalence class is definable by a formula in $\mathcal{F}_\infty(\Lambda)$ of modal rank $\leq k$.*

2 From Coalgebraic Models to First-Order Structures

The characterization of (coalgebraic) modal logic as a fragment of first-order logic stands or falls with the first-order correspondence language. In general, one needs to balance expressivity of the correspondence language against the characterization results, and the value of our results increases with the expressive power of the correspondence language. The first-order correspondence language that we use here is inspired by [11] as it is multi-sorted and includes specific sorts for states and neighbourhoods. However, it also includes a third sort for *structured successors*, i.e. elements of the set TS where S is the state set. This is, to our taste, not only the most natural first-order modelling of coalgebras, but also strengthens our main result as it increases the expressivity of the correspondence language. Even more expressivity is owed to a slightly surprising feature, whose motivation is more technical in nature: one needs expressive means for the notion of support (Definition 2), which serves the dual purpose of restricting neighbourhoods (thus keeping the logic away from full monadic second-order logic) and on the other hand to avoid vacuity of Gaifman locality (see Remark 10 for details). The following non-essential assumption simplifies the presentation.

Assumption 8. We assume that $T\emptyset$ has a distinguished element $\bot_T$, and hence that every set TX has a distinguished element $Ti(\bot_T)$, also denoted $\bot_T$, where $i : \emptyset \to X$ is inclusion. Moreover, we assume that $\bot_T \notin [\![\heartsuit]\!]_\emptyset(\emptyset, \dots, \emptyset)$ for every k-ary operator $\heartsuit \in \Lambda$, and hence, by naturality, that $\bot_T \notin [\![\heartsuit]\!]_X(A_1, \dots, A_k)$ for all sets X and all $A_1, \dots, A_k \subseteq X$. This is mainly for the sake of readability, as it makes the definition of the standard translation more straightforward. In our running examples, $\bot_T$ can be taken to be

- $\bot_T = \emptyset \in \mathcal{P}(\emptyset)$ for the modal logic K (presented in terms of $\Diamond$);
- $\bot_T = \emptyset \in \mathcal{P}(\mathcal{P}(\emptyset))$ for classical and monotone modal logic;
- $\bot_T = \lambda A.\, \emptyset \in \mathcal{P}(\emptyset) \to \mathcal{P}(\emptyset)$ for conditional logic.
- $\bot_T = \lambda x.\, 0$ for graded modal logic (presented in terms of the $\Diamond_k$);
- $\bot_T = \lambda x.\, 0$ for the modal logic of subprobabilities.

Definition 9. The *(coalgebraic) correspondence language* associated with the modal similarity type Λ is the first order language with equality $\mathcal{L}(\Lambda)$ over three sorts s, t, n of *states*, *successor structures*, and *neighbourhoods*, respectively, consisting of the sorted relation symbols

– $\mathsf{tr} : s \times t$ (the coalgebraic transition structure)
– $\heartsuit : t \times n \times \cdots \times n$ (k copies of n) for all k-ary $\heartsuit \in \Lambda$ (the modal operators)
– $\in : s \times n$ (membership of points in neighbourhoods)
– $\mathsf{supp} : t \times n$ (support, see Definition 2)

The relation tr represents the transition structure, and is notionally treated as a partial map where undefined represents absence of successors. Whenever we use the term $\mathsf{tr}(x)$ for some x, we implicitly assume that $\mathsf{tr}(x)$ is defined. The (functional) relation supp encodes a particular choice of support for every $x : t$. Given a first-order $\mathcal{L}(\Lambda)$-structure M, we denote the constituents of M by indexing as usual; e.g. M_s is the state set of M, and M_{tr} the successor relation. We say that M is *based on* a T-coalgebra (C, γ) if the following holds.

– $M_s = C$, $M_t \subseteq TC$ and $M_n \subseteq \mathcal{P}(C)$
– The relation M_{tr} is right-unique, and hence will be written as a partial map. It represents the transition structure γ with default value $\bot_T$; i.e. for each $c \in C$, $\gamma(c) = M_{\mathsf{tr}}(c)$ whenever $M_{\mathsf{tr}}(c)$ is defined, and $\gamma(c) = \bot_T$ otherwise.
– The relation M_{supp} is functional, and will also be written as a map. It picks a distinguished support (Definition 2) for every $\mu \in M_t$, i.e. $\mu \in T(M_{\mathsf{supp}}(\mu))$.
– The relations $M_{\heartsuit}$ represent the predicate liftings for every $\heartsuit \in \Lambda$ relative to the support, i.e. $M_{\heartsuit} = \{(\mu, A_1, \ldots, A_n) \in M_t \times \mathcal{P}(M_{\mathsf{supp}}(\mu))^n \mid \mu \in [\![\heartsuit]\!]_{M_{\mathsf{supp}}(\mu)}(A_1, \ldots, A_n)\}$ for $\heartsuit \in \Lambda$.
– $M_{\in}$ is membership: $M_{\in} = \{(s, A) \in C \times M_n \mid s \in A\}$.

We write $\mathsf{Mod}(\mathcal{L}(\Lambda))$ for the class of all $\mathcal{L}(\Lambda)$-structures that are based on some T-coalgebra, briefly referred to as *T-structures*. As every T-structure induces a uniquely defined T-coalgebra (C, γ) we occasionally regard T-structures as T-coalgebras, and in particular use notions such as behavioural equivalence. If M is a first-order structure for $\mathcal{L}(\Lambda)$ and $\phi(x) \in \mathcal{L}(\Lambda)$ is a formula with at most one free variable x of sort s, we write $M, m \models \phi(x)$ if the structure M with the free variable interpreted as m satisfies the formula $\phi(x)$, and $[\![\phi(x)]\!]_M$ is the set of all m such that $M, m \models \phi(x)$. We say that $\phi(x)$ is *invariant under behavioural equivalence* if $M, m \models \phi$ whenever $N, n \models \phi$ and $N, n \approx M, m$.

Our main interest in the present work is to establish results following van Benthem [22] and Rosen [18] which state that every first-order formula which is invariant under behavioural equivalence is equivalent to a modal formula, valid over the class of all structures and over the class of all finite structures, respectively. Occasionally we shall refer to results of the former kind as *van-Benthem-type* theorems, and to results of the latter kind as *Rosen-type* theorems.

Remark 10. Some explanations are in order concerning some aspects of the above definition. We first note that it is crucial that we do not require that the sort n of neighbourhoods is interpreted by the entire powerset of states. Otherwise, we would essentially arrive at monadic second-order logic, and hence invalidate the main theorem already for the case of the modal logic K as the bisimulation-invariant fragment of monadic second-order logic is the μ-calculus rather than the basic modal logic K [13]. Definition 9 restricts the interpretation of n only by the clause on the interpretation of the modal operators. Technically, this makes less formulas invariant under behavioural equivalence, as the interpretation of n may differ among behaviourally equivalent models. This is the first effect of support: without support, the interpretation of $\heartsuit \in \Lambda$ would need to be defined as something like $M_\heartsuit = \{(\mu, (A_1, \dots, A_n)) \in M_t \times \mathcal{P}(C)^n \mid \mu \in [\![\heartsuit]\!]_{r_s(\mu)}(A_1, \dots, A_n)\}$ which would constrain the interpretation of n much more strongly, and e.g. in the case of the modal logic K (with $\heartsuit = \Diamond$) would imply $A \in M_n$ for every $A \in \mathcal{P}(C)$ containing some state that has a predecessor.

The second technical point where support is needed is the following. The core of the proof of Rosen's theorem, as adapted below, is *locality*. In particular, we use Gaifman's theorem stating that every first-order formula is essentially local (see Section 3). Without support, however, locality becomes a void notion in many logics. E.g. in the extension of classical modal logic with necessitation, i.e. with an axiom $\Box\top$, any two points in the model would be connected by a path of length 3 (via the successor structure of the first point and the neighbourhood $\top$). The formalization of support as a functional relation therefore pre-empts this trivialization. As support has already played an important technical role in other contexts [21], and is also at the heart of our unravelling construction, we are beginning to believe it may be more than just a technical nuisance.

Finally, the purpose of the default value in the above definition of T-structure is to deal with substructures that arise by cutting off transitions after a fixed number of steps. In the setting of Kripke frames, this corresponds to all successors of a node x being lost in a substructure, so that x has the empty successor set, and hence fails to satisfy diamond formulas. This notion of cutting off transitions is made explicit by our default mechanism.

The correspondence language contains standard or natural correspondence languages when applied to the running examples, as illustrated next. In most cases, the generic language is even substantially more expressive than the 'natural' correspondence language, and van Benthem/Rosen-type results become *stronger* in the context of more expressive languages, as they apply to more formulas. Some of the examples moreover highlight the importance of support.

Example 11. 1. The coalgebraic correspondence language for K differs rather substantially from the standard first-order correspondence language, the language with a unary predicate p for every propositional variable p and a binary predicate R for the transition relation — it does not explicitly talk about the transition relation, and instead has types for successor sets and neighbourhoods (one of which could be dispensed with in this case). Crucially, the standard correspondence language can be embedded into the language used here, so that our characterization result established in Section 4 does reprove, and in fact strengthen, the classical van Benthem/Rosen theorem. The embedding is defined by mapping atomic formulas xRy in the standard correspondence language to the formula $\exists A.\,(\mathsf{tr}(x)\Diamond A \wedge \forall z.\,(z \in A \leftrightarrow z = y))$.

2. In the case of classical modal logic with neighbourhood frame semantics, our correspondence language has a sort s of states, a sort t of sets of neighbourhoods, and a sort n of neighbourhoods, with the successor map tr which maps states to sets of neighbourhoods, the relation $\Box$ which represents the $\ni$ relation between sets of neighbourhoods and neighbourhoods, membership $\in$ between states and neighbourhoods, and additionally the support map supp from set of neighbourhoods to neighbourhoods. Superficially, this appears to be an extension of the correspondence language for neighbourhood frames used in [11], which has two sorts of states and neighbourhoods, respectively, corresponding to our sorts s and n, and two relations, corresponding to the composite $\mathsf{tr};\Box$ and to $\in$, respectively, in our setting. However, the two languages have a subtly different semantics in that in our language, the sets required to be present in the neighbourhood type are determined by the support, while in [11], the neighbourhood type contains precisely the image of the neighbourhood relation between states and neighbourhoods. At present, it is unclear whether one language can be embedded in the other so that the van Benthem-type result of [11] remains independent of our characterization theorem result below (there is no correspondent in [11] to our Rosen-type result).

We emphasize that our results will apply without further ado to extensions of classical modal logic by rank-1 frame conditions, i.e. axioms where the nesting depth of modal operators is uniformly equal to 1. A simple example is the monotonicity axiom $\Box(a \wedge b) \to \Box a$, which axiomatizes monotone neighbourhood frames (Example 3.3). It is unclear to what extent this is possible following [11]. In particular, one cannot interpret the type of neighbourhoods as the set of all sets that are a neighbourhood of some state, as the induced logic would be able to express some (if not all) μ-calculus formulas, such as $(\Box\top \wedge \neg\Box\neg\Box\bot) \to \mu X.p \wedge \Box X$: as soon as the antecedent $\Box\top \wedge \neg\Box\neg\Box\bot$ is satisfied, there is a state which has $\emptyset$ as a neighbourhood, so that the neighbourhood type would contain *all* sets of states by monotonicity, thus allowing us to define the fixpoint above by quantification over neighbourhoods. As formulas of this type are invariant under behavioural equivalence but not equivalent to any modal formula, the analogue of the van Benthem/Rosen theorem fails. In the approach presented here, the notion of support enables sufficiently small interpretations of the neighbourhood type and handles rank-1 frame conditions smoothly.

3. One natural correspondence language for graded modal logic, interpreted over multigraphs (thus recovering invariance under behavioural equivalence, which fails over the relational semantics) is an extension of first order logic with counting quantifiers where the counting is relative to local weights induced by the weighting of successors in a multigraph. This induces counting quantifiers $\exists^x_{>k} y.\,\phi$ read 'in the local weighting at x, there exist more than k y satisfying ϕ'. This language can be mapped into our language by a recursive translation $(\_)^t$, where the clause for a counting quantifier is $(\exists^x_{>k} y.\,\phi)^t = (\exists A : n.\,\mathsf{tr}(x)\Diamond_k A \wedge \forall y.\,(\phi)^t \leftrightarrow y \in A)$. Support is uncritical in this case (as already for

item [1]), as the first-order language with counting quantifiers does not contain a sort for neighbourhoods. The standard translation factors through the language of counting quantifiers via translations $(\_)^s_x$ taking $\Diamond_k \phi$ to $\exists^x_{>k} y.\, (\phi)^t_y$. As a consequence, the characterization results proved below apply *a forteriori* also to the language with counting quantifiers.

4. Similarly, the correspondence language for the modal logic of subprobabilities contains a sublanguage that speaks about locally determined weights of formulas: borrowing notation from Halpern's first-order logic of so-called type-1 probability structures [10] (which we extend from a single, global probability distribution to Markov chains, i.e. local (sub-)probability distributions), we may write $w^x_y(\phi) \geq p$ to denote that the set of all y satisfying ϕ has, in the local distribution at x, probability at least p.

5. The correspondence language for conditional logic contains the following more natural language, consisting of three sorts s, t, n for states, selection functions, and neighbourhoods, respectively, unary state predicates for the propositional variables, and a ternary relation R of type $s \times n \times s$ giving for each neighbourhood a transition relation on states. As neighbourhoods are explicit, we need to retain the support function supp. This corresponds to the view of a selection function model as a multi-relational Kripke model where relations are indexed over propositions, i.e. a structure of the type $(X, (R_A \subseteq X \times X)_{A \subseteq X})$, where we retain information only about those R_A for which A is in the neighbourhood type. Here, it is again crucial that the neighbourhood type is required to contain only those sets that are contained in the support of some element – without the support, neighbourhoods would be the full powerset, thus affording the full expressive power of monadic second order logic, as every selection function on a set C is contained in $[\![\Rightarrow]\!](A, C)$ for every $A \in \mathcal{P}(C)$.

The translation of modal formulas to first-order logic takes the following form:

Definition 12. The *standard translation* $\mathsf{ST}_x(\phi)$ of a modal formula $\phi \in \mathcal{F}(\Lambda)$ is a first-order formula with one free variable $x : s$ of sort s defined inductively by commutation with all boolean operators and

$$\begin{aligned}\mathsf{ST}_x(\heartsuit(\phi_1, \dots, \phi_k)) = \exists A_1, \dots, A_k : n.\, (\mathsf{tr}(x)\heartsuit(A_1, \dots, A_k) \wedge \\ \textstyle\bigwedge_{i=1}^k \forall y : s.\, (y \in A_i \leftrightarrow y \in \mathsf{supp}(\mathsf{tr}(x)) \wedge \mathsf{ST}_y(\phi_i))).\end{aligned}$$

The default value $\bot_T$ is compatible with the above definition: $\mathsf{ST}_x(\heartsuit(\phi_1, \dots, \phi_k))$ is not satisfied in case $\mathsf{tr}(x)$ is undefined, which agrees precisely with the behaviour of the default value $\bot_T$ inserted as the successor structure of x in this case. Correctness is a straightforward calculation:

Lemma 13. *Suppose $\phi = \mathsf{ST}_x(\psi)$ for a modal formula $\psi \in \mathcal{F}(\Lambda)$. Let M be a first-order structure based on a coalgebra C. Then $[\![\phi]\!]_M = [\![\psi]\!]_C$. In particular, ϕ is invariant under behavioural equivalence.*

Remark 14. Unlike Rosen's proof [18], the original proof of van Benthem's characterization result, as well as e.g. the van-Benthem-type result proved for neighbourhood structures in [11], rely on standard machinery from first-order logic, in particular compactness. There are at least two sources of non-compactness in the overall setup used here: one, of course, rests in the fact that we are aiming for a Rosen-type theorem over *finite models*; and the other is the functor T. Not only may T impose finite branching; e.g. in case T is the probability distribution functor, the set of formulas $\{\neg L_0 \neg p\} \cup \{M_{1-1/n} p \mid n \in \mathbb{N}\}$ for a propositional variable p is finitely satisfiable but not satisfiable, no matter what type of model (finite, infinite, finitely or infinitely branching) we consider.

3 Gaifman's Theorem and Coalgebraic Unravelling

We recall Gaifman's locality theorem [8], and derive a simple corollary that asserts locality of coproduct-invariant formulas (we claim no originality here). The basic idea is taken from [15], where the same statement is proved as Lemma 3.5 for at most binary relational structures. We apply a simpler if somewhat wholesale argument using Gaifman locality.

Definition 15. The *Gaifman graph* of a relational structure A is the graph whose nodes are the elements of A and contains and edge from a to b iff a and b occur together in one of the tuples in the interpretations of the relation symbols in A. (E.g., the Gaifman graph of a single-relation Kripke structure is just its symmetric closure.) *Gaifman distance* is graph distance in the Gaifman graph. For $l \geq 0$, the *l-neighbourhood* $N_l^A(a)$ of $a \in A$ is the induced substructure of A containing all points with Gaifman distance at most l from a. A first-order formula $\phi(x)$ with a single free variable x is *l-local* if for every relational structure A and every $a \in A$, $A, a \models \phi(x)$ iff $N_l^A(a), a \models \phi(x)$. Moreover, $\phi(x)$ is *Gaifman l-local* if for any two points $a, b \in A$ with isomorphic l-neighbourhoods, $A, a \models \phi(x)$ iff $A, b \models \phi(x)$.

Gaifman locality is weaker than locality in that it admits statements about the global structure of models. We need a special case of Gaifman's theorem:

Theorem 16 (Gaifman [8]). *Every first-order formula $\phi(x)$ is Gaifman l-local for some $l \geq 0$, exponentially bounded in the quantifier rank of ϕ.*

Gaifman's theorem is usually formulated in single-sorted logic, but readily extends to multiple sorts, with the obvious definition of Gaifman distance, using the standard encoding of multiple sorts as unary predicates in single-sorted logic.

Definition 17. A formula $\phi(x)$ with a single free variable x is *invariant under coproducts* if for all relational structures A, B and all points $a \in A$, $A, a \models \phi(x)$ iff $A + B, a \models \phi(x)$, where $A + B$ is the coproduct (disjoint union) of A and B.

Corollary 18. *If $\phi(x)$ is invariant under coproducts, then $\phi(x)$ is l-local for some $l \geq 0$, exponentially bounded in the quantifier rank of $\phi(x)$.*

The previous does not immediately apply to our framework, as neighbourhoods in T-structures are not in general T-structures. The following lemma brings us back into the realm of T-structures, thanks to the default element $\bot_T \in T\emptyset$.

Lemma 19. *Let A be a T-structure, let a be a state in A, and let $k \geq 0$. Then $N_{3k}(a)$ is a T-structure.*

We proceed to develop some facts concerning (partial) tree unravellings of coalgebras, in generalization of corresponding techniques for Kripke frames, including a not entirely trivial coalgebraic generalization of the fact that on trees, behavioural equivalence is equivalent to bounded behavioural equivalence (Lemma 21). The basic notion underlying these concepts is the following.

Definition 20. Let A be a T-structure for the correspondence language. The *supporting Kripke frame* of A relates states $a, b \in A_s$ iff $b \in A_{\mathsf{supp}}(A_\gamma(a))$; i.e. its transition relation is $A_\gamma; A_{\mathsf{supp}}; A_\ni$ where $A_\ni$ is the inverse relation of $A_\in$. If this Kripke frame is a tree of depth l (with root a), i.e., is loop-free and every state is reachable from a by a unique path of length at most l, and moreover all leaves of this tree have the default successor structure (i.e. they do not have an R-successor) then we say that A (or (A, a)) is a *tree of depth l.*

Lemma 21. *Let A, B be T-structures with states $a \in A$, $b \in B$. If (A, a) and (B, b) are trees of depth at most l, then $A, a \approx B, b$ iff $A, a \approx_l B, b$.*

The core construction is described in the following lemma.

Lemma 22 (Unravelling). *Let A be a T-structure, let a be a state in A, and let $k \geq 0$. Then there exists a T-structure B and $b \in B$ such that $A, a \approx B, b$, and moreover $(N^B_{3k}(b), b)$ is a tree of depth at most k.*

Finally, we note that bounded behavioural equivalence is indeed local.

Lemma 23. *Let A be a T-coalgebra, let $a \in A$, and let $k \geq 0$. Then $A, a \approx_k N^A_{3k}(a), a$.*

4 A Coalgebraic van Benthem/Rosen Theorem

The core result proved in relational versions of the van Benthem/Rosen theorem is that every bisimulation-invariant formula can be expressed by a collection of modal formulas of bounded rank. In the relational case, this immediately implies equivalence to a single modal formula, as the set of modal formulas of a given maximal rank is finite up to logical equivalence. Coalgebraically, the situation turns out to be the same as long as the modal similarity type is finite. For infinite modal similarity types, the infinitary version of the van Benthem/Rosen theorem cannot be improved, as we demonstrate by means of a simple counterexample later. We thus tend to regard the infinitary version, stated next, as the most fundamental incarnation of the van Benthem/Rosen theorem. We emphasize that the bound on the rank in the statement of the theorem is the core of the result – without it, the claim is a trivial consequence of the (coalgebraic) Hennessy-Milner property for infinitary languages [17]. As in [15], the theorem below has two readings, for finite and infinite models.

Theorem 24 (Coalgebraic van Benthem/Rosen theorem, infinitary version). *Let Λ be separating. A first-order formula $\phi(x)$ over T with a single free variable x is invariant under behavioural equivalence (over finite models) iff it is equivalent (over finite models) to a modal formula in $\mathcal{F}_\infty(\Lambda)$ with finite modal rank.*

The proof uses the Lemmas established in Section 3 in sequence. As announced, the finitary version of the theorem follows immediately for finite similarity types:

Corollary 25 (Coalgebraic van Benthem/Rosen theorem, finitary version). *Let Λ be finite and separating. Then a first-order formula $\phi(x)$ over T with a single free variable x is invariant under behavioural equivalence (over finite models) iff it is equivalent (over finite models) to a modal formula in $\mathcal{F}(\Lambda)$.*

For the logics introduced in Example 3, the situation is as follows.

Example 26. Theorem 24 applies to all logics of Example 3, and Corollary 25 applies to those logics that only have finitely many modalities, i.e. all of them except (unbounded) graded modal logic and probabilistic modal logic; we note that Corollary 25 does apply to our bounded version of graded modal logic. We emphasize that Theorem 24 does yield a characterization of the behavioural-equivalence-invariant fragment of a first-order logic with counting quantifiers; while a similar van Benthem-type result is known [6], our result seems to be the first Rosen-type result (i.e. over finite structures) for graded modal logic.

As indicated above, a simple example shows that in the full correspondence language for an infinite modal similarity type, one can express properties which are invariant under behavioural equivalence but not expressible by a finitary modal formula, even in the standard coalgebraic modelling of Kripke models with infinitely many variables; in other words, the infinitary version of the van Benthem/Rosen theorem cannot be improved for infinite modal similarity types.

Example 27. Recall that standard Kripke models over the set P of variables are modelled by the functor $TX = \mathcal{P}(X) \times \mathcal{P}(\mathsf{P})$ (Example 3.1). Then the following formula is invariant under behavioural equivalence:

$$\begin{aligned}\exists y, z : s, Y, Z, A : n. \,(\forall w : s. \,((w \in Y \leftrightarrow w = y) \wedge (w \in Z \leftrightarrow w = z) \wedge w \in A)\\ \wedge \, \mathsf{tr}(x)\Diamond Y \wedge \mathsf{tr}(x)\Diamond Z \wedge \mathsf{tr}(y)\neg\Diamond A \wedge \mathsf{tr}(z)\neg\Diamond A \wedge \mathsf{tr}(y) \neq \mathsf{tr}(z))\end{aligned}$$

This formula states that x has two successors y, z which are both deadlocks but disagree on the value of at least one propositional variable. This formula is evidently not equivalent to any finitary modal formula. However, it is expressible by the infinitary modal formula $\bigvee_{p\in\mathsf{P}}(\Diamond(p \wedge \neg\Diamond\top) \wedge \Diamond(\neg p \wedge \neg\Diamond\top))$, which has modal depth 2, thus illustrating Theorem 24.

Note that proofs of the Rosen theorem in a relational setting begin with a (trivial) reduction to finitely many variables, which is possible precisely because the standard correspondence language does not allow one to say that two states agree on all propositional variables. Of course, the example above depends heavily on the use of equality on t. We state the following nagging open question:

Problem 28. Let Λ be separating. Is every formula of the correspondence language that is invariant under behavioural equivalence and does not mention equality on t equivalent to a finitary modal formula?

We note that in the case of infinite collections of independent modal operators, such as infinitely many propositional variables, or boxes for infinitely many unrelated agents, the question is answered positively by a trivial reduction to the finite case. The problematic case are infinite collections of interdependent operators as, e.g., in graded modal logic.

5 Conclusions and Related Work

We have introduced a correspondence language for coalgebraic modal logic, and proved two van Benthem/Rosen type theorems using this language: an infinitary version which applies to *every* separating coalgebraic modal logic, and states that every formula which is invariant under behavioural equivalence is equivalent to an infinitary modal formula *of bounded depth*; and, as an easy corollary to this, a finitary version which improves this to equivalence to a finitary modal formula for *finite* modal similarity types, a condition which in connection with separation implies that the type functor preserves finite sets. The infinitary result yields e.g. that a formula in a natural first order logic of multigraphs with counting quantifiers is invariant under behavioural equivalence iff it is equivalent to a bounded-depth infinitary graded modal formula. The finitary result yields characterizations of conditional logic, classical modal logic, monotone modal logic, and a bounded version of graded modal logic as the invariant fragment under behavioural equivalence in the respective correspondence languages.

It remains an open problem to extend the finitary result to infinite modal similarity types; a simple example shows that this can work only for a restricted correspondence language that excludes equality on the type of successor structures. This would in particular imply finitary van Benthem/Rosen theorems for graded and probabilistic modal logic. The former would complement a van Benthem-type result for graded modal logic proved in [6] by a Rosen-type result (i.e. over finite structures). A further interesting direction for future investigation is to extend the ambient logic, in particular to obtain a coalgebraic analogue of the characterization of the modal μ-calculus as the bisimulation-invariant fragment of monadic second order logic due to Janin and Walukiewicz [13].

Acknowledgement. The authors wish to thank Helle Hvid Hansen for useful discussions and pointers, and Erwin R. Catesbeiana for suggesting the use of $\perp$.

References

1. Barr, M.: Terminal coalgebras in well-founded set theory. Theoret. Comput. Sci. 114, 299–315 (1993)
2. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge University Press, Cambridge (2001)

3. Chellas, B.: Modal Logic. Cambridge University Press, Cambridge (1980)
4. D'Agostino, G., Visser, A.: Finality regained: A coalgebraic study of Scott-sets and multisets. Arch. Math. Logic 41, 267–298 (2002)
5. Dawar, A., Otto, M.: Modal characterisation theorems over special classes of frames. In: Panangaden, P. (ed.) Logic in Computer Science, LICS 2005, pp. 21–30. IEEE Computer Society, Los Alamitos (2005)
6. de Rijke, M.: A note on graded modal logic. Stud. Log. 64, 271–283 (2000)
7. Fine, K.: In so many possible worlds. Notre Dame J. Formal Logic 13, 516–520 (1972)
8. Gaifman, H.: On local and non-local properties. In: Logic Colloquium 1981, pp. 105–135. North Holland, Amsterdam (1982)
9. Gumm, H.P.: From T-coalgebras to filter structures and transition systems. In: Fiadeiro, J.L., Harman, N.A., Roggenbach, M., Rutten, J. (eds.) CALCO 2005. LNCS, vol. 3629, pp. 194–212. Springer, Heidelberg (2005)
10. Halpern, J.Y.: An analysis of first-order logics of probability. Artif. Intell. 46, 311–350 (1990)
11. Hansen, H.H., Kupke, C., Pacuit, E.: Bisimulation for neighbourhood structures. In: Mossakowski, T., Montanari, U., Haveraaen, M. (eds.) CALCO 2007. LNCS, vol. 4624, pp. 279–293. Springer, Heidelberg (2007)
12. Heifetz, A., Mongin, P.: Probabilistic logic for type spaces. Games and Economic Behavior 35, 31–53 (2001)
13. Janin, D., Walukiewicz, I.: Automata for the modal μ-calculus and related results. In: Hájek, P., Wiedermann, J. (eds.) MFCS 1995. LNCS, vol. 969, pp. 552–562. Springer, Heidelberg (1995)
14. Larsen, K., Skou, A.: Bisimulation through probabilistic testing. Inform. Comput. 94, 1–28 (1991)
15. Otto, M.: Bisimulation invariance and finite models. In: Logic Colloquium 2002. Lect. Notes Log., vol. 27, pp. 276–298. ASL (2006)
16. Pattinson, D.: Coalgebraic modal logic: Soundness, completeness and decidability of local consequence. Theoret. Comput. Sci. 309, 177–193 (2003)
17. Pattinson, D.: Expressive logics for coalgebras via terminal sequence induction. Notre Dame J. Formal Logic 45, 19–33 (2004)
18. Rosen, E.: Modal logic over finite structures. J. Logic, Language and Information 6(4), 427–439 (1997)
19. Schröder, L.: A finite model construction for coalgebraic modal logic. J. Log. Algebr. Prog. 73, 97–110 (2007)
20. Schröder, L.: Expressivity of coalgebraic modal logic: The limits and beyond. Theoret. Comput. Sci. 390, 230–247 (2008)
21. Schröder, L., Pattinson, D.: Strong completeness of coalgebraic modal logics. In: Albers, S., Marion, J.-Y. (eds.) Theoretical Aspects of Computer Science, STACS 2009, Leibniz International Proceedings in Informatics, pp. 673–684. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl (2009)
22. van Benthem, J.: Modal Correspondence Theory. PhD thesis, Department of Mathematics, University of Amsterdam (1976)

Untyped Recursion Schemes and Infinite Intersection Types

Takeshi Tsukada and Naoki Kobayashi

Tohoku University

Abstract. A new framework for higher-order program verification has been recently proposed, in which higher-order functional programs are modelled as higher-order recursion schemes and then model-checked. As recursion schemes are essentially terms of the *simply-typed* lambda-calculus with recursion and tree constructors, however, it was not clear how the new framework applies to programs written in languages with more advanced type systems. To circumvent the limitation, this paper introduces an *untyped* version of recursion schemes and develops an infinite intersection type system that is equivalent to the model checking of untyped recursion schemes, so that the model checking can be reduced to type checking as in recent work by Kobayashi and Ong for typed recursion schemes. The type system is undecidable but we can obtain decidable subsets of the type system by restricting the shapes of intersection types, yielding a sound (but incomplete in general) model checking algorithm.

1 Introduction

Model checking of recursion schemes [1, 2] has recently been applied to higher-order program verification [3, 4]. A recursion scheme is a grammar for generating a possibly infinite tree; from a programming language point of view, it is a term of the simply-typed λ-calculus with recursion and tree constructors. The idea of the higher-order program verification is to transform a higher-order program into a recursion scheme that generates a tree representing event sequences of the program, so that temporal properties of the program can be verified by model-checking the recursion scheme. The main advantage of the verification method is that it is sound and *complete* for a certain class of higher-order programs – the *simply-typed* λ-calculus with recursion and finite base types.

There is however a gap between the class of higher-order programs that can be handled directly by the above method (the *simply-typed* λ-calculus with finite base types) and the class of programs written in real programming languages. One of the main restrictions on the former class is that programs must be *simply-typed*. It was unclear how the method above can be applied to programs that use more advanced types, such as polymorphism and recursive types.

To address the problem above, we remove the restriction that recursion schemes must be simply-typed, by extending Kobayashi and Ong's type-based approach to model-checking recursion schemes [3, 5]. Instead of considering each extension of the simple type system (such as ML type system and System F with/without recursive types), we study the most expressive type system – an infinite intersection sort system for recursion schemes, and develop a type-based method for model-checking intersection-typed recursion schemes. (The infinite intersection sort system is the most expressive in the sense that all untyped recursion schemes that generate valid trees are typable in the type system.) Since various type systems, possibly with polymorphic and recursive types, can be regarded as restricted forms of the intersection type system, our studies of the intersection-typed recursion schemes can serve as a common foundation for recursion schemes extended with various type systems.

In the paper, in Section 3, we first give an intersection *sort* system that exactly characterizes the (untyped) recursion schemes that generate well-ranked trees, in the sense that a recursion scheme is well-sorted if and only if it generates a well-ranked tree. We then (in Section 4) introduce an intersection type system, parameterized by a Büchi automaton $\mathcal{A}$ with a trivial acceptance condition, such that a recursion scheme is well-typed if and only if it generates a well-ranked tree accepted by $\mathcal{A}$. One of the main results, presented in Section 5, is that if a recursion scheme is well-sorted under a sort environment Γ, the recursion scheme is well-typed if and only if it is so under a refinement of Γ. Thus, although the infinite intersection type system is undecidable in general, it is decidable whether, given a recursion scheme well-sorted under a *finite* sort environment, the recursion scheme is well-typed. As a consequence, the model checking of recursion schemes with ML-style polymorphism is decidable. Model checking of recursion schemes with more advanced type systems is undecidable, but we can still obtain a sound and decidable (but incomplete) type system by restricting the syntax of intersection types so that there can be finitely many refinements for each sort environment.

For the space restriction, we omit some proofs. They are found in a longer version available from `http://www.kb.ecei.tohoku.ac.jp/~tsukada/papers/fossacs10-full.pdf`.

2 Preliminaries

Trees Let $T \subseteq (\omega - \{0\})^*$ be a subset of finite sequences of natural numbers without 0. T is a *tree* if it satisfies the following conditions: (i) $\varepsilon \in T$, (ii) if $pi \in T$ for $p \in (\omega - \{0\})^*, i \in \omega - \{0\}$, then $p \in T$, and (iii) if $pi \in T$ for $p \in (\omega - \{0\})^*, i \in (\omega - \{0\})$, then $pj \in T$ for every $1 \leq j \leq i$. Note that 0 is not used as an index of trees; This convention makes some definitions and proofs simple. If $p, pi \in T$, we say p is the *parent* of pi and pi is a *child* of p. The *rank* of the node p, written $\#_T(p)$, is the cardinality of the set of its children. Note that for every $p \in T$, $\#_T(p) \leq \omega$. We omit the subscript T when it is clear from the context. We assume that the elements of the set $(\omega - \{0\})^*$ are ordered by the prefix ordering, i.e., $p_0 \leq p_1$ if and only if there is some p_1' such that $p_1 = p_0p_1'$.

Let A be an alphabet (i.e. a set of symbols). An *A-labeled tree* is a function $r : T \to A$ from a tree T to the alphabet A. Let Σ be a *ranked alphabet*, i.e. a map from an alphabet A to $\omega \cup \{\omega\}$. Σ is *finitely ranked* if $\Sigma(a) < \omega$ for every $a \in dom(\Sigma)$. By abuse of notation, we often write $a \in \Sigma$ for $a \in dom(\Sigma)$. A $dom(\Sigma)$-labeled tree r is also called a Σ-labeled tree. It is *well-ranked* if for every $p \in dom(r)$, $\Sigma(r(p)) = \#p$.

Trivial Automata. A Büchi tree automaton $\mathcal{A}$ with a trivial acceptance condition (called a *trivial automaton*, for short) is a quadruple (Q, Σ, q_S, Δ)[1] where Q is a finite set of states, Σ is a finitely-ranked alphabet, $q_S \in Q$ is an initial state, and $\Delta \subseteq Q \times \Sigma \times Q^*$ is a transition relation. The transition relation must respect the rank, i.e., if $(q, a, q_1, \ldots, q_n) \in \Delta$, then $n = \Sigma(a)$. A trivial automaton is *deterministic* if, for each pair $(q, a) \in Q \times dom(\Sigma)$, there is at most one element of the form $(q, a, q_1, \ldots, q_n)$ in Δ.

For a well-ranked Σ-labeled tree $r : T \to \Sigma$, a *run* of $\mathcal{A}$ on r is a Q-labeled tree $\varrho : T \to Q$, satisfying $(\varrho(p), r(p), \varrho(p1), \ldots, \varrho(pn)) \in \Delta$ for each $p \in T$, where $n = \Sigma(r(p))$.

For a state q, a Σ-labeled well-ranked tree r is *accepted by $\mathcal{A}$ from the state q*, if there is a run ϱ of $\mathcal{A}$ that satisfies $\varrho(\varepsilon) = q$. We write $\mathcal{L}_{\mathcal{A}}(q)$ for the set of trees accepted from q. The *language recognized by $\mathcal{A}$*, written $\mathcal{L}_{\mathcal{A}}$, is $\mathcal{L}_{\mathcal{A}}(q_S)$.

We assume that there is one distinguished element $\perp \in \Sigma$ with $\Sigma(\perp) = 0$, and for any state q of any trivial automata, $(q, \perp) \in \Delta$. Intuitively, $\perp$ is the undefined tree, which is accepted from any state.

Recursion Schemes. An *(untyped) recursion scheme* $\mathcal{G}$ is a quadruple $(\Sigma, \mathcal{N}, \mathcal{R}, S)$, where: (i) Σ is a ranked alphabet with a distinguished element $\perp$ of rank 0. An element of Σ is called a *terminal*; (ii) $\mathcal{N}$ is a finite set of symbols called *non-terminals*; (iii) $S \in \mathcal{N}$ is the *start symbol*; and (iv) $\mathcal{R}$ is a set of rewriting rules of the form $\{F_1 \tilde{x_1} \to t_1, \ldots, F_n \tilde{x_n} \to t_n\}$. Here, F_i is a nonterminal, $\tilde{x}$ abbreviates a sequence of variables and t_i is an applicative term over $\mathcal{N} \cup (\Sigma - \{\perp\}) \cup \{\tilde{x_i}\}$ (i.e. a term constructed from $\mathcal{N} \cup (\Sigma - \{\perp\}) \cup \{\tilde{x_i}\}$ and applications). There must be exactly one rule for each non-terminal. If $(F\tilde{x} \to t) \in \mathcal{R}$, we write $\mathcal{R}(F) = \lambda\tilde{x}.t$, where $\lambda\tilde{x}$ abbreviates a sequence of lambda abstractions. We identify an applicative term over X with an X-labeled tree in the standard manner: a term $x\,t_1\,\ldots\,t_n$ (where $x \in X$) as a tree whose root is labeled by x, having $t_1, \cdots, t_n$ as subtrees.

In the definition of standard recursion schemes [2], there are additional conditions that each symbol or variable is assigned a simple type (with $\mathtt{o}$, the type of trees, as a unique base type), and that both the left- and right-hand sides of each rule must have type $\mathtt{o}$. In this paper, we call the standard recursion schemes *(simply-)typed recursion schemes*, and call recursion schemes without the typing constraint *untyped recursion schemes* or simply *recursion schemes*.

The rewriting relation $\longrightarrow_{\mathcal{G}}$ is defined inductively by: (i) $F\tilde{s} \longrightarrow_{\mathcal{G}} [\tilde{s}/\tilde{x}]t$, if $\mathcal{R}(F) = \lambda\tilde{x}.t$, and (ii) if $t \longrightarrow_{\mathcal{G}} t'$, then $ts \longrightarrow_{\mathcal{G}} t's$ and $st \longrightarrow_{\mathcal{G}} st'$.

For a term t, we define $t^{\perp}$ by: (i) $a^{\perp} = a$ for each terminal a; (ii) $(t_1 t_2)^{\perp} = t_1^{\perp}\, t_2^{\perp}$ if $t_1^{\perp} \neq \perp$; and (iii) $t^{\perp} = \perp$ otherwise. The partial order $\sqsubseteq$ on Σ is defined as $a \sqsubseteq b$ if and only if $a = b$ or $a = \perp$. We extend this ordering to the ordering

1 A trivial automaton is a Büchi automaton where all the states are final.

on Σ-labeled trees by: $r_1 \sqsubseteq r_2$ if and only if $T_1 \subseteq T_2$ and $r_1(p) \sqsubseteq r_2(p)$ for every $p \in T_1$. The *value tree* of $\mathcal{G}$, written $[\![\mathcal{G}]\!]$, is $\bigsqcup\{t^{\perp} \mid S \longrightarrow^*_{\mathcal{G}} t\}$, where $\bigsqcup S$ is the least upper bound with respect to $\sqsubseteq$. The value tree is always well-defined, as the rewriting relation is confluent.

In this paper, we are interested in the model checking problem: "Given a recursion scheme $\mathcal{G}$ and a trivial automaton $\mathcal{A}$, does $[\![\mathcal{G}]\!] \in \mathcal{L}_{\mathcal{A}}$ hold?" In the case of *simply-typed* recursion schemes, the problem is known to be decidable [2].[2] In the case of *untyped* recursion schemes, however, the model checking problem above (or, even the problem of checking whether the value tree is well-ranked) is undecidable, as the untyped recursion schemes are essentially terms of the untyped λ-calculus (with uninterpreted function symbols).

Remark 1. The reader may wonder why we have recursion as primitives, even though a fixed-point combinator can be encoded in the untyped λ-calculus. That is because we later impose various type constraints corresponding to those of advanced type systems (e.g. a type system with ML polymorphism), in which a fixed-point combinator may no longer be encoded. Note that our main interests are in extending Kobayashi and Ong's type-based method [3, 5] for model-checking simply-typed recursion schemes to handle recursion schemes *with various advanced type systems*, and that we study infinite intersection type systems for untyped recursion schemes to establish common foundations.

3 Infinite Intersection Sorts

The goal of this section is to characterize the class of recursion schemes whose value trees are well-ranked. For this purpose, we construct an intersection type system in which a recursion scheme is well-typed if, and only if, the value trees of recursion schemes are well-ranked. In the following, we call this type system the *sort system* to avoid confusion with the type system for model checking introduced in Section 4.

In the sort system, intersection types are infinite both in width (i.e. we allow $\bigwedge_{i<\omega} \kappa_i$) and in depth (i.e. we allow sorts having infinite paths, like $o \to o \to o \to \cdots$). Note that such infinite intersection types are necessary for the complete characterization of recursion schemes that generate well-ranked trees: See Remark 2.

As defined below, a *sort* is a (possibly infinite) tree labeled by o, $\to$, and $\wedge$. The sort o (i.e. a tree consisting of a single node labelled by o) describes trees. The other constructors $\to$ and $\wedge$ describe functions and intersections as usual; for example, $(\mathsf{o} \wedge (\mathsf{o} \to \mathsf{o})) \to \mathsf{o}$ describes a function that takes as input a term that can be both used as a tree and a tree function, and returns a tree.

Definition 1 (sorts). *Let $K = \{(\mathsf{o}, 0), (\to, 2)\} \cup \{(\wedge^{\alpha}, \alpha) \mid \alpha \leq \omega\}$ be a ranked alphabet. A sort κ is a well-ranked K-labeled tree that satisfies: (i) $\kappa(\varepsilon) \in \{\mathsf{o}, \to\}$;*

[2] Ong [2] proved the decidability for a more general case, where $\mathcal{A}$ is an alternating parity tree automaton.

(ii) If $\kappa(p) = \rightarrow$, then $\kappa(p1) = \wedge^\alpha$ and $\kappa(p2) \in \{\mathsf{o}, \rightarrow\}$; and (iii) If $\kappa(p) = \wedge^\alpha$, then $\kappa(pi) \in \{\mathsf{o}, \rightarrow\}$ for every i such that $pi \in dom(\kappa)$.

We often omit the superscript and simply write $\wedge$ for $\wedge^\alpha$. When κ_i's ($i < \alpha$) are sorts, we write $\bigwedge_{i<\alpha} \kappa_i$ for the tree whose root is labelled by $\wedge^\alpha$ and whose children are κ_i's. We write $\top$ for the empty intersection $\bigwedge_{i<0} \kappa_i$. Similarly, we write $\bigwedge_{i<\alpha} \kappa_i \rightarrow \kappa$ for the sort whose root is labelled by $\rightarrow$, and whose children are $\bigwedge_{i<\alpha} \kappa_i$ and κ. When $\alpha = 1$, we just write $\kappa_0 \rightarrow \kappa$ for $\bigwedge_{i<\alpha} \kappa_i \rightarrow \kappa$ (e.g. $(\mathsf{o} \rightarrow \mathsf{o}) \rightarrow \mathsf{o}$ for $\bigwedge_{i<1} \kappa_i \rightarrow \mathsf{o}$ where $\kappa_0 = \mathsf{o} \rightarrow \mathsf{o}$). We give a higher precedence to $\wedge$ than to $\rightarrow$.

The three conditions in the definition above are imposed just for a technical convenience (more specifically, for removing the introduction and elimination rules for intersection types). Note that, for example, $(\kappa_1 \rightarrow \kappa_2) \rightarrow \kappa_3$ (which is prohibited by the restriction (ii)) can be represented as $\bigwedge_{i<1} \kappa'_i \rightarrow \kappa_3$, where $\kappa'_0 = \kappa_1 \rightarrow \kappa_2$. A similar restriction on the syntax of types has been used for finite intersection type systems [6].

A type environment, denoted by Γ, is a (possibly infinite) set of bindings of the form $x{:}\tau$ (where non-terminals of recursion schemes are treated as variables). We often omit the curly brackets, and simply write $x_1 : \kappa_1, \ldots, x_n : \kappa_n$ for $\{x_1 : \kappa_1, \ldots, x_n : \kappa_n\}$. Note that we allow multiple bindings for the same variable, as in $\{x : \kappa_1, x : \kappa_2\}$. We abbreviate $\{x : \kappa_i \mid i < \alpha\}$ as $x : \bigwedge_{i<\alpha} \kappa_i$. We write $dom(\Gamma)$ for the set $\{x \mid \exists\tau.(x : \tau \in \Gamma)\}$.

We fix a finitely ranked alphabet Σ below. The typing rules for λ-terms are given as follows:

$$\frac{}{\Gamma, x : \kappa \vdash x : \kappa} \qquad \frac{\Gamma \vdash t_1 : \bigwedge_{i<\alpha} \kappa_i \rightarrow \kappa \quad \Gamma \vdash t_2 : \kappa_i (\text{for every } i < \alpha)}{\Gamma \vdash t_1 t_2 : \kappa}$$

$$\frac{}{\Gamma \vdash a : \underbrace{\mathsf{o} \rightarrow \cdots \rightarrow \mathsf{o}}_{\Sigma(a)} \rightarrow \mathsf{o}}$$

$$\frac{}{\Gamma \vdash \lambda x.t : o} \qquad \frac{\Gamma \cup \{x : \kappa_i \mid i < \alpha\} \vdash t : \kappa \quad x \notin dom(\Gamma)}{\Gamma \vdash \lambda x.t : \bigwedge_{i<\alpha} \kappa_i \rightarrow \kappa}$$

The rules above are standard, except the rule on the left bottom. It is due to our definition of the value tree of a recursion scheme. To see why, consider the recursion scheme $\mathcal{G}$, consisting of the two rules $S \rightarrow F$ and $F\,x \rightarrow \mathtt{a}$. The rewriting of S gets stuck as $S \rightarrow F$, so that the value tree of $\mathcal{G}$ is $\perp$. Since $\perp$ is a well-ranked tree, F (which is essentially $\lambda x.\mathtt{a}$) should be assigned type o.

A recursion scheme $\mathcal{G} = (\mathcal{N}, \Sigma, \mathcal{R}, S)$ is well-typed under Γ, written $\vdash \mathcal{G} : \Gamma$, if $dom(\Gamma) \subseteq \mathcal{N}$, $\Gamma \vdash \mathcal{R}(F) : \tau$ holds for every $F : \tau \in \Gamma$, and $S : \mathsf{o} \in \Gamma$. We write $\vdash \mathcal{G}$ if there exists Γ such that $\vdash \mathcal{G} : \Gamma$.

The following theorem states soundness and completeness of the sort system.

Theorem 1. *For any recursion scheme $\mathcal{G}$, $[\![\mathcal{G}]\!]$ is well-ranked if and only if $\vdash \mathcal{G}$.*

Proof. A special case of Theorems 2 and Theorem 3 in Section 4, where the automaton $\mathcal{A}$ is $(\{o\}, \Sigma, o, \Delta)$ such that $(o, a, \overbrace{o, \ldots, o}^{n}) \in \Delta$ for every $a \in \Sigma$ of rank n. □

Remark 2. As already mentioned, intersection sorts in our type system may be infinite both in width and depth, and non-regular (i.e. may not be expressed by finite recursive types). The following observations explain why simpler intersection sorts are insufficient. First, intersection types that are infinite in width but *finite in depth* guarantee that λ-terms are strongly normalizing [7]; thus the Y combinator (which can be defined by $Y\, f \to (A\, f)(A\, f)$ and $A\, f\, x \to f\,(x\, x)$) would not be typable. Secondly, the restriction of intersection sorts to finite recursive types (i.e. sorts that are finite in width and infinite but regular in depth) is also insufficient. This is because under this restriction, the typability would be recursively enumerable,[3] but the well-rankedness of the tree generated by an untyped recursion scheme is not. The latter can be easily proved, for example, by a reduction from the halting problem of a Minsky machine [8]. A Minsky machine consists of two counters holding natural numbers, and has instructions for counter increment/decrement, conditional/unconditional jumps, and halting [8]. We can use the standard Church encoding for natural numbers and booleans to simulate those instructions, and replace the halt command with a term generating an ill-ranked tree. The resulting untyped recursion scheme generates a well-ranked tree if and only if the Minsky machine does not halt.

4 Type System for Model Checking Untyped Recursion Schemes

In this section, we extend Kobayashi's type system [3] (for model-checking recursion schemes wrt safety properties) to deal with untyped recursion schemes.

Let $\mathcal{A} = (Q, \Sigma, q_S, \Delta)$ be a trivial automaton. We shall extend the sort system of the previous section by refining the sort o into a type of the form $\bigwedge\{q_1, \ldots, q_k\}$ where $q_1, \ldots, q_k \in Q$. Intuitively, a state q of the automaton is regarded as a type that describes the trees accepted by $\mathcal{A}$ from the state q, i.e., t has type q if and only if $[\![t]\!] \in \mathcal{L}_{\mathcal{A}}(q)$.

Definition 2 (Types). *Let $T = \{(q, 0) \mid q \in Q\} \cup \{(\to, 2)\} \cup \{(\wedge^{\alpha}, \alpha) \mid \alpha \leq \omega\}$ be the set of ranked alphabets. A type τ is a well-ranked T-labeled tree that satisfies the following conditions: (i) $\tau(\varepsilon) \neq \wedge$ (ii) If $\tau(p) = \to$, then $\tau(p1) = \wedge$, $\tau(p2) \neq \wedge$, and (iii) If $\tau(p) = \wedge$, then $\tau(pi) \neq \wedge$ for every $i \in \omega - \{0\}$.*

The typing rules are almost the same as the sorting rules, except that the type of a terminal is determined by the transition function of the automaton.

$$\frac{}{\Gamma, x : \tau \vdash_{\mathcal{A}} x : \tau} \qquad \frac{(q, a, q_1, \ldots, q_n) \in \Delta}{\Gamma \vdash_{\mathcal{A}} a : q_1 \to \cdots \to q_n \to q} \qquad \frac{}{\Gamma \vdash_{\mathcal{A}} \lambda x.t : q}$$

[3] Note that since the set of finite recursive sorts is recursively enumerable, the set of valid type judgements is also recursively enumerable.

$$\frac{\Gamma \vdash_{\mathcal{A}} t_1 : \bigwedge_{i<\alpha} \tau_i \to \tau \qquad \Gamma \vdash_{\mathcal{A}} t_2 : \tau_i \text{ for all } i < \alpha}{\Gamma \vdash_{\mathcal{A}} t_1\ t_2 : \tau} \qquad \frac{\Gamma \cup \{x : \tau_i \mid i < \alpha\} \vdash_{\mathcal{A}} t : \tau \qquad x \notin dom(\Gamma)}{\Gamma \vdash_{\mathcal{A}} \lambda x.t : \bigwedge_{i<\alpha} \tau_i \to \tau}$$

We write $\vdash_{\mathcal{A}} \mathcal{G} : \Gamma$ if (i) $dom(\Gamma) \subseteq \mathcal{N}$, (ii) $\Gamma \vdash_{\mathcal{A}} \mathcal{R}(F) : \tau$ holds for every $F : \tau \in \Gamma$, and (iii) $S : q_S \in \Gamma$.

4.1 Soundness of the Type System

We use the following lemma to establish the soundness of our type system.

Lemma 1 (Subject Reduction). *Suppose $\vdash_{\mathcal{A}} \mathcal{G} : \Gamma$. If $\Gamma \vdash_{\mathcal{A}} t : \tau$ and $t \to_{\mathcal{G}}^* t'$, then $\Gamma \vdash_{\mathcal{A}} t' : \tau$.*

Theorem 2 (Soundness). *Let $\mathcal{G}$ be a recursion scheme and $\mathcal{A}$ be a trivial automaton. If $\vdash_{\mathcal{A}} \mathcal{G}$, then $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{A}$.*

Proof Sketch. It suffice to show that if $S \to_{\mathcal{G}}^* t$, then $t^\perp$ is accepted by $\mathcal{A}$. Assume $S \to_{\mathcal{G}}^* t$. By the definition of $\vdash_{\mathcal{A}} \mathcal{G}$, there is a type environment such that $\vdash_{\mathcal{A}} \mathcal{G} : \Gamma$. By Lemma 1, we get $\Gamma \vdash_{\mathcal{A}} t : q_S$, from which $t^\perp \in \mathcal{L}_{\mathcal{A}}$ follows. □

4.2 Completeness of the Type System

We show that our type system is complete in the sense that a recursion scheme is well-typed if its value tree is accepted by $\mathcal{A}$. The overall idea of the proof is similar to the completeness proof of Kobayashi and Ong's type system for the modal μ-calculus model checking of typed recursion schemes [5]; Type information is extracted from a reduction sequence of the recursion scheme, by observing how each non-terminal is used in the reduction. Some non-trivial adjustments are necessary, however, since sorts (which are called kinds in [5]) are infinite (thus, we cannot use induction on sorts unlike in [5]).

Theorem 3 (Completeness). *Let $\mathcal{G}$ be a recursion scheme and $\mathcal{A}$ be a trivial automaton. If $[\![\mathcal{G}]\!]$ is well-ranked and accepted by $\mathcal{A}$, then $\vdash_{\mathcal{A}} \mathcal{G}$.*

We fix below a recursion scheme $\mathcal{G}$ and a trivial automaton $\mathcal{A}$. By the assumption that $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{A}$, there is a run tree of $\mathcal{A}$ over $[\![\mathcal{G}]\!]$ whose root is labeled by q_S. We fix such a run tree r below.

We shall define a *reduction tree* $\mathcal{T}^\infty$, which expresses the process of the value tree of $\mathcal{G}$ being constructed in a reduction sequence. In the reduction tree, each term is annotated with a label to keep track of the origin of the term. The set Term^L of *annotated terms*, ranged over by u, is given by:

$$u ::= x^l \mid a^l \mid F^l \mid (u_1 u_2)^l,$$

where $l \subseteq \omega^* \times \omega$. Intuitively, u^l with $(p, n) \in l$ means that the term u has occurred in the node p of $\mathcal{T}^\infty$, in the form $c \cdots u\, u_{n-1} \cdots u_1$ (where c is a non-terminal or a terminal). We often write u^l for u when the outermost label is l.

We define $(u^l)^{+(p,i)}$ as $u^{l\cup(p,i)}$. We write $\flat(u)$ for the term obtained by removing all the labels from u. We omit some of the labels of an annotated term when they are not important. The substitution of annotated terms is defined as a homomorphism satisfying $[v^l/x](x^{l'}) = v^{l\cup l'}$.

The *expansion relation* $\triangleright$ on (finite and unranked) $(\omega^* \times \mathrm{Term}^L)$-labelled trees is the least relation that satisfies the following condition.

If T is a $(\omega^* \times \mathrm{Term}^L)$-labelled tree with $T(p) = (p', c\, u_n \dots u_1)$, where $c \in \Sigma \cup \mathcal{N}$, and there is no child of p in T, then:

1. If $c = F^l$ and $\mathcal{R}(F) = \lambda x_n \dots x_k.s$ and $k \geq 1$, then

$$T \triangleright T \cup \{(p1, (p', ([u_n^{+(p,n)} \dots u_k^{+(p,k)}/x_n \dots x_k]s)u_{k-1}^{+(p,k-1)} \dots u_1^{+(p,1)}))\}$$

2. If $c = a^l$ (which implies $n = \Sigma(a)$), then:

$$T \triangleright T \cup \{(pi, (p'i, u_{n-i+1}^{+(p,n-i+1)}))) \mid 1 \leq i \leq n\}$$

In a label of the form (p', u), the annotated term u represents the term being reduced, and p' represents the corresponding position in the value tree of $\mathcal{G}$. In other words, the subtree of $[\![\mathcal{G}]\!]$ at position p' will be generated by reducing u.

The *reduction tree* $\mathcal{T}^\infty$ is the $(\omega^* \times \mathrm{Term}^L)$-labelled tree obtained by an infinite expansion of $(\epsilon, S^{\{\}})$, i.e. $\mathcal{T}^\infty = \bigcup\{T \mid \{\ (\epsilon, S^{\{\}})\ \} \triangleright^* T\}$.

It is easy to see that $\mathcal{T}^\infty$ is well-defined. Note that $V = \{(p', a) \mid \exists p \in \omega^*, \mathcal{T}^\infty(p) = (p', a\, u_n \dots u_1))\}$ is essentially equal to the value tree $[\![\mathcal{G}]\!]$. The only difference is that if $[\![\mathcal{G}\ ]\!]\ (p') = \bot$, then $V(p')$ is undefined.

Example 1. Let $\mathcal{G}_0 = (\Sigma, \{S, F\}, \mathcal{R}, S)$, where $\Sigma = \{(\mathtt{br}, 2), (\mathtt{a}, 1), (\mathtt{b}, 1), (\mathtt{c}, 0)\}$, $\mathcal{R} = \{S \to F\ \mathtt{c},\ F\ x \to \mathtt{br}\ \mathtt{c}\ (\mathtt{a}(F(\mathtt{b}(x))))\}$. The reduction tree $\mathcal{T}^\infty$ is shown in Figure 4.2. We have $\mathcal{T}^\infty(1121) = (21, \cdots \mathtt{c}^{(1,1)} \cdots)$, which means this occurrence $\mathtt{c}$ appears at 1 as the 1st argument (counting from right to left). The node at 112 is of the form $(2, \mathtt{a}\, u)$, which means $[\![\mathcal{G}_0\ ]\!]\ (2) = \mathtt{a}$. □

Lemma 2. *Let $u_1^{l_1}$ be an annotated term occurring in $\mathcal{T}^\infty$, i.e., there exists a path p_1 such that $\mathcal{T}^\infty(p_1) = (p_1', u)$ and $u_1^{l_1}$ is a subterm of u. If $(p, n) \in l_1$ then $\mathcal{T}^\infty(p) = (p', v\, v_n^l \dots v_1)$ with $u_1^{l_1} = v_n^{l\cup l'}$ for some l'.*

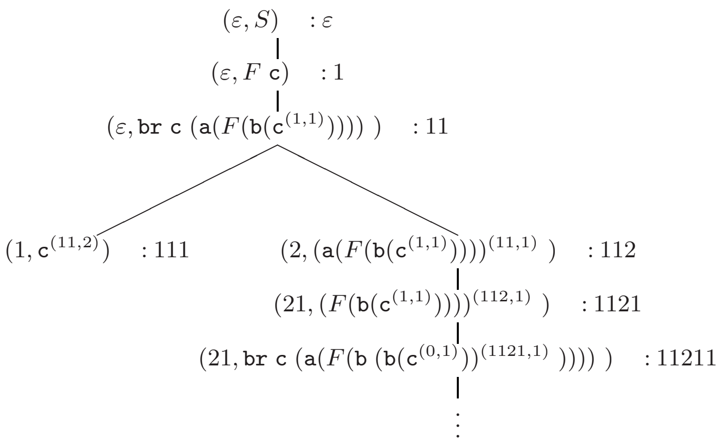
We define $\theta_{(\wedge,p,i)}$ and $\theta_{(\to,p,i)}$ as the (possibly infinite) trees that satisfy the following equations.

$$\theta_{(\wedge,p,i)} = \bigwedge\{\theta_{(\to,p_0,n)} \mid \mathcal{T}^\infty(p_0) = (p_0', u^l\, u_n \dots u_1) \text{ and } (p, i) \in l\}$$
$$\theta_{(\to,p,i)} = \theta_{(\wedge,p,i)} \to \cdots \to \theta_{(\wedge,p,1)} \to r(p')$$
$$\text{where } \mathcal{T}^\infty(p) = (p', u) \text{ and } r \text{ is the run tree.}$$

In the first equation, we assume that $\theta_{(\to,p_0,i)}$'s are ordered according to a certain linear order among elements of $(\omega - \{0\})^*$. Thus, $\theta_{(\wedge,p,i)}$ and $\theta_{(\to,p,i)}$ are uniquely determined, and every $\theta_{(\to,p,i)}$ is a type.

Let $\Gamma_{\mathcal{G}}^r$ be $\{F : \theta_{(\to,p,n)} \mid \mathcal{T}^\infty(p) = (p', F^l\, u_n \dots u_1)\}$. We show that $\Gamma_{\mathcal{G}}^r$ is the witness of well-typing of $\mathcal{G}$, i.e. $\vdash \mathcal{G} : \Gamma_{\mathcal{G}}^r$.

We first show that each term occurring in $\mathcal{T}^\infty$ is well-typed under $\Gamma_{\mathcal{G}}^r$.



$(\varepsilon, S) \quad : \varepsilon$
$(\varepsilon, F\ \mathtt{c}) \quad : 1$
$(\varepsilon, \mathtt{br}\ \mathtt{c}\ (\mathtt{a}(F(\mathtt{b}(\mathtt{c}^{(1,1)})))) \ ) \quad : 11$
$(1, \mathtt{c}^{(11,2)}) \quad : 111$
$(2, (\mathtt{a}(F(\mathtt{b}(\mathtt{c}^{(1,1)}))))^{(11,1)} \ ) \quad : 112$
$(21, (F(\mathtt{b}(\mathtt{c}^{(1,1)})))^{(112,1)} \ ) \quad : 1121$
$(21, \mathtt{br}\ \mathtt{c}\ (\mathtt{a}(F(\mathtt{b}\ (\mathtt{b}(\mathtt{c}^{(0,1)}))^{(1121,1)} \ )))) \ ) \quad : 11211$

Fig. 1. $\mathcal{T}^\infty$ for $\mathcal{G}_0$. We omit the empty annotation. Here p of $(p_0, u) : p$ is the path of the node (p_0, u), hence they are not a part of the nodes.

Lemma 3. *If* $\mathcal{T}^\infty(p) = (p', uu_n \dots u_1)$*, then* $\Gamma^r_{\mathcal{G}} \vdash \flat(u) : \theta_{(\rightarrow,p,n)}$.

The following lemma is a kind of the inverse of the substitution lemma; we derive a typing for t from that of $[s_1^{l_1} \dots s_k^{l_k}/x_1 \dots x_{l_k}]t$. Recall that $\Gamma, x : \bigwedge_{i<\alpha} \tau_i \vdash_{\mathcal{A}} t : \tau$ is the abbreviation of $\Gamma, x : \tau_0, x : \tau_1, \dots \vdash_{\mathcal{A}} t : \tau$. We define $\Gamma \vdash_{\mathcal{A}} t : \bigwedge_{i<\alpha} \tau_i$ as the abbreviation of $\Gamma \vdash_{\mathcal{A}} t : \tau_i$ for all $i < \alpha$.

Lemma 4. *Suppose that* $\mathcal{T}^\infty(p) = (p', u\, u_n \dots u_1)$ *and* $u = [s_1^{l_1} \dots s_k^{l_k}/x_1 \dots x_k]v$*, with* $(p_i, j_i) \in l_i$ *for each* $i \in \{1, \dots, k\}$*. Let* Γ *be* $x_1 : \theta_{(\wedge,p_1,j_1)}, \dots, x_k : \theta_{(\wedge,p_k,j_k)}$*. Then* $\Gamma^r_{\mathcal{G}}, \Gamma \vdash_{\mathcal{A}} \flat(v) : \theta_{(\rightarrow,p,n)}$.

Proof. By induction on the structure of v.

– Case where v does not contain x_i for any i: Immediate from Lemma 3.

– Case $v = x_i$: Note that $\mathcal{T}^\infty(p) = (p', s_i^{l_i}\, u_1 \dots u_n)$. By the assumption $(p_i, j_i) \in l_i$ and the definition of $\theta_{(\wedge,p_i,j_i)}$, we have $x : \theta_{(\rightarrow,p,n)} \in \Gamma$. So, by using (T-VAR), we obtain $\Gamma^r_{\mathcal{G}}, \Gamma \vdash x_i : \theta_{(\rightarrow,p,n)}$ as required.

– Case $v = v_{00}v_{01}$: Let $u_{0j} = [s_1 \dots s_k/x_1 \dots x_k]v_j$ for $j = 0, 1$ and l be the outermost label of v_{01}, i.e., v_{01}^l. Note that $u = u_{00}u_{01}$ and $\mathcal{T}^\infty(p) = (p', u_{00}^{l_{00}} u_{01}^{l_{01}} u_n \dots u_1)$. By the induction hypothesis, $\Gamma^r_{\mathcal{G}}, \Gamma \vdash_{\mathcal{A}} \flat(v_{00}) : \theta_{(\rightarrow,p,n+1)}$. Because $\theta_{(\rightarrow,p,n+1)} = \theta_{(\wedge,p,n+1)} \rightarrow \theta_{(\rightarrow,p,n)}$, what we should show is $\Gamma^r_{\mathcal{G}}, \Gamma \vdash_{\mathcal{A}} \flat(v_{01}) : \theta_{(\wedge,p,n+1)}$. Let p_0 be any path such that $\mathcal{T}^\infty(p_0) = (p_0', v'^{l'}\, v_m', \dots, v_1')$ and $(p, n+1) \in l'$. By Lemma 2, we obtain $v'^{l'} = u_{01}^{l_{01} \cup l_{01}'}$. Therefore $v'^{l'} = [s_1 \dots s_k/x_1 \dots x_k](v_{01}^{l \cup l_{01}'})$. By using induction hypothesis, we have $\Gamma^r_{\mathcal{G}}, \Gamma \vdash_{\mathcal{A}} \flat(v_{01}) : \theta_{(\rightarrow,p_0,m)}$. So by the definition of $\theta_{(\wedge,p,n+1)}$, we obtain $\Gamma^r_{\mathcal{G}}, \Gamma \vdash_{\mathcal{A}} \flat(v_{00}) : \theta_{(\wedge,p,n+1)}$ as required. $\square$

We are now ready to prove the completeness of the type system.

Proof of Theorem 3. It is easy to see that $S : q_S \in \Gamma^r_{\mathcal{G}}$. Thus, it remains to show that $\Gamma^r_{\mathcal{G}} \vdash \mathcal{R}(F) : \tau$ holds for each $F : \tau \in \Gamma^r_{\mathcal{G}}$. By the construction of $\Gamma^r_{\mathcal{G}}$, there is a path p that satisfies $\mathcal{T}^\infty(p) = (p', F^l u_n \dots u_1)$ and $\tau = \theta_{(\wedge,p,n)} \to \cdots \to \theta_{(\wedge,p,1)} \to r(p')$. Suppose $F : \tau \in \Gamma^r_{\mathcal{G}}$ and $\mathcal{R}(F) = \lambda x_n \lambda x_{n-1} \dots \lambda x_k.s$ (here, k may be a negative integer).

– Case $k \leq 0$: By using (T-Bot), we obtain $\Gamma^r_{\mathcal{G}} \cup \{x_i : \theta_{(\wedge,p,i)} \mid 1 \leq i \leq n\} \vdash \lambda x_0 \lambda x_{-1} \dots \lambda x_k.s : r(p')$. By using (T-Abs), we obtain $\Gamma^r_{\mathcal{G}} \vdash \lambda x_n \dots \lambda x_k : \tau$.

– Case $k > 0$: By the definition of $\mathcal{T}^\infty$, we have:

$$\mathcal{T}^\infty(p1) = (p', ([u_n^{+(p,n)} \dots u_k^{+(p,k)}/x_n \dots x_k]s) u_{k-1}^{(p,k-1)} \dots u_1^{(p,1)}).$$

By Lemma 4, $\Gamma^r_{\mathcal{G}}, \{x_i : \theta_{(\wedge,p,i)} \mid k \leq i \leq n\} \vdash s : \theta_{(\wedge,p,k-1)} \to \cdots \to \theta_{(\wedge,p,1)} \to r(p')$. By using (T-Abs), we obtain $\Gamma^r_{\mathcal{G}} \vdash \lambda x_n \dots \lambda x_k.s : \theta_{(\wedge,p,n)} \to \cdots \to \theta_{(\wedge,p,1)} \to r(p')$ as required. □

5 Theory of Refinement

The purpose of this section is to develop a theory for applying our type system to verification of recursion schemes that model programs written in *typed* programming languages. The soundness and completeness theorems in the previous section imply that our type system for untyped recursion schemes is undecidable in general (as the model-checking problem is undecidable). Note, however, that a recursion scheme obtained from a typed program is well-typed under a certain type system. Thus, we are interested in the model checking problem: "Given a recursion scheme $\mathcal{G}$ *that is well-typed in a type system* T, is the value tree $\mathcal{G}$ accepted by $\mathcal{A}$?" Note that the type system T may ensure, in addition to the well-rankedness, certain properties of the value tree, like "every child of the node labelled by *cons* is *true* or *false*" (e.g. when the source program has type **bool list**). Thus, the type system T can be regarded as a restricted form of the intersection type system $\mathcal{T}_{\mathcal{A}'}$ for another automaton $\mathcal{A}'$. Therefore, the above model checking problem is refined to:

> Given a recursion scheme $\mathcal{G}$ *well-typed in a certain restriction of* $\mathcal{T}_{\mathcal{A}'}$, is the value tree $\mathcal{G}$ accepted by $\mathcal{A}$ (or equivalently, is $\mathcal{G}$ well-typed in $\mathcal{T}_{\mathcal{A}}$)?

Because of the assumption that $\mathcal{G}$ is well-typed, the model checking problem can be solved in certain cases. For example, Kobayashi and Ong's work [3, 5] can be considered as studies of a special case of the problem above, where $\mathcal{A}'$ is the Büchi automaton with a single state $\mathtt{o}$, and sorts are restricted to simple and finite ones (without intersections). The main result of this section (Theorem 5) is that if $\mathcal{G}$ is well-typed in $\mathcal{T}_{\mathcal{A}'}$, then $\mathcal{G}$ is well-typed in $\mathcal{T}_{\mathcal{A}}$ if, and only if, $\mathcal{G}$ is so under certain restricted type environments of $\mathcal{T}_{\mathcal{A}}$, so that the model checking problem can sometimes be solved effectively. Below we focus on the case where there is an *automata homomorphism* from $\mathcal{A}$ to $\mathcal{A}'$.

Definition 3. *Let $\mathcal{A}_1 = (Q^1, \Sigma, q^1_S, \Delta^1)$ and $\mathcal{A}_2 = (Q^2, \Sigma, q^2_S, \Delta^2)$ be trivial automata. A homomorphism $f : \mathcal{A}_1 \to \mathcal{A}_2$ of automata is a map from the states of $\mathcal{A}_1$ to the states of $\mathcal{A}_2$ satisfying the following conditions: (i) f maps the initial state to the initial state, i.e. $f(q^1_S) = q^2_S$. (ii) f respects the transitions, i.e. $(q, a, q_1, \dots, q_n) \in \Delta^1$ implies $(f(q), a, f(q_1), \dots, f(q_n)) \in \Delta^2$.*

For a given homomorphism $f : \mathcal{A}_1 \to \mathcal{A}_2$ of automata, we extend this function to a map $\hat{f}$ from types of $\mathcal{T}_{\mathcal{A}_1}$ to those of $\mathcal{T}_{\mathcal{A}_2}$, by:

$$\hat{f}(\tau)(p) = \begin{cases} f(\tau(p)) & (\text{if } \tau(p) \in Q^1) \\ \tau(p) & (\text{if } \tau(p) = \to \text{ or } \wedge) \end{cases}$$

and we define $\hat{f}(\Gamma)$ by $\hat{f}(\{F_1 : \tau_1, \dots\}) = \{F_1 : \hat{f}(\tau_1), \dots\}$. Then we can see that any homomorphism f preserves typability.

Theorem 4. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be automata, $\mathcal{G}$ be a recursion scheme and $f : \mathcal{A}_1 \to \mathcal{A}_2$ be a homomorphism of automata. If $\Gamma \vdash_{\mathcal{A}_1} \mathcal{G}$, then $f(\Gamma) \vdash_{\mathcal{A}_2} \mathcal{G}$.*

Note that the sort system given in Section 3 is the same as $\mathcal{T}_{\mathcal{A}^\top}$, where $\mathcal{A}^\top$ is the trivial automaton $(\{o\}, \Sigma, o, \Delta)$ such that $(o, a, \overbrace{o, \dots, o}^{n}) \in \Delta$ for every $a \in \Sigma$ of rank n. Moreover, for any trivial automaton $\mathcal{A}$, $f : \mathcal{A} \to \mathcal{A}^\top : q \mapsto o$ is a (unique) homomorphism.

The main goal of this section is to establish a "backward" property. Given a homomorphism $f : \mathcal{A}_1 \to \mathcal{A}_2$, we want to derive a property about the typability of $\mathcal{G}$ in $\mathcal{T}_{\mathcal{A}_1}$, assuming that $\Gamma \vdash_{\mathcal{A}_2} \mathcal{G}$. For that purpose, we define a *refinement relation* $\sqsubseteq_f$ between types of $\mathcal{A}_1$ and those of $\mathcal{A}_2$.

Definition 4. *The binary relation $\sqsubseteq$ on types is the largest relation that satisfies the following conditions: (i) If $\tau \sqsubseteq q$, then $\tau = q$. (ii) If $\tau \sqsubseteq \bigwedge_{j<\beta} \sigma_j \to \sigma'$, then $\tau = \bigwedge_{i<\alpha} \tau_i \to \tau'$, with $\tau' \sqsubseteq \sigma'$ and $\forall i < \alpha . \exists j < \beta . \tau_i \sqsubseteq \sigma_j$. Given a homomorphism $f : \mathcal{A}_1 \to \mathcal{A}_2$, we define the refinement relation $\tau_1 \sqsubseteq_f \tau_2$ by $f(\tau_1) \sqsubseteq \tau_2$.*

The refinement relation can be considered as a generalization of the kinding relation $\tau :: \kappa$ (which means "a type τ has sort κ") used in Kobayashi and Ong's type systems [3, 5]. Note that the relation $\sqsubseteq$ is *not* a subtyping relation; the type constructor $\to$ is *covariant* wrt $\sqsubseteq$.

Example 2. $\sqsubseteq$ is reflexive and transitive. If $q_1 \neq q_2$, $q_1 \to q \sqsubseteq (q_1 \wedge q_2) \to q$ holds but $(q_1 \wedge q_2) \to q \sqsubseteq q_1 \to q$ does not. If $f(q_1) = f(q_2) = q$, then $(q_1 \wedge q_2) \to q_1 \sqsubseteq_f q \to q$. □

We write $\Gamma_1 \sqsubseteq \Gamma_2$ if for each $F : \tau \in \Gamma_1$, there exists $F : \sigma \in \Gamma_2$ such that $\tau \sqsubseteq \sigma$. The definition of $\Gamma_1 \sqsubseteq_f \Gamma_2$ is similar. The following is a key lemma to obtain the main result.

Lemma 5. *Let $\mathcal{A}$ be a* deterministic *trivial automaton and $\mathcal{G}$ be a recursion scheme. If $\mathcal{G}$ is typable in $\mathcal{T}_{\mathcal{A}}$, then $\Gamma^r_{\mathcal{G}}$ (where r is a run tree of $\mathcal{A}$ over $[\![\mathcal{G}]\!]$, which is unique by the assumption that $\mathcal{A}$ is deterministic) is the minimum type environment in $\{\Gamma \mid \vdash_{\mathcal{A}} \mathcal{G} : \Gamma\}$ with respect to $\sqsubseteq$.*

Proof Sketch. Assume $\vdash_{\mathcal{A}} \mathcal{G} : \Gamma$. By the proof of Lemma 1 (which is constructive in the sense that it gives a procedure to construct a derivation tree for t' from that of t), we can construct a type derivation for each term occurring in $\mathcal{T}^\infty$ (recall that $\mathcal{T}^\infty$ is a tree representing an infinite reduction sequence). For $\mathcal{T}^\infty(p) = (p', uv_n \dots v_1))$, let $\phi_{(p,n)}$ be the type assigned to the term u. Then, we can prove by co-induction that $\theta_{(\rightarrow,p,n)} \sqsubseteq \phi_{(p,n)}$ holds for every p and n. Let $\mathcal{T}^{-1}(F) = \{(p,n) \mid \mathcal{T}^\infty(p) = (p', Fu_n \dots u_1)\}$. Thus, we have:

$$\Gamma^r_{\mathcal{G}} = \{F : \theta_{(\rightarrow,p,n)} \mid (p,n) \in \mathcal{T}^{-1}(F)\} \sqsubseteq \{F : \phi_{(p,n)} \mid (p,n) \in \mathcal{T}^{-1}(F)\} \subseteq \Gamma$$

as required. □

The following main result implies that if $\Gamma \vdash_{\mathcal{A}_2} \mathcal{G}$, then it suffices to consider only refinements of Γ to check whether $\mathcal{G}$ is well-typed in $\mathcal{T}_{\mathcal{A}_1}$.

Theorem 5. *Let $\mathcal{A}_1$ be a trivial automaton, $\mathcal{A}_2$ be a deterministic trivial automaton, $\mathcal{G}$ be a recursion scheme and $f : \mathcal{A}_1 \rightarrow \mathcal{A}_2$ be a homomorphism of automata. Assume $\Gamma \vdash_{\mathcal{A}_2} \mathcal{G}$. Then, $\mathcal{G}$ is typable in $\mathcal{T}_{\mathcal{A}_1}$ iff $\mathcal{G}$ is so under some type environment Γ' such that $\Gamma' \sqsubseteq_f \Gamma$.*

Proof. The "if" direction is trivial. To prove the converse, assume $\mathcal{G}$ is well-typed in $\mathcal{T}_{\mathcal{A}_1}$. By the soundness of the type system, $[\![\mathcal{G}]\!] \in \mathcal{L}_{\mathcal{A}_1}$ holds, i.e. there exists a run tree r_1 of $\mathcal{A}_1$ over $[\![\mathcal{G}]\!]$. Let r_2 be a unique run tree of $\mathcal{A}_2$ over $[\![\mathcal{G}]\!]$. It is easy to show that $r_2(p) = f(r_1(p))$. Then, by the definition of $\Gamma^{r_1}_{\mathcal{G}}$ and $\Gamma^{r_2}_{\mathcal{G}}$, $f(\Gamma^{r_1}_{\mathcal{G}}) \sqsubseteq \Gamma^{r_2}_{\mathcal{G}}$. By Lemma 5, for any type environment Γ such that $\Gamma \vdash_{\mathcal{A}_2} \mathcal{G}$, we have $f(\Gamma^{r_1}_{\mathcal{G}}) \sqsubseteq \Gamma^{r_2}_{\mathcal{G}} \sqsubseteq \Gamma$, which implies $\Gamma^{r_1}_{\mathcal{G}} \sqsubseteq_f \Gamma$. □

As the sort system is equivalent to the type system for the automaton $\mathcal{A}^\top$ and there are only finitely many refinements of a finite sort, we obtain:

Corollary 1. *If $\mathcal{G}$ is a finitely sorted recursion scheme (i.e. the sort of every non-terminal in $\mathcal{G}$ is finite), then it is decidable whether $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{A}$.*

We can obtain a model checking algorithm by modifying Kobayashi's algorithm [9] for simply-typed recursion schemes.

6 Applications

We now discuss how the foregoing theory can be applied to verification of functional programs written in languages with advanced type systems (like polymorphism and recursive types). As shown in [3, 4], a higher-order functional program can be easily transformed into a recursion scheme that simulates the output or the event sequences of the source program, and then model-checked. Since the recursion scheme thus obtained is well-sorted under a similar type system, we focus here on model-checking of recursion schemes that are well-sorted under advanced type systems.

As already mentioned, polymorphic types and recursive types may be regarded as restricted forms of infinite intersection types (or sorts). For example, a (predicative) polymorphic type $\forall\alpha.\tau$ can be regarded as $\bigwedge\{[\sigma/\alpha]\tau \mid \sigma \text{ is a } \textit{finite} \text{ sort}\}$

with infinite width, and a recursive sort $\mu X.o \to X$ as a sort $o \to o \to \dots$ with an infinite path. Thus, the model-checking problem of interest is: "Given a recursion scheme $\mathcal{G}$ well-sorted under a certain fragment $\mathcal{S}$ of the sort system of Section 3 and an automaton $\mathcal{A}$, is $[\![\mathcal{G}]\!]$ accepted by $\mathcal{A}$?" We discuss below decidable and undecidable fragments of the sort system.

Decidable fragments. By Corollary 1, if $\mathcal{S}$ allows only finite sorts (intersection sorts with finite width and depth), then the model-checking problem is decidable. The ML-style let-polymorphism (where the set of rewriting rules is of the form $F_1\ \widetilde{x} \to t_1; \cdots ; F_m\ \widetilde{x} \to t_m; S \to t_{m+1}$ and F_i can have polymorphic sorts only in $t_j (j > i)$) satisfies this condition. It is easy to see that if $F_1\ \widetilde{x} \to t_1; \cdots ; F_m\ \widetilde{x} \to t_m; S \to t_{m+1}$ is well-typed under the ML-style polymorphic type system, then the recursion scheme is well-sorted by using only finite sorts.

Another fragment that has only finite sorts is the system **S** studied in [10]. As system **S** contains rank-2 intersection with polymorphic recursion ($\mathbf{I_2}$ + **REC-INT**, for which the typability is undecidable [11]) as a subsystem, this is an interesting example for which well-sorting is undecidable, but the model-checking problem for well-sorted recursion schemes is decidable. We do not know whether the model-checking problem is decidable for the fragment with Milner-Mycroft-style polymorphic recursion.

Remark 3. Note that if a recursion scheme is well-sorted by using finite sorts, then it can be transformed into an equivalent, simply-typed recursion scheme. (To see why, observe that a function of sort $\tau_1 \wedge \tau_2 \to \tau$ can be transformed into a function of sort $\tau_1 \to \tau_2 \to \tau$). Thus, extending simply-typed recursion schemes to those with finite intersection sorts does not increase the expressive power of recursion schemes. Our approach of using intersection sorts would, however, be more efficient, as the transformation from a finitely-sorted recursion scheme into a simply-typed recursion scheme will blow up the size of the recursion scheme.

Undecidable Fragments. The model-checking problem is undecidable for fragments that contain either recursive types or System F-style polymorphic types. With System F-style (impredicative) polymorphism, we can encode a natural number as a term of type $\forall\alpha.(\alpha \to \alpha) \to \alpha \to \alpha$ and express the successor, predecessor, and zero-equality test on natural numbers. Thus, by a combination with recursion,[4] we can encode a Minsky machine [8] M into a recursion scheme $\mathcal{G}$ such that M halts if and only if $[\![\mathcal{G}]\!]$ contains a terminal `h`. A similar reasoning applies to recursion schemes with recursive types.

For these undecidable fragments, we can still obtain a sound but incomplete model-checking algorithm, by restricting the refinement. For example, for recursive sorts, we can restrict their refinements to recursive types (i.e. regular infinite intersection types) of a fixed size, so that the number of possible refinements for each recursive sort is finite. Then Theorem 5 can be used to obtain a sound model-checking algorithm.

[4] Note that although terms of System F are strongly normalizing, we have recursion as a primitive.

7 Related Work

The model checking of recursion schemes has been studied extensively [1, 2, 5, 12, 13], and applied to higher-order program verification [3, 4, 9]. Knapik et al. [1] showed the decidability of the modal μ-calculus model-checking of *safe* recursion schemes, and Ong [2] showed the decidability for arbitrary (typed) recursion schemes. Kobayashi and Ong [3, 5] recently proposed type-based model checking algorithms for recursion schemes. To our knowledge, all the previous studies dealt with *simply-typed* recursion schemes (simply-typedness was a part of the definition of recursion schemes).

Infinite intersection types have been studied by Leivant [7] and Bonsangue and Kok [14]. One of the main advantages of their type systems is that infinite intersection types give a "natural master formalism" [7] for various type disciplines. It is for this reason that we have introduced infinite intersection types for recursion schemes. To our knowledge, the previous type systems [7, 14] do not allow types having infinite paths (like the type τ defined by $\tau((11)^*) = \rightarrow$, $\tau((11)^*1) = \wedge^1$, and $\tau((11)^*2) = \mathtt{o}$).

There may be some connection between our work and studies on infinitely λ-calculi [15–17], where infinite objects generated by infinite reductions of λ-terms are considered. Note that the model checking of recursion schemes is also concerned about properties of the infinite objects generated by λ-terms. Direct connection is however unclear, since different properties of the infinite objects are considered. Tatsuta [16] showed that there is no decidable type system that characterizes the class of hereditary head-normalizing terms (λ-terms whose Böhm trees do not have $\perp$), and also gave a type-based characterization of that class by using an intersection type system with a countably infinite set of types.

8 Conclusion

We have developed an infinite intersection type system that is equivalent to the model-checking of untyped recursion schemes for safety properties (the properties expressed by trivial automata). Future work includes an extension of the type system to deal with the full modal μ-calculus, along the line of Kobayashi and Ong's work for typed recursion schemes [5]. It is also left for future work to find good decidable restrictions of the infinite intersection type system and apply them to verification of programs written in a language with polymorphic and/or recursive types.

Acknowledgments. We would like to thank anonymous referees for useful comments. This work was partially supported by Kakenhi 20240001.

References

1. 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)

2. Ong, C.-H.L.: On model-checking trees generated by higher-order recursion schemes. In: LICS 2006, pp. 81–90. IEEE Computer Society Press, Los Alamitos (2006)
3. Kobayashi, N.: Types and higher-order recursion schemes for verification of higher-order programs. In: Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages, pp. 416–428 (2009)
4. Kobayashi, N., Tabuchi, N., Unno, H.: Higher-order multi-parameter tree transducers and recursion schemes for program verification. In: POPL, pp. 495–507 (2010)
5. Kobayashi, N., Ong, C.-H.L.: A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In: Proceedings of LICS 2009, pp. 179–188. IEEE Computer Society Press, Los Alamitos (2009)
6. van Bakel, S.: Intersection type assignment systems. Theor. Comput. Sci. 151(2), 385–435 (1995)
7. Leivant, D.: Discrete polymorphism. In: LISP and Functional Programming, pp. 288–297 (1990)
8. Minsky, M.L.: Computation: Finite and infinite Machines. Prentice-Hall, Englewood Cliffs (1967)
9. Kobayashi, N.: Model-checking higher-order functions. In: Proceedings of PPDP 2009. ACM Press, New York (2009)
10. Hallett, J.J., Kfoury, A.J.: Programming examples needing polymorphic recursion. Electr. Notes Theor. Comput. Sci. 136, 57–102 (2005)
11. Terauchi, T., Aiken, A.: On typability for rank-2 intersection types with polymorphic recursion. In: LICS, pp. 111–122. IEEE Computer Society, Los Alamitos (2006)
12. Knapik, T., Niwinski, D., Urzyczyn, P.: Deciding monadic theories of hyperalgebraic trees. In: Abramsky, S. (ed.) TLCA 2001. LNCS, vol. 2044, pp. 253–267. Springer, Heidelberg (2001)
13. Aehlig, K., de Miranda, J.G., Ong, C.-H.L.: The monadic second order theory of trees given by arbitrary level-two recursion schemes is decidable. In: Urzyczyn, P. (ed.) TLCA 2005. LNCS, vol. 3461, pp. 39–54. Springer, Heidelberg (2005)
14. Bonsangue, M.M., Kok, J.N.: Infinite intersection types. Inf. Comput. 186(2), 285–318 (2003)
15. Berarducci, A., Dezani-Ciancaglini, M.: Infinite lambda-calculus and types. Theor. Comput. Sci. 212(1-2), 29–75 (1999)
16. Tatsuta, M.: Types for hereditary head normalizing terms. In: Garrigue, J., Hermenegildo, M.V. (eds.) FLOPS 2008. LNCS, vol. 4989, pp. 195–209. Springer, Heidelberg (2008)
17. Tatsuta, M.: Types for hereditary permutators. In: LICS, pp. 83–92. IEEE Computer Society, Los Alamitos (2008)

Solvability in Resource Lambda-Calculus[*]

Michele Pagani and Simona Ronchi della Rocca

Dipartimento di Informatica – Università di Torino
C.so Svizzera 185 – 10149 Torino (IT)
{pagani,ronchi}@di.unito.it

Abstract. The resource calculus is an extension of the λ-calculus allowing to model resource consumption. Namely, the argument of a function comes as a finite multiset of resources, which in turn can be either linear or reusable, giving rise to non-deterministic choices, expressed by a formal sum. Using the λ-calculus terminology, we call *solvable* a term that can interact with the environment: solvable terms represent meaningful programs. Because of the non-determinism, different definitions of solvability are possible in the resource calculus. Here we study the optimistic (angelical, or may) notion, and so we define a term *solvable* whenever there is a simple head context reducing the term into a sum where at least one addend is the identity. We give a syntactical, operational and logical characterization of this kind of solvability.

1 Introduction

The resource calculus (Λ^r) is an extension of the λ-calculus allowing to model resource consumption. Namely, the argument of a function comes as a finite multiset of resources, which in turn can be either linear or reusable. A linear resource needs to be used exactly once, while a reusable one can be called ad libitum. In this setting the evaluation of a function applied to a multiset of resources gives rise to different possible choices, because of the different possibilities of distributing the resources among the occurrences of the formal parameter. So the calculus is not deterministic, but no internal choice is performed actually, the result being a formal sum of all the possible cases. In case of a multiset of linear resources, also a notion of *crash* arises, whenever the cardinality of the multiset does not fit exactly the number of occurrences. Then the resource calculus is a useful framework for studying the notions of linearity and non-determinism, and the relation between them.

Λ^r is an evolution of the calculus of multiplicities, this last introduced by Boudol in order to study the semantics of the lazy λ-calculus [1]. Ehrhard and Regnier designed the differential λ-calculus [2], drawing on insights gained from an analysis of some denotational models of linear logic. As the authors remarked the differential λ-calculus seemed quite similar to Boudol's calculus of multiplicities. Indeed this was formalized by Tranquilli, which defined the Λ^r syntax, and

[*] Partially founded by the Italian MIUR project CONCERTO, and the French ANR projet blanc CHOCO, ANR-07-BLAN-0324.

showed a Curry-Howard correspondence between this calculus and Ehrhard and Regnier's differential nets [3]. The main differences between Boudol's calculus and Λ^r are that the former is equipped with explicit substitution and lazy operational semantics, while the latter is a true extension of the classical λ-calculus.

One way to appreciate the resource calculus is by observing the various subcalculi it contains. Clearly, usual λ-calculus can be embedded into Λ^r translating the application MN into $M[N^!]$, where $[N^!]$ represents the multiset containing one copy of the resource N, which is reusable (see the grammar of Figure 1(a)). Forbidding linear terms but allowing non-empty finite multisets of reusable terms yields a purely non-deterministic extension of λ-calculus, which recalls de Liguoro and Piperno's $\lambda_\oplus$-calculus [4]. On the other side, allowing only multisets of linear terms gives the linear fragment of Λ^r, used by Ehrhard and Regnier for giving a quantitative account to λ-calculus β-reduction through Taylor expansion [5, 6].

The aim of this paper is to study the operational behaviour of the full resource calculus. It has been already proved that it enjoys the properties of confluence and a sort of standardization [7]. In particular confluence does not clash with non-determinism since the sum carries all the possibilities. Here we study the solvability property. Namely, following the λ-calculus terminology, we use the word *solvable* in order to denote a term that can interact operationally with the environment, i.e., that can produce a given output when inserted into a context supplying it with suitable resources. According to this definition, in a computer science setting the solvable terms represent the meaningful programs.

Let us recall that in the λ-calculus a term M is defined to be solvable if and only if there is a context $C(\cdot)$ (of a non constant behaviour) such that $C(M)$ reduces to the identity. λ-solvability has been completely characterized, by different points of view. Syntactically a term is solvable if and only if it reduces to a head-normal form [8], operationally if and only if the head reduction strategy applied to it eventually stops [8], logically if and only if it can be typed in a suitable intersection type assignment system [9], denotationally if and only if its denotation is not minimal in a suitable sensible model [10, 11]. Our aim is to characterize the notion of solvability in Λ^r following the same lines.

The first problem we meet is the definition of solvability in Λ^r. In this paper we decided to follow an optimistic (angelical, or may) approach, and so we define a term to be solvable whenever there is a context, of a non constant behaviour, that, when filled by the term, reduces to a sum of terms, at least one of these being the identity. Other possible definitions of solvability (on which we are currently working) are discussed in the conclusion of the paper.

Our result is a characterization of solvability in Λ^r from a syntactical, operational and logical point of view (Theorem 19). It turns out that an extended notion of head-normal form can be defined, such that a term is solvable if and only if it can reduce to a term of such form. From an operational point of view, we use the notion of outer-reduction strategy, defined in [7], where no reduction is made inside reusable resources, and we prove that in order to reach the head-normal form we can restrict ourselves to use just reduction strategies of this kind. Moreover we give also a logical characterization of solvability, through a

Λ^r:	M, N, L	$::= x \mid \lambda x.M \mid MP$	terms
$\Lambda^{(!)}$:	$M^{(!)}, N^{(!)}$	$::= M \mid M^!$	resources
Λ^b:	P, Q, R	$::= [M_1^{(!)}, \dots, M_n^{(!)}]$	bags
$\Lambda^{(b)}$:	A, B	$::= M \mid P$	expressions

$\mathbb{M}, \mathbb{N} \in \mathcal{N}\langle\Lambda^r\rangle \quad \mathbb{P}, \mathbb{Q} \in \mathcal{N}\langle\Lambda^b\rangle \quad \mathbb{A}, \mathbb{B} \in \mathcal{N}\langle\Lambda^{(b)}\rangle := \mathcal{N}\langle\Lambda^r\rangle \cup \mathcal{N}\langle\Lambda^b\rangle$ sums

(a) Grammar of terms, bags, expressions, sums.

$$\lambda x.(\textstyle\sum_i M_i) := \sum_i \lambda x.M_i \qquad (\sum_i M_i)P := \sum_i M_i P \qquad M(\sum_i P_i) := \sum_i MP_i$$

$$[(\textstyle\sum_i M_i)]\cdot P := \sum_i [M]_i \cdot P \qquad [(\sum_i M_i)^!]\cdot P := [M_1^!, \dots, M_k^!]\cdot P$$

(b) Notation on $\mathcal{N}\langle\Lambda^{(b)}\rangle$.

Fig. 1. Syntax of resource calculus. The symbol $\mathcal{N}$ denotes the set of natural numbers, $\mathcal{N}\langle\Lambda^r\rangle$ (resp. $\mathcal{N}\langle\Lambda^b\rangle$) denotes the set of finite formal sums of terms (resp. bags), with 0 referring to the neutral element.

type assignment system, assigning to terms suitable non-idempotent intersection types. All these characterizations are conservative with respect to the λ-calculus.

The type assignment system we define is strongly related to the relational semantics of linear logic. It can be seen, basically, as an extension to Λ^r of the type system introduced by de Carvalho in the restricted case of λ-calculus [12]. We plan to continue our investigation in the direction of giving a clear setting where our type assignment can be presented as a logical description of a denotational model for the resource calculus, where all the unsolvable terms are equated. Indeed such a goal seems to us non immediate, since a quantitative account of resources does not fit well with the contextual closure of the interpretation function. For a discussion about this point see [12]. A possible solution might be achieved following the ideas in [13].

The paper is organized as follows. Section 2 contains a syntactical description of the resource calculus. Section 3 is dedicated to the definition of solvability and of head-normal form. In Section 4 the intersection type assignment system is presented and its properties are stated. In Section 5 there is the proof of the main theorem, showing all the characterizations of solvability. In Section 6 alternative notions of solvability are discussed.

2 Resource Calculus

Syntax. Basically, we have three syntactical sorts: terms, that are in functional position, bags, that are in argument position and represent multisets of resources, and finite formal sums, that represent the possible results of a computation.

$$y\langle N/x\rangle := \begin{cases} N & \text{if } y = x, \\ 0 & \text{otherwise,} \end{cases} \qquad (\lambda y.M)\langle N/x\rangle := \lambda y.(M\langle N/x\rangle),$$
$$(MP)\langle N/x\rangle := M\langle N/x\rangle P + M(P\langle N/x\rangle),$$
$$[M]\langle N/x\rangle := [M\langle N/x\rangle], \qquad 1\langle N/x\rangle := 0,$$
$$[M^!]\langle N/x\rangle := [M\langle N/x\rangle, M^!], \qquad (P\cdot R)\langle N/x\rangle := P\langle N/x\rangle\cdot R + P\cdot R\langle N/x\rangle.$$

Fig. 2. Linear substitution, in the abstraction case we suppose $y \notin \mathrm{FV}(N) \cup \{x\}$

Precisely, Figure 1(a) gives the grammar for generating the set Λ^r of **terms** and the set Λ^b of **bags** (which are in fact finite multisets of **resources** $\Lambda^{(!)}$) together with their typical metavariables. A resource can be linear (it must be used exactly once) or not (it can be used ad libitum), in the last case it is written with a ! apex. Bags are multisets presented in multiplicative notation, so that $P{\cdot}Q$ is the multiset union, and $1 = [\,]$ is the empty bag. It must be noted though that we will never omit the dot $\cdot$, to avoid confusion with application. An **expression** (whose set is denoted by $\Lambda^{(b)}$) is either a term or a bag. Though in practice only **sums** of terms are needed, for the sake of the proofs we also introduce sums of bags. In writing $\mathcal{N}\langle\Lambda^{(b)}\rangle$ we are abusing the notation, as it does not denote the $\mathcal{N}$-module generated over $\Lambda^{(b)} = \Lambda^r \cup \Lambda^b$ but rather the union of the two $\mathcal{N}$-modules. This amounts to say that sums may be taken only in the same sort.

The grammar for terms and bags does not include sums in any point, so that in a sense they may arise only on the "surface". However as an inductive notation (and *not* in the actual syntax) we extend all the constructors to sums as shown in Figure 1(b). In fact all constructors but the $(\cdot)^!$ are, as expected, linear. Notice the similarity between the equation $[(M+N)^!] = [M^!]\cdot[N^!]$ and $\mathrm{e}^{x+y} = \mathrm{e}^x\cdot\mathrm{e}^y$: this is far from a coincidence, as Taylor expansion and linear logic semantics show well [6]. We adopt the usual λ-calculus conventions as in [8]. Also we use the following notation for terms useful to build examples:

$$\mathbf{I} := \lambda x.x\,, \qquad \mathbf{F} := \lambda xy.y\,, \qquad \mathbf{\Delta} := \lambda x.x[x^!]\,, \qquad \mathbf{\Omega} := \mathbf{\Delta}[\mathbf{\Delta}^!]\,.$$

There is no technical difficulty in defining α-equivalence and the set $\mathrm{FV}(\mathbb{A})$ of free variables as in ordinary λ-calculus. Due to the presence of two kinds of resources, we need two different notions of substitutions, so to capture both the linear and non linear behaviour. Moreover we define also a resource substitution, which is expressed in function of the first two, useful for defining the reduction.

Definition 1 (Substitutions). *We define the following substitution operations.*

1. *$A\{N/x\}$ is the usual λ-calculus (i.e. capture free) substitution of N for x. It is extended to sums as in $\mathbb{A}\{\mathbb{N}/x\}$ by linearity in $\mathbb{A}$[1] and using the notations of Figure 1(b) for $\mathbb{N}$. The form $A\{x+N/x\}$ is called* **partial substitution**.

[1] $F(A)$ (resp. $F(A,B)$) is extended by linearity (resp. bilinearity) by setting $F\left(\sum_i A_i\right) = \sum_i F(A_i)$ (resp. $F\left(\sum_i A_i, \sum_j B_j\right) = \sum_{i,j} F(A_i, B_j)$).

2. *$A\langle N/x\rangle$ is the* **linear substitution** *defined inductively in Figure [2]. It is extended to $\mathbb{A}\langle\mathbb{N}/x\rangle$ by bilinearity in both $\mathbb{A}$ and $\mathbb{N}$.*
3. **Resource substitution** *$A\langle\langle N^{(!)}/x\rangle\rangle$ is the disjoint union of the partial and linear substitutions, i.e. $A\langle\langle N/x\rangle\rangle := A\langle N/x\rangle$ and $A\langle\langle N^!/x\rangle\rangle := A\{N+x/x\}$.*

Roughly speaking, the linear substitution corresponds to the replacement of the resource to exactly one *linear* occurrence of the variable. In the presence of multiple occurrences, all the possible choices are made, and the result is the sum of them. For example $(y[x][x])\langle M/x\rangle = y[M][x] + y[x][M]$. Indeed linear substitution bears resemblance to differentiation, as it is in Ehrhard and Regnier's differential λ-calculus [2]. We refer to [3, 14] for the mathematical intuitions underlying the resource calculus. The following are examples with sums

$$\begin{aligned}(x[x^!])\langle M+N/x\rangle &= (x[x^!])\langle M/x\rangle + (x[x^!])\langle N/x\rangle\\ &= M[x^!] + x[M,x^!] + N[x^!] + x[N,x^!],\\ (x[x^!])\{M+N/x\} &= (M+N)[(M+N)^!] = M[M^!,N^!] + N[M^!,N^!].\end{aligned}$$

Substitutions commute as stated in the following.

Lemma 2 ([2, 3, 14]). *For $\mathbb{A}$ a sum of expressions, $\mathbb{M},\mathbb{N}$ sums of terms and x,y variables such that $y \notin \mathrm{FV}(\mathbb{M}) \cup \mathrm{FV}(\mathbb{N})$, we have*

$$\begin{aligned}\big(\mathbb{A}\langle\mathbb{N}/y\rangle\big)\langle\mathbb{M}/x\rangle &= \big(\mathbb{A}\langle\mathbb{M}/x\rangle\big)\langle\mathbb{N}/y\rangle + \mathbb{A}\langle\mathbb{N}\langle\mathbb{M}/x\rangle/y\rangle\\ \big(\mathbb{A}\{y+\mathbb{N}/y\}\big)\langle\mathbb{M}/x\rangle &= \big(\mathbb{A}\langle\mathbb{M}/x\rangle\big)\{y+\mathbb{N}/y\} + \mathbb{A}\langle\mathbb{N}\langle\mathbb{M}/x\rangle/y\rangle\{y+\mathbb{N}/y\}.\end{aligned}$$

In particular if $x \notin \mathrm{FV}(\mathbb{N})$ then the second addend of both sums is 0 and the two substitutions commute.

Furthermore we have, if $x \notin \mathrm{FV}(\mathbb{M}) \cup \mathrm{FV}(\mathbb{N})$,

$$(\mathbb{A}\{x+\mathbb{M}/x\})\{x+\mathbb{N}/x\} = \mathbb{A}\{x+\mathbb{M}+\mathbb{N}/x\} = (\mathbb{A}\{x+\mathbb{N}/x\})\{x+\mathbb{M}/x\}.$$

Reductions. A (monic) context $C(\cdot)$ is a term that uses a distinguished free variable called its **hole** exactly once. Formally, the set of **simple contexts** is given by the following grammar

$$\Lambda_{(\cdot)}: \qquad C(\cdot), D(\cdot) ::= (\cdot) \mid \lambda x.C(\cdot) \mid C(\cdot)P \mid M[C(\cdot)]\cdot P \mid M[(C(\cdot))^!]\cdot P$$

A **context** $\mathbb{C}(\cdot)$ is a simple context in $\Lambda_{(\cdot)}$ summed to any sum in $\mathcal{N}\langle\Lambda^r\rangle$. The expression $\mathbb{C}(M)$ denotes the result of blindly replacing M to the hole (allowing variable capture) in $C(\cdot)$. We generalize to sums applying the notations of Figure 1(b). For example $C(\cdot) := \lambda x.y[(\cdot)^!]$ and $D(\cdot) := \lambda x.y[(\cdot)]$ are simple contexts. If $\mathbb{M} = x+y$, then $C(\mathbb{M}) = \lambda x.y[x^!, y^!]$ and $D(\mathbb{M}) = \lambda x.y[x] + \lambda x.y[y]$.

A relation r in $\Lambda^r \times \mathcal{N}\langle\Lambda^r\rangle$ is extended to one in $\mathcal{N}\langle\Lambda^r\rangle \times \mathcal{N}\langle\Lambda^r\rangle$ by **context closure**[2] by setting: $\mathbb{M}\ \tilde{r}\ \mathbb{N}$ iff $\exists\mathbb{C}(\cdot)$ and $M'\ r\ \mathbb{N}'$ s.t. $\mathbb{M} = \mathbb{C}(M'), \mathbb{N} = \mathbb{C}(\mathbb{N}')$.

[2] In [3, 14] bag contexts are defined too, so that context closure extends a relation to $\mathcal{N}\langle\Lambda^{(b)}\rangle \times \mathcal{N}\langle\Lambda^{(b)}\rangle$. In fact we prefer to introduce the term contexts only, making clear that the set $\mathcal{N}\langle\Lambda^r\rangle$ is the actual protagonist of the calculus. However our choice is a matter of taste, affecting no main property of the calculus.

$$\begin{tikzcd}
(\lambda x.x[\mathbf{\Delta}^!])[\mathbf{\Delta}^!,\mathbf{I}] \arrow[r,"\mathtt{b}"] \arrow[rrrd, bend right, "\mathtt{g}"] & (\lambda x.(\mathbf{\Delta}+x)[\mathbf{\Delta}^!])[\mathbf{I}] \arrow[r,"\mathtt{b}"] & (\lambda x.x[\mathbf{\Delta}^!])[\mathbf{I}] \arrow[r,"\mathtt{b}"] & (\lambda x.\mathbf{I}[\mathbf{\Delta}^!])1 \arrow[d,"\mathtt{b}"] & \\
& & & \mathbf{I}[\mathbf{\Delta}^!] \arrow[r,"\mathtt{b}"] \arrow[rd, bend right, "\mathtt{g}"'] & (\lambda x.\mathbf{\Delta})1 \arrow[d,"\mathtt{b}"] \\
& & & & \mathbf{\Delta}
\end{tikzcd}$$

Fig. 3. An example of baby- and giant-step reductions. We use the notation of Fig. 1(b): after the first $\mathtt{b}$-step the term $(\lambda x.(\mathbf{\Delta}+x)[\mathbf{\Delta}^!])[\mathbf{I}]$ stands for $(\lambda x.\mathbf{\Delta}[\mathbf{\Delta}^!])[\mathbf{I}] + (\lambda x.x[\mathbf{\Delta}^!])[\mathbf{I}]$, and the term after the following step is equal to $0+(\lambda x.x[\mathbf{\Delta}^!])[\mathbf{I}]$. In fact 0 is the neutral element of the sum and $(\lambda x.\mathbf{\Delta}[\mathbf{\Delta}^!])[\mathbf{I}] \xrightarrow{\mathtt{b}} 0$.

A context is **linear** if its hole is not under the scope of a $(\,)^!$ operator. Linear contexts can be defined inductively omitting the $M[(C(\cdot))^!]\cdot P$ generation rule in the simple context definition. A **head context** is a context having the hole not in a bag. Head contexts can be defined inductively omitting the rules $M[C(\cdot)]\cdot P$ and $M[(C(\cdot)^!)]\cdot P$ in the simple context definition. Notice that the composition $\mathbb{C}(\mathbb{D}(\cdot))$ of two head (resp. linear) contexts $\mathbb{C}(\cdot)$, $\mathbb{D}(\cdot)$ is head (resp. linear).

We introduce two kinds of reduction rule, baby-step and giant-step reduction, the former being a decomposition of the latter. Both are meaningful: baby-step is more atomic, performing one substitution at a time, while the giant-step is closer to λ-calculus β-reduction, wholly consuming its redex in one shot.

Definition 3 ([3, 14]). *The* **baby-step** *reduction* $\xrightarrow{\mathtt{b}}$ *is defined by the context closure of the following relation (supposing x not free in N):*

$$(\lambda x.M)1 \xrightarrow{\mathtt{b}} M\{0/x\} \qquad (\lambda x.M)[N]\cdot P \xrightarrow{\mathtt{b}} (\lambda x.M\langle N/x\rangle)P$$

$$(\lambda x.M)[N^!]\cdot P \xrightarrow{\mathtt{b}} (\lambda x.M\{N+x/x\})P$$

The **giant-step** *reduction* $\xrightarrow{\mathtt{g}}$ *is defined by the context closure of the following relation, for $n \geq 0$: $(\lambda x.M)[N_1^{(!)},\ldots,N_n^{(!)}] \xrightarrow{\mathtt{g}} M\langle\langle N_1^{(!)}/x\rangle\rangle\ldots\langle\langle N_n^{(!)}/x\rangle\rangle\{0/x\}$.*

For any reduction $\xrightarrow{\mathtt{x}}$, we denote by $\xrightarrow{\mathtt{x}+}$ and $\xrightarrow{\mathtt{x}*}$ its transitive and reflexive-transitive closure respectively.

Notice that giant-step reduction is defined independently of the ordering of the resource substitutions, as shown by the substitution commutations stated above. Baby-step and giant-step reductions are clearly related each other.

Proposition 4 ([3, 14]). *We have $\xrightarrow{\mathtt{g}} \subset \xrightarrow{\mathtt{b}*} \subset \xrightarrow{\mathtt{g}*}\xleftarrow{\mathtt{g}*}$, where the last denotes the composition between $\xrightarrow{\mathtt{g}*}$ and its inverse $\xleftarrow{\mathtt{g}*}$.*

Figure 3 shows an example of baby-step and giant-step reduction sequences. The reader can check, in this example, that the two reductions are related as stated in Proposition 4. By the way, let us mention that although giving the same normal forms, baby-step and giant-step reductions might have different properties: for example, the starting term in the Figure 3 is strongly normalizing for giant-step but only weakly normalizing for baby-step reduction (an infinite reduction sequence can be obtained by firing the $\mathbf{\Delta}[\mathbf{\Delta}^!]$ redex in the first addend of $(\lambda x.(\mathbf{\Delta} + x)[\mathbf{\Delta}^!])[\mathbf{I}]$).

3 Solvability

Using the λ-calculus terminology, we will call *solvable* the terms representing meaningful programs, i.e., the ones that can interact with the environment. Let us recall that in λ-calculus a term is solvable whenever there is a head context reducing it to the identity [8]. In resource calculus terms appear in formal sums, where repetitions do matter, hence various notions of solvability can arise, depending on the number of times one gets the identity. This paper deals extensively with the weakest notion of solvability, which asks that a term is solvable whenever a suitable context filled with it reduces to a sum, where at least one addend is the identity. This notion is related in some sense to a *may*-semantics of Λ^r, which arises naturally because of the definition of 0 as the neutral element of the sum. However different notions of solvability could be proposed, and we will discuss them in Section 6.

Definition 5. *A term M is* **solvable** *whenever there are a head simple context $C(\cdot)$ and a sum of terms $\mathbb{N}$, possibly 0, such that $C(M) \xrightarrow{g*} \mathbf{I} + \mathbb{N}$.*

The above definition considers giant-step reduction, however one can replace it with baby-step reduction, obtaining an equivalent notion of solvability, as easily argued from Proposition 4. Instead, it is crucial the restriction to *simple* and *head* contexts: there are general contexts reducing constantly to $\mathbf{I}$, disregarding the term they are applied to. For example consider the head but non-simple context $\mathbf{I} + (\cdot)[\mathbf{I}1]$ or the simple but non-head context $(\lambda x.\mathbf{I})[(\cdot)^!]$: we have $\mathbb{C}(M) \xrightarrow{g} \mathbf{I}$ for every term M.

One major outcome of this paper is the characterization of solvability by means of the following notion of head-normalizability (see Theorem 19).

Definition 6. *A term is a* **head-normal form**, hnf *for short, if it has no redex but under the scope of a $(\,)^!$. The set of* hnf *can be defined inductively as follows.*

$$\lambda x.M \text{ is hnf if } M \text{ is hnf},$$
$$xP_1 \ldots P_p \text{ is hnf if } p \geq 0 \text{ and } \forall i \leq p, \text{ every linear resource in } P_i \text{ is a hnf}.$$

A sum $\mathbb{M}$ of terms is a head-normal form whenever it contains an addend in head-normal form. A term M is **head-normalizable** *iff it is reducible to a* hnf.

Notice that for the resource terms corresponding to λ-terms these notions coincide with the usual ones.

The term $\lambda x.y1[x^!, \mathbf{\Omega}^!]$ is a hnf, and it is solvable via $(\lambda y.(\cdot))[\mathbf{F}]$: indeed, $(\lambda y.(\lambda x.y1[x^!, \mathbf{\Omega}^!]))[\mathbf{F}] \xrightarrow{g} \lambda x.\mathbf{F}1[x^!, \mathbf{\Omega}^!] \xrightarrow{g} \lambda x.\mathbf{I}[x^!, \mathbf{\Omega}^!] \xrightarrow{g} \mathbf{I} + \lambda x.\mathbf{\Omega}$. The terms

$$\dfrac{}{x : \sigma \vdash x : \sigma}\,\mathtt{v} \qquad \dfrac{}{\vdash 1 : \omega}\,\mathtt{1} \qquad \dfrac{\Gamma \vdash A : \pi, \quad \mathbb{B} \neq 0}{\Gamma \vdash A + \mathbb{B} : \pi}\,\oplus$$

$$\dfrac{\Gamma, x : \sigma_1, \ldots, x : \sigma_n \vdash M : \tau,\ x \notin d(\Gamma)}{\Gamma \vdash \lambda x.M : \sigma_1 \wedge \ldots \wedge \sigma_n \to \tau}\,{\to}\mathtt{I}_n \qquad \dfrac{\Gamma \vdash M : \pi \to \tau \quad \Delta \vdash P : \pi}{\Gamma, \Delta \vdash MP : \tau}\,{\to}\mathtt{E}$$

$$\dfrac{\Gamma \vdash M : \sigma \quad \Delta \vdash P : \pi}{\Gamma, \Delta \vdash [M]{\cdot}P : \sigma \wedge \pi}\,\ell \qquad \dfrac{\Gamma_i \vdash M : \sigma_i, \text{ for } 1 \leq i \leq n \quad \Delta \vdash P : \pi}{\Gamma_1, \ldots, \Gamma_n, \Delta \vdash [M^!]{\cdot}P : \sigma_1 \wedge \ldots \wedge \sigma_n \wedge \pi}\,!_n$$

Fig. 4. The type assignment system $\vdash$. The rules $\to\mathtt{I}_n$ and $!_n$ are parametrized by a natural number n, their 0-ary versions $\to\mathtt{I}_0$ and $!_0$ yield $\omega \to \tau$ and π respectively.

$\mathbf{F}[x^!, \mathbf{\Omega}^!]$, $\mathbf{I}[x^!, \mathbf{\Omega}^!]$ are not hnf but both are head-normalizable (the former reducing to $\mathbf{I}$, the latter to $x + \mathbf{\Omega}$); both are clearly solvable, also. The terms $\mathbf{F}[x]$ or $\mathbf{I}[\mathbf{\Omega}^!]$ are not head-normalizable: they reduce to 0 and $\mathbf{\Omega}$, respectively. The notion of head-reduction is extended in this non-deterministic setting as follows.

Definition 7 ([7]). *Let $\epsilon \in \{\mathtt{b}, \mathtt{g}\}$. The* **outer ϵ-reduction** $\xrightarrow{\mathrm{o}\epsilon}$ *is the closure to* linear *contexts of the ϵ steps given in Definition 3.*

4 An Intersection Type Assignment System

In this section we present an intersection type system assigning types to all and only the expressions having head-normal form (Theorem 19). This system lacks idempotency ($\sigma \wedge \sigma \neq \sigma$): in fact we use the intersection as logical counterpart of the multiset union. The system has some similarities with that one in [15], which supplies a logical semantics of the language in [1]. The main logical difference between the two systems is that the one in [15] is affine and describes a lazy operational semantics. In the restricted setting of λ-calculus similar non-idempotent systems have been considered starting from [16], e.g. [17, 18, 19, 12].

Definition 8. *The set of types is the union of the set of* linear types *and that of* intersection types*, given by the following grammars*

$$\begin{aligned} \sigma, \tau &::= a \mid \pi \to \sigma && \textit{linear types} \\ \pi, \zeta &::= \sigma \mid \omega \mid \pi \wedge \zeta && \textit{intersection types} \end{aligned}$$

where the variable a varies on an infinite set of atoms and ω is a constant. We consider types modulo the equivalence $\sim$ generated by the following rules:

$$\pi \wedge \zeta \sim \zeta \wedge \pi, \quad \pi \wedge \omega \sim \pi, \quad \pi_1 \wedge (\pi_2 \wedge \pi_3) \sim (\pi_1 \wedge \pi_2) \wedge \pi_3.$$

The last two rules allow us to consider n-ary intersections $\sigma_1 \wedge \ldots \wedge \sigma_n$, for any $n \in \mathcal{N}$, ω being the 0-ary intersection.

A basis *is a finite multiset of assignments of the shape $x : \sigma$, where x is a variable and σ is a linear type. Capital Greek letters Γ, Δ range over bases. We denote by* $\mathrm{d}(\Gamma)$ *the set of variables occurring in Γ and by Γ, Δ the multiset union between the bases Γ and Δ. A* typing judgement *is a sequent $\Gamma \vdash \mathbb{A} : \pi$.*

The $\vdash$ type assignment system derivates typing judgements for $\mathcal{N}\langle\Lambda^{(b)}\rangle$. Its rules are defined in Figure 4. Capital Greek letters Φ, Ψ range over derivations, $\Phi :: \Gamma \vdash \mathbb{A} : \pi$ denoting a derivation Φ with conclusion $\Gamma \vdash \mathbb{A} : \pi$.

Rule $\oplus$ assigns to a sum the type of one of its addends, and it reflects the may-semantics we chose. The condition $\mathbb{B} \neq 0$ is not necessary for characterizing head-normalizable terms, but it is useful to avoid redundant applications of $\oplus$. In the rule $!_n$ the parameter n takes into account the number of times the reusable resource $M^!$ will be called, whereas the rule ℓ assigns just one type to the linear resource M. Note that any bag containing only reusable resources can be typed by ω using the rules $\mathtt{1}$ and $!_0$. All other rules are almost standard.

Let us recall that types are modulo the equivalence $\sim$, which means that all the rules of Figure 4 must be considered closed under the $\sim$.

Definition 9. *The* measure *of a derivation Φ is the number* $\mathrm{m}(\Phi)$ *of axioms (i.e.* $\mathtt{v}$ *and* $\mathtt{1}$ *rules) in Φ. The measure* $\mathrm{m}(\mathbb{A})$ *of a sum of expressions $\mathbb{A}$ is*

$$\mathrm{m}(\mathbb{A}) := \inf\{\mathrm{m}(\Phi)\ ;\ \Phi :: \Gamma \vdash \mathbb{A} : \pi, \textit{ for } \Gamma \textit{ a basis and } \pi \textit{ a type}\}.$$

The following lemmata (Lemma 10-14) state basically that the typing system behaves well with respect to the substitutions of resource calculus. They are needed to prove Proposition 15: typing judgements are invariant under baby and giant-step reductions.

Lemma 10 (Linear Substitution). *Let $\Phi :: \Gamma, x : \tau \vdash \mathbb{A} : \pi$ and $\Psi :: \Delta \vdash N : \tau$. There is a derivation $\mathcal{L}(\Phi, \Psi) :: \Gamma, \Delta \vdash \mathbb{A}\langle N/x\rangle : \pi$ with $\mathrm{m}(\mathcal{L}(\Phi, \Psi)) = \mathrm{m}(\Phi) + \mathrm{m}(\Psi) - 1$.*

Proof (Sketch). By induction on Φ, splitting depending on its last rule. We treat in detail only the case of a terminal $!_n$ rule. The base of induction is trivial: $\mathtt{v}$ is immediate, while $\mathtt{1}$ does not meet the condition of having $x : \tau$ in the basis. The cases $\to\mathtt{I}_n$, $\oplus$ are immediate consequences of the induction hypothesis, the cases $\to\mathtt{E}$, ℓ are easier variant of the $!_n$ case. So let us assume

$$\Phi := \quad \dfrac{\begin{array}{c}\vdots\ \Phi_i \\ \underline{\Gamma_i} \vdash M : \sigma_i,\end{array} \quad \text{for } 1 \leq i \leq n \qquad \begin{array}{c}\vdots\ \Phi_P \\ \Gamma_P \vdash P : \zeta\end{array}}{\Gamma_1, \ldots, \Gamma_n, \Gamma_P \vdash [M^!]\cdot P : \sigma_1 \wedge \ldots \wedge \sigma_n \wedge \zeta}\ !_n$$

We suppose the underlined hypothesis $x : \tau$ is in Γ_1, i.e. $\Gamma_1 = \Gamma_1', x : \tau$ (the case $x : \tau$ is in another Γ_i or in Γ_P being an easy variant). Notice that supposing $x : \tau$ in Γ_1 entails $n \geq 1$. By induction there is $\mathcal{L}(\Phi_1, \Psi) :: \Gamma_1', \Delta \vdash M\langle N/x\rangle : \sigma_1$ s.t. $\mathrm{m}(\mathcal{L}(\Phi_1, \Psi)) = \mathrm{m}(\Phi_1) + \mathrm{m}(\Psi) - 1$. Let $M\langle N/x\rangle = \sum_{j=1}^{k} L_j$. In order to be typable $M\langle N/x\rangle$ must have an addend (i.e. $k > 0$), say L_1, and a proof $\Phi_1' :: \Gamma_1', \Delta \vdash L_1 : \sigma_1$ s.t. $\mathrm{m}(\Phi_1') = \mathrm{m}(\mathcal{L}(\Phi_1, \Psi))$. We define

$$\mathcal{L}(\Phi,\Psi) := \quad \dfrac{\dfrac{\begin{array}{c}\vdots\ \Phi'_1\\ \Gamma'_1, \Delta \vdash L_1 : \sigma_1\end{array} \quad \dfrac{\begin{array}{c}\vdots\ \Phi_i\\ \Gamma_i \vdash M : \sigma_i, \quad \text{for } 2 \leq i \leq n\end{array} \qquad \begin{array}{c}\vdots\ \Phi_P\\ \Gamma_P \vdash P : \zeta\end{array}}{\Gamma_2, \ldots, \Gamma_n, \Gamma_P \vdash [M^!]\cdot P : \sigma_2 \wedge \ldots \wedge \sigma_n \wedge \zeta}\,!_{n-1}}{\Gamma'_1, \Delta, \Gamma_2, ..., \Gamma_n, \Gamma_P \vdash [L_1, M^!]\cdot P : \sigma_1 \wedge ... \wedge \sigma_n \wedge \zeta}\,\ell}{\Gamma'_1, \Delta, \Gamma_2, \ldots, \Gamma_n, \Gamma_P \vdash \sum_{j=1}^{k}[L_j, M^!]\cdot P : \sigma_1 \wedge ... \wedge \sigma_n \wedge \zeta}\,\oplus$$

where by definition $[M\langle N/x\rangle, M^!]\cdot P = \sum_{j=1}^{k}[L_j, M^!]\cdot P$, and if $k = 1$ the last $\oplus$ rule is omitted. Moreover, $\mathrm{m}(\mathcal{L}(\Phi,\Psi)) = \mathrm{m}(\Phi'_1) + \mathrm{m}(\Phi_P) + \sum_{i=2}^{n}\mathrm{m}(\Phi_i) = \mathrm{m}(\mathcal{L}(\Phi_1,\Psi)) + \mathrm{m}(\Phi_P) + \sum_{i=2}^{n}\mathrm{m}(\Phi_i) = \mathrm{m}(\Phi_1) + \mathrm{m}(\Psi) - 1 + \mathrm{m}(\Phi_P) + \sum_{i=2}^{n}\mathrm{m}(\Phi_i) = \mathrm{m}(\Phi) + \mathrm{m}(\Psi) - 1$. □

Lemma 11 (Linear Expansion). *Let $\Phi :: \Gamma \vdash \mathbb{A}\langle N/x\rangle : \pi$. There are a linear type τ and derivations $\Phi_1 :: \Gamma_1, x : \tau \vdash \mathbb{A} : \pi$ and $\Phi_2 :: \Gamma_2 \vdash N : \tau$ with $\Gamma = \Gamma_1, \Gamma_2$.*

Proof (Sketch). By structural induction on $\mathbb{A}$, splitting depending on the top level constructor. We detail the case $\mathbb{A} = [M^!]\cdot P$, the other cases being easy variants. If $\mathbb{A}\langle N/x\rangle = [M\langle N/x\rangle, M^!]\cdot P + [M^!]\cdot P\langle N/x\rangle$, then Φ types only one addend of the sum $\mathbb{A}\langle N/x\rangle$ through a $\oplus$ rule. Let us suppose this addend is in $[M\langle N/x\rangle, M^!]\cdot P$ (the case it is in $[M^!]\cdot P\langle N/x\rangle$ being easier), so being of the form $[M', M^!]\cdot P$, with $M\langle N/x\rangle = M' + \mathbb{M}$. By inspecting the rules in Figure 4 one can deduce from Φ a derivation $\overline{\Phi}^1 :: \Gamma^1 \vdash M' : \sigma$ and a derivation $\overline{\Phi}^2 :: \Gamma^2 \vdash [M^!]\cdot P : \overline{\pi}$ s.t. $\Gamma = \Gamma^1, \Gamma^2$, $\pi = \sigma \wedge \overline{\pi}$ and $\overline{\Phi}^2$ ends in a $!_n$ rule with n premises typing M and one premise typing P. Possibly applying one $\oplus$ rule to $\overline{\Phi}^1$ we get a derivation of $\Gamma^1 \vdash M\langle N/x\rangle : \sigma$, hence by induction hypothesis we have $\overline{\Phi}^1_1 :: \Gamma^1_1, x : \tau \vdash M : \sigma$ and $\overline{\Phi}^1_2 :: \Gamma^1_2 \vdash N : \tau$. Then we define $\Phi_1 :: \Gamma_1, x : \tau \vdash [M^!]\cdot P : \pi$ as a $!_{n+1}$ rule with premise $\overline{\Phi}^1_1$ plus the premises of the $!_n$ rule ending $\overline{\Phi}^2$, and Φ_2 as $\overline{\Phi}^1_2$. □

Lemma 12 (Partial Substitution). *Let $m \geq 0$, $\Phi :: \Gamma, x : \sigma_1, \ldots, x : \sigma_m \vdash \mathbb{A} : \pi$ and $\forall i \leq m$, $\Psi_i :: \Delta_i \vdash N : \sigma_i$ with $\Delta = \Delta_1, \ldots, \Delta_m$. There is $\mathcal{P}(\Phi, \Psi_{i\leq m}) :: \Gamma, \Delta \vdash \mathbb{A}\{(N + x)/x\} : \pi$ with $\mathrm{m}(\mathcal{P}(\Phi, \Psi_{i\leq m})) = \mathrm{m}(\Phi) - m + \sum_{i=1}^{m}\mathrm{m}(\Psi_i)$.*

Proof (Sketch). Like in the proof of Linear Substitution (Lemma 10) we do induction on Φ, splitting depending on its last rule. We detail only the case of a terminal $!_n$ rule, the other cases being immediate or easier variants. So let

$$\Phi := \quad \dfrac{\begin{array}{c}\vdots\ \Phi_j\\ \Gamma_j, \Gamma^x_j \vdash M : \tau_j, \ \text{for } 1 \leq j \leq n\end{array} \qquad \begin{array}{c}\vdots\ \Phi_{n+1}\\ \Gamma_{n+1}, \Gamma^x_{n+1} \vdash P : \zeta\end{array}}{\Gamma, x : \sigma_1, \ldots, x : \sigma_m \vdash [M^!]\cdot P : \tau_1 \wedge \ldots \wedge \tau_n \wedge \zeta}\,!_n$$

where $\Gamma = \Gamma_1, \ldots, \Gamma_{n+1}$ and $x : \sigma_1, \ldots, x : \sigma_m = \Gamma^x_1, \ldots, \Gamma^x_{n+1}$. Notice $\mathrm{m}(\Phi) = \sum_{j=1}^{n+1}\mathrm{m}(\Phi_j)$. For every $j \leq n+1$, let I^j be the set of $i \leq m$ s.t. $x : \sigma_i$ is in Γ^x_j, m^j being the cardinality of I^j, possibly 0. Notice $m = \sum_{j=1}^{n+1} m^j$. Let Δ_{I^j} be the multiset union of the Δ_i bases with $i \in I^j$.We apply the induction hypothesis to each pair Φ_j and $\Psi_{i\in I^j}$, getting a derivation $\mathcal{P}(\Phi_j, \Psi_{i\in I^j}) :: \Gamma_j, \Delta_{I^j} \vdash M\{N + x/x\} : \tau_j$

for every $j \leq n$, and $\mathcal{P}(\Phi_{n+1}, \Psi_{I^{n+1}}) :: \Gamma_{n+1}, \Delta_{I^{n+1}} \vdash P\{N + x/x\} : \zeta$, such that $\mathrm{m}(\mathcal{P}(\Phi_j, \Psi_{i \in I^j})) = \mathrm{m}(\Phi_j) - m^j + \sum_{i \in I^j} \mathrm{m}(\Psi_i)$ for every $j \leq n + 1$.

As always, $M\{N + x/x\}$ (resp. $P\{N + x/x\}$) is in general a sum $\sum_{h=1}^{k} M_h$ (resp. $\mathbb{P}$). Let us suppose $k \geq 2$, the case $k = 0$ not holding since $M\{N + x/x\}$ is typed and the case $k = 1$ being immediate. By inspecting the rules of Figure 4, we obtain a function $f : \{0, \ldots, n-1\} \to \{0, \ldots, k-1\}$, an addend P' in $\mathbb{P}$, and a family of derivations $\Phi'_j :: \Gamma_j, \Delta_{I^j} \vdash M_{f(j)} : \tau_j$ for $j \leq n$, and $\Phi'_{n+1} :: \Gamma_{n+1}, \Delta_{I^{n+1}} \Delta_i \vdash P' : \zeta$, s.t. $\mathrm{m}(\mathcal{P}(\Phi_j, \Psi_{i \in I^j})) = \mathrm{m}(\Phi'_j)$ for $j \leq n + 1$. For every $h \leq k$, let $J_h = f^{-1}(h)$, and l_h be the cardinality of J_h; for h, $0 \leq h < k$, let $\pi_0 = \zeta$, $\pi_{h+1} = \pi_h \wedge \bigwedge_{j \in f^{-1}(h+1)} \tau_j$, and $\overline{\Gamma}_0 = \Gamma_{n+1}$, $\overline{\Gamma}_{h+1} = \overline{\Gamma}_h, \Gamma_{f^{-1}(h)}$, and $\overline{\Delta}_0 = \Delta_{I^{n+1}}$, $\overline{\Delta}_{h+1} = \overline{\Delta}_h, \Delta_{\substack{j \in f^{-1}(h) \\ i \in I^j}}$, where, consistency as before, $\Gamma_{f^{-1}(h)}$ (resp. $\Delta_{\substack{j \in f^{-1}(h) \\ i \in I^j}}$) denotes the multiset union of the Γ_j (resp. Δ_i) bases with $j \in f^{-1}(h)$ (resp. $i \in \bigcup_{j \in f^{-1}(h)} I^j$). Recalling $\pi_k = \tau_1 \wedge \ldots \wedge \tau_n \wedge \zeta = \pi$, we have $\mathcal{P}(\Phi, \Psi_{i \leq m}) :=$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\, \Phi'_j \\ \Gamma_j, \Delta_{I^j} \vdash M_k : \tau_k, \text{ for } j \in J_k\end{array} \qquad \begin{array}{c} \dfrac{\begin{array}{c}\vdots\, \Phi'_j \\ \Gamma_j, \Delta_{I^j} \vdash M_1 : \tau_j, \text{ for } j \in J_1\end{array} \quad \begin{array}{c}\vdots\, \Phi'_{n+1} \\ \Gamma_{n+1}, \Delta_{I^{n+1}} \vdash P' : \zeta\end{array}}{\overline{\Gamma}_1, \overline{\Delta}_1 \vdash [M_1^!]\cdot P' : \pi_1}\,!_{l_1} \\ \vdots \\ \overline{\Gamma}_{k-1}, \overline{\Delta}_{k-1} \vdash [M_1^!, \ldots, M_{k-1}^!]\cdot P' : \pi_{k-1}\end{array}}{\Gamma, \Delta \vdash [(M\{N + x/x\})^!]\cdot P' : \tau_1 \wedge \ldots \wedge \tau_n \wedge \zeta}\,!_{l_k}}{\Gamma, \Delta \vdash [(M\{N + x/x\})^!]\cdot \mathbb{P} : \tau_1 \wedge \ldots \wedge \tau_n \wedge \zeta}\,\oplus$$

We have $\mathrm{m}(\mathcal{P}(\Phi, \Psi_{i \leq m})) = \sum_{j=1}^{n+1} \mathrm{m}(\phi'_j) = \sum_{j=1}^{n+1} \mathrm{m}(\mathcal{P}(\Phi_j, \Psi_{i \in I^j})) = \sum_{j=1}^{n+1} \big(\mathrm{m}(\Phi_j) + \sum_{i \in I^j} \mathrm{m}(\Psi_i)\big) - \sum_{j=1}^{n+1} m^j = \mathrm{m}(\Phi) - m + \sum_{i=1}^{m} \mathrm{m}(\Psi_i)$. □

The next lemmata have proofs similar to the previous ones (by induction on $\mathbb{A}$ or Φ). We omit to sketch their proofs.

Lemma 13 (Partial Expansion). *Let $\Phi :: \Gamma \vdash \mathbb{A}\{N + x/x\} : \pi$, then there is a number $m \geq 0$, linear types $\tau_1, \ldots, \tau_m$ and derivations $\Phi_1 :: \Gamma_1, x : \tau_1, \ldots, x : \tau_m \vdash \mathbb{A} : \pi$ and $\Psi_i :: \Delta_i \vdash N : \tau_i$ for $i \leq m$ and $\Gamma = \Gamma_1, \Delta_1, \ldots, \Delta_m$.*

Lemma 14. *Let $x \notin d(\Gamma)$, then for every $\Phi :: \Gamma \vdash \mathbb{A} : \pi$ there is $\Psi :: \Gamma \vdash \mathbb{A}\{0/x\} : \pi$ with $\mathrm{m}(\Phi) = \mathrm{m}(\Psi)$, and vice versa.*

Proposition 15. *Let $\epsilon \in \{\mathsf{b}, \mathsf{g}\}$ and $M \xrightarrow{\epsilon} \mathbb{M}$, then M and $\mathbb{M}$ share the same judgements, i.e. $\Gamma \vdash M : \tau$ iff $\Gamma \vdash \mathbb{M} : \tau$. Also, $M \xrightarrow{\mathsf{og}} \mathbb{M}$ entails $\mathrm{m}(M) > \mathrm{m}(\mathbb{M})$.*

Proof (Scketch). The proof is by structural induction on M. The induction step splits depending on the top-level constructor of M. All cases are easy consequences of the induction hypothesis, taking into account that, whenever the redex is inside a reusable resource $N^!$ (so the reduction is not outer) the measure m may not decrease since (the bag containing) $N^!$ may be typed by ω.

The base of induction is when M is the redex fired by the reduction $M \xrightarrow{\epsilon} \mathbb{M}$. One can consider only the baby-step cases, the giant one will follow since it corresponds to a sequence of baby-steps. In particular one proves that the measure

$$\dfrac{\dfrac{\overset{\vdots\ \Psi_1}{\Gamma_1, x:\tau, \vec{x}:\vec{\tau} \vdash L:\sigma}}{\Gamma_1 \vdash \lambda x.L : \tau \wedge \zeta \to \sigma}\,{\scriptstyle \to\mathrm{I}_n} \qquad \dfrac{\overset{\vdots\ \Psi_2}{\Gamma_2 \vdash N:\tau} \quad \overset{\vdots\ \Psi_3}{\Gamma_3 \vdash P:\zeta}}{\Gamma_2, \Gamma_3 \vdash [N]\cdot P : \tau \wedge \zeta}\,{\scriptstyle \ell}}{\Gamma \vdash (\lambda x.L)[N]\cdot P : \sigma}\,{\scriptstyle \to\mathrm{E}}$$

(a) definition of Ψ

$$\dfrac{\dfrac{\dfrac{\overset{\vdots\ \Phi_1}{\Gamma_1, \Gamma_2, \vec{x}:\vec{\tau} \vdash \overline{L}:\sigma}}{\Gamma_1, \Gamma_2 \vdash \lambda x.\overline{L} : \zeta \to \sigma}\,{\scriptstyle \to\mathrm{I}_n} \qquad \overset{\vdots\ \Phi_3}{\Gamma_3 \vdash P:\zeta}}{\Gamma \vdash (\lambda x.\overline{L})P : \sigma}\,{\scriptstyle \to\mathrm{E}}}{\Gamma \vdash (\lambda x.L\langle N/x\rangle)P : \sigma}\,{\scriptstyle \oplus}$$

(b) definition of Φ

Fig. 5. Definition of the derivations Ψ and Φ used in the proof of Proposition 15

m is monotone strictly decreasing on every baby-step but the one choosing a bang element from the bag, in which case m is monotone decreasing. Then m strictly decreases on giant-steps since they correspond to sequences of baby-steps ending always in an empty bag baby-step.

The baby-step has three cases (recall Definition 3), depending on the resource involved in the reduction. The case of the empty bag is proven using Lemma 14, the case of the bag having one underlined linear resource uses Lemma 10 and 11, and the last case of a bag having one underlined reusable resource uses Lemma 12 and 13. We detail only the linear resource case. Let $M = (\lambda x.L)[N]\cdot P \xrightarrow{\mathtt{b}} (\lambda x.L\langle N/x\rangle)P = \mathbb{M}$ and suppose $\Psi :: \Gamma \vdash (\lambda x.L)[N]\cdot P : \sigma$. By inspecting the rules of Figure 4 we can assume Ψ to be as in Figure 5(a), where by $\vec{x}:\vec{\tau}$ we are meaning $x:\tau_1, \ldots, x:\tau_m$, and ζ is $\tau_1 \wedge \ldots \wedge \tau_m$ (in case $m = 0$, $\zeta = \omega$). By Linear Substitution (Lemma 10) we get $\mathcal{L}(\Psi_1, \Psi_2) :: \Gamma_1, \vec{x}:\vec{\tau}, \Gamma_2 \vdash L\langle N/x\rangle : \sigma$, with $\mathrm{m}(\mathcal{L}(\Psi_1,\Psi_2)) = \mathrm{m}(\Psi_1) + \mathrm{m}(\Psi_2) - 1$. As usual, we should notice that $L\langle N/x\rangle : \sigma$ might not be a simple term but a sum: in that case $\mathcal{L}(\Psi_1,\Psi_2)$ ends in a $\oplus$ rule with premise a derivation $\Phi_1 :: \Gamma_1, \vec{x}:\vec{\tau}, \Gamma_2 \vdash \overline{L} : \sigma$ with $\overline{L}$ a simple term in the sum $L\langle N/x\rangle$ and $\mathrm{m}(\Phi_1) = \mathrm{m}(\mathcal{L}(\Psi_1,\Psi_2))$. Then we define Φ as in Figure 5(b), with $\Phi_3 = \Psi_3$. We remark that $\mathrm{m}(\Phi) = \mathrm{m}(\Phi_1) + \mathrm{m}(\Phi_3) = \mathrm{m}(\mathcal{L}(\Psi_1,\Psi_2)) + \mathrm{m}(\Psi_3) = \mathrm{m}(\Psi) - 1$. We conclude that every type of $(\lambda x.L)[N]\cdot P$ is also a type of $(\lambda x.L\langle N/x\rangle)P$ and $\mathrm{m}((\lambda x.L)[N]\cdot P) \geq \mathrm{m}((\lambda x.L\langle N/x\rangle)P) + 1$.

Conversely, assume $\Phi :: \Gamma \vdash (\lambda x.L\langle N/x\rangle)P : \sigma$. We can suppose Φ as in Figure 5(b), where as above $\vec{x}:\vec{\tau}$ denotes the basis $x:\tau_1, \ldots, x:\tau_m$ with $\zeta = \tau_1 \wedge \ldots \wedge \tau_m$, and in case $L\langle N/x\rangle$ is a simple term the terminal $\oplus$ rule is omitted. By possibly adding one $\oplus$ rule to Φ_1 one get $\overline{\Phi}_1 :: \Gamma_1, \Gamma_2, \vec{x}:\vec{\tau} \vdash L\langle N/x\rangle : \sigma$. Applying Linear Expansion (Lemma 11) we have $\Psi_1 :: \Gamma_1, \vec{x}:\vec{\tau}, x:\tau \vdash L:\sigma$ and $\Psi_2 :: \Gamma_2 \vdash N:\tau$ (where we recall $x \notin \mathrm{FV}(N)$). Then we set Ψ as in Figure 5(a). This proves that the types of $(\lambda x.L\langle N/x\rangle)P$ are also of $(\lambda x.L)[N]\cdot P$. $\square$

5 Main Theorem

We prove the equivalence among solvability, typability and head-normalizability (Theorem 19). As a byproduct we achieve also an operational characterization through the notion of outer reduction (Definition 7). In various calculi the implication *typable* $\Rightarrow$ *head-normalizable* is often proven using suitable notions of computability (e.g. [20]) or reducibility candidates (e.g. [21]), whereas the implication *solvable* $\Rightarrow$ *head-normalizable* is argued through a standardization theorem (e.g. [8]). Our proof is instead based on a different method, namely both implications are easy consequences of Lemma 16, which is argued by induction on the measure on the type derivations given in Definition 9. In the λ-calculus setting, a similar approach can be found in [22]. More in general, the idea of measuring quantitative properties of terms using non-idempotent intersection types can be found also in [12, 23].

Lemma 16. *Let M be a resource term and $C(\cdot)$ be a simple head context. If $C(M)$ is typable, then M is reducible to a* hnf *by outer reduction.*

Proof. We do induction on $\mathrm{m}\big(C(M)\big)$, which is a finite number, being $C(M)$ typable. If M is a hnf we are done. Otherwise it has an outer redex, so let $M \xrightarrow{\mathsf{og}} \mathbb{M}$. Since $C(\cdot)$ is a head context, every outer redex of M is outer in $C(M)$, hence we have $C(M) \xrightarrow{\mathsf{og}} C(\mathbb{M})$. By Proposition 15 $\mathrm{m}\big(C(M)\big) > \mathrm{m}\big(C(\mathbb{M})\big)$. Let $\mathbb{M} = M' + \mathbb{M}''$ be such that $\mathrm{m}\big(C(M')\big) = \mathrm{m}\big(C(\mathbb{M})\big)$: the fact that M' exists is due to $C(\cdot)$ being simple, as every addend in $C(\mathbb{M})$ is obtained by plugging an addend of $\mathbb{M}$ in $C(\cdot)$. By induction hypothesis M' is outer reducible to a hnf $\mathbb{L}$. We conclude by context closure: $M \xrightarrow{\mathsf{og}} M' + \mathbb{M}'' \xrightarrow{\mathsf{og}*} \mathbb{L} + \mathbb{M}''$. □

Lemma 17. *Every term in head-normal form is solvable.*

Proof. By structural induction on a hnf M. The case $M = \lambda x.M'$ with M' hnf is a trivial consequence of the induction hypothesis. The case $M = xP_1 \dots P_p$ splits in two subcases, depending whether P_1 contains linear resources.

Case I: $P_1 = [L]{\cdot}\overline{P}_1$. We do induction on L and $x\overline{P}_1 P_2 \dots P_p$, which are hnf by definition. Thus we obtain two simple head contexts $C(\cdot)$ and $D(\cdot)$ s.t. $C(L) \xrightarrow{\mathsf{g}*} \mathbf{I}+\mathbb{G}_C$ and $D(x\overline{P}_1 P_2 \dots P_p) \xrightarrow{\mathsf{g}*} \mathbf{I}+\mathbb{G}_D$. Let H be the simple term $\lambda y.C(y)[x[y^!]]$, we have $H[L]{\cdot}\overline{P}_1 \xrightarrow{\mathsf{g}} C(L)[x\overline{P}_1]+\mathbb{G}_H \xrightarrow{\mathsf{g}*} (\mathbf{I}+\mathbb{G}_C)[x\overline{P}_1]+\mathbb{G}_H \xrightarrow{\mathsf{g}} x\overline{P}_1+\mathbb{G}_C[x\overline{P}_1]+\mathbb{G}_H$, where $\mathbb{G}_H$ is the garbage, possibly 0, generated by putting an element of the bag $\overline{P}_1$ into $C(\cdot)$, instead of L. Then define $E(\cdot) := (\lambda x.(\cdot))[H, x^!]$ and notice $E(x[L]{\cdot}\overline{P}_1 P_2 \dots P_p) \xrightarrow{\mathsf{g}} H[L]{\cdot}\overline{P}_1 P_2 \dots P_p+\mathbb{G}_E$, where $\mathbb{G}_E$ is the garbage, possibly 0, obtained by linearly substituting H for some free occurrence of x in one P_i's. Finally, we define the context $F(\cdot) := D(E(\cdot))$, which is simple and head, being the composition of simple and head contexts. We have $F(x[L]\cdot\overline{P}_1 P_2 \dots P_p) \xrightarrow{\mathsf{g}*} D(H[L]\cdot\overline{P}_1 P_2 \dots P_p) + \mathbb{G}' \xrightarrow{\mathsf{g}*} D(x\overline{P}_1 P_2 \dots P_p) + \mathbb{G}'' \xrightarrow{\mathsf{g}*} \mathbf{I} + \mathbb{G}'''$, where, noted in passing, $\mathbb{G}' = D(G_E)$, $\mathbb{G}'' = \mathbb{G}' + D((\mathbb{G}_C[x\overline{P}_1] + \mathbb{G}_H)P_2 \dots P_p)$ and finally $\mathbb{G}''' = \mathbb{G}'' + \mathbb{G}_D$.

Case II: No linear resource in P_1. We do induction on $xP_2 \dots P_p$, thus getting a simple head context $D(\cdot)$ reducing $xP_2 \dots P_p$ to a sum containing the identity. We set $F(\cdot) := (\lambda x.(\cdot))[\lambda y_1 \dots y_p.D(x[y_2^!] \dots [y_p^!]), x^!]$. Easily one checks $F(xP_1P_2 \dots P_p) \xrightarrow{\mathsf{g}*} D(xP_2 \dots P_p) + \mathbb{G} \xrightarrow{\mathsf{g}*} \mathbf{I} + \mathbb{G}'$, for suitable $\mathbb{G}$, $\mathbb{G}'$. □

In the λ-calculus the above lemma is trivial since contexts can reduce a head-normal form into the identity simply replacing the head variable with a term erasing all its resources. In the resource calculus this is not possible, because of the linear resources, that cannot be erased but must be used.

Lemma 18. *Every head-normal form is typable.*

Proof. By structural induction on a hnf $\mathbb{M}$. The only interesting case is when $\mathbb{M}$ is of the form $xP_1 \dots P_p$ with each P_i of the form $[M_{i,1}, \dots, M_{i,m_i}] \cdot [N^!_{i,1}, \dots, N^!_{i,n_i}]$, with $m_i, n_i \geq 0$ and for each $j \leq m_i$, M_{j,m_i} hnf. By induction hypothesis we have derivations $\Psi_{i,j} :: \Gamma_{i,j} \vdash M_{i,j} : \tau_{i,j}$ for each $i \leq p$, $j \leq m_i$ hence we can construct a derivation $\Phi_i :: \Gamma_{i,1}, \dots, \Gamma_{i,m_i} \vdash P_i : \tau_{i,1} \wedge \dots \wedge \tau_{i,m_i}$ by applying a tree of m_i rules ℓ having as premises the $\Psi_{i,1}, \dots, \Psi_{i,m_i}$ respectively and, as the rightmost leaf, a derivation of $\vdash [N^!_{i,1}, \dots, N^!_{i,n_i}] : \omega$ made of n_i rules $!_0$ and one rule $\mathtt{1}$. Similarly we get a derivation typing $xP_1 \dots P_p$ by applying a tree of p rules $\to\mathtt{E}$ having as premises the Φ_i's derivations and, as the leftmost leaf, a $\mathtt{v}$ rule typing x with $(\bigwedge_{j \leq m_1} \tau_{1,j}) \wedge \dots \wedge (\bigwedge_{j \leq m_p} \tau_{p,j}) \to \sigma$, for a linear type σ. □

Theorem 19. *Given a resource term M, the following are equivalent:*

1. *M is head-normalizable,*
2. *M is typable by $\vdash$,*
3. *M is reducible to a* hnf *by outer reduction,*
4. *M is solvable.*

Proof. 1 ⇒ 2: by Prop. 15 and Lemma 18. 2 ⇒ 3: by Lemma 16, merely taking the hole as the simple head context. 3 ⇒ 4: by Lemma 17 and context closure. 4 ⇒ 1: if there is a head simple context $C(\cdot)$ s.t. $C(M)$ has a hnf, by the already proven implication 1 ⇒ 2, $C(M)$ is typable, we conclude by Lemma 16. □

The implication 1 ⇒ 3 can also be argued as a corollary of the standardization proven in [7]. However our proof uses the type assignment system, namely Lemma 16, so it adopts a different approach with respect to the techniques in [7].

6 Concluding Remarks

Theorem 19 achieves a weak operational characterization of solvability. In fact, the non-deterministic nature of the calculus makes terms having different outer reduction sequences, some terminating in a (head-)normal form, others infinite. Take for example $\mathbf{I}[(\mathbf{I}[x])^!, (\mathbf{\Omega}[\mathbf{I}1])^!]$: a first outer step gives $\mathbf{I}[x] + \mathbf{\Omega}[\mathbf{I}1]$, from which starts an outer reduction terminating in x (i.e. $\mathbf{I}[x] + \mathbf{\Omega}[\mathbf{I}1] \xrightarrow{\mathtt{g}} \mathbf{I}[x] + \mathbf{\Omega}[0] = \mathbf{I}[x] \xrightarrow{\mathtt{g}} x$), as well as infinite outer reductions looping on the head-normal form $x + \mathbf{\Omega}[\mathbf{I}1]$ or looping on $\mathbf{I}[x] + \mathbf{\Omega}[\mathbf{I}1]$.

As already mentioned, other notions of solvability are meaningful, depending on the number of times one requires the identity in the resulting sum. The following two seem quite interesting:

– a term M is **must-solvable** whenever there are an applicative simple context $C(\cdot)$ and $n > 0$ such that $C(M) \xrightarrow{\mathtt{g}*} n\mathbf{I}$;

– a term M is **exactly-solvable** whenever there is an applicative simple context $C(\cdot)$ such that $C(M) \xrightarrow{g*} \mathbf{I}$

Clearly exact-solvability implies must-solvability, which in its turn implies the, let us say, *may-solvability* of Definition 5. Also, these three notions do not collapse one another: for example, the term $\mathbf{I}[\mathbf{I}^!, \mathbf{\Omega}^!]$ is may-solvable but not must- nor exactly-solvable, in fact $\mathbf{I}[\mathbf{I}^!, \mathbf{\Omega}^!] \xrightarrow{g} \mathbf{I} + \mathbf{\Omega}$; the term $\mathbf{I}[\mathbf{I}^!, \mathbf{I}^!]$ is may- and must-solvable, but not exactly-solvable, in fact $\mathbf{I}[\mathbf{I}^!, \mathbf{I}^!] \xrightarrow{g} 2\mathbf{I}$; the term $\mathbf{I}[\mathbf{I}, \mathbf{I}^!]$ is exactly solvable, hence also may and must solvable, in fact $\mathbf{I}[\mathbf{I}, \mathbf{I}^!] \xrightarrow{g} \mathbf{I}$. We will give an analysis of all these kinds of solvability in a future work.

Acknowledgements. We are grateful to Rocco De Nicola, Giulio Manzonetto, Mauro Piccolo and Paolo Tranquilli for useful discussions and hints.

References

[1] Boudol, G.: The Lambda-Calculus with Multiplicities. INRIA Report 2025 (1993)
[2] Ehrhard, T., Regnier, L.: The Differential Lambda-Calculus. Theor. Comput. Sci. 309(1), 1–41 (2003)
[3] Tranquilli, P.: Intuitionistic Differential Nets and Lambda-Calculus. Theor. Comput. Sci. (2008) (to appear)
[4] de'Liguoro, U., Piperno, A.: Non Deterministic Extensions of Untyped Lambda-Calculus. Inf. Comput. 122(2), 149–177 (1995)
[5] Ehrhard, T., Regnier, L.: Böhm trees, Krivine's Machine and the Taylor Expansion of Lambda-Terms. In: Beckmann, A., Berger, U., Löwe, B., Tucker, J.V. (eds.) CiE 2006. LNCS, vol. 3988, pp. 186–197. Springer, Heidelberg (2006)
[6] Ehrhard, T., Regnier, L.: Uniformity and the Taylor Expansion of Ordinary Lambda-Terms. Theor. Comput. Sci. 403(2-3), 347–372 (2008)
[7] Pagani, M., Tranquilli, P.: Parallel Reduction in Resource Lambda-Calculus. In: Hu, Z. (ed.) APLAS 2009. LNCS, vol. 5904, pp. 226–242. Springer, Heidelberg (2009)
[8] Barendregt, H.: The Lambda-Calculus, its Syntax and Semantics, 2nd edn. Stud. Logic Found. Math., vol. 103. North-Holland, Amsterdam (1984)
[9] Coppo, M., Dezani-Ciancaglini, M., Venneri, B.: Functional Characters of Solvable Terms. Zeitschrift für Mathematische Logik 27, 45–58 (1981)
[10] Hyland, J.M.E.: A Syntactic Characterization of the Equality in Some Models of the Lambda Calculus. J. London Math. Soc. 2(12), 361–370 (1976)
[11] Ronchi Della Rocca, S., Paolini, L.: The Parametric λ-Calculus: a Metamodel for Computation. EATCS Series. Springer, Berlin (2004)
[12] de Carvalho, D.: Execution Time of λ-Terms via Denotational Semantics and Intersection Types (2009) (submitted for publication)
[13] Bucciarelli, A., Ehrhard, T., Manzonetto, G.: Not Enough Points Is Enough. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 298–312. Springer, Heidelberg (2007)
[14] Tranquilli, P.: Nets between Determinism and Nondeterminism. Ph.D. thesis, Università Roma Tre/Université Paris Diderot (Paris 7) (April 2009)
[15] Boudol, G., Curien, P.L., Lavatelli, C.: A Semantics for Lambda Calculi with Resources. MSCS 9(5), 437–482 (1999)

[16] Coppo, M., Dezani-Ciancaglini, M., Venneri, B.: Principal Type Schemes and Lambda-Calculus Semantics. In: To H.B. Curry, (ed.) Essays on Combinatory Logic, Lambda-calculus and Formalism, pp. 480–490. Academic Press, London (1980)
[17] Kfoury, A.J.: A Linearization of the Lambda-Calculus and Consequences. Journal of Logic and Computation 10(3), 411–436 (2000)
[18] Wells, J.B., Dimock, A., Muller, R., Turbak, F.: A Calculus with Polymorphic and Polyvariant Flow Types. J. Funct. Program. 12(3), 183–227 (2002)
[19] Neergaard, P.M., Mairson, H.G.: Types, Potency, and Idempotency: why Nonlinearity and Amnesia Make a Type System Work. In: ICFP, pp. 138–149. ACM, New York (2004)
[20] Coppo, M., Dezani-Ciancaglini, M., Zacchi, M.: Type Theories, Normal Forms and D_∞-Lambda-Models. Inf. Comput. 72(2), 85–116 (1987)
[21] Girard, J.Y.: Interprétation Fonctionnelle et Élimination des Coupures de l'Arithmétique d'Ordre Supérieur. Thèse de doctorat, Université Paris 7 (1972)
[22] Valentini, S.: An elementary proof of strong normalization for intersection types. Archive for Mathematical Logic 40(7), 475–488 (2001)
[23] de Carvalho, D., Pagani, M., Tortora de Falco, L.: A Semantic Measure of the Execution Time in Linear Logic. Theor. Comput. Sci. (2008) (to appear)

A Hierarchy for Delimited Continuations in Call-by-Name

Alexis Saurin

PPS & INRIA πr^2[⋆]
saurin@pps.jussieu.fr

Abstract. $\Lambda\mu$-calculus was introduced as a Böhm-complete extension of Parigot's $\lambda\mu$-calculus. $\Lambda\mu$-calculus, contrarily to Parigot's calculus, is a calculus of CBN *delimited control* as evidenced by Herbelin and Ghilezan. In their seminal paper on (CBV) delimited control, Danvy and Filinski introduced the CPS Hierarchy of control operators $(\mathtt{shift}_i/\mathtt{reset}_i)_{i\in\omega}$.

In a similar way, we introduce in the present paper the *Stream Hierarchy*, a hierarchy of calculi extending and generalizing $\Lambda\mu$-calculus. The $(\Lambda^n)_{n\in\omega}$-calculi have Church-Rosser and Böhm theorems. We then present sound and complete CPS translations for the hierarchy. Next, we investigate the operational content of the hierarchy through its abstract machines, the $(\Lambda^n)_{n\in\omega}$-KAM. Finally, we establish that the Stream hierarchy is indeed a CBN analogue to the CPS hierarchy.

Keywords: $\Lambda\mu$-calculus, delimited control, CPS hierarchy, Böhm theorem, CPS translation, Abstract machine, Streams.

1 Introduction

Curry-Howard in Classical Logic, $\lambda\mu$-calculus and Separation. Curry-Howard correspondence [17] was first designed as a correspondence between intuitionistic natural deduction (NJ) and simply typed λ-calculus. The extension of the correspondence to classical logic resulted in strong connections with control operators in functional languages as first noticed [15] by Griffin who analysed the logical interpretation of Felleisen's $\mathcal{C}$ operator [12]. Shortly after Griffin, Parigot introduced $\lambda\mu$-calculus [27] as an extension of λ-calculus corresponding to minimal classical natural deduction [1,26] in which one can encode usual control operators. $\lambda\mu$-calculus became one of the most widely studied classical λ-calculi, both in the typed and untyped setting, for several reasons: it naturally extends λ-calculus while retaining most of λ-calculus standard properties and intuitionistic natural deduction in a straightforward way. However, a fundamental property of pure λ-calculus, known as separation property (or Böhm theorem [6]), does not hold for $\lambda\mu$-calculus [29,9]. In a previous work, we introduced $\Lambda\mu$-calculus, an extension to $\lambda\mu$-calculus, for which we proved that separation holds [31].

[⋆] This work was partially funded by the french foreign office with the support of a *Bourse Lavoisier*.

Delimited control and the CPS hierarchy. Delimited control refers to a class of control operators which are much more expressive than non-delimited control operators (like `call/cc` for instance) in that they allow to simulate various side-effects [13], the monadic side-effects. In their seminal paper on `shift`/`reset` [7], Danvy and Filinski defined `shift`/`reset` delimited-control operators by their CPS semantics. They also introduced a hierarchy of such control operators, $(\mathtt{shift}_i/\mathtt{reset}_i)_{i\in\omega}$, which are obtained by iterating CPS translations and that is known as the CPS hierarchy. Delimited control and the CPS hierarchy found applications in linguistics, normalization by evaluation, partial evaluation or concurrency. While the emphasis was traditionally given to the delimited-control languages in call-by-value, recent works [16,21] have advocated the reasons for studying CBN delimited control.

In this paper, we develop a CBN analogue to the CPS hierarchy, based on $\Lambda\mu$-calculus. We develop our work on the strong connections between $\Lambda\mu$-calculus and calculi with delimited continuations in call-by-name evidenced by Herbelin and Ghilezan [16].

Structure of the Paper. In Section 2, we first review Parigot's $\lambda\mu$-calculus and $\Lambda\mu$-calculus as well as the main properties of those calculi. In Section 3, we motivate and define the $(\Lambda^n)_{n\in\omega}$-calculi which we refer to as the *stream hierarchy*. We establish two essential results of its meta-theory: Church-Rosser and Böhm theorems. Section 4 is concerned with translations of the stream hierarchy into λ-calculus which are sound and complete and we develop in Section 5 Krivine's style abstract machines [23] for the hierarchy. Finally, Section 6 makes precise the relationships between the Stream hierarchy and the CPS hierarchy. A long version of this paper can be found on the author's webpage [30].

2 Background and Notations: From $\lambda\mu$ to $\Lambda\mu$.

In this section, we recall some background on $\Lambda\mu$-calculus: starting with Parigot's $\lambda\mu$, we introduce $\Lambda\mu$-calculus via the property of Separation.

Parigot's Original Calculus: $\lambda\mu$. In 1992, Parigot proposed an extension of λ-calculus providing "an algorithmic interpretation of classical natural deduction" [27]: $\lambda\mu$-calculus is in Curry-Howard correspondence [17] with classical natural deduction [26,27]. Although initially motivated by the correspondence with classical logic, $\lambda\mu$-calculus is now widely studied in its untyped version as we do in the rest of this paper.

Definition 1. **$\lambda\mu$-terms** *($t, u, v, \dots \in \Sigma_{\lambda\mu}$) are defined by the following syntax:*

$$\boxed{\Sigma_{\lambda\mu} \qquad t,u ::= \ x \mid \lambda x.t \mid (t)u \mid \mu\alpha.(t)\beta}$$

with $x \in \mathcal{V}$ and $\alpha, \beta \in \mathcal{V}_c$, $\mathcal{V}$ and $\mathcal{V}_c$ being two disjoint infinite sets of variables.

In $\mu\alpha.(t)\beta$, variable β is in the scope of $\mu\alpha$. For $t \in \Sigma_{\lambda\mu}$, $(t)\alpha$ is not *in $\Sigma_{\lambda\mu}$, but we refer to such it as a* **named term** *and generically write n (and thus we write $\mu\alpha.n$). The set of closed $\lambda\mu$-terms is denoted by $\Sigma^c_{\lambda\mu}$.*

Remark 1. *The reader may have noticed that we use an alternative notation for $\lambda\mu$-terms that we introduced and justified in previous works [31,33], writing $(t)\alpha$ instead of the more common $[\alpha]t$ (this shall later be extended to the $(\Lambda^i)_{i\in\omega}$).*

In this paper, we shall use Krivine's notation [22] for terms of λ-calculus and its various extensions considered here: we write $(t)u$ for λ-application (instead of (MN)). As usual we consider λ-application to be left-associative, that is $(t)u_1 \dots u_{k-1}u_k$ shall be read as $(\dots((t)u_1)\dots u_{k-1})u_k$. This notation is extended to variables of $\mathcal{V}_c$ (and later on to the variables of the hierarchy). For instance, we shall write $\mu\alpha.(t)u\beta$ instead of $\mu\alpha.((t)u)\beta$.

Definition 2. **$\lambda\mu$-reduction**, *written $\longrightarrow_{\lambda\mu}$, is induced by the following rules:*

$$\boxed{\begin{array}{llll} (\lambda x.t)u \longrightarrow_\beta t\{u/x\} & & (\mu\alpha.n)\beta \longrightarrow_\rho n\{\beta/\alpha\} & \\ (\mu\alpha.n)u \longrightarrow_\mu \mu\alpha.n\{(v)u\alpha/(v)\alpha\} & & \mu\alpha.(t)\alpha \longrightarrow_\theta t & \textit{if } \alpha \notin FV(t) \end{array}}$$

$n\{(v)u\alpha/(v)\alpha\}$ substitutes (without variable-capture) every named term $(v)\alpha$ in n by $(v)u\alpha$. This substitution is called ***structural substitution*** [27].

A $\lambda\mu$-calculus Satisfying Böhm Theorem: $\Lambda\mu$-calculus. $\lambda\mu$ satisfies standard properties of λ-calculus such as confluence [27,29], subject reduction [27] and SN [28]. However, Böhm theorem fails in $\lambda\mu$-calculus (more precisely in its extensional version, $\lambda\mu\eta$-calculus [29,9]). This led us [31] to define an extension to $\lambda\mu\eta$, $\Lambda\mu$-calculus, for which we proved Böhm theorem: the more liberal syntax of $\Lambda\mu$ makes new contexts available and thus achieves a Böhm Out.

Definition 3. *$\Lambda\mu$-terms ($t, u, v \dots \in \Sigma_{\Lambda\mu}$) are defined by the following syntax:*

$$\boxed{\Sigma_{\Lambda\mu} \qquad t,u ::= \ x \mid \lambda x.t \mid (t)u \mid \mu\alpha.t \mid (t)\alpha}$$

where x (resp. α) ranges over an infinite set $\mathcal{V}_t$ (resp. $\mathcal{V}_s$) of term (resp. stream) variables. $\mathcal{V}_t$ and $\mathcal{V}_s$ are disjoint. The set of closed $\Lambda\mu$-terms is denoted by $\Sigma^c_{\Lambda\mu}$.

Remark 2. *Since $\alpha \notin \Sigma_{\Lambda\mu}$, it is clear that notations $(t)\alpha$ and $(t)u$ are not ambiguous. Notice that $\Sigma_{\lambda\mu} \subsetneq \Sigma_{\Lambda\mu}$ and that named terms of definition 1 are now elements of $\Sigma_{\Lambda\mu}$. Moreover, terms such as $\mu\alpha.\mu\beta.t$ or $\lambda x.(t)\alpha y$ are in $\Sigma_{\Lambda\mu} \setminus \Sigma_{\lambda\mu}$.*

Definition 4. **$\Lambda\mu$-reduction**, *written $\longrightarrow_{\Lambda\mu}$, is induced by the following rules:*

$$\boxed{\begin{array}{llll} (\lambda x.t)u \longrightarrow_{\beta_T} t\{u/x\} & \lambda x.(t)x & \longrightarrow_{\eta_T} t & \textit{if } x \notin FV(t) \\ (\mu\alpha.t)\beta \longrightarrow_{\beta_S} t\{\beta/\alpha\} & \mu\alpha.(t)\alpha & \longrightarrow_{\eta_S} t & \textit{if } \alpha \notin FV(t) \\ & \mu\alpha.t & \longrightarrow_{fst} \lambda x.\mu\alpha.t\{(v)x\alpha/(v)\alpha\} & \textit{if } x \notin FV(t) \end{array}}$$

Remark 3. *Notice that μ is not part of $\Lambda\mu$-calculus reduction system. It can indeed be simulated by a sequence of fst and β_T-reduction; see [31,33] for details. Names for reductions in $\Lambda\mu$ come from the stream interpretation of $\Lambda\mu$: $\mathcal{V}_S$-variables are place-holders for streams of $\Lambda\mu$-terms; see next section for details.*

The Böhm theorem for $\Lambda\mu$ is stated with respect to a set of *canonical normal forms* (corresponding to $\beta\eta$-normal forms in λ-calculus), which are terms in $\beta_T\eta_T\beta_S\eta_S$-normal form such that no *fst*-reduction step creates a non-*fst* redex:

Definition 5. *A $\Lambda\mu$-term t is in* **canonical normal form (CNF)** *if it is $\beta_T\eta_T\beta_S\eta_S$-normal and if it contains no subterm of the form $(\lambda x.u)\alpha$ nor $(\mu\alpha.u)v$.*

Theorem 4 (Böhm theorem [31]). *Let $t, t' \in \Sigma^c_{\Lambda\mu}$ in CNF. If $t \neq_{\Lambda\mu} t'$, then there exists a context*[1] *$C[]$ st. $C[t] \longrightarrow^{\star}_{\Lambda\mu} \lambda x.\lambda y.x$ and $C[t'] \longrightarrow^{\star}_{\Lambda\mu} \lambda x.\lambda y.y$.*

Confluence holds in $\Lambda\mu$ [32,34] under the same hypothesis as in $\lambda\mu\eta$-calculus:

Theorem 5. *$\forall t, t', t'' \in \Sigma^c_{\Lambda\mu}$, $\exists u \in \Sigma_{\Lambda\mu}$ s.t. $t \longrightarrow^{\star}_{\Lambda\mu} t', t'' \Rightarrow t', t'' \longrightarrow^{\star}_{\Lambda\mu} u$.*

3 $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$ and Beyond: The Stream Hierarchy

In the present section, we introduce the $(\Lambda^n)_{n\in\omega}$-calculi that we refer to as the *stream hierarchy*. This hierarchy of calculi is intended to be a call-by-name analogous to the CPS hierarchy. We first motivate our approach before defining the hierarchy and focusing on the metatheory of $(\Lambda^n)_{n\in\omega}$-calculi (they satisfy confluence and separation). In the following sections, we shall then study CPS translations and abstract machines for the hierarchy and finally, we shall establish that the Stream Hierarchy is indeed a CBN analogue to the CPS hierarchy in the final section of the paper.

3.1 Motivating the Stream Hierarchy

$\Lambda\mu$-calculus, a CBN calculus of delimited control. Separation theorem for $\Lambda\mu$-calculus can be seen as a consequence of the fact that $\Lambda\mu$-calculus admits more contexts than Parigot's $\lambda\mu$. As a consequence, it allows for a more powerful exploration of terms. Typical contexts used in the separation proofs are $[]u_1 \ldots u_m\beta_u v_1 \ldots v_n\beta_v$. This exploits the fact that a context of the form $[]u_1 \ldots u_m\beta_u$ *delimits* the part of the environment that can be passed through the left-most μ-abstracted variable (*i.e.* α) when term $\mu\alpha.\mu\alpha'.t$ is placed in the hole. As a result, one can access to the second μ-abstracted variable α' thanks to the second portion of the context, $v_1 \ldots v_n\beta_v$.

Based on this fact, Herbelin and Ghilezan [16] evidenced strong connections between $\Lambda\mu$-calculus and calculi with delimited continuations in the spirit of Danvy and Filinski `shift`/`reset` operators [7] using the calculus $\lambda\mu\widehat{\mathtt{tp}}$. In its call-by-value version, $\lambda\mu\widehat{\mathtt{tp}}$ is equivalent to Danvy-Filinski's `shift`/`reset` operators while in its call-by-name version the calculus is equationally correspondent to $\Lambda\mu$-calculus. This led Herbelin & Ghilezan to assert that $\Lambda\mu$-calculus is a CBN calculus of delimited control.

[1] The context may be asked to be "stream applicative", *ie.* of the form: $[]t_{1,1} \ldots t_{1,n_1}\alpha_1 \ldots t_{k,1} \ldots t_{k,n_k}\alpha_k$.

CPS Hierarchy. In their seminal paper on `shift`/`reset` [7], Danvy and Filinski introduced a hierarchy of control operators, $(\texttt{shift}_i/\texttt{reset}_i)_{i\in\omega}$, which are obtained by iterated CPS translations. This is known as the *CPS hierarchy*. In the following, we shall refer to it as the CPS hierarchy or $\lambda\mathcal{S}_n$ and adopt Kameyama's terminology [19]:

Definition 6 $(\lambda\mathcal{S}_n)$

$$\boxed{\begin{array}{llll}
\Sigma_{\lambda\mathcal{S}_n} & t,u ::= x \mid \lambda x.t \mid (t)u \mid \langle t\rangle_i \mid \mathcal{S}_i k.t & & 1\leq i\leq n\\
 & E_v^i ::= [] \mid (E_v^i)t \mid (V)E_v^i \mid \langle E_v^i\rangle_j & & 1\leq j\leq i\\
 & V ::= x \mid \lambda x.t & & \\
 & & & \\
 & (\lambda x.t)V & \longrightarrow t\,\{V/x\} & \\
 & \langle V\rangle_i & \longrightarrow V & \\
 & \langle E_v^{j-1}[\mathcal{S}_j k.t]\rangle_i & \longrightarrow \langle t\,\{\lambda x.\langle E_v^{j-1}[x]\rangle_j/k\}\rangle_i &
\end{array}}$$

While the emphasis was traditionally given to the delimited-control languages in call-by-value, recent works have advocated the interest of studying call-by-name delimited control [16,21], although CBN delimited control behaves quite differently from call-by-value. In particular, in pursuing the investigation of call-by-name delimited control, it is quite natural to wonder whether an analogous to the CPS hierarchy exists in the call-by-name world.

$\Lambda\mu$-calculus, Streams and Infinitary λ-calculi. The *fst*-rule allows for an operational interpretation of $\Lambda\mu$-calculus as a stream calculus with the ability to abstract over streams of $\Lambda\mu$-terms. With this interpretation of $\mathcal{V}_S$-variables as place-holders for streams of $\Lambda\mu$-terms:

– the effect of the *fst*-rule is to instantiate the *first* elements of a stream:

$$\mu\alpha.t \longrightarrow^{\star}_{fst} \lambda x_1\ldots\lambda x_n.\mu\alpha.t\,\{(v)x_1\ldots x_n\alpha/(v)\alpha\}$$

– $\mu\alpha$ is considered as an ***abstraction over streams of terms*** $(\lambda x_1^\alpha\ldots x_n^\alpha\ldots.t)$ while $(t)\alpha$ can be seen as the construction ***passing a stream as an argument*** to t $((t)x_1^\alpha\ldots x_n^\alpha\ldots)$;
– β_S and η_S are respectively the corresponding of β-reduction and η-reduction for streams (or an infinite reduction sequence of β, resp η) and rule *fst* corresponds to popping the first element of a stream (or matching it);
– actually, $\Lambda\mu$-calculus can be seen as a core functional language for stream, this direction being investigated in a current work with M. Gaboardi (see long version of the paper for details).

Parigot already noticed some (weak) form of this in his seminal paper where "the operator μ looks like a λ having potentially infinite number of arguments" [27]. Viewing μ as an operator iterating λ-abstraction until limit ordinal ω, the parallel with infinitary λ-calculi is natural. Such infinitary calculi have been considered in the literature [3,4,20] both to study infinite structures arising in lazy languages or to study consistency problems in λ-calculus. Though, infinitary λ-calculi have been designed in a much different way from the infinitary calculus underlying $\Lambda\mu$-calculus: while a reduction sequence may have transfinite length, depths of terms are bounded by ω (that is any subterm of an infinite term is at finite depth): subterms at transfinite depths are considered meaningless. On the contrary, with $\Lambda\mu$-calculus, limit ordinal ω is reached by one μ-abstraction which is a limit ordinal construction: $\mu\alpha.\mu\beta.\lambda x.x$ would correspond to transfinite term $\lambda x_0, x_1 \ldots x_\omega, x_{\omega+1} \ldots x_{\omega 2}.x_{\omega 2}$ in which $\lambda x_{\omega 2}.x_{\omega 2}$ is at depth $\omega 2$.

Even though we will not pursue this direction in this paper, this theme has been extremely influential in developing the stream hierarchy. Indeed, once a transfinite calculus is unveiled, the question of the ordinal by which it is indexed (if any) is pending: λ-calculus corresponds to ordinal ω while $\Lambda\mu$-calculus corresponds to ordinal ω^2 but what about other ordinals such as ω^3 for instance? The stream hierarchy is actually related to this question.

3.2 Definition of the Hierarchy of $(\Lambda^n)_{n\in\omega}$-calculi

Definition 7. *Let $\mathcal{V}$ be a countable set of variables ($x, y, \dots \in \mathcal{V}$). For any $i \in \omega$, one considers a copy of $\mathcal{V}$, named $\mathcal{V}^i$ ($x^i, y^i, \ldots$ denoting the elements of $\mathcal{V}^i$), those copies being pairwise disjoint. Λ^ω-terms ($t, u, v, \dots \in \Sigma_{\Lambda^\omega}$) are defined by the following grammar (closed Λ^ω-terms are denoted by $\Sigma^c_{\Lambda^\omega}$):*

$$\boxed{\begin{array}{lllll} \Sigma_{\Lambda^\omega} & t, u ::= x^0 & \mid \lambda^0 x.t & \mid (t)u & \\ & & \mid \lambda^i x.t & \mid (t)x^i & \textit{for any } i > 0 \end{array}}$$

In $\lambda^i x.t$ (resp x^i), i is the **level** *of the abstraction (resp. variable) and $\lambda^i x$ binds every variable x^i which is free in t. An α-equivalence straightforwardly follows.*

Definition 8. *For $n \in \omega$, Σ_{Λ^n} (resp. $\Sigma^c_{\Lambda^n}$) is the restriction of Σ_{Λ^ω} (resp. $\Sigma^c_{\Lambda^\omega}$) to terms with binders and variables of level lower or equal to n, for $i \le n$.*

Definition 9. *For $n \in \omega$, $\longrightarrow_{\Lambda^n}$ is the reduction on Σ_{Λ^n} induced by rules:*

$$\boxed{\begin{array}{llll} (\lambda^0 x.t)u & \longrightarrow_{\beta^0} & t\left\{u/x^0\right\} & \\ (\lambda^i x.t)y^i & \longrightarrow_{\beta^i} & t\left\{y^i/x^i\right\} & \textit{if } 0 < i \le n \\ (\lambda^i x.t)u & \longrightarrow_{\mu^{i/0}} & \lambda^i x.t\left\{(v)ux^i/(v)x^i\right\} & \textit{if } 0 < i \le n \\ (\lambda^i x.t)y^j & \longrightarrow_{\mu^{i/j}} & \lambda^i x.t\left\{(v)y^j x^i/(v)x^i\right\} & \textit{if } 0 < j < i \le n \end{array}}$$

Definition 10. *For $n \in \omega$, $\longrightarrow_{\Lambda^n_\eta}$ is the reduction on Σ_{Λ^n} induced by rules:*

$$\boxed{\begin{array}{llll} (\lambda^0 x.t)u & \longrightarrow_{\beta^0} & t\left\{u/x^0\right\} & \\ (\lambda^i x.t)y^i & \longrightarrow_{\beta^i} & t\left\{y^i/x^i\right\} & \textit{if } 0 < i \le n \\ \lambda^i x.(t)x^i & \longrightarrow_{\eta^i} & t & \textit{if } x^i \notin FV(t), 0 \le i \le n \\ \lambda^i x.t & \longrightarrow_{fst^{i/j}} & \lambda^j x.\lambda^i x.t\left\{(v)x^j x^i/(v)x^i\right\} & \textit{if } x^j \notin FV(t) \textit{ and } 0 \le j < i \le n \end{array}}$$

Proposition 1. *For any $0 \le j < i \le n$, $\mu^{i/j}$ can be derived from $fst^{i/j}$ and β^j.*

Definition 11. *We consider the following subsystems of Λ^n_η-reduction:*

- $\boldsymbol{\beta}$ *(resp.* $\boldsymbol{\eta}$*) is the subsystem of reductions* $(\beta^i)_{0\leq i\leq n}$ *(resp.* $(\eta^i)_{0\leq i\leq n}$*);*
- ***fst*** *is the subsystem made of reductions* $(fst^{i/j})_{0\leq j<i\leq n}$*;*
- β^0_{var} *is the restriction of* β^0 *to redex where the argument is a level-0 variable;*
- $\boldsymbol{\beta}_{var}$ *is the subsystem made of reductions* β^0_{var} *and* $(\beta^i)_{1\leq i\leq n}$*.*

Example 1. Λ^0 and Λ^1 are respectively λ-calculus and $\Lambda\mu$-calculus.

We shall consider here an example in Λ^i which is a CBN correspondent to the level i `Shift` of the CPS-hierarchy $\mathcal{S} = \lambda^0 x.\lambda^i y.(x^0)\lambda^0 z.(z^0)y^i$.

Consider $\mathcal{C}^{<i} = []ut_{1,1}\dots t_{1,n_1}x_1^{j_1}\dots t_{k,1}\dots t_{k,n_k}x_k^{j_k}$ such that for all $l \leq k$, $j_l < i$, we have $\mathcal{C}^{<i}(\mathcal{S}) \longrightarrow^\star_{\Lambda^i} \lambda^i y.(u)\lambda^0 z.(z^0)t_{1,1}\dots t_{1,n_1}x_1^{j_1}\dots t_{k,1}\dots t_{k,n_k}x_k^{j_k}y^i$ that is $\mathcal{S}$ stores any context of level strictly less than i in a continuation that can later be manipulated (for instance it can be composed with itself if $u = \lambda^0 x.(u')\lambda^0 y.(x^0)(x^0)y^0$). The flow of control is given to u only once an argument of level i (or higher) is reached, in which case $\lambda^i y$ is destroyed.

3.3 Meta-theory of the Stream Hierarchy

In this section, we state two essential theorems of Λ^n-calculi: confluence and separation. More details can be found in [30].

Confluence theorem. Confluence holds on closed terms. Such a restriction is necessary: $(\lambda^2 y.x)z^2$ reduces to x and to $(\lambda^0 y.\lambda^1 y'.\lambda^2 y''.x)z^2$ which cannot reduce to the same term.

Theorem 6. *For any* $n \in \omega$ *and any* $t, u, v \in \Sigma^c_{\Lambda^n}$*, if* $t \longrightarrow^\star_{\Lambda^n_\eta} u, v$ *then there exists* $w \in \Sigma^c_{\Lambda^n}$ *such that* $u, v \longrightarrow^\star_{\Lambda^n_\eta} w$*.*

As a corollary, Λ^j is a conservative extension of Λ^i, for any $i < j$:

Corollary 1. *Let* $i < j \in \omega$ *and* $t, u \in \Sigma^c_{\Lambda^i}$*. Then* $t =_{\Lambda^i_\eta} u$ *iff* $t =_{\Lambda^j_\eta} u$*.*

Böhm theorem. To state the separation theorem (*aka* Böhm theorem) for the stream hierarchy, we first define *canonical normal forms* for the hierarchy using the notion of *pre-redex*.

Definition 12. $t \in \Sigma_{\Lambda^n}$ *is a* **pre-redex** *if it is of the form* $(\lambda^i x.t)y^j$ *or* $(\lambda^i x.t)u$ *for* $0 \leq i, j \leq n$*.*

Canonical normal forms (Λ^n-CNF) can be considered as those terms containing only **fst**-redexes such that a **fst**-reduction does not create any redex other than **fst**-redexes:

Definition 13. *A* Λ^n**-CNF** *is a* $\boldsymbol{\beta\eta}$*-normal form with no pre-redex.*

We can now state the separation result:

Theorem 7. *Let* $n \in \omega$*,* $t, u \in \Sigma^c_{\Lambda^n}$*. If* t, u *are non* ***fst****-equivalent* Λ^n*-CNF then there exists a context* $\mathcal{C}[]$ *st.* $\mathcal{C}[t] \longrightarrow^\star_{\Lambda^n_\eta} \lambda^0 x, y.x^0$ *and* $\mathcal{C}[u] \longrightarrow^\star_{\Lambda^n_\eta} \lambda^0 x, y.y^0$*.*

4 Translating the Stream Hierarchy into λ-Calculus

We define in this section sound and complete translations of the stream hierarchy into λ-calculus with pairs. These translations are inspired by the recent CPS translation for $\lambda\mu\widehat{\mathsf{tp}}$-calculus by Herbelin and Ghilezan [16]. Several translations into λ-calculus have been proposed for $\lambda\mu$-calculus in the literature. de Groote [10] was the first to study CPS translations for $\lambda\mu$-calculus. Lafont, Reus and Streicher [24] proposed a CPS translation for λ-calculus into λ-calculus with pairs which later led to a continuation semantics for $\lambda\mu$-calculus[36] and is very much related to CPS translations for $\lambda\mu$-calculus by Fujita [14] or Lassen [25]. A by-product of this section is to provide a sound and complete CPS translation for $\Lambda\mu$-calculus. We recall the definition of the λ-calculus with pairs.

Definition 14. *Terms of* λ**-calculus with pairs** *are given by the following syntax:*

$$\boxed{\Sigma_{\lambda\pi} \qquad t,u ::= x \mid \lambda x.t \mid (t)u \mid \langle t,u\rangle \mid (\pi_1)t \mid (\pi_2)t}$$

Definition 15. *Equations of $\lambda\pi$ are $\beta\eta$ (equationally) plus the following:*

$$\boxed{(\pi_1)\langle t_1,t_2\rangle =_{\pi_1} t_1 \qquad (\pi_2)\langle t_1,t_2\rangle =_{\pi_2} t_2 \qquad \langle(\pi_1)t,(\pi_2)t\rangle =_{SP} t}$$

Definition 16. *We assume that the set of variables of λ-calculus with pairs is $\mathcal{V} = \{k\} \uplus \mathcal{V}_0 \uplus \cdots \uplus \mathcal{V}_n$ and we define a translation $[-] : \Sigma_{\Lambda^n} \longrightarrow \Sigma_{\lambda\pi}$ as follows:*

$$\begin{aligned}
\left[x^0\right] &= \lambda k.(x^0)k \\
\left[\lambda^i x.t\right] &= \lambda k.((\lambda x^i.\,[t])(\pi_1)^{n-i+1}k)\langle \ldots \langle(\pi_2)(\pi_1)^{n-i}k,(\pi_2)(\pi_1)^{n-i-1}k\rangle \ldots,(\pi_2)k\rangle \\
\left[(t)x^i\right] &= \lambda k.([t])\langle \ldots \langle\langle x^i,(\pi_1)^{n-i}k\rangle,(\pi_2)(\pi_1)^{n-i-1}k\rangle \ldots,(\pi_2)k\rangle \\
[(t)u] &= \lambda k.([t])\langle \ldots \langle\langle [u],(\pi_1)^{n}k\rangle,(\pi_2)(\pi_1)^{n-1}k\rangle \ldots,(\pi_2)k\rangle
\end{aligned}$$

with $0 \leq i \leq n$ for $\left[\lambda^i x.t\right]$ and $0 < i \leq n$ for $\left[(t)x^i\right]$.

Remark 8. *In the previous definition, we abbreviated $(\pi_i)(\pi_i)\ldots(\pi_i)t$ as $(\pi_i)^n t$.*

The definition for $[(t)u]$ when $u = x_0$ corresponds to instantiating the definition for $\left[(t)x^i\right]$ with $i = 0$. An alternative definition for $[(t)u]$ is thus possible: $[(t)u] = \left[(t)x^0\right]\left\{[u]/x^0\right\}$ if $x^0 \notin FV(t)$, if clause for $\left[(t)x^i\right]$ is extended to $i = 0$.

Example 2. Consider $t = \lambda^1 y_0 \ldots y_n.(x^0)t_1 \ldots t_m \in \Sigma_{\Lambda^1}$. Then one has:

$$[t] \longrightarrow^\star$$
$$\lambda k.(x^0)\langle\langle [t_1],\ldots\langle [t_m],(\pi_1)(\pi_2)^{m+1}k\rangle\rangle,(\pi_2)^{m+2}k\rangle \left\{(\pi_1)(\pi_2)^i k/y_i^1,\ 0 \leq i \leq m\right\}$$

The translation is sound and complete with respect to Λ^n_η-equational theory:

Theorem 9. *For any $n \in \omega$, $t,u \in \Sigma^c_{\Lambda^n}$, $t =_{\Lambda^n_\eta} u$ iff $[t] =_{\beta\eta\pi SP} [u]$.*

For the completeness part, we study the image of Σ_{Λ^n} terms by the translation which is characterized by the terms T defined by the following grammar:

Definition 17. *The target of the CPS can be defined by the following grammar:*

$$\begin{array}{llllll} T, K_0 & ::= x^0 & \mid \lambda k.(T)K_{n+1} & \mid (\lambda x^i.T)K_i & \mid (\pi_1)K_1 & \textit{(for } 0 \leq i \leq n\textit{)} \\ K_i & ::= x^i & \mid \langle K_{i-1}, K_i\rangle & \mid (\pi_2)K_i & \mid (\pi_1)K_{i+1} & \textit{(for } 0 < i \leq n\textit{)} \\ K_{n+1} & ::= k & \mid \langle K_n, K_{n+1}\rangle & \mid (\pi_2)K_{n+1} & & \end{array}$$

Proof. We only sketch the proof, more details are available in appendix and in the long version. Soundness is obtained by induction on the length of a proof of equality between t and u. Completeness is more involved. It mainly amounts to the following arguments:

– an inverse translation, $\_^\sharp$, is defined from the target language of Λ^n to Λ^{n+1};
– one proves that the inverse translation preserves equality *in* Λ^{n+1}, and thus: if $[t] =_{\beta\eta\pi SP} [u]$, then $[t]^\sharp =_{\Lambda_\eta^{n+1}} [u]^\sharp$;
– one then shows that $[t]^\sharp =_{\Lambda_\eta^{n+1}} t$ so that we can deduce that $t =_{\Lambda_\eta^{n+1}} u$ and
– finally we conclude thanks to the fact that Λ_η^{n+1} is a conservative extension of Λ_η^n (corollary 1): $t =_{\Lambda_\eta^n} u$. □

Remark 10. *It shall be noted that the proof of completeness is greatly simplified by the use of the hierarchy in the sense that the inverse translation translates back to Λ^{n+1} and not to Λ^n. Indeed, it can take advantage of the regularity of the structure of the $n+1^{th}$ continuation used in the translation.*

A sound and complete CPS translation for $\Lambda\mu$-calculus, $[]^{\Lambda\mu}$, is obtained by instantiating the previous result with $n = 1$. We have the following corollary:

Corollary 2. *For any $t, u \in \Sigma_{\Lambda\mu}^c$, $t =_{\Lambda\mu} u$ if, and only if, $[t]^{\Lambda\mu} =_{\beta\eta\pi SP} [u]^{\Lambda\mu}$.*

5 An Operational Investigation of the Stream Hierarchy

In the final section of his seminal paper, Parigot outlined an abstract machine for $\lambda\mu$-calculus. Later, de Groote [11] and Streicher and Reus [36] studied abstract machines for $\lambda\mu$-calculus. We shall be interested in this section in abstract machines for the Stream hierarchy. We shall now define abstract machines which compute Λ^n-head normal forms. In the following, we do not consider extensionality rules which are not necessary to compute head normal forms.

Definition 18. *Λ^n-head normal forms are defined by the following grammar:*

$$\boxed{\begin{array}{llll} h ::= g & \mid \lambda^i x.h & & \textit{for } 0 \leq i \leq n \\ g ::= x^0 & \mid (g)t & \mid (g)x^i & \textit{for } 0 < i \leq n,\ t \textit{ denotes an arbitrary } \Lambda^n\textit{-term} \end{array}}$$

In the next definition we introduce constants representing variables in order to compute head normal forms (and not only weak head normal forms).

Definition 19. *Let $0 \leq i \leq n$. Constants of level i are defined as*

$$\boxed{\mathfrak{c}^i = x^i_{\rho_i} \mid v(\mathfrak{c}^j)^i_{\rho_i}}$$

with $x \in \mathcal{V}$, $i < j$ and ρ_i a finite sequence of integers $k, 0 \leq k < i$ (ϵ denotes the empty sequence). We shall also consider particular constants which shall represent empty contexts: $\bot_i$, $0 \leq i \leq n{+}1$. The structure of constants takes into account the need for treating the case of fst-rules. For instance x^j_ϵ will represent the variable x^j in the left-hand side of $\lambda^j x.t \longrightarrow_{fst^{j/i}} \lambda^i y.\lambda^j x'.t'$, while $v(x^j_\epsilon)^i_\epsilon$ and $x^j_{[i]}$ will respectively represent variable y^i and x'^j in the right-hand side.

Moreover, the following notions are needed to define the machine:

Definition 20. *We define by mutual recursion* **contexts of level** *i, $0 \leq i \leq n+1$,* **closures** *and* **environments***:*

- *a closure is a pair of a term t and an environment e, denoted $t[e]$;*
- *an environment e is a partial function which, when defined, associates to a variable of level i a context of level i;*
- *a context S_0 of level 0 is defined as follows: $S_0 ::= \bot_0 \mid \mathfrak{c}^0 \mid u[e]$;*
- *a context S_i of level i ($i \geq 1$) is defined as follows: $S_i ::= \bot_i \mid \mathfrak{c}^i \mid S_{i-1} \cdot S_i$.*

We set $\bot_i \cdot S_{i+1}$ to be equal to S_{i+1}, and $(S^1_i \cdot \ldots (S^n_i \cdot \bot_{i+1})) \cdot (S_{i+1} \cdot S_{i+2})$ to $(S^1_i \cdot \ldots (S^n_i \cdot S_{i+1})) \cdot S_{i+2}$. These equalities allow us to assume that if S is of the form $(((S^1_i \cdot \ldots (S^n_i \cdot \bot_{i+1})) \cdot S_{i+2}) \ldots S_k)$, then either $\forall j, i+2 \leq j \leq k, S_j = \bot_j$ or it is of the form $((((((S^1_i \cdot \ldots (S^n_i \cdot \bot_{i+1})) \cdot \bot_{i+2}) \ldots \bot_{j-1}) \cdot \mathfrak{c}^j_\rho) \cdot S_{j+1}) \ldots S_k)$.

Definition 21. *We define $\textbf{\textit{pop}}^i(S_{n+1})$ and $\textbf{\textit{push}}(S_i, S_{n+1})$ as follows:*

- *$\textbf{\textit{push}}(S_i, S_j)$ (with $i < j$):*
  - *$\textbf{\textit{push}}(\bot_i, S_j) = S_j$;*
  - *$\textbf{\textit{push}}(S_i, \bot_j) = ((S_i \cdot \bot_{i+1}) \cdot \cdots \cdot \bot_j)$ if $S_i \neq \bot_i$;*
  - *$\textbf{\textit{push}}(S_i, \mathfrak{c}^j_\rho) = ((S_i \cdot \bot_{i+1}) \cdot \cdots \cdot \bot_{j-1}) \cdot \mathfrak{c}^j_\rho$ if $S_i \neq \bot_i$;*
  - *$\textbf{\textit{push}}(S_i, S_{i+1}) = (S_i \cdot S_{i+1})$ if $S_i \neq \bot_i$;*
  - *$\textbf{\textit{push}}(S_i, S_j \cdot S_{j+1}) = (\textbf{\textit{push}}(S_i, S_j) \cdot S_{j+1})$ if $S_i \neq \bot_i$.*
- *$\textbf{\textit{pop}}^i(S_{n+1}) = \textbf{\textit{pop}}^{i,n+1}(S_{n+1})$ with $\textbf{\textit{pop}}^{i,j}(S_j)$ (for $i < j$):*
  - *$\textbf{\textit{pop}}^{i,j}(\bot_j) = (\bot_i, \bot_j)$;*
  - *$\textbf{\textit{pop}}^{i,j}(\mathfrak{c}^j_\rho) = (v(\mathfrak{c}^j_\rho)^i_\epsilon, \mathfrak{c}^j_{\rho \cdot i})$;*
  - *$\textbf{\textit{pop}}^{i,j+1}(((S^1_{i-1} \cdot \ldots S^n_{i-1} \cdot \bot_i) \cdot \bot_{i+1}) \cdot \ldots \bot_{j+1}) = (S^1_{i-1} \cdot \ldots S^n_{i-1} \cdot \bot_i, \bot_{j+1})$;*
  - *$\textbf{\textit{pop}}^{i,j+1}(((((S^1_{i-1} \cdot \ldots S^n_{i-1} \cdot \bot_i) \cdot \cdots \cdot \bot_{k-1}) \cdot \mathfrak{c}^k_\rho) \cdot S_{k+1}) \cdot \cdots \cdot S_{j+1}) = (S^1_{i-1} \cdot \ldots S^n_{i-1} \cdot v(\mathfrak{c}^k_\rho)^i_\epsilon, ((\mathfrak{c}^k_{\rho \cdot i} \cdot S_{k+1}) \ldots S_{j+1}))$. Otherwise, one has:*
  - *$\textbf{\textit{pop}}^{i,j+1}(S_j \cdot S_{j+1}) = (S'_i, S'_{j+1})$ if $\textbf{\textit{pop}}^{i,j}(S_j) = (S'_i, S''_j)$ and $S'_{j+1} = \textbf{\textit{push}}(S''_j, S_{j+1})$.*

We now define the Λ^n-KAM:

Definition 22. States of Λ^n-KAM *have the form $\lambda^{i_1} {x_1}^{i_1}_\epsilon \ldots \lambda^{i_n} {x_n}^{i_n}_\epsilon . \langle t, [e], S_{n+1} \rangle$ where $t \in \Sigma_{\Lambda^n}$, e is an environment and S_{n+1} is a context of level $n+1$. States are abbreviated as $\overrightarrow{\lambda} \langle t[e]\ S_{n+1} \rangle$ when the prefix of abstractions is irrelevant.*

An **initial state** *of Λ^n-KAM is of the form $\langle t, [\emptyset], \bot_{n+1} \rangle$.*

Definition 23. *The transitions of the machines are the following:*

$$\begin{array}{lllll}
\overrightarrow{\lambda}\langle x^0 \; [e] \; S\rangle & \longrightarrow & \overrightarrow{\lambda}\langle t \; [e'] & S\rangle & \textit{if } e(x^0) = t[e'] \\
\overrightarrow{\lambda}\langle (t)u \; [e] \; S\rangle & \longrightarrow & \overrightarrow{\lambda}\langle t \; [e] & S'\rangle & \textit{with } S' = \textbf{\textit{push}}(u[e], S) \\
\overrightarrow{\lambda}\langle (t)x^i \; [e] \; S\rangle & \longrightarrow & \overrightarrow{\lambda}\langle t \; [e] & S'\rangle & \textit{with } S' = \textbf{\textit{push}}(e(x^i), S) \\
\overrightarrow{\lambda}\langle \lambda^i x.t \; [e] \; S\rangle & \longrightarrow & \overrightarrow{\lambda}\langle t \; [e'] & S'\rangle & \textit{if } \textbf{\textit{pop}}^i(S) = (S'_i, S'),\ e' = [e, x^i = S'_i] \\
 & & & & \textit{and } S'_i \neq S^1_{i-1} \cdot \ldots (S^n_{i-1} \cdot \bot_i) \\
\overrightarrow{\lambda}\langle \lambda^i x.t \; [e] \; S\rangle & \longrightarrow & \overrightarrow{\lambda}\lambda^i x^i_\epsilon.\langle t \; [e'] & \bot_{n+1}\rangle & \textit{if } \textbf{\textit{pop}}^i(S) = (S^1_{i-1} \cdot \ldots (S^n_{i-1} \cdot \bot_i), \bot_{n+1}) \\
 & & & & \textit{and } e' = [e, x^i = S^1_{i-1} \cdot \ldots (S^n_{i-1} \cdot x^i_\epsilon)]
\end{array}$$

The only case when the machine cannot reduce is when the machine state is in case $\lambda^{i_1} x^{i_1}_{1\epsilon}. \ldots \lambda^{i_n} x^{i_n}_{n\epsilon}.\langle x^0, [e], S\rangle$ and x^0 is associated by e to a variable constant of level 0, $\mathfrak{c}^0$, and not to a closure $t[e']$ since there is no rule for reducing this case (it is easy to check that when the initial state is made of a closed term, this is indeed the only case which can stop the machine). The final states of the machine are thus of the form:

$$\boxed{\lambda^{i_1} x^{i_1}_{1\epsilon}. \ldots \lambda^{i_n} x^{i_n}_{n\epsilon}.\langle \mathfrak{c}^0, [e], S\rangle}$$

In that case, we have reached the head variable and obtained the head normal form, the prefix of $\lambda^i x^i_\epsilon$ which has been gathered during the computation is the prefix of abstractions of the head normal form (up to some **fst**-reduction which have been lazily performed in the term and shall be propagated during the reconstruction of the Λ^n-term). One actually has the following:

Theorem 11. *If t is a closed Λ^n-term, Λ^n-KAM stops after a computation from initial state $\langle t[\emptyset], \bot_{n+1}\rangle$ if and only if t has a head normal form.*

Moreover, from the constant of level 0 *which is the left-component of the final state, one can compute the head variable of the head normal form and recursively the complete head normal form.*

6 Relating the Stream Hierarchy and the CPS Hierarchy

The aim of this section is to make clear how the Stream hierarchy relates to Danvy & Filinski's CPS hierarchy and to actually show that the Stream hierarchy is indeed a call-by-name analogous to the CPS hierarchy, that is a CBN hierarchy of delimited continuations. For this purpose, we first introduce a new hierarchy of calculi, the $\lambda\mu\widehat{\mathsf{tp}}_n$-calculi that we use as mediators between the two CBN/CBV hierarchies, following a method recently developed by Herbelin and Ghilezan.

6.1 $\lambda\mu\widehat{\mathsf{tp}}_n$-calculi

Definition 24 ($\lambda\mu\widehat{\mathsf{tp}}_n$-calculi). *Let $n \in \omega$. $\lambda\mu\widehat{\mathsf{tp}}_n$***-terms** *($t, u, v, \cdots \in \Sigma_{\lambda\mu\widehat{\mathsf{tp}}_n}$) are defined by the following syntax (with $q ::= \alpha \mid \widehat{\mathsf{tp}}$):*

$$\boxed{\Sigma_{\lambda\mu\widehat{\mathsf{tp}}_n} \qquad t, u ::= x \mid \lambda x.t \mid (t)u \mid \mu^i q.c_i \qquad c_i ::= \left[q^i\right]t \quad (1 \leq i \leq n)}$$

CBV and CBN $\lambda\mu\widehat{\mathsf{tp}}_n$-calculi can be naturally considered. In the CBV case, values and evaluation contexts are defined as $V ::= x \mid \lambda x.t$ and $E^i_v ::= [] \mid (E^i_v)t \mid (V)E^i_v \mid \mu^j\widehat{\mathsf{tp}}.[q^j]E^i_v$, $1 \leq j < i$ while in the CBN case, every term is a value and evaluation contexts are $E^i ::= [] \mid (E^i)t \mid \mu^j\widehat{\mathsf{tp}}.[q^j]E^i$, $1 \leq j < i$.

Definition 25 (CBN/CBV $\lambda\mu\widehat{\mathsf{tp}}_n$equational theories)
CBV $\lambda\mu\widehat{\mathsf{tp}}_n$ equational theory (written $=_{\lambda\mu\widehat{\mathsf{tp}}_n^{cbv}}$) is defined by the following rules:

$$
\boxed{
\begin{array}{lll|llll}
(\beta_v) & (\lambda x.t)V & = t\left\{V/x\right\} & (\eta_v) & \lambda x.(V)x & = V & \text{if } x \notin FV(V) \\
(\eta^i_{\widehat{\mathsf{tp}}v}) & \mu^i\widehat{\mathsf{tp}}.\left[\widehat{\mathsf{tp}}^i\right]V & = V & (\eta^i_\mu) & \mu^i\alpha.\left[\alpha^i\right]t & = t & \text{if } \alpha^i \notin FV(t) \\
(\mu^i_{\widehat{\mathsf{tp}}}) & \left[\widehat{\mathsf{tp}}^i\right]\mu^i\widehat{\mathsf{tp}}.c_i & = c_i & (\mu^i_v) & \left[q^i\right]E^{i-1}_v\left[\mu^i\alpha.c_i\right] & = c_i\left\{\left[q^i\right]E^{i-1}_v[u]/\left[\alpha^i\right]u\right\} &
\end{array}
\\
\begin{array}{llll}
(\beta^i_\Omega) & (\lambda x.E^i_v[x])\mu^i\widehat{\mathsf{tp}}.c_i & = E^i_v\left[\mu^i\widehat{\mathsf{tp}}.c_i\right] & \\
(\mu'^i_{\widehat{\mathsf{tp}}}) & \left[\widehat{\mathsf{tp}}^l\right]\mu^i\alpha.c_i & = \left[\widehat{\mathsf{tp}}^l\right]\mu^i\widehat{\mathsf{tp}}.ci\left\{\widehat{\mathsf{tp}}^i/\alpha^i\right\} & (i \leq l) \\
(\mu^i_{let}) & \mu^j\alpha.\left[q^j\right](\lambda x.t)\mu^i\widehat{\mathsf{tp}}.c_i & = (\lambda x.\mu^j\alpha.\left[q^j\right]t)\mu^i\widehat{\mathsf{tp}}.c_i & (j \leq i+1)
\end{array}
}
$$

$=_{\lambda\mu\widehat{\mathsf{tp}}_n^{cbn}}$ is defined by keeping only rules $\beta_v, \eta_v, \mu^i_v, \mu^i_{\widehat{\mathsf{tp}}}, \eta^i_{\widehat{\mathsf{tp}}v}$ and η^i_μ, by considering every terms as a value, E^i as evaluation contexts and constraining q^i to be α^i. CBN rules are denoted by droping the v subscripts in the rule names.

Definition 26 (Translations between $\lambda\mathcal{S}_n$ and $\lambda\mu\widehat{\mathsf{tp}}_n$)

$$
\boxed{
\begin{array}{ll|ll}
|\langle t\rangle_i|^{\mathcal{S}\rangle\widehat{\mathsf{tp}}} & = \mu^i\widehat{\mathsf{tp}}.[\widehat{\mathsf{tp}}^i]|t|^{\mathcal{S}\rangle\widehat{\mathsf{tp}}} & |\mathcal{S}_i k.t|^{\mathcal{S}\rangle\widehat{\mathsf{tp}}} & = \mu^i\alpha.[\widehat{\mathsf{tp}}^i](\lambda k.|t|^{\mathcal{S}\rangle\widehat{\mathsf{tp}}})\lambda x.\mu^i\widehat{\mathsf{tp}}.[\alpha^i]x \\
|\mu^i\widehat{\mathsf{tp}}.c_i|^{\widehat{\mathsf{tp}}\rangle\mathcal{S}} & = \langle|c_i|^{\widehat{\mathsf{tp}}\rangle\mathcal{S}}\rangle_i & |\mu^i\alpha.c_i|^{\widehat{\mathsf{tp}}\rangle\mathcal{S}} & = \mathcal{S}_i k^i_\alpha.|c_i|^{\widehat{\mathsf{tp}}\rangle\mathcal{S}} \\
|[\widehat{\mathsf{tp}}^i]t|^{\widehat{\mathsf{tp}}\rangle\mathcal{S}} & = |t|^{\widehat{\mathsf{tp}}\rangle\mathcal{S}} & |[\alpha^i]q|^{\widehat{\mathsf{tp}}\rangle\mathcal{S}} & = (k^i_\alpha)|t|^{\widehat{\mathsf{tp}}\rangle\mathcal{S}}
\end{array}
}
$$

Definition 27 (Translations between Λ^n and $\lambda\mu\widehat{\mathsf{tp}}_n$)

$$
\boxed{
\begin{array}{ll|ll|ll}
|\lambda^i x.t|^{\Lambda\rangle\widehat{\mathsf{tp}}} & = \mu^i\alpha_x.[\widehat{\mathsf{tp}}^i]|t|^{\Lambda\rangle\widehat{\mathsf{tp}}} & |\mu^i\widehat{\mathsf{tp}}.c_i|^{\widehat{\mathsf{tp}}\rangle\Lambda} & = |c_i|^{\widehat{\mathsf{tp}}\rangle\Lambda} & |\mu^i\alpha.c_i|^{\widehat{\mathsf{tp}}\rangle\Lambda} & = \lambda^i x_\alpha.|c_i|^{\widehat{\mathsf{tp}}\rangle\Lambda} \\
|(t)x^i|^{\Lambda\rangle\widehat{\mathsf{tp}}} & = \mu^i\widehat{\mathsf{tp}}.[\alpha^i_x]|t|^{\Lambda\rangle\widehat{\mathsf{tp}}} & |[\widehat{\mathsf{tp}}^i]t|^{\widehat{\mathsf{tp}}\rangle\Lambda} & = |t|^{\widehat{\mathsf{tp}}\rangle\Lambda} & |[\alpha^i]t|^{\widehat{\mathsf{tp}}\rangle\Lambda} & = (|t|^{\widehat{\mathsf{tp}}\rangle\Lambda})x^i_\alpha
\end{array}
}
$$

Theorem 12. *For any $n \in \omega$, Λ^n is in eq. correspondence with CBN $\lambda\mu\widehat{\mathsf{tp}}_n$:*
let $t, u \in \Sigma^c_{\Lambda^n}$, $t =_{\Lambda^n_\eta} u \Rightarrow |t|^{\Lambda\rangle\widehat{\mathsf{tp}}} =_{\lambda\mu\widehat{\mathsf{tp}}_n^{cbn}} |u|^{\Lambda\rangle\widehat{\mathsf{tp}}}$
let $t, u \in \Sigma^c_{\lambda\mu\widehat{\mathsf{tp}}_n}$, $t =_{\lambda\mu\widehat{\mathsf{tp}}_n^{cbn}} u \Rightarrow |t|^{\widehat{\mathsf{tp}}\rangle\Lambda} =_{\Lambda^n_\eta} |u|^{\widehat{\mathsf{tp}}\rangle\Lambda}$

In order to study the correspondence with CPS hierarchy, we recall Kameyama's axiomatization of $\lambda\mathcal{S}_n$ [19]:

Definition 28. *$=_{\lambda\mathcal{S}_n}$ is defined as:*

$$
\boxed{
\begin{array}{llll}
(\beta_v) & (\lambda x.t)V & = t\left\{V/x\right\} & \\
(\eta_v) & \lambda x.(V)x & = V & \text{if } x \notin FV(V) \\
(\beta_\Omega) & (\lambda x.E^0_v[x])t & = E^0_v[t] & \text{if } x \notin FV(E^0_v) \\
(Reset-Value) & \langle V\rangle_i & = V & \\
(Reset-lift) & \langle(\lambda x.t)\langle u\rangle_i\rangle_j & = (\lambda x.\langle t\rangle_j)\langle u\rangle_i & j \leq i \\
(\mathcal{S}-reset) & \mathcal{S}_i k.\langle t\rangle_i & = \mathcal{S}_i k.t & \\
(\mathcal{S}-elim) & \mathcal{S}_i k.(k)\langle t\rangle_{i-1} & = \langle t\rangle_{i-1} & k \notin FV(t) \\
(\mathcal{S}-lift) & \langle E^{j-1}_v[\mathcal{S}_j k.t]\rangle_i & = \langle t\left\{\lambda x.\langle E^{j-1}_v[x]\rangle_j/k\right\}\rangle_i & x \notin FV(kE^{j-1}_v)
\end{array}
}
$$

Theorem 13. *For any $n \in \omega$, CBV $\lambda\mu\widehat{\mathsf{tp}}_n$ simulates $\lambda\mathcal{S}_n$: let $t, u \in \Sigma^c_{\lambda\mathcal{S}_n}$, $t =_{\lambda\mathcal{S}_n} u \Rightarrow |t|^{\mathcal{S}\rangle\widehat{\mathsf{tp}}} =_{\lambda\mu\widehat{\mathsf{tp}}_n^{cbv}} |u|^{\mathcal{S}\rangle\widehat{\mathsf{tp}}}$.*

Remark 14. *If we have only implication $\Rightarrow$ and not the converse, it is solely because $\lambda\mu\widehat{\mathsf{tp}}_n$ makes use of structural substitution and thus that some reductions are anticipated in $\lambda\mu\widehat{\mathsf{tp}}_n$ compared to the reduction in $\lambda\mathcal{S}_n$. This already occurs at the first level of the hierarchy [16] and is analyzed in [2].*

7 Conclusion

In this paper we introduced a new hierarchy of calculi, the $(\Lambda^n)_{n\in\omega}$-calculi, called the *stream hierarchy*. This hierarchy generalizes both λ-calculus and $\Lambda\mu$-calculus. $(\Lambda^n)_{n\in\omega}$-calculi have layered, or hierarchical, abstractions as well as variables with levels and its reduction system naturally extends the one for $\Lambda\mu$-calculus. The main related works are the CBV studies of delimited continuations and of the CPS hierarchiy and most notably works by Danvy, Filinski, Hasegawa and Kameyama [5,8,13,18,19] and the works on CBN delimited control by Ghilezan, Herbelin and Kiselyov [16,21]. The main results of the paper are:

- Church-Rosser and Böhm theorem for the hierarchy which ensures that the hierarchy is well-structured;
- sound and complete CPS translations for the hierarchy. The completeness proof strongly relies on conservativity results between different layers of the hierarchy allowing for simpler completeness proofs compared to more traditional translations as Fujita's CPS adapted to $\Lambda\mu$-calculus;
- an operational semantics for the hierarchy obtained by constructing abstract machines, the Λ^n-KAM, inspired from Krivine abstract machine for λ-calculus. The Λ^n-KAMs compute Λ^n-head normal forms;
- finally, we established that the stream hierarchy is indeed a hierarchy of delimited continuations in call-by-name, by mediating between the CPS hierarchy and the stream hierarchy thanks to the $\lambda\mu\widehat{\mathsf{tp}}_n$-calculi.

As a conclusion, we have developed a(n almost) complete study of the stream hierarchy. Our contribution evidences that the Stream hierarchy is a CBN hierarchy of delimited continuations and that fruitful connections exist between delimited control and infinitary calculi which underly $\Lambda\mu$-calculus and the entire stream hierarchy. However, some more developments are still to be done, which are left for future work:

- the CPS translations for the hierarchy can be used for a semantical study of the hierarchy. However, we are also interested in developping Böhm tree semantics for $\Lambda\mu$-calculus and the stream hierarchy;
- the CPS translations and the abstract machines considered in this paper have many similarities. It would be of interest to study how the abstract machines can be generated from the CPS semantics;
- the Λ^n-KAM has a structure (states and reductions) very similar to abstract machines for the CPS hierarchy [8,5]. We shall make this relation clear;

- we developed an untyped study of the stream hierarchy but a typed study of the hierarchy would also be of interest;
- the stream hierarchy that we considered here is indexed by ω. However, it can straightforwardly be made more general by indexing the hierarchy by a larger ordinal while preserving most results. We limited our presentation to ω for two reasons: for simplicity, first, but also because the CPS hierarchy is itself limited to ω. We conjecture that the CPS hierarchy can as well be extended above ω which could actually be interesting for several applications of the hierarchy where it might be of interest to have a delimiter that can delimit an infinite number of different `shift` operators;
- the Stream interpretation of $\Lambda\mu$-calculus and the links with infinitary calculi have been very influential. We shall develop these directions in future works. See [35] for some early developments.

Finally, we think that the ability to develop the stream hierarchy as a natural generalization of $\Lambda\mu$-calculus is a hint of the fact that $\Lambda\mu$-calculus is a calculus with a strong structure: this hierarchical extension could not have been developed based on Parigot's syntax for instance (but for adding a dynamically bound variable as we did with $\lambda\mu\widehat{\mathtt{tp}}_n$-calculi).

Acknowledgments. The author wishes to thank Hugo Herbelin, Luca Paolini, Mauro Piccolo, Kazushige Terui and Simona Ronchi della Rocca for helpful discussions and fruitful comments as well as the anonymous reviewers.

References

1. Ariola, Z., Herbelin, H.: Minimal classical logic and control operators. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719. Springer, Heidelberg (2003)
2. Ariola, Z., Herbelin, H.: Control Reduction Theories: the Benefit of Structural Substitution. In: JFP. Includes a Historical Note by Matthias Felleisen (2007)
3. Berarducci, A.: Infinite λ-calculus and non-sensible models. In: Logic and Algebra 1994. Lect. Notes in Pure and App. Math. Series, vol. 180. Marcel Dekker, New York (1996)
4. Berarducci, A., Dezani, M.: Infinite λ-calculus and types. TCS, 212 (1999)
5. Biernacka, M., Biernacki, D., Danvy, O.: An operational foundation for delimited continuations in the CPS hierarchy. Logical Meth. in Comp. Science 1(2) (2005)
6. Böhm, C.: Alcune proprietà delle forme $\beta\eta$-normali nel λK-calcolo. Publicazioni dell'Istituto per le Applicazioni del Calcolo 696 (1968)
7. Danvy, O., Filinski, A.: Abstracting control. In: LISP and Funct. Prog. (1990)
8. Danvy, O., Yang, Z.: An operational investigation of the CPS hierarchy. In: Swierstra, S.D. (ed.) ESOP 1999. LNCS, vol. 1576, pp. 224–242. Springer, Heidelberg (1999)
9. David, R., Py, W.: $\lambda\mu$-calculus and Böhm's theorem. J. of Symb. Logic (2001)
10. de Groote, P.: A CPS-translation of the $\lambda\mu$-calculus. In: Tison, S. (ed.) CAAP 1994. LNCS, vol. 787, pp. 85–99. Springer, Heidelberg (1994)
11. de Groote, P.: An environment machine for the λ-calculus. MSCS 8 (1998)

12. Felleisen, M., Friedman, D.P., Kohlbecker, E.E., Duba, B.F.: A syntactic theory of sequential control. TCS 52, 205–237 (1987)
13. Filinski, A.: Representing monads. In: POPL 1994, pp. 446–457. ACM, New York (1994)
14. Fujita, K.: A sound and complete cps-translation for λ-calculus. In: TLCA (2003)
15. Griffin, T.: A formulae-as-types notion of control. In: POPL. IEEE, Los Alamitos (1990)
16. Herbelin, H., Ghilezan, S.: An approach to CBN delimited continuations. In: POPL. ACM Sigplan, New York (2008)
17. Howard, W.A.: The formulae-as-type notion of construction, 1969. In: Essays in Comb. Logic, λ-Calculus, and Formalism, pp. 479–490. Academic Press, London (1980)
18. Kameyama, Y., Hasegawa, M.: A Sound and Complete Axiomatization of Delimited Continuations. In: ICFP 2003, pp. 177–188. SIGPLAN Notices (2003)
19. Kameyama, Y.: Axioms for control operators in the cps hierarchy. In: HOSC (2007)
20. Kennaway, R., Klop, J.W., Sleep, M.R., de Vries, F.-J.: Infinitary lambda calculus. TCS 175(1), 93–125 (1997)
21. Kiselyov, O.: Call-by-name linguistic side effects. In: ESSLLI 2008 Workshop on Symmetric calculi and Ludics for the semantic interpretation (2008)
22. Krivine, J.-L.: Lambda-calculus, Types and Models. Ellis Horwood (1993)
23. Krivine, J.-L.: A call-by-name lambda-calculus machine. In: HOSC (2005)
24. Lafont, Y., Reus, B., Streicher, T.: Continuations semantics or expressing implication by negation. Tech. Rep. 9321, Ludwig-Maximilians-Universität (1993)
25. Lassen, S.: Head normal form bisimulation for pairs and the λ-calculus. In: Logic In Computer Science. IEEE Computer Society Press, Los Alamitos (2006)
26. Parigot, M.: Free deduction: An analysis of "computations" in classical logic. In: Voronkov, A. (ed.) RCLP 1990 and RCLP 1991. LNCS (LNAI), vol. 592, pp. 361–380. Springer, Heidelberg (1992)
27. Parigot, M.: $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In: Voronkov, A. (ed.) LPAR 1992. LNCS, vol. 624. Springer, Heidelberg (1992)
28. Parigot, M.: Proofs of strong normalisation for second order classical natural deduction. Journal of Symbolic Logic 62(4), 1461–1479 (1997)
29. Py, W.: Confluence en $\lambda\mu$-calcul. PhD thesis, Université de Savoie (1998)
30. Saurin, A.: A hierarchy for delimited continuations in call-by-name. long version at, `http://www.pps.jussieu.fr/~saurin/Publi/LM_hierarchy_long.pdf`
31. Saurin, A.: Separation with streams in the $\Lambda\mu$-calculus. In: LICS. IEEE, Los Alamitos (2005)
32. Saurin, A.: On the relations between the syntactic theories of $\lambda\mu$-calculi. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 154–168. Springer, Heidelberg (2008)
33. Saurin, A.: Une étude logique du contrôle, appliquée à la programmation fonctionnelle et logique. PhD thesis, École Polytechnique (September 2008)
34. Saurin, A.: Typing streams in the $\Lambda\mu$-calculus. ACM ToCL (to appear)
35. Saurin, A.: Standardization and Böhm trees for $\Lambda\mu$-calculus. In: FLOPS 2010. LNCS, vol. 6009. Springer, Heidelberg (to appear, 2010)
36. Streicher, T., Reus, B.: Classical logic, continuation semantics and abstract machines. Journal of Functional Programming 8(6), 543–572 (1998)

Author Index